ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

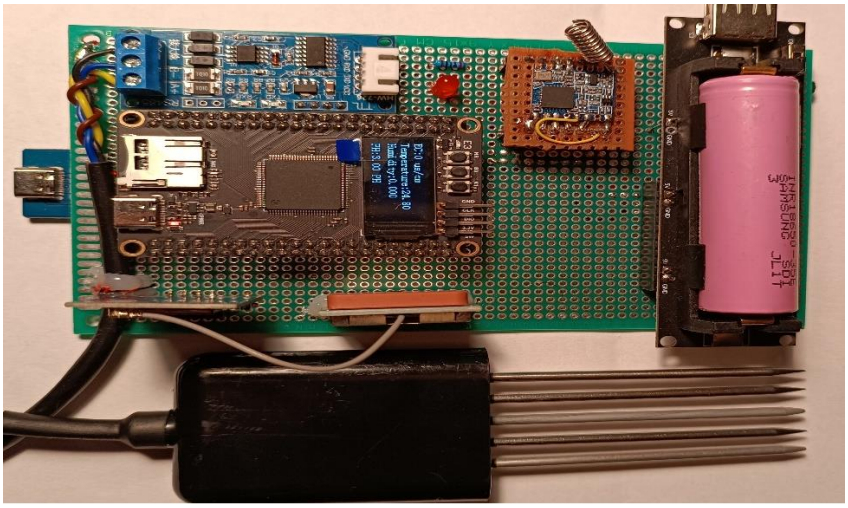
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΕΦΑΡΜΟΣΜΕΝΑ ΗΛΕΚΤΡΟΝΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«ΑΝΑΛΥΤΗΣ ΕΔΑΦΟΛΟΓΙΚΩΝ ΘΡΕΠΤΙΚΩΝ ΣΥΣΤΑΤΙΚΩΝ
ΒΑΣΙΖΟΜΕΝΟΣ ΣΕ ΜΙΚΡΟΕΛΕΓΚΤΗ ARM»



Του φοιτητή
Κολάση Χρυσοβέργη-Γρηγόριου
Αρ. Μητρώου: 520012Μ

Επιβλέπων
Γιακουμής Άγγελος
Επίκουρος Καθηγητής

Ημερομηνία 26/02/2024

Αναλυτής εδαφολογικών θρεπτικών συστατικών βασιζόμενος σε μικροελεγκτή ARM
Κωδικός Δ.Ε. 21156
Ονοματεπώνυμο φοιτητή: Κολάσης Χρυσοβέργης-Γρηγόριος
Ονοματεπώνυμο εισηγητή: Γιακουμής Άγγελος
Ημερομηνία ανάληψης Δ.Ε.: 24-02-2021
Ημερομηνία περάτωσης Δ.Ε.: 28-02-2024

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

«Αφιέρωση»

Θα ήθελα να αφιερώσω αυτήν την διπλωματική εργασία στην αγαπημένη μου σύντροφο Βασιλική, για την υπομονή της και την στήριξη σε όλη αυτήν την απαιτητική ακαδημαϊκή μου πορεία.

Πρόλογος

Η επιλογή του θέματος της διπλωματικής εργασίας ενός φοιτητή αδιαμφισβήτητα παίζει καθοριστικό ρόλο στην βελτίωση των ικανοτήτων του στην έρευνα υπάρχουσας τεχνολογίας, στην ανάπτυξη συσκευών ή λογισμικού για την λύση σύνθετων προβλημάτων. Η παρούσα εργασία ασχολείται με κλάδο της Γεωπονίας και πιο συγκεκριμένα με την μέτρηση συστατικών και ιδιοτήτων του εδάφους. Το έδαφος και κατ' επέκταση ό,τι καλλιεργείται σε αυτό παίζει καθοριστικό ρόλο στην ποιότητα της τροφής των ζώων και ως επακόλουθο στην ανθρώπινη τροφή. Έχοντας ως κριτήριο την σημαντικότητα του εδάφους και των καλλιεργειών στις ζωές μας η παρούσα εργασία αποσκοπεί στην μέτρηση των συστατικών του εδάφους με την βοήθεια της τεχνολογίας. Η ενασχόληση με τον μικροελεγκτή STM32 αρχιτεκτονικής ARM και ο προγραμματισμός του σε περιβάλλον ανάπτυξης CUBE IDE ήταν κάτι πολύ απαιτητικό και ταυτόχρονα εποικοδομητικό διότι προϋποθέτει βαθιά κατανόηση των λειτουργιών του για την επιτυχημένη ανάπτυξη του προγράμματος. Η εμβάθυνση στις τεχνολογίες LoRa, GPS, RS45 που αναπτύχθηκαν στην συσκευή καθώς και η γλώσσα Python, τα Websockets, η USB επικοινωνία, οι χάρτες Google Maps και η βάση δεδομένων MySQL που χρησιμοποιήθηκαν στο λογισμικό υψηλού επιπέδου, εξέλιξαν τις δεξιότητες μου.

Περίληψη

Στην παρούσα διπλωματική εργασία γίνεται μελέτη, σχεδίαση και κατασκευή μια ολοκληρωμένης συσκευής που μετράει τα συστατικά του εδάφους. Η εργασία αποτελείται από την έρευνα μεθόδων μέτρησης εδάφους , την επιλογή αισθητήρα από το εμπόριο που στηρίζεται στην μέθοδο μέτρησης TDR, την κατασκευή του πρωτοτύπου της συσκευής μέτρησης και την ασύρματη μεταφορά των δεδομένων μέσω LoRa σε έναν δέκτη ESP32. Ο πυρήνας της συσκευής είναι ο μικροελεγκτής STM32H750, που επικοινωνεί με το NEO-6M GPS για την απόκτηση της γεωγραφικής θέσης, μέσω RS485 επικοινωνεί με τον αισθητήρα μέτρησης και μέσω SPI με το τσιπ LoRa SX1276. Οι τιμές της μέτρησης μεταφέρονται μέσω του δέκτη στον υπολογιστή μέσω USB επικοινωνίας. Ο υπολογιστής τρέχει το κεντρικό λογισμικό υψηλού επιπέδου application script σε γλώσσα python. Το script αυτό λαμβάνει τα δεδομένα μέσω USB τα αποθηκεύει σε βάση δεδομένων MySQL και τα στέλνει μέσω Websockets σε ένα web application τοπικά στον υπολογιστή. Το web app αυτό είναι ένας χάρτης που αναπτύχθηκε βασιζόμενος στο API της Google και περιέχει όλες τις μετρήσεις και τις γεωγραφικές τους θέσεις τοποθετημένες σαν πινέζες. Επιπλέον, διεξάγονται πειράματα για την ορθή λειτουργία του αισθητήρα και αναλύονται τα συμπεράσματα που προέκυψαν καθώς και τρόποι βελτίωσης συσκευής και λογισμικού για περαιτέρω ανάπτυξη.

«SOIL NUTRIENT ANALYZER BASED ON ARM MICROCONTROLLER»

«Kolasis Chrysovergis-Grigorios»

Abstract

In this thesis, the study, design and construction of an integrated device that measures soil components is carried out. The work consists of surveying ground measurement methods and selecting a commercially available sensor based on the TDR measurement. The construction of the measuring device prototype, the wireless transfer of the data via LoRa to an ESP32 receiver. The heart of the device is the STM32H750 microcontroller, where it communicates with the NEO-6M GPS to obtain the geographic position, communicates via RS485 with the measuring sensor and via SPI with the LoRa SX1276 chip. The measurement values are transferred via the receiver to the computer via USB communication. The computer runs the main high-level application script in python language. This script receives the data via USB, stores it in a MySQL database and sends it via Websockets to a web application locally on the computer. This web app is a map developed based on Google's API and contains all measurements and their geographic locations placed like markers. Experiments are conducted for the proper functioning of the sensor and the conclusions drawn are analyzed as well as ways to improve the device and software for further development.

Ευχαριστίες

Θα ήθελα να εκφράσω την βαθύτατη ευγνωμοσύνη μου στην οικογένεια μου καθώς και τον επιβλέποντα καθηγητή της διπλωματικής μου εργασίας τον κ. Γιακουμή.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
IoT	Internet of Things
AI	Artificial Intelligence
ML	Machine Learning
EC	Electrical Conductivity
TDR	Time Domain Reflectometry
TDS	Total Dissolved Solids
NPK	Nitrogen Phosphorus Potassium
HVAC	Heating Ventilation and Air Conditioning
ASCII	American Standard Code for Information Interchange
RTU	Remote Terminal Unit
TCP	Transmission Control Protocol
IP	Internet Protocol
CRC	Cyclic Redundancy Check
LORA	Long Range
GPS	Global Positioning System
USB	Universal Serial Bus
HTTP	Hypertext Transfer Protocol
SQL	Structured Query Language
CSS	Cascading Style Sheets
UART	Universal Asynchronous Receiver Transmitter
SPI	Serial Peripheral Interface
PCB	Printed Circuit Board

Κεφάλαιο 1ο: Έδαφος και καλλιέργεια

1.1 Εισαγωγή

Στο πρώτο εισαγωγικό κεφάλαιο της εργασίας , γίνεται έρευνα για την έξυπνη γεωργία και τις εφαρμογές στης στο πεδίο. Έπειτα γίνεται αναφορά στο έδαφος και την βασική εδαφική ανάλυση που μετράει τα βασικότερα συστατικά του εδάφους όπως είναι το pH , η ηλεκτρική αγωγιμότητα, το άζωτο , ο φώσφορος και το κάλιο. Τέλος , αναλύονται διάφοροι μέθοδοι μέτρησης αυτών των στοιχείων από ήδη υπάρχων μελέτες και επιλέγεται ο τρόπος μέτρησης που χρησιμοποιεί η συσκευή καθώς και οι στόχοι της εργασίας.

1.2 Έξυπνη Γεωργία

Ο όρος έξυπνη γεωργία, γνωστός και ως Smart Farming, αναφέρεται στην υιοθέτηση προηγμένων τεχνολογιών και λειτουργιών που βασίζονται σε συλλογή και ανάλυση δεδομένων για τη βελτιστοποίηση της γεωργικής παραγωγής σε αντίθεση με τις αρχικές γεωργικές πρακτικές που επικεντρώνονταν εξ ολοκλήρου στη χρήση ανθρώπινου δυναμικού και απλών εργαλείων. Στο σύνολο τους τα συστήματα που υποστηρίζουν την έξυπνη γεωργία βασίζονται στη συγκέντρωση και μετάδοση δεδομένων σε συστήματα απομακρυσμένης αποθήκευσης για να επιτραπεί ο συνδυασμός και η ανάλυση τους με σκοπό την αξιοποίηση τους για τη λήψη αποφάσεων. Τα σύγχρονα συστήματα Έξυπνης Γεωργίας τροφοδοτούνται από προηγμένες τεχνολογίες που φέρνουν επανάσταση στη γεωργική παραγωγή σε μεγάλες και μικρές αγροτικές επιχειρήσεις και μπορούν να καταταχθούν στις παρακάτω γενικές κατηγορίες:

Internet of Things (IoT)

Η κατηγορία αυτή αναφέρεται σε ένα δίκτυο συσκευών, οχημάτων και λοιπών φυσικών αντικειμένων που διαθέτουν ενσωματωμένους αισθητήρες και λογισμικό, καθώς και συνδεσιμότητα δικτύου που τους επιτρέπει να συλλέγουν δεδομένα. Ο αισθητήρας είναι μια ηλεκτροτεχνική συσκευή που μετρά φυσικές ποσότητες από το περιβάλλον, όπως θερμοκρασία, υγρασία, παρουσία ή απουσία ορισμένων στοιχείων, και μετατρέπει αυτές τις μετρήσεις σε σήμα που μπορεί να διαβαστεί από ένα όργανο. Στην περίπτωση της έξυπνης γεωργίας, οι συνδεδεμένες συσκευές IoT περιλαμβάνουν πολλά είδη αισθητήρων (smart sensors) όπως μεταξύ άλλων αυτούς που χρησιμοποιούνται για την παρακολούθηση των καλλιεργειών ή την παρατήρηση της κατάστασης του αγροτικού εξοπλισμού καθώς και μη επανδρωμένα εναέρια οχήματα (UAV) ή drones εξοπλισμένα με LiDAR που συλλέγουν δεδομένα μέσω τηλεπισκόπησης. Με τη χρήση των παραπάνω μέσων δυνατή η εφαρμογή της γεωργίας ακριβείας (Precision Farming). Ο όρος αυτός χρονολογείται τουλάχιστον στις αρχές του 1980 με το έργο του Dr. Pierre Robert, ο οποίος ερεύνησε τη μεταβλητότητα του εδάφους και την ιδέα ότι τα διαφορετικά μέρη ενός χωραφιού απαιτούν διαφορετικά επίπεδα θρεπτικών ουσιών για να επιτρέψουν τη βέλτιστη απόδοση των καλλιεργειών. Στη δεκαετία του 1990 σημειώθηκε ένα άλλο άλμα προς τα εμπρός στην τεχνολογία που χρησιμοποιούν οι αγροτικές επιχειρήσεις με την εφεύρεση της ψηφιακής επιτήρησης απόδοσης καλλιεργειών και την αυξανόμενη χρήση των παγκόσμιων συστημάτων εντοπισμού θέσης (GPS) που βασίζονται σε δορυφόρους. Ο συνδυασμός των δεδομένων επιτήρησης απόδοσης με τη

χαρτογράφηση GPS επέτρεψε τη χαρτογράφηση απόδοσης, η οποία παρείχε σημαντικές πληροφορίες σχετικά με τα χαρακτηριστικά και την ποιότητα των καλλιεργειών σε πραγματικό χρόνο, κατά τη συγκομιδή τους.

Σχήμα 1.1 Χρήση IoT στα πλαίσια του Smart Farming

Τεχνητή νοημοσύνη (AI) και μηχανική μάθηση (ML)

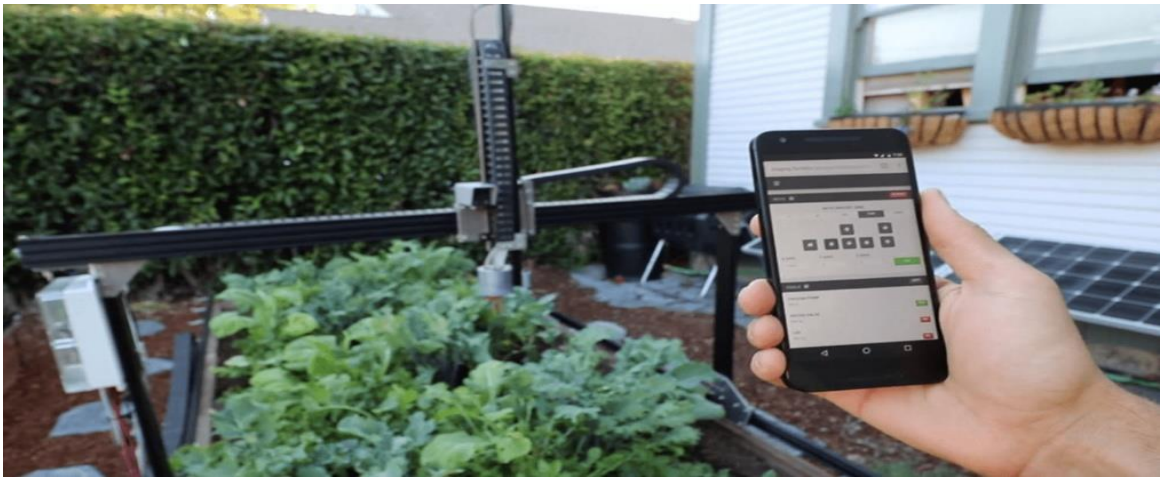
Η τεχνητή νοημοσύνη και η μηχανική μάθηση μπορούν να βοηθήσουν τους αγρότες να αντλήσουν πληροφορίες από μεγάλα και πολύπλοκα σύνολα δεδομένων – τα λεγόμενα (Big Data) – που προέρχονται από εφαρμογές IoT. Η ανάλυση δεδομένων και η μοντελοποίηση μέσω cloud-based AI και μηχανικής μάθησης μπορούν να συμβάλλουν στη λήψη αποφάσεων και στην ανάπτυξη τεχνικών έξυπνης γεωργίας. Τα παραπάνω δεδομένα μπορούν να βοηθήσουν τη γεωργική βιομηχανία να διαχειριστεί πιο εύκολα και πιο αποδοτικά τη διαδικασία γεωργικής παραγωγής, συμπεριλαμβανομένης της συγκομιδής, της εκμετάλλευσης του εκάστοτε αγροτεμαχίου και του σχεδιασμού της αλυσίδας εφοδιασμού.

Αυτοματισμός και ρομποτική

Ο αυτοματισμός και η ρομποτική κατέχουν εξέχουσα θέση στις σύγχρονες πρακτικές έξυπνης γεωργίας. Τα αυτόνομα τρακτέρ αποτελούν σημαντικό παράδειγμα στη συγκεκριμένη κατηγορία. Το τρακτέρ αυτόνομης οδήγησης γεννήθηκε από μια συνεργασία μεταξύ της εταιρείας αγροτικού εξοπλισμού John Deere και της NASA στις αρχές της δεκαετίας του 2000. Η συγκεκριμένη καινοτομία, εφόσον είχε καταχωρηθεί από τον αγρότη η ακριβής τοποθεσία του χωραφιού, είτε μέσω αισθητήρων είτε οδηγώντας το τρακτέρ στην περίμετρό του, παρείχε τη δυνατότητα αυτόματου προγραμματισμού της ακριβούς διαδρομής για το όργωμα του χωραφιού και αυτόνομου ελέγχου του τιμονιού (σύστημα AutoTrac της John Deere).

Σχήμα 1.2. Το πρώτο πλήρως αυτόνομο τρακτέρ της John Deere

Τα τελευταία χρόνια , όμως, παρουσιάστηκε από την ίδια εταιρεία και ένα πλήρως αυτόνομο τρακτέρ με τη δυνατότητα απουσίας του αγρότη από την καμπίνα του και ελέγχου της λειτουργίας του αποκλειστικά μέσω smartphone. Πιο συγκεκριμένα, το πεδίο καταγράφεται από 6 κάμερες και από τα δεδομένα που προκύπτουν το μηχάνημα κάνει τους απαραίτητους χειρισμούς για την αποφυγή εμποδίων. Ωστόσο, το συγκεκριμένο σύστημα αυτονομίας καλύπτει τις ανάγκες του οργώματος για την προετοιμασία του εδάφους για τη σπορά, και όχι παντός είδους γεωργική εργασία. Εκτός από αυτόνομα τρακτέρ, στην κατηγορία αυτή περιλαμβάνονται και η χρήση ρομπότ (Farmbot) για γεωργικές εργασίες όπως σπορά, συγκομιδή και κλάδεμα καθώς και η χρήση των UAV (Farmdrones) , τα οποία χρησιμοποιούνται συνήθως για συλλογή δεδομένων και για τον ψεκασμό λιπασμάτων και φυτοφάρμακων με τρόπο πιο ακριβή και περιορισμένο σε σχέση με τις παραδοσιακές τεχνικές, γεγονός που συμβάλλει στην μείωση των περιβαλλοντικών επιπτώσεων του ψεκασμού λιπασμάτων.



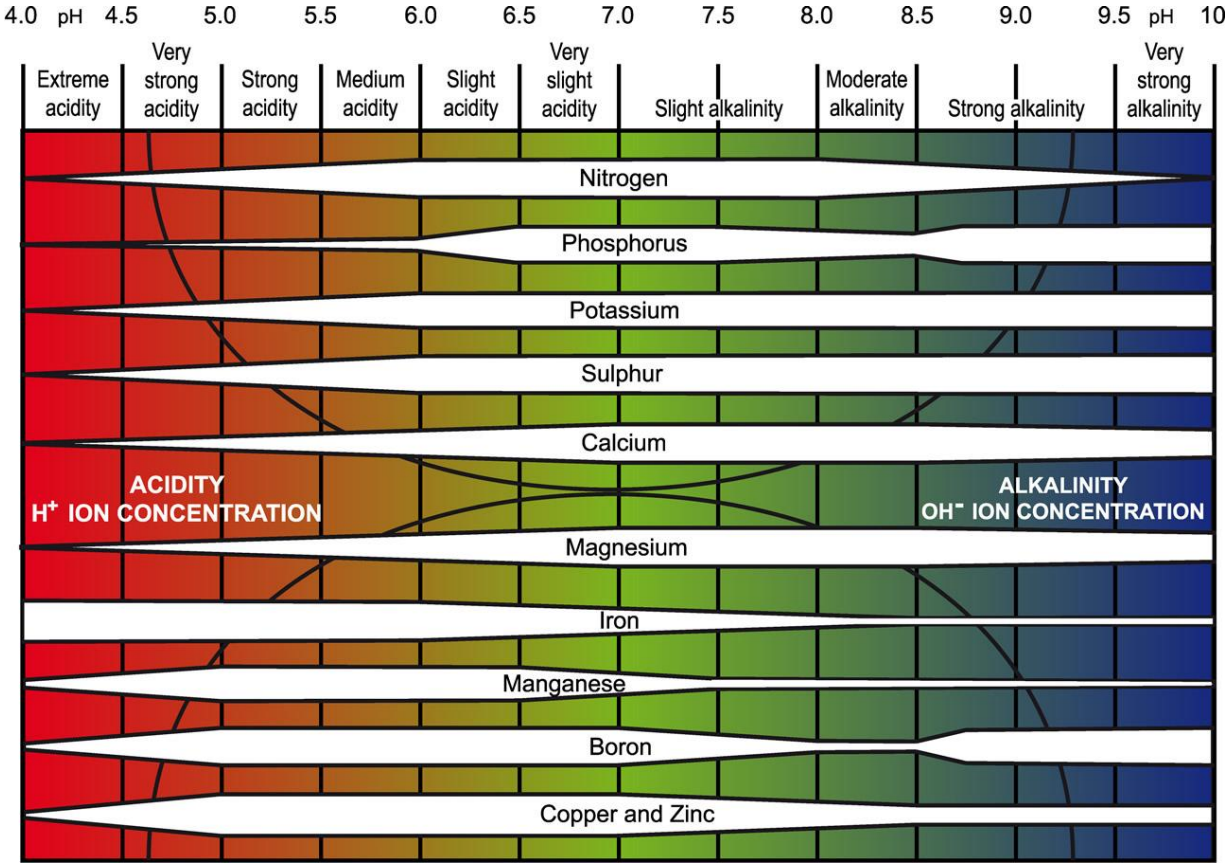
Σχήμα 1.3. Χρήση ρομπότ στην Έξυπνη Γεωργία (Farmbot)

Σχήμα 1.4. Χρήση UAV (Farmdrone) στην Έξυπνη Γεωργία

Ο τομέας της Έξυπνης Γεωργίας είναι διαρκώς εξελισσόμενος, ικανός να διασφαλίσει τη βιωσιμότητα της πρωτογενούς παραγωγής καθώς απευθύνεται σε μικρές, μεσαίες και μεγάλες αγροτικές επιχειρήσεις μειώνοντας το ρίσκο και αυξάνοντας το κέρδος και την ανταγωνιστικότητά τους. Πέραν αυτού, δύναται να συμβάλει και στον ευρύτερο στόχο της κάλυψης της αυξανόμενης ζήτησης για τρόφιμα με βάση μια πιο ακριβή και αποδοτική από πλευράς πόρων προσέγγιση για τη διαχείριση των καλλιεργειών.

1.3 Βασική Εδαφολογική Ανάλυση

Ένας ιδιαίτερα σημαντικός παράγοντας προκειμένου να εξασφαλιστεί η καλύτερη δυνατή διαχείριση των καλλιεργειών από τους αγρότες και να ενισχυθεί η γονιμότητα και αποδοτικότητα των εδαφών είναι η ανάλυση της σύστασης και των φυσικοχημικών ιδιοτήτων του εδάφους. Στα πλαίσια της Βασικής Εδαφολογικής Ανάλυσης μετριέται το pH, η ηλεκτρική αγωγιμότητα (Electrical Conductivity - EC) καθώς και η περιεκτικότητα του εδάφους στα κύρια θρεπτικά συστατικά αυτού που είναι τα άζωτο (N), φώσφορος (P) και κάλιο (K). Πρέπει να σημειωθεί ότι τα θρεπτικά στοιχεία βρίσκονται στο εδαφικό διάλυμα με τη μορφή ιόντων και προσλαμβάνονται μέσω του ριζικού συστήματος των φυτών συμβάλλοντας στην ανάπτυξη τους. Παρακάτω αναλύονται εκτενέστερα οι παραπάνω παράγοντες και η επίδραση τους στο έδαφος και στην ανάπτυξη των φυτών.



Σχήμα 1.5. Επίδραση του ακραίου pH στις απορροφήσεις μακροστοιχείων και μικροστοιχείων των φυτών από το έδαφος

1. Επίδραση pH.

Το pH του εδάφους μετράται στο υδατικό του εκχύλισμα και επιδρά στην ανάπτυξη των φυτών μέσω επίδρασης στη λειτουργία του ριζικού συστήματος καθώς και στις χημικές ιδιότητες αυτού. Από τους πιο σημαντικούς παράγοντες που επηρεάζουν το pH του εδάφους είναι τα μητρικά πετρώματα, τα αρχικά πετρώματα δηλαδή από τα οποία έχει προκύψει το έδαφος, καθώς και οι κλιματολογικές συνθήκες, όπως οι βροχοπτώσεις και τα επίπεδα θερμοκρασίας. Πιο συγκεκριμένα, το pH του νερού της βροχής συνήθως κυμαίνεται γύρω στο 5,6 λόγω του ατμοσφαιρικού CO_2 που αντιδρά με το H_2O και δίνει ανθρακικό οξύ (H_2CO_3). Όταν αυτό καταλήγει στο έδαφος προκαλεί την έκπλυση των θετικών

κατιόντων , όπως είναι τα ιόντα ασβεστίου και μαγνησίου, ως διττανθρακικά HCO_3^-. Επίσης, η όξινη βροχή περιέχει νιτρικό και θειικό οξύ που προέρχονται από οξείδια του θείου και του αζώτου που απελευθερώνονται στην ατμόσφαιρα μέσω της καύσης ορυκτών καυσίμων. Το αδιάλυτο νιτρικό και θειικό οξύ καταλήγουν στο έδαφος και απορροφώνται από τα βαθύτερα στρώματα αυτού προκαλώντας την οξίνιση τους εδάφους. Τα παραπάνω δικαιολογούν την παρατήρηση ότι τα εδάφη που δέχονται έντονες βροχοπτώσεις έχουν όξινο pH σε αντίθεση με πιο ξηρές περιοχές στις οποίες παρατηρείται αλκαλικό pH καθώς δεν υπάρχει αρκετό νερό για την έκπλυση των διαλυτών αλάτων. Πέραν των παραπάνω υπάρχουν και άλλοι πολύπλευροι παράγοντες που επιδρούν στο pH του εδάφους καθώς επίσης είναι ξεκάθαρο ότι κάθε μεταβολή του pH επιδρά στη διαθεσιμότητα των θρεπτικών στοιχείων, αλλά και στη συγκέντρωση βαρέων μετάλλων και λοιπών στοιχείων με πιθανή επιζήμια δράση στη θρέψη των φυτών. Όλα τα παραπάνω οδηγούν στο συμπέρασμα ότι η μέτρηση του pH του εδάφους είναι σημαντική για τους αγρότες προκειμένου να λάβουν αποφάσεις σχετικά με τη διαχείριση των αγροτεμαχίων σχετικά με την προσθήκη λιπασμάτων και θρεπτικών ουσιών για τη δημιουργία του βέλτιστου περιβάλλοντος για την ανάπτυξη της επιθυμητής καλλιέργειας αλλά και την υπόδειξη πιθανών καλλιεργειών που μπορούν να ευδοκιμήσουν. Γι' αυτό λοιπόν στα πλαίσια της γεωργίας ακρίβειας θα πρέπει να γίνεται διαρκώς έλεγχος της τιμής του PH και ταυτόχρονα βελτίωσή της ώστε να επιτυγχάνεται άριστη ποιότητα και καλύτερη δυνατή γονιμότητα.

2. Επίδραση ηλεκτρικής αγωγιμότητας (Electrical Conductivity-EC).

Η ηλεκτρική αγωγιμότητα αποτελεί σημαντική χημική ιδιότητα τους εδάφους καθώς αποτελεί ένδειξη παρουσίας θρεπτικών στοιχείων σε αυτό και παρέχει τη δυνατότητα προσδιορισμού της αλατότητας του. Πιο συγκεκριμένα, υψηλές τιμές ηλεκτρικής αγωγιμότητας συνεπάγονται και αυξημένες συγκεντρώσεις αλάτων στο έδαφος. Η αυξημένη παρουσία αλάτων στο έδαφος είναι επιζήμια για την ανάπτυξη των φυτών που καλλιεργούνται σε αυτά και οι επιδράσεις αυτές συνοψίζονται ως εξής:

- Υδατική καταπόνηση φυτών. Τα φυτά προκειμένου να εξασφαλίσουν την ποσότητα νερού που απαιτείται για την κάλυψη των αναγκών τους πρέπει να αντλούν το νερό με μεγαλύτερη δύναμη από αυτήν που το συγκρατεί το έδαφος. Η παρουσία αλάτων στο εδαφικό διάλυμα αυξάνει αυτή την απαιτούμενη δύναμη (οσμωτικό δυναμικό) . Ουσιαστικά, όταν έχουμε δύο κατά τα άλλα όμοια εδάφη εκ των οποίων το ένα είναι αλατούχο ενώ το άλλο δεν είναι αλατούχο τα φυτά δύναται να αντλήσουν μεγαλύτερη ποσότητα νερού από το μη αλατούχο έδαφος. Ως εκ τούτου, φυτά που αναπτύσσονται σε αλατούχα εδάφη συχνά υποφέρουν από την έλλειψη του νερού και απαιτούν συχνότερη άρδευση.
- Επίδραση στους ρυθμούς αύξησης των φυτών (νανισμός φυτών). Τα φυτά που αναπτύσσονται σε αλατούχα εδάφη παρουσιάζουν μειωμένο ρυθμό ανάπτυξης κάτι το οποίο δεν είναι ορατό στην περίπτωση που δεν μπορεί να συγκριθεί με την περίπτωση ανάπτυξης σε μη αλατούχο έδαφος.
- Σε ορισμένα είδη φυτών η επιζήμια δράση των αλάτων γίνεται αντιληπτή μέσω εμφάνισής συνηθέστερα κοκκινοπράσινου χρώματος στα φύλλα και κηρώδους επίστρωσης.
- Επιζήμια επίδραση παρατηρείται και σε μηχανισμούς του φυτού και ειδικότερα σε αυτούς της φωτοσύνθεσης .

Είναι σαφές λοιπόν ότι οι υψηλές συγκεντρώσεις αλάτων στο έδαφος υποβαθμίζουν τη γονιμότητά του και περιορίζουν τη δυνατότητα καλλιέργειας ορισμένων φυτών όπως το φασόλι, το καλαμπόκι, το λινάρι και αλλά φυτά που είναι ευαίσθητα στην αλατότητα. Η αλάτωση των εδαφών εξαρτάται σημαντικά από τη γεωγραφική τους θέση με παράδειγμα τα δέλτα ποταμών και παραθαλάσσιες

περιοχές που πλημυρίζουν εποχικά καθώς και από τη σύσταση του νερού άρδευσης. Πρακτικά, η μέτρηση της ηλεκτρικής αγωγιμότητας (ECe) πραγματοποιείται στο εκχύλισμα του εδαφικού πολτού στους 25 °C και χρησιμοποιείται για την μέτρηση της αλατότητας. Η μέτρηση εκφράζεται σε decisiemens/m ή σε millisiemens/cm και βάση αυτής τα εδάφη κατατάσσονται χονδρικά σε αλατούχα εδάφη (ECe)> 2ds/m και σε μη αλατούχα εδάφη με (ECe)< 2ds/m.

3. Άζωτο και νιτρικά ιόντα.

Τα άζωτο εντάσσεται στα μακροστοιχεία του εδάφους και αποτελεί κατασταλτικό παράγοντα στην αύξηση και ανάπτυξη των φυτών. Ωστόσο, σε περιπτώσεις υψηλών συγκεντρώσεων το νιτρικό άζωτο επηρεάζει αρνητικά την ανάπτυξη των φυτών μέσω ανταγωνισμού με λοιπά μακροστοιχεία προκαλώντας την έλλειψή τους. Τα νιτρικά ιόντα και το άζωτο, λοιπόν, επηρεάζουν την βιολογική παραγωγικότητα, την ποιότητα του περιβάλλοντος αλλά και την ανθρώπινη υγιεινή καθώς είναι παρόντα στη διατροφή και το πόσιμο νερό. Η πολύπλευρη επίδραση του αζώτου, λοιπόν, καθιστά απαραίτητο τον προσδιορισμό του στα πλαίσια της γεωργίας ακριβείας με στόχο την ορθολογική χρήση των αζωτούχων λιπασμάτων.

4. Φώσφορος.

Ο φώσφορος (P) αποτελεί σημαντικό θρεπτικό στοιχείο του εδάφους και ανιχνεύεται σε μικρό ποσοστό σε αυτό είτε με την ανόργανη (σχηματισμός συμπλοκών με Al, Ca, Fe) είτε με την οργανική μορφή του (λιπίδια, νουκλεϊκά οξέα), η οποία και αποτελεί το 20-80% του συνολικού P. Η σημαντικότητα του P για το έδαφος έγκειται στη θετική του επίδραση στην ανάπτυξη των φυτών, στην ενδυνάμωση του ριζικού τους συστήματος, στην ανθεκτικότητα σε ασθένειες ενώ φαίνεται να αναστέλλει και τις επιπτώσεις της υπερβολικής χρήσης αζωτούχων λιπασμάτων. Η μικρή περιεκτικότητα του εδάφους σε P αλλά και η περιορισμένη διαλυτότητα των ενώσεων του P έχει ως αποτέλεσμα την μειωμένη διαθεσιμότητα του. Γενικά, η διαθεσιμότητα του φωσφόρου στο έδαφος εξαρτάται από τη συγκέντρωση του στο εδαφικό διάλυμα, από το μητρικά πετρώματα, από το κλίμα αλλά και από διάφορους φυσικοχημικούς παράγοντες του εδάφους όπως το pH. Όπως αναφέρθηκε το μεγαλύτερο ποσοστό του P είναι αδιάλυτο λόγω των ενώσεων που αυτό σχηματίζει και κατ΄ επέκταση μη απορροφήσιμο από τα φυτά. Αυτό καθιστά τον προσδιορισμό του σημαντικό κυρίως για να προσδιοριστεί η δυνατότητα του εδάφους να τροφοδοτεί με το απαραίτητο ποσοστό φωσφόρου προκειμένου να πραγματοποιηθεί λίπανση του εδάφους σε περίπτωση που απαιτείται αλλά και να αποφευχθεί υπερλίπανση σε περίπτωση επάρκειας η οποία είναι επιβλαβής σε περιβαλλοντικό επίπεδο.

5. Κάλιο.

Το κάλιο προσλαμβάνεται εύκολα από τους φυτικούς ιστούς από τα πρώτα στάδια της ανάπτυξής τους και συμμετέχει σε πλήθος διεργασιών των φυτών. Το κάλιο εμφανίζεται στο έδαφος σε διάφορες μορφές: υδατοδιαλυτό , ανταλλάξιμο, μη ανταλλάξιμο και δομικό κάλιο. Όλες αυτές οι μορφές του καλίου βρίσκονται σε μια δυναμική ισορροπία μεταξύ τους Η μορφή του καλίου που προσλαμβάνεται από τα φυτά είναι η υδατοδιαλυτή και η οποία αναπληρώνεται εύκολα από το ανταλλάξιμο κάλιο. Σε περίπτωση εξάντλησης του K είτε λόγω απορρόφησης του από τα φυτά είτε λόγω έκπλυσης , ποσότητα μη ανταλλάξιμου K μετατρέπεται σε ανταλλάξιμη. Από την άλλη, η μορφή του δομικού K, η οποία και αποτελεί το μεγαλύτερο ποσοστό του συνολικού εδαφικού καλίου, μπορεί να γίνει διαθέσιμη προς αφομοίωση από τα φυτά με πολύ αργούς ρυθμούς μέσω της διαδικασίας της χημικής αποσάθρωσης.

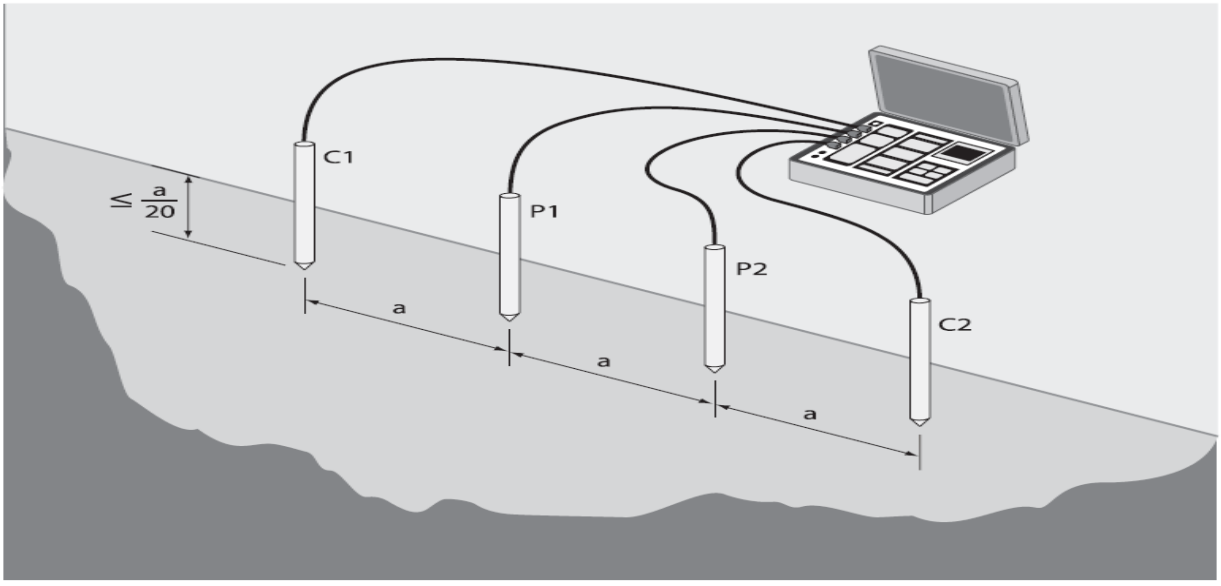
1.4 Μέτρηση εδάφους

Η μέτρηση της ηλεκτρικής αγωγιμότητας του εδάφους έχει απασχολήσει πολλούς στο παρελθόν και πολλοί έχουν πειραματιστεί με διάφορες μεθόδους. Η αγωγιμότητα είναι η αντίστροφη ιδιότητα από την αντίσταση. Δηλαδή είναι η ιδιότητα του πόσο εύκολα ρέει ηλεκτρικό ρεύμα μέσα από αυτό.

Ηλεκτρική αγωγιμότητα μέσω μέτρησης ηλεκτρικής αντίστασης

Η ηλεκτρική ειδική αντίσταση χρησιμοποιήθηκε αρχικά από τους γεωφυσικούς για τη μέτρηση της ειδικής αντίστασης του γεωλογικού υπεδάφους. Οι μέθοδοι ηλεκτρικής ειδικής αντίστασης περιλαμβάνουν τη μέτρηση της αντίσταση στη ροή ρεύματος στα τέσσερα ηλεκτρόδια που εισάγονται σε ευθεία γραμμή στην επιφάνεια του εδάφους με καθορισμένη απόσταση μεταξύ τους. Παρόλο που μπορούν να χρησιμοποιηθούν δύο ηλεκτρόδια, η σταθερότητα της ένδειξης βελτιώθηκε με τη χρήση τεσσάρων ηλεκτροδίων. Με δύο ηλεκτρόδια, οι μετρήσεις αποκρύπτονται από την πόλωση των ηλεκτροδίων που προκύπτει από την αλληλεπίδραση της επιφάνειας του φορτισμένου ηλεκτροδίου με ελεύθερα φορτία στον ηλεκτρολύτη. Τα τέσσερα ηλεκτρόδια μετριάζουν το πρόβλημα της πόλωσης των ηλεκτροδίων παρέχοντας ένα δεύτερο ζεύγος ηλεκτροδίων για τη μέτρηση της τάσης στο έδαφος. Όταν οι αποστάσεις των ηλεκτρόδιων είναι ίσες μεταξύ τους τότε η διάταξη με τα τέσσερα ηλεκτρόδια ονομάζεται ‘Wenner array’. Χρησιμοποιώντας τη διάταξη Wenner Array, τέσσερα ηλεκτρόδια τοποθετούνται σε ευθεία γραμμή σε διαστήματα του a, σε βάθος b. Ένα ρεύμα διέρχεται από τα δύο εξωτερικά ηλεκτρόδια C1,C2 και στη συνέχεια μετριέται η διαφορά δυναμικού μεταξύ των δύο εσωτερικών ηλεκτροδίων P1,P2. Μια απλή εξίσωση του νόμου του Ohm καθορίζει την αντίσταση. Από αυτές τις πληροφορίες, είναι πλέον δυνατός ο υπολογισμός της ειδικής αντίστασης του εδάφους. Για τις περισσότερες πρακτικές περιπτώσεις, το "a" είναι 20 φορές μεγαλύτερο από το "b", οπότε μπορεί να γίνει η υπόθεση ότι b=0 και η εξίσωση υπολογισμού της αντίστασης του εδάφους να μετατραπεί σε:

$$\rho = 2*\pi*a*Re$$

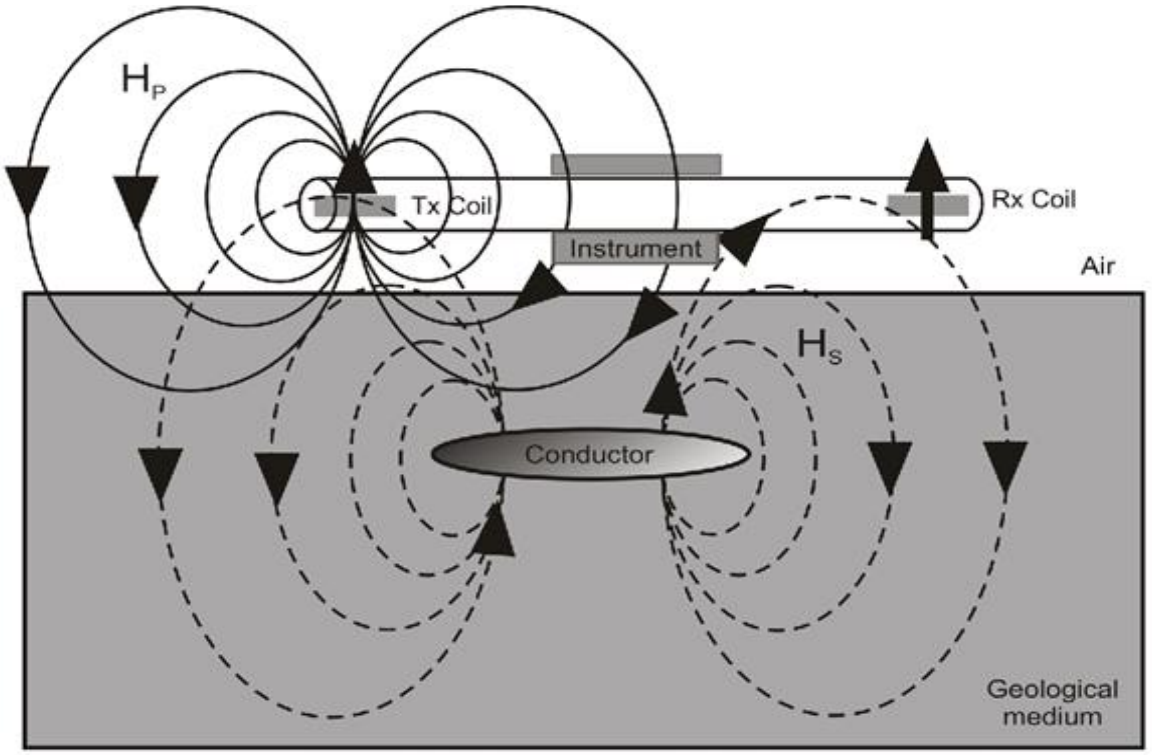


Σχήμα 1.6. Μέτρηση ηλεκτρικής αντίστασης εδάφους με την διάταξη Wenner Array

Η ηλεκτρική αντίσταση είναι μια επεμβατική τεχνική που απαιτεί καλή επαφή μεταξύ του εδάφους και των τεσσάρων ηλεκτροδίων που έχουν εισαχθεί στο χώμα. Κατά συνέπεια, παράγει λιγότερο αξιόπιστες μετρήσεις σε ξηρά ή πετρώδη εδάφη από μια μη επεμβατική μέτρηση όπως η EMI. Ωστόσο, έχει μια ευελιξία μέτρησης βάθους για εφαρμογή στο πεδίο, Το βάθος της μέτρησης μπορεί εύκολα να αλλάξει αλλάζοντας την απόσταση μεταξύ των ηλεκτροδίων.

Ηλεκτρική αγωγιμότητα μέσω μέτρησης ηλεκτρομαγνητικής επαγωγής

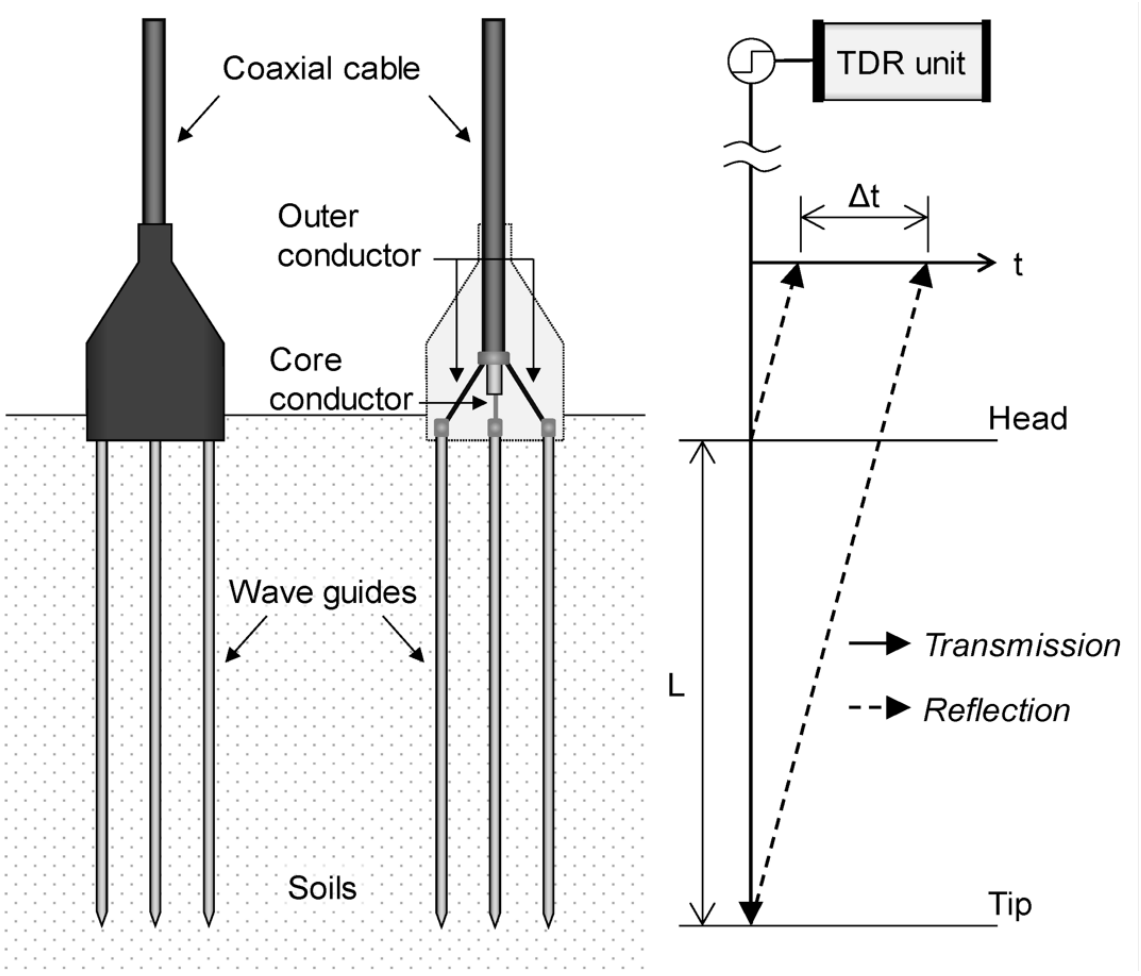
Η ηλεκτρική αγωγιμότητα του εδάφους μπορεί να μετρηθεί μη επεμβατικά με την ηλεκτρομαγνητική παρεμβολή (EMI). Η ηλεκτρομαγνητική επαγωγή χρησιμοποιείται σε οικόπεδα ή μεγαλύτερες χωρικές εκτάσεις. Ένα πηνίο πομπού που βρίσκεται στο ένα άκρο του οργάνου προκαλεί κυκλικούς βρόχους δινορευμάτων (eddy-current loops) στο έδαφος, με το μέγεθος αυτών των βρόχων ευθέως ανάλογο με την ηλεκτρική αγωγιμότητα κοντά σε αυτόν τον βρόχο. Κάθε βρόχος ρεύματος δημιουργεί ένα δευτερεύον ηλεκτρομαγνητικό πεδίο που είναι ανάλογο με την τιμή της ροής ρεύματος εντός του βρόχου. Ένα κλάσμα του δευτερεύοντος επαγόμενου ηλεκτρομαγνητικού πεδίου από κάθε βρόχο λαμβάνεται από το πηνίο δέκτη του οργάνου, και το άθροισμα αυτών των σημάτων ενισχύεται και διαμορφώνεται σε μια τάση εξόδου, η οποία σχετίζεται με την ηλεκτρική αγωγιμότητα. Το πλάτος και η φάση του δευτερεύοντος ηλεκτρομαγνητικού πεδίου θα διαφέρει από εκείνο του εκπέμφθηκε από τον πομπό ως αποτέλεσμα των ιδιοτήτων του εδάφους (π.χ. περιεκτικότητα σε άργιλο, περιεκτικότητα σε νερό, αλατότητα), της απόστασης των πηνίων και τον προσανατολισμό, τη συχνότητα και την απόστασή τους από την επιφάνεια του εδάφους.



Σχήμα 1.7 Μέτρηση διαφοράς ηλεκτρομαγνητικού πεδίου εκπομπής και λήψης για τον προσδιορισμό της ηλεκτρικής αγωγιμότητας τους εδάφους

Ηλεκτρική αγωγιμότητα και υγρασία εδάφους μέσω μέτρησης διηλεκτρικής διαπερατότητας

Οι αισθητήρες TDR (Time Domain Reflectometry) και χωρητικότητας ομαδοποιούνται επειδή και τα δύο μετρούν τη διηλεκτρική διαπερατότητα. Η μέθοδος TDR καθορίζει τα χαρακτηριστικά των ηλεκτρικών γραμμών παρατηρώντας τις ανακλώμενες κυματομορφές. Μια παρόρμηση της ενέργειας διαδίδεται στο έδαφος, ακολουθούμενη από την επακόλουθη μέτρηση της ενέργειας που αντανακλάται από το έδαφος. Μια ανάλυση του σχήματος, της διάρκειας και του μεγέθους της ανακλώμενης κυματομορφής καθορίζει τη φύση του η διακύμανση της σύνθετης αντίστασης του πορώδους μέσου. Σε αντίθεση οι αισθητήρες χωρητικότητας καθορίζουν τη διηλεκτρική διαπερατότητα του εδάφους μετρώντας το χρόνο φόρτισης ενός πυκνωτή, ο οποίος χρησιμοποιεί το έδαφος ως διηλεκτρικό.



Σχήμα 1.8 Γράφημα απεικόνισης TDR probe τριών κυματοδηγών για μέτρηση στο έδαφος

Η τεχνική TDR βασίζεται στον χρόνο που κάνει ένας παλμός τάσης για να ταξιδέψει και να επιστρέψει σε όλο το μήκος του ηλεκτροδίου που βρίσκεται μέσα στο έδαφος, που είναι συνάρτηση της διηλεκτρικής διαπερατότητας (e) του μέσου που μετριέται. Η μέτρηση της ηλεκτρικής αγωγιμότητας EC με TDR βασίζεται στην απόσβεση της εφαρμοζόμενης τάση σήματος καθώς διασχίζει το μέσο ενδιαφέροντος, με το σχετικό μέγεθος της απώλειας ενέργειας που σχετίζεται με την EC.

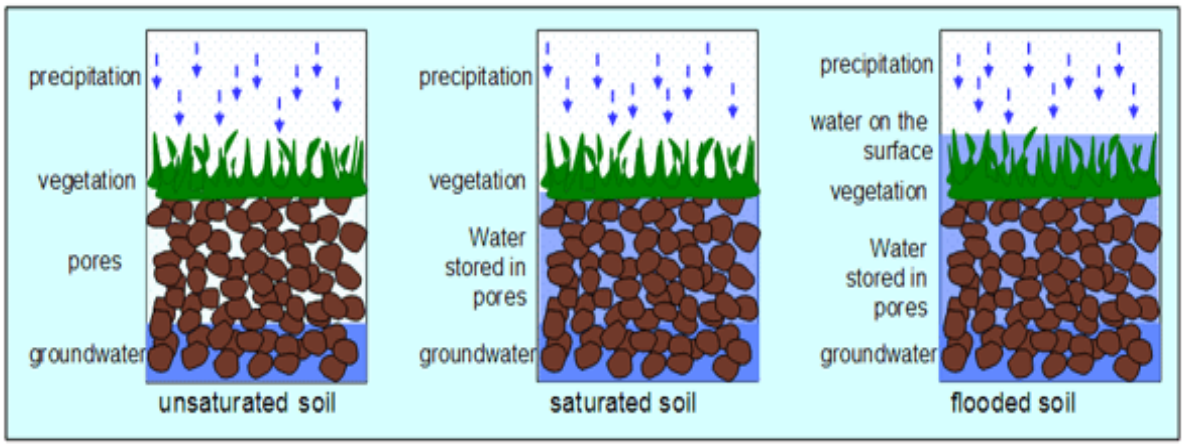
Ένα μειονέκτημα τόσο του TDR όσο και ων αισθητήρων χωρητικότητάς είναι ότι η ερμηνεία της εξόδου τους μπορεί να είναι προβληματική σε εδάφη με υψηλή αλατότητα. Κατά συνέπεια, οι αισθητήρες χωρητικότητας και οι TDR κυρίως χρησιμοποιήθηκαν για τη μέτρηση της περιεκτικότητας σε νερό και δεν ήταν τα όργανα επιλογής για τη μέτρηση της αλατότητας, είτε σε στο εργαστήριο ή στο πεδίο.

Ωστόσο, βελτιωμένα μοντέλα με καλή βαθμονόμηση και διηλεκτρικούς αισθητήρες με υψηλότερες συχνότητες λειτουργίας επεκτείνουν το εύρος και την ακρίβεια της περιεκτικότητας σε νερό του εδάφους και προσδιορισμό της αλατότητας του.

Τα πλεονεκτήματα του TDR για τη μέτρηση της EC περιλαμβάνουν:

- σχετικά μη επεμβατική φύση γιατί υπάρχει μόνο μικρή παρέμβαση στις διεργασίες του εδάφους
- ικανότητα ανίχνευσης μικρών αλλαγών στην EC υπό αντιπροσωπευτικές εδαφικές συνθήκες,
- ικανότητα λήψης συνεχών μετρήσεων χωρίς όριο
- έλλειψη απαίτησης βαθμονόμησης για τις μετρήσεις της περιεκτικότητας σε νερό του εδάφους σε πολλές περιπτώσεις

Οι αισθητήρες χωρητικότητας και το TDR χρησιμοποιούνται συχνότερα σε στήλες εδάφους ή μικρά αγροτεμάχια όπου δεν υπάρχει χωρική μεταβλητότητα μια ιδιαίτερη ανησυχία, ελαχιστοποιώντας έτσι την ανάγκη τοποθέτησης πολλών αισθητήρων για την ποσοτική ανομοιομορφία του εδάφους.



Σχήμα 1.9 Οι διάφορες καταστάσεις του εδάφους αναλόγως το την ποσότητα νερού που περιέχει

Η ηλεκτρική αγωγιμότητα ενός δείγματος εδάφους μετριέται στο εργαστήριο χρησιμοποιώντας ειδικό όργανο μέτρησης αγωγιμότητας στο εκχύλισμα του δείγματος. Η μορφή της ύλης τους δείγματος (αέρια, υγρή, στερεή) και τα ποσοστά αυτής επηρεάζεται από την αναλογία εδάφους/νερού στο οποίο γίνεται το εκχύλισμα, επομένως η αναλογία πρέπει να ακολουθεί κάποιο πρότυπο παρασκευής.

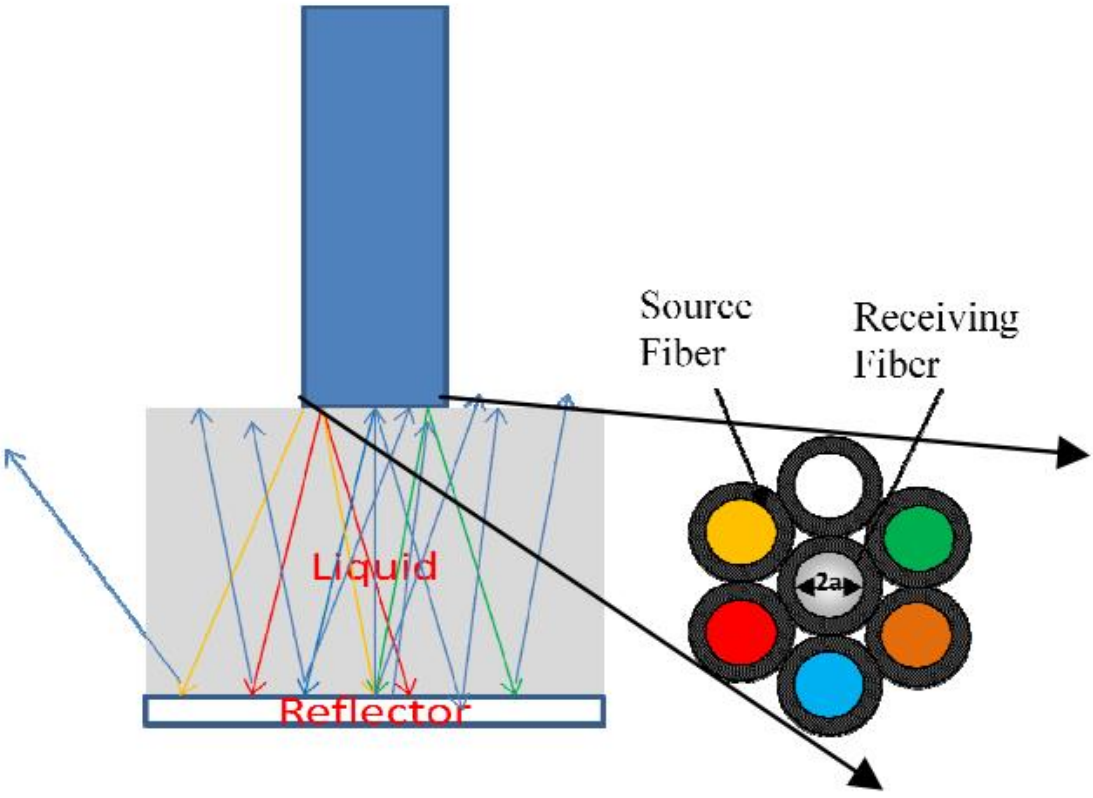
Μια ευρέως χρησιμοποιούμενη τεχνική είναι η μέτρηση του εκχυλίσματος εδάφους της πάστας κορεσμού που παρασκευάζεται με απεσταγμένο νερό με διήθηση υπό κενό. Συχνά χρησιμοποιούμενες αναλογίες εκχυλίσματος εκτός από το εκχύλισμα κορεσμού από μια πάστα κορεσμένου εδάφους είναι 1:1, 1:2, και μίγματα εδάφους/νερού 1:5. Ωστόσο, εκχυλίσματα σε αυτές τις αναλογίες παρέχουν μόνο σχετική μέτρηση επειδή τα εδάφη είναι προσαρμοσμένα αφύσικα σε υψηλή περιεκτικότητα νερού που δεν συμβαίνει στις πραγματικές συνθήκες.

Δείγματα εδαφικών διαλυμάτων σε περιεκτικότητα νερού που συνήθως βρίσκονται στα πεδία απαιτούν πολύ χρόνο, εργασία και κόστος για να είναι πρακτικά. Κατά συνέπεια, λαμβάνονται υδατικά εκχυλίσματα δειγμάτων εδάφους σε υψηλότερη από την κανονική περιεκτικότητα σε νερό (δηλαδή κατάσταση κορεσμού και πάνω) για ανάλυση ρουτίνας του εδάφους. Τυποποίηση της αναλογίας

εκχυλίσματος εδάφους/νερού είναι απαραίτητη για να ληφθούν αποτελέσματα που μπορούν να ερμηνευθούν και να συγκριθούν επειδή οι απόλυτες και οι σχετικές ποσότητες διαλυμένων ουσιών επηρεάζονται από το εκχύλισμα εδάφους/νερού.

Το TDS στο έδαφος αναφέρεται στη συνολική ποσότητα διαλυτών αλάτων σε ένα εκχύλισμα πάστας κορεσμένου εδάφους εκφρασμένο σε μέρη ανά εκατομμύριο ή χιλιοστόγραμμα ανά λίτρο (ppm ή mg/L). Δεν υπάρχει ακριβής ποσοτική σχέση μεταξύ της ηλεκτρικής αγωγιμότητας με το TDS επειδή τα χημικά συστατικά και τα είδη αλατιού επηρεάζουν τις τιμές μετατροπής. Ωστόσο, υπάρχει μια προσέγγιση όταν το χλωριούχο νάτριο είναι το κυρίαρχο άλας, οπότε το TDS προκύπτει πολλαπλασιάζοντας την EC με συντελεστή 500 έως 670. Ο συντελεστής που χρησιμοποιείται συνήθως είναι ο 640. Η αλατότητα είναι μια μέτρηση όπως το TDS που υπολογίζει το επίπεδο αλατιού σε ένα δείγμα νερού. Για να υπολογιστούν τα επίπεδα αλατότητας, μια προσέγγιση είναι με έναν συντελεστή μετατροπής (συνήθως ~0,5) στις μετρήσεις αγωγιμότητας.

Δεν υπάρχει κάποια ευρέως διαδομένη μέθοδος η οποία κάνει μέτρηση των μακροστοιχείων του εδάφους (Άζωτο, Φώσφορο, Κάλιο) με ακρίβεια μέσω ηλεκτρικών μεγεθών. Υπάρχει έρευνα επιστημόνων που χρησιμοποιεί απορροφήσεις του φωτός σε διάφορα μήκη κύματος για να καθορίσει την συγκέντρωση των στοιχείων αλλά με περιγραφική προσέγγιση (λίγη, μεσαία, υψηλή).



Σχήμα 1.10 Μέθοδος ανίχνευσης μακροστοιχείων στο έδαφος μέσω απορρόφησης φωτός με περιγραφική προσέγγιση

Η λειτουργία του αισθητήρα NPK απεικονίζεται στο Σχ. 1.10 Τα τρία διαφορετικά χρώματα LED εκπέμπουν έγχρωμο φως ίδιας έντασης. Μέσω πολυτροπικής οπτικής ίνας το φως εισέρχεται στο υγρό

δείγμα. Σε κατάλληλη απόσταση τοποθετείται ανακλαστήρας. Όταν το φως ταξιδεύει μέσω του διαλύματος από τον ανιχνευτή προς το ανακλαστήρα και πίσω, ανάλογα με τις χρωματικές απορροφήσεις του διαλύματος θα ληφθούν διαφορετικές εντάσεις για διαφορετικά χρώματα. Ο οι ανακλώμενες εντάσεις μπορούν να σχεδιαστούν έναντι του μήκος κύματος της πηγής για να κατασκευαστεί ένα αρκετά απλό φάσμα απορρόφησης σε διακριτά μήκη κύματος. Το φάσμα απορρόφησης είναι διαφορετικό για κάθε υδατικό διάλυμα εδάφους και έτσι μπορούν να εξαχθούν συμπεράσματα για τις ποσότητες μακροστοιχείων μέσα τους.

1.5 Επιλογή μεθόδου και στόχοι

Στην παρούσα εργασία εξετάζεται η μέθοδος με την χρήση αισθητήρα που βασίζεται η μέτρηση του στην μέθοδο TDR. Για λόγους χρονικούς και οικονομικούς αλλά και ενδιαφέροντος αποφασίστηκε να αγορασθεί κάποιος αισθητήρας από το εμπόριο, παρά να κατασκευαστεί κάτι εξατομικευμένο από την αρχή. Οι αισθητήρες που υπάρχουν για μετρήσεις εδάφους

Κύριο σκοπός είναι η κατασκευή μια ολοκληρωμένης μετρητικής συσκευής σε πρωτότυπο στάδιο για την μέτρηση όσο των δυνατών περισσότερων μεγεθών για το έδαφος. Πιο συγκεκριμένα:

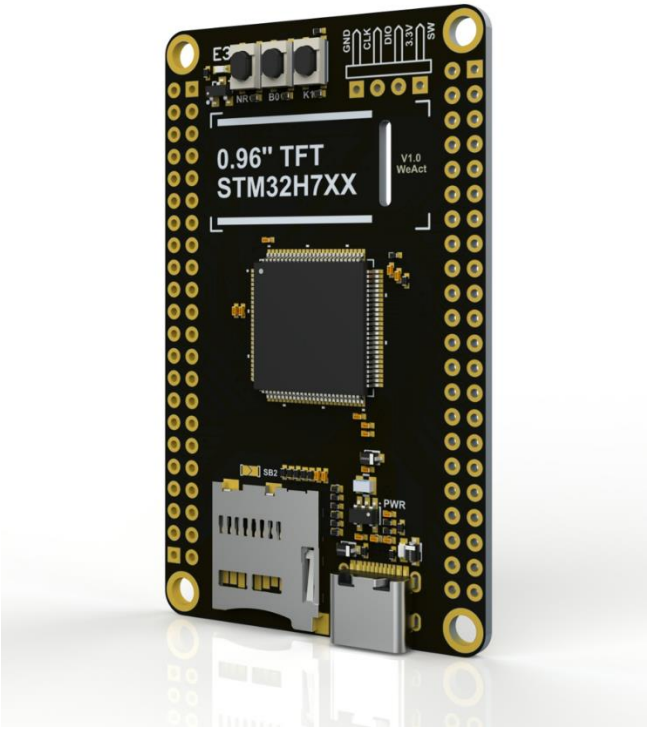
- Η εύρεση ενός κυκλώματος σωστής επικοινωνίας με τον αισθητήρα μέσω κάποιου πιθανού πρωτοκόλλου επικοινωνίας
- Η ασύρματη μετάδοση των δεδομένων σε κάποια συχνότητα RF ώστε να υπάρχει απομακρυσμένη μεταφορά δεδομένων σε συσκευή κοντά στον υπολογιστή. Το πρωτόκολλο RF που χρησιμοποιήθηκε είναι το LoRa.
- Η επικοινωνία της συσκευής με έναν ηλεκτρονικό υπολογιστή για την μεταφορά των δεδομένων μέτρησης , απεικόνιση σε κάποιον χάρτη για αποτύπωση της μέτρησης για μελλοντική επίσκεψη από τον χρήστη ώστε να συμβουλευθεί για τα εδάφη.
- Η σωστή και εργονομική σύνδεση όλων των εξαρτημάτων μεταξύ τους και η παροχή φορητότητας στην συσκευή.

Παρακάτω αναλύονται οι μέθοδοι που χρησιμοποιήθηκαν για να επιτύχουν όσο το δυνατόν κοντά σε αυτό το επιθυμητό αποτέλεσμα που περιγράφεται παραπάνω. Σημαντικά σημεία που αξίζει να αναφερθούν κατά την διαδικασία κατασκευής της συσκευής είναι η εν βάθος ενασχόληση με τον μικροεπεξεργαστή αφού αποτελεί τον εγκέφαλο του συστήματος και απαιτείται η κατανόηση της δομής του , των λειτουργιών του αλλά και ο προγραμματισμός του. Η επικοινωνία που εξυπηρετεί τον δίαυλο επικοινωνίας της συσκευής με τον κεντρικό υπολογιστή. Η κατασκευή web application τοπικού δικτύου και η ανάγκη δημιουργίας βάσης δεδομένων για την αποθήκευση των μετρήσεων.

Κεφάλαιο 2ο: Συσκευή μέτρησης

2.1 Εισαγωγή

Στην προσπάθεια ανάπτυξης μιας αποτελεσματικής συσκευής για την ακριβή μέτρηση των θρεπτικών στοιχείων του εδάφους, η επιλογή ενός κατάλληλου μικροελεγκτή παίζει καθοριστικό ρολό. Η σειρά STM32, με τις αξιοσημείωτες δυνατότητες επεξεργασίας, την εκτεταμένη γκάμα ενσωματωμένων περιφερειακών και την προηγμένη απόδοση ισχύος, ξεχωρίζει ως πρωταρχικός υποψήφιος για μια τέτοια εφαρμογή. Η σειρά που προτιμήθηκε από την STMicroelectronics, περιλαμβάνει μια ευέλικτη οικογένεια μικροελεγκτών ARM Cortex-M 32-bit που εκτιμώνται ιδιαίτερα για την επεκτασιμότητα, την απόδοση σε πραγματικό χρόνο και την αποδοτικότητα τους, καθιστώντας τους εξαιρετικά ευνοϊκούς για τέτοιου είδους εργασίες.



Σχήμα 2.1 Αναπτυξιακή πλακέτα βασισμένη στον STM32H750 της εταιρείας WeAct Studio

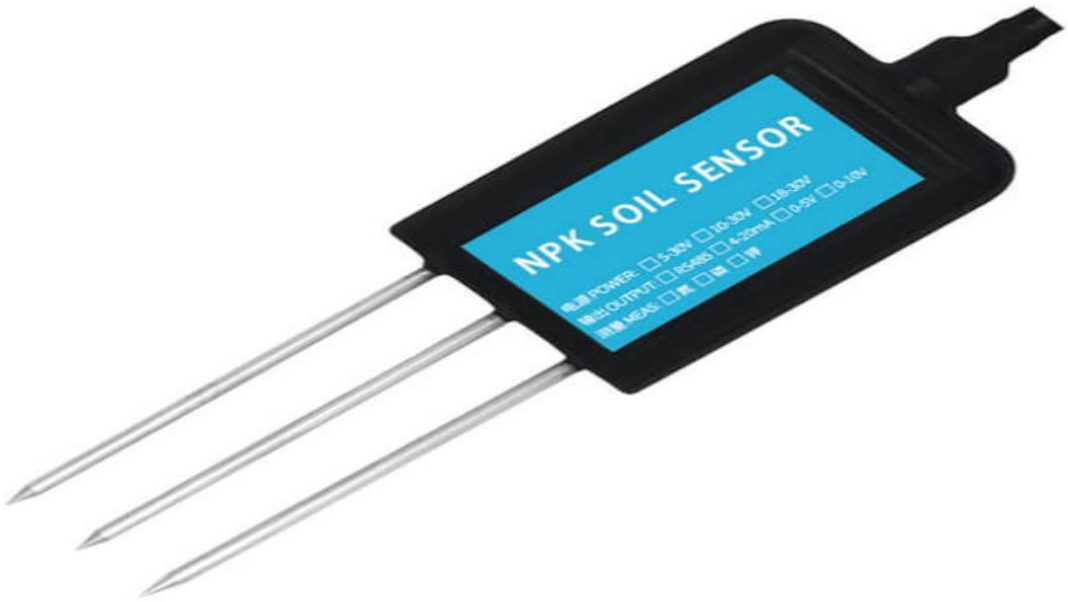
Το συγκεκριμένο μοντέλο που επιλέχθηκε σε αυτήν την συσκευή μέτρησης , είναι ο STM32H750VBT6, που προσφέρει ένα ανώτερο σύνολο χαρακτηριστικών, όπως προηγμένη υπολογιστική ικανότητα με ταχύτητα ρολογιού έως και 400 MHz, υψηλή χωρητικότητα 128 KB μνήμης RAM και μνήμη Flash μέσω εξωτερικών μέσων, εξασφαλίζοντας τεράστιο χειρισμό δεδομένων και ταχεία επεξεργασία για πολύπλοκες αναλύσεις. Αυτός ο μικροελεγκτής διαθέτει επίσης μια μονάδα κινητής υποδιαστολής (FPU) που διευκολύνει αποτελεσματικούς υπολογισμούς με Float/Double τύπους μεταβλητών, μια κρίσιμη απαίτηση για την ερμηνεία των δεδομένων του αισθητήρα NPK. Χρησιμοποιήθηκε το αναπτυξιακό της WeAct Studio καθώς διαθέτει περιφερειακά για ανάπτυξη όπως οθόνη, κουμπιά, RGB-LED, ακίδες προγραμματισμού και πολλά άλλα που το καθιστούν μια ελκυστική λύση για αυτήν την εφαρμογή.

Επιπλέον, η βελτιωμένη λειτουργικότητα της θύρας I/O και οι διάφορες διεπαφές επικοινωνίας εξασφαλίζουν απρόσκοπτη ενσωμάτωση αισθητήρα και μεταφορά δεδομένων. Τα χαρακτηριστικά της υψηλής απόδοσης, της ευελιξίας και του πλούτου των χαρακτηριστικών επικοινωνίας του STM32H750VBT6 το καθιστούν υποδειγματικό πυρήνα για μια συσκευή μέτρησης θρεπτικών συστατικών στο έδαφος, που υπόσχεται να αυξήσει την ευκολία και την αξιοπιστία των γεωργικών διαγνωστικών. Ένα ακόμα κριτήριο για την επιλογή του συγκεκριμένου μοντέλου είναι η εύκολη εύρεση αναπτυξιακού κυκλώματος στην αγορά καθώς και γρήγορη προμήθεια του όπως και το κόστος του. Στα παρακάτω υπό-κεφάλαια όλα τα καθοριστικά κομμάτια της συσκευής ως προς τον τρόπο λειτουργίας τους , τα τεχνικά τους χαρακτηριστικά καθώς και ο προγραμματισμός του.

2.2 Αισθητήρες μέτρησης στοιχείων εδάφους εμπορίου

Η επιλογή του αισθητήρα έγινε με βάση την διαθεσιμότητα του στην αγορά , το κόστος του , την προσαρμογή του και την μεγάλη γκάμα μετρούμενων μεγεθών όσο αναφορά τα στοιχεία του εδάφους. Στο εμπορείο υπάρχουν πολλοί αισθητήρες που έχουν να κάνουν με μετρήσεις εδάφους. Κάποιες κατηγορίες αισθητήρων είναι αυτοί που μετράνε:

- Άζωτο, φώσφορο και κάλιο ως γνωστοί με τον όρο NPK. (Σχήμα 1.5)
- Το pH του εδάφους αποκλειστικά. (Σχήμα 2.2)
- Θερμοκρασία , υγρασία , ηλεκτρική αγωγιμότητα , pH , άζωτο , φώσφορο, κάλιο (7 σε 1)



Σχήμα 2.2 Αισθητήρας μέτρησης θρεπτικών στοιχείων εδάφους τύπου NPK τριών μεγεθών

Σχήμα 2.3 Αισθητήρας αποκλειστικής μέτρηση pH στο έδαφος

Σχήμα 2.4 Αισθητήρας μέτρησης θερμοκρασίας, υγρασίας, pH, αζώτου , φωσφόρου , καλίου και ηλεκτρικής αγωγιμότητας

Στην παρούσα εργασία εξετάστηκε ο τελευταίος τύπος αισθητήρα , ώστε να υπάρξουν όσα περισσότερα δεδομένα είναι δυνατόν για την μέτρηση του εδάφους. Τα χαρακτηριστικά του αισθητήρα που επιλέχτηκε αναλύονται παρακάτω:

- Τάση τροφοδοσίας : 4.5 - 30V (DC)

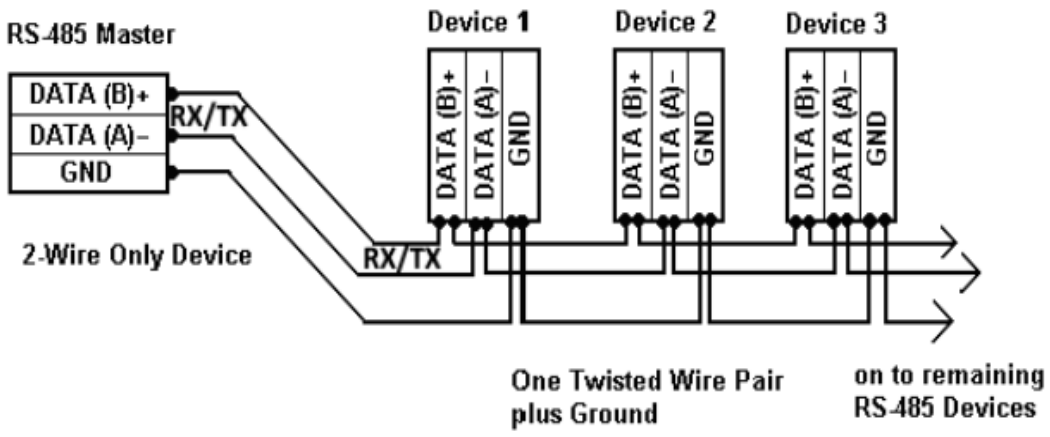
- Μέγιστη κατανάλωση σε ισχύς : 0.5W (με 24V τροφοδοσία)
- Υλικό ακίδας μέτρησης : Αντιδιαβρωτικό ειδικό ηλεκτρόδιο
- Πρωτόκολλο επικοινωνίας : RS485
- Μέτρηση Θερμοκρασίας:
    - Εύρος : -40~80℃
    - Ακρίβεια : ±0.5℃ (25℃)
    - Ανάλυση : 0.1℃
- Μέτρηση Ηλεκτρικής αγωγιμότητα:
    - Εύρος : 0~20000 us/cm
    - Ακρίβεια : 1 us/cm
    - Ανάλυση : ±3%
- Μέτρηση Υγρασίας:
    - Εύρος : 0~100 %
    - Ακρίβεια : 0.1%
    - Ανάλυση : 2% στο όριο 0-50%, 3% στο όριο 50-100%
- Μέτρηση pH:
    - Εύρος : 3~9 PH
    - Ακρίβεια : 0.1
    - Ανάλυση : ±3%
- Μέτρηση NPK:
    - Εύρος : 1-1999 mg/kg
    - Ακρίβεια : 1 mg/kg
    - Ανάλυση : ±5%

2.3 RS485 και Modbus επικοινωνία

Το πρωτόκολλο επικοινωνίας RS485 είναι ένα πρότυπο για τη σειριακή μετάδοση δεδομένων που επιτρέπει την ισχυρή μετάδοση δεδομένων σε μεγάλες αποστάσεις και σε ηλεκτρικά θορυβώδη περιβάλλοντα. Αναπτύχθηκε ως βελτίωση σε σχέση με τον προκάτοχό του, RS232, κυρίως για να αντιμετωπίσει την ανάγκη για επικοινωνία πολλαπλών σημείων, επιτρέποντας τη σύνδεση πολλαπλών συσκευών μέσω ενός μόνο δικτύου. Το RS485 λειτουργεί χρησιμοποιώντας μια μέθοδο διαφορικής σηματοδότησης, πράγμα που σημαίνει ότι μεταδίδει κάθε bit δεδομένων ως διαφορά τάσης μεταξύ δύο καλωδίων, γνωστή ως συνεστραμμένο ζεύγος. Αυτή η μέθοδος είναι ιδιαίτερα ανθεκτική στον θόρυβο κοινής λειτουργίας που μπορεί να προκληθεί στη διαδρομή σήματος, καθώς οποιοσδήποτε θόρυβος συλλαμβάνεται στη γραμμή είναι πιθανό να επηρεάσει εξίσου και τα δύο καλώδια και μπορεί να ακυρωθεί στο άκρο του δέκτη. Λόγω αυτής της διαφορικής λειτουργίας, το RS485 μπορεί να στείλει αξιόπιστα δεδομένα έως περίπου 1200 μέτρα και σε ταχύτητες έως 10 Mbps, αν και το μέγιστο μήκος καλωδίου μειώνεται καθώς αυξάνεται η ταχύτητα.

Ένα από τα καθοριστικά χαρακτηριστικά του RS485 είναι η ικανότητά του να υποστηρίζει πολλές ταυτόχρονα συσκευές στον ίδιο δίαυλο. Αυτό επιτυγχάνεται μέσω μιας τοπολογίας διαύλου δεδομένων κομματικής γραμμής, όπου όλοι οι πομποί και οι δέκτες συνδέονται σε ένα ενιαίο, μακρύ καλώδιο. Οι συσκευές σε ένα δίκτυο RS485 μπορούν να επικοινωνούν σε half-duplex λειτουργία, όπου μιλούν και ακούν εκ περιτροπής, γεγονός που απαιτεί ένα master-slave πρωτόκολλο ή μια ιεραρχία επικοινωνίας peer-to-peer για τη διαχείριση της ροής δεδομένων και την αποφυγή συγκρούσεων. Τα ηλεκτρικά

χαρακτηριστικά του RS485 επιτρέπουν τόσο την ευελιξία όσο και την απλότητα στο σχεδιασμό του δικτύου. Το πρότυπο καθορίζει μια ονομαστική διαφορική τάση από 1,5 βολτ ελάχιστη έως 5 βολτ μέγιστη που αντιπροσωπεύει δυαδικό ένα και μηδέν, αλλά οι δέκτες έχουν σχεδιαστεί για να ανιχνεύουν διαφορές τόσο μικρές όσο 200mV, γεγονός που συμβάλλει στην ανοσία του σήματος σε θορυβώδη περιβάλλοντα. Η δημοτικότητα του RS485 σε βιομηχανικές εφαρμογές οφείλεται επίσης στην αξιοπιστία και την ευκολία χρήσης του. Χρησιμοποιείται συνήθως σε εφαρμογές όπως ο εργοστασιακός αυτοματισμός, ο αυτοματισμός κτιρίων και άλλες καταστάσεις όπου ένα δίκτυο αισθητήρων και ενεργοποιητών πρέπει να επικοινωνεί αξιόπιστα σε μεγάλες αποστάσεις. Όταν οι συσκευές επικοινωνούν μέσω RS485, μπορούν να χρησιμοποιήσουν απλά πρωτόκολλα start stop bits ή πιο περίπλοκα με μεθόδους διευθυνσιοδότησης πακέτων, ανίχνευσης σφαλμάτων και διόρθωσης που ταιριάζουν στις ανάγκες της εφαρμογής. Συμπερασματικά, η ικανότητα του RS485 για επικοινωνία μεγάλων αποστάσεων, η ανθεκτικότητα απέναντι στον ηλεκτρικό θόρυβο, η υποστήριξη πολλαπλών συσκευών και οι ευέλικτες τοπολογίες δικτύου το καθιστούν μια διαρκή επιλογή για βιομηχανική και εμπορική επικοινωνία δεδομένων. Ένα επίπεδο πάνω στο hardware είναι το Modbus πρωτόκολλο που χρησιμοποιεί και ο NPK αισθητήρας που χρησιμοποιήθηκε στην μελέτη.



Σχήμα 2.5 Modbus-RTU Half-Duplex communication

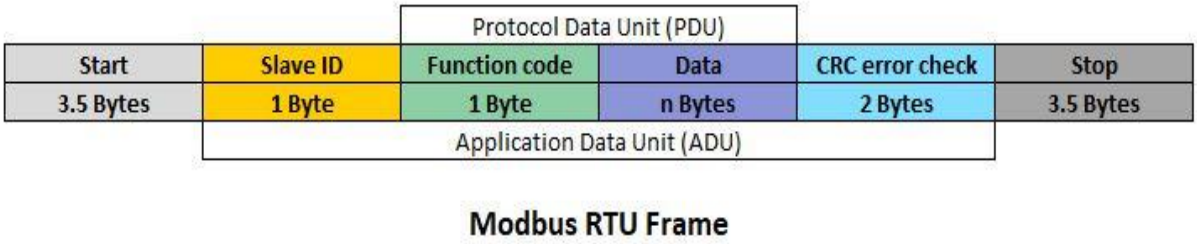
Το Modbus, το οποίο ιδρύθηκε το 1979 από τη Modicon, είναι ένα πρωτόκολλο επικοινωνίας που έχει αντέξει εντυπωσιακά στο εξελισσόμενο τοπίο των βιομηχανικών περιβαλλόντων, διατηρώντας την ευρεία χρήση και τη συνάφειά του. Αυτή η εις βάθος ανάλυση του Modbus τονίζει γιατί παραμένει η βέλτιστη επιλογή για τα σύγχρονα βιομηχανικά συστήματα επικοινωνίας, δεδομένης της απλότητας, της προσαρμοστικότητας και της αποδεδειγμένης αποτελεσματικότητάς του. Στη συνεχώς εξελισσόμενη σφαίρα του βιομηχανικού αυτοματισμού και της επικοινωνίας, η ανάγκη για ισχυρά αλλά απλά πρωτόκολλα είναι πρωταρχικής σημασίας. Το Modbus, που φέρει μια κληρονομιά που εκτείνεται σε δεκαετίες, είναι συχνά η καλύτερη λύση σε διάφορες σύγχρονες εφαρμογές. Αυτή η διαρκής προτίμηση για το Modbus πηγάζει από τον απλό σχεδιασμό του, την προσβασιμότητα του ανοιχτού κώδικα και την ευκολία με την οποία μπορεί να ενσωματωθεί σε διαφορετικά συστήματα.

Ως πρωτόκολλο σειριακής επικοινωνίας, το Modbus διευκολύνει κυρίως τη σύνδεση βιομηχανικών ηλεκτρονικών συσκευών. Λειτουργεί με βάση μια δυναμική master-slave όπου μια συσκευή, η κύρια, εκκινεί συναλλαγές ή ερωτήματα, στα οποία απαντούν οι άλλες συσκευές, οι slaves. Αυτή η δομή επιτρέπει σε ένα μόνο master να αλληλοεπιδρά με έως και 247 slaves μέσω ενός δικτύου. Ιδιαίτερα

προσαρμόσιμο, το Modbus σχεδιάστηκε αρχικά για επικοινωνία RS-232, αλλά έκτοτε εξελίχθηκε ώστε να είναι συμβατό με το RS-485 και το TCP/IP, αντανακλώντας την ικανότητά του να παραμένει σχετικό εν μέσω μεταβαλλόμενων τεχνολογικών τοπίων. Η απλότητα και η διαλειτουργικότητα του Modbus είναι από τα πιο καθοριστικά χαρακτηριστικά του. Η απλή φύση του πρωτοκόλλου απαιτεί ελάχιστη προσπάθεια ανάπτυξης, ενισχύοντας υψηλά επίπεδα διαλειτουργικότητας μεταξύ συσκευών από πολλούς κατασκευαστές. Η ευελιξία του είναι ένας άλλος ακρογωνιαίος λίθος, υποστηρίζοντας διάφορους τύπους δεδομένων και κατάλληλο για διάφορα δίκτυα, συμπεριλαμβανομένου του Ethernet. Αυτή η προσαρμοστικότητα επιτρέπει την απρόσκοπτη ενσωμάτωση σε προϋπάρχοντα συστήματα. Η φύση ανοιχτού κώδικα του Modbus, χωρίς τέλη αδειοδότησης, προωθεί την ευρεία υιοθέτηση και υποστήριξη σε ένα ευρύ φάσμα συσκευών και προμηθευτών. Κρίσιμης σημασίας, η αξιοπιστία του Modbus, ειδικά όσον αφορά τη δομή αιτήματος/απάντησής του, είναι απαραίτητη σε βιομηχανικά περιβάλλοντα όπου η αξιόπιστη παράδοση δεδομένων είναι αδιαπραγμάτευτη απαίτηση.

Η χρήση του σε σύγχρονα συστήματα επιφέρει πολλά οφέλη. Η εγγενής του απλότητα και η κατάσταση ανοιχτού κώδικα μειώνουν σημαντικά το κόστος ανάπτυξης και υλοποίησης. Η ευκολία με την οποία το Modbus μπορεί να ενσωματωθεί σε υπάρχοντα συστήματα παρουσιάζει μια λύση χωρίς προβλήματα για την αναβάθμιση ή τη βελτίωση των βιομηχανικών δυνατοτήτων επικοινωνίας. Με την ευρεία αποδοχή του στον κλάδο, υπάρχει ένα εκτεταμένο οικοσύστημα εργαλείων και συσκευών που υποστηρίζουν το Modbus, διασφαλίζοντας την εφαρμογή του σε ποικίλα σενάρια. Επιπλέον, είναι επεκτάσιμο, από μικρά, απλά δίκτυα έως πολύπλοκα συστήματα που περιλαμβάνουν πολλαπλές συσκευές. Σε πρακτικές εφαρμογές, το Modbus δείχνει την ευελιξία του. Στον βιομηχανικό αυτοματισμό, χρησιμοποιείται ευρέως για επικοινωνία μεταξύ προγραμματιζόμενων λογικών ελεγκτών (PLC), αισθητήρων και ενεργοποιητών. Τα συστήματα διαχείρισης κτιρίων χρησιμοποιούν επίσης Modbus, ιδιαίτερα στον έλεγχο συστημάτων HVAC, φωτισμού και πρόσβασης. Επιπλέον, η χρησιμότητά του στην απομακρυσμένη παρακολούθηση είναι εμφανής στα συστήματα τηλεμετρίας και SCADA για την επίβλεψη απομακρυσμένων λειτουργιών. Σε σύγκριση με άλλα πρωτόκολλα όπως το PROFIBUS και το Ethernet/IP, τα οποία προσφέρουν υψηλότερες ταχύτητες και πιο προηγμένες δυνατότητες, το Modbus διατηρεί τη θέση του λόγω της απλής φύσης και της φιλικότητας προς τον χρήστη. Διατίθεται σε διάφορες παραλλαγές , καθεμία προσαρμοσμένη σε συγκεκριμένους τύπος δικτύων και απαιτήσεων επικοινωνίας. Τα πιο σημαντικά μεταξύ αυτών είναι τα Modbus RTU, Modbus ASCII και Modbus TCP. Παρακάτω θα γίνει ανάλυση του Modbus RTU που είναι η παραλλαγή του Modbus που θα αναλυθεί παρακάτω και χρησιμοποιείται από τον εξεταζόμενο αισθητήρα NPK.

Το πρωτόκολλο Modbus RTU (Remote Terminal Unit) είναι μια μέθοδος επικοινωνίας master-slave που χρησιμοποιείται ευρέως σε βιομηχανικά περιβάλλοντα για τη σύνδεση ενός εποπτικού υπολογιστή με τις απομακρυσμένες τερματικές του μονάδες. Η αποτελεσματικότητα αυτού του πρωτοκόλλου έγκειται στην απλότητα και τη στιβαρότητά του, καθιστώντας το κατάλληλο για διάφορες εφαρμογές. Η ανάλυση του Modbus RTU απαιτεί κατανόηση της δομής δεδομένων του και του μηχανισμού επικοινωνίας master-slave.



Σχήμα 2.6 Modbus-RTU δομή πακέτου επικοινωνίας σε byte ανάλυση

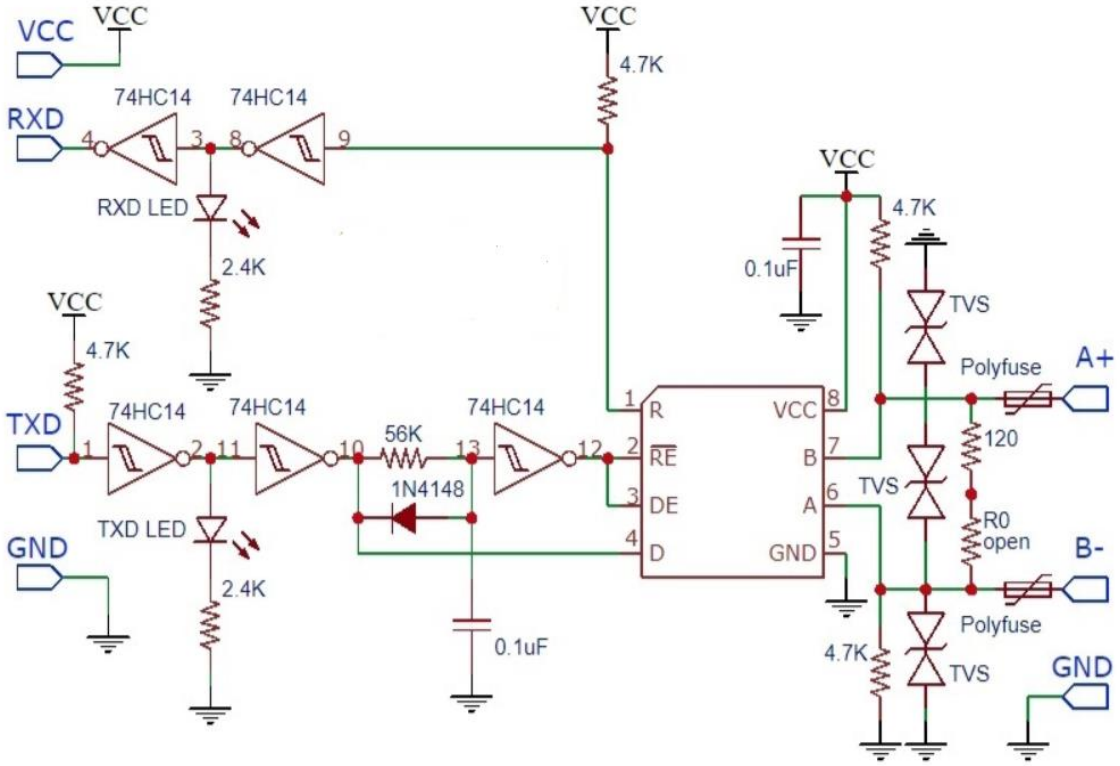
Δομή δεδομένων στο Modbus RTU:

- Το Modbus RTU μεταδίδει δεδομένα σε δυαδική μορφή, γεγονός που το καθιστά αποτελεσματικό και συμπαγές. Κάθε πακέτο δεδομένων στο Modbus RTU αποτελείται από πολλά πεδία:
- Διεύθυνση συσκευής: Το πρώτο πεδίο σε ένα μήνυμα Modbus RTU καθορίζει τη διεύθυνση της εξαρτημένης συσκευής. Αυτό είναι σημαντικό σε ένα δίκτυο με πολλαπλούς slave, επιτρέποντας στον master να επικοινωνεί με έναν συγκεκριμένο slave.
- Function Code: Αυτό το πεδίο λέει στο slave τι είδους ενέργεια να εκτελέσει. Για παράδειγμα, θα μπορούσε να δώσει εντολή στο slave να διαβάσει ή να γράψει στους καταχωρητές τους.
- Data: Τα πραγματικά δεδομένα που αποστέλλονται ή η θέση δεδομένων προς πρόσβαση στην υποτελή. Σε ένα αίτημα για ανάγνωση δεδομένων, αυτό μπορεί να καθορίσει ποιους καταχωρητές θα διαβάσει, ενώ σε μια απάντηση, θα ήταν οι πραγματικές τιμές καταχωρητή.
- CRC Error Check: Αυτός είναι συνήθως ένας κυκλικός έλεγχος πλεονασμού (CRC) για τη διασφάλιση της ακεραιότητας των δεδομένων. Το CRC υπολογίζεται από τον αποστολέα (είτε τον master είτε τον slave) και υπολογίζεται εκ νέου από τον παραλήπτη για έλεγχο τυχόν αλλοίωσης κατά τη μετάδοση.

Επικοινωνία Master-Slave:

- Στο Modbus RTU, η επικοινωνία ξεκινά από την κύρια συσκευή και ακολουθεί έναν αυστηρό κύκλο ερωτήματος-απόκρισης. Κάθε slave στο δίκτυο έχει μια μοναδική διεύθυνση και μόνο ο slave που απευθύνεται ο master θα απαντήσει σε ένα ερώτημα.
- Ερώτημα από τον Master: Ο Master στέλνει ένα μήνυμα ζητώντας ή στέλνοντας δεδομένα στον slave. Αυτό το μήνυμα περιλαμβάνει τη διεύθυνση του slave, τον κωδικό λειτουργίας (ποια ενέργεια πρέπει να πραγματοποιηθεί) και τυχόν σχετικά δεδομένα ή διευθύνσεις δεδομένων.
- Απάντηση από τον Slave: Με τη λήψη ενός ερωτήματος, ο slave που απευθύνεται εκτελεί την ενέργεια που ζητήθηκε και στέλνει μια απάντηση πίσω στον master. Αυτή η απάντηση περιέχει τον ίδιο κωδικό λειτουργίας, τα δεδομένα που ζητήθηκαν (αν ήταν συνάρτηση ανάγνωσης) ή μια επιβεβαίωση (αν ήταν συνάρτηση εγγραφής).

2.4 RS485 Κύκλωμα



Σχήμα 2.7 Κύκλωμα RS485 half-duplex επικοινωνίας του αισθητήρα NPK με τον STM32 με χρήση αυτόματου τρόπου εναλλαγής DE/RE.

Το κεντρικό τσιπ για την επίτευξη της επικοινωνίας RS485 που χρησιμοποιήθηκε είναι το SP3485 το οποίο βρίσκεται στο εμπόριο σε χαμηλή τιμή και βρίσκεται σε πολλά αναπτυξιακά modules. Η προτεινόμενη τάσης λειτουργίας είναι στα 3.3V που προτιμάται εφόσον και ο STM32 δουλεύει στα 3.3V. Κάποια επιπλέον χαρακτηριστικά είναι ότι:

• Χρησιμοποιεί διαφορικό σήμα για ανοχή στον θόρυβο
• Υποστηρίζει αποστάσεις έως 1200 μέτρα
• Ταχύτητες έως 10 Mbit/Sec
• Η Multi-drop αρχιτεκτονική υποστηρίζει έως και 256 συσκευές στον ίδιο δίαυλο.

Κατά τη λήψη, το TX είναι active HIGH, επιβάλλοντας έτσι κατάσταση LOW στο DE/RE (όπως αναμένεται). Όταν το TX πέσει χαμηλά, εκφορτώνει γρήγορα το ο πυκνωτής 0.1uF μέσω της αντίστασης 56K και ρυθμίζει τη ροή κατεύθυνσης για τη μετάδοση άρα DE/RE σε κατάσταση HIGH. Όταν η μετάδοση και τα δεδομένα TX αλλάζουν κατάσταση σε λογική HIGH, ο πυκνωτής 0.1uF παραμένει σε κατάσταση αποφόρτισης αλλά μέσω της αντίστασης 56K επαναφορτίζεται αργά και η εκ φόρτιση θα ολοκληρωθεί όχι σε χρονικό διάστημα μικρότερο μετά την αποστολή του τελευταίου byte μετάδοσης. Οι τιμές που εμφανίζονται, δηλαδή 56k και 0.1uF εξαρτώνται και από τον ρυθμό μετάδοσης baud-rate. Οι δίοδοί TVS χρησιμοποιούνται για προστασία από υπερτάσεις καθώς η αντίσταση τερματισμού του δίαυλου 120Ω είναι προαιρετική και μπαίνει συνήθως στον τελευταίο slave στον δίαυλο για την εξάλειψη των ανακλάσεων των σημάτων.

2.5 Υλοποίηση επικοινωνίας με τον αισθητήρα NPK

Ο κατασκευαστής τους αισθητήρα υιοθετεί το πρωτόκολλο Modbus-RTU για την εξαγωγή των μετρήσεων του. Η προκαθορισμένη τιμή για τον ρυθμό μετάδοσής είναι στα 4800 bits/sec καθώς και τα δεδομένα 2 byte επιστρέφονται με προτεραιότητα το HIGH byte.

address code	function code	Register start address	Register length	Check code low	Check code high
1 byte	1 byte	2 bytes	2 bytes	1 byte	1 byte

Σχήμα 2.8 Δομή δεδομένων μηνύματος ερώτησης προς τον αισθητήρα

- Address Code: Είναι η διεύθυνση του αισθητήρα και είναι μοναδική στον δίαυλο του RS485 δικτύου. Προκαθορισμένη τιμή είναι η 0x01.
- Function Code: Η εντολή λειτουργίας που αποστέλλεται από τον STM32, ο αισθητήρας χρησιμοποιεί μόνο τον κωδικός συνάρτησης 0x03 (διάβασμα δεδομένων καταχωρητή).
- Register Start Address: Η αρχική διεύθυνση καταχωρητή που απευθύνεται το διάβασμα των δεδομένων
- Register length: Το μήκος της διεύθυνσης των καταχωρητών που απευθύνεται το διάβασμα των τιμών. Για παράδειγμα αν η αρχική διεύθυνση είναι η 0x0000 και θέσεις μήκος 0x0004 , θα διαβάσει τις θέσεις 0x0000,0x0001,0x0002,0x0003.
- Check Code Low: Το λιγότερο σημαντικό byte του CRC
- Check Code High: Το περισσότερο σημαντικό byte του CRC

address code	function code	Effective number of bytes	Data area	Second data area	Nth data area	Check code
1 byte	1 byte	1 byte	2 bytes	2 bytes	2 bytes	2 bytes

Σχήμα 2.9 Δομή δεδομένων μηνύματος απάντησης από τον αισθητήρα

- Effective number of bytes: Το μήκος των byte που πρόκειται να διαβαστούν και περιέχουν δεδομένα.
- Data Area, Second Data Area, Nth Data Area: Τα bytes που έρχονται κατά την ιδιά σειρά που δηλώθηκαν οι διευθύνσεις των καταχωρητών στο ερώτημα διαβάσματος και περιέχουν τις μετρήσεις του αισθητήρα με πρώτο byte επιστροφής το περισσότερο σημαντικό.

Register address	PLC or configuration address	Content	Operation	Definition description
0000 H	40001 (decimal)	Moisture content	Read only	Real-time value of water content (expanded by 10 times)
0001 H	40002 (decimal)	Temperature value	Read only	Real-time temperature value (expanded 10 times)
0002 H	40003 (decimal)	Conductivity	Read-only	Conductivity real-time value
0003 H	40004 (decimal)	PH value	Read only	PH real-time value (expanded ten times)
0004H	40005 (decimal)	Nitrogen content	Read only	Actual value of nitrogen content
0005H	40006 (decimal)	Phosphorus content	Read only	Actual value of phosphorus content
0006H	40007 (decimal)	Potassium content	Read only	Actual value of potassium content
0007 H	40008 (decimal)	Salinity	Read only	Salinity real-time value
0008 H	40009 (decimal)	Total dissolved solids TDS	Read only	TDS real-time value

Σχήμα 2.10 Πίνακας διευθύνσεων καταχωρητών και δεδομένων του αισθητήρα NPK

Για να γίνει εξαγωγή όλων των δεδομένων που μπορεί να μετρήσει ο αισθητήρας στο μήνυμα ερώτησης θα συμπεριληφθούν όλες οι διευθύνσεις των καταχωρητών , ο κάθε καταχωρητής περιέχει διαφορετικό δεδομένο μέτρησης του εδάφους όπως φαίνεται στον παραπάνω πίνακα.

- Moisture Content: Υγρασία του εδάφους που είναι η συνολική ποσότητα νερού, συμπεριλαμβανομένων των υδρατμών, σε ένα ακόρεστο έδαφος.
- Temperature Value: Η θερμοκρασία του εδάφους.
- Conductivity: Αγωγιμότητα του εδάφους μετρά πόσο καλά ένα έδαφος μπορεί να μεταφέρει ένα ηλεκτρικό ρεύμα. Είναι μια βασική παράμετρος στην επιστήμη του εδάφους και τη γεωργία, καθώς

παρέχει πληροφορίες σχετικά με την ικανότητα του εδάφους να μεταδίδει νερό και θρεπτικά συστατικά

- PH Value: Το PH του εδάφους είναι σημαντικό γιατί το pH του εδάφους καθορίζει ποια θρεπτικά συστατικά είναι διαθέσιμα στις ρίζες. Άζωτο, φώσφορος και κάλιο είναι διαθέσιμα όταν διαλυθούν στο νερό ή στην υγρασία του εδάφους. Τα θρεπτικά συστατικά δεν διαλύονται όταν το pH του εδάφους είναι πολύ όξινο ή αλκαλικό.
- Nitrogen Content: Το άζωτο στο έδαφος είναι πολύ σημαντικό για την ανάπτυξη των φυτών (δομή), την επεξεργασία των φυτικών τροφίμων (μεταβολισμό) και τη δημιουργία χλωροφύλλης. Χωρίς αρκετό άζωτο στο φυτό, το φυτό δεν μπορεί να ψηλώσει ή να παράγει αρκετή τροφή εμφανίζοντας συνήθως χρώμα κίτρινο.
- Phosphorus Content: Ο φώσφορος είναι ένα από τα κύρια θρεπτικά συστατικά των φυτών στο έδαφος. Είναι συστατικό των φυτικών κυττάρων, απαραίτητο για την κυτταρική διαίρεση και την ανάπτυξη της αναπτυσσόμενης άκρης του φυτού. Για το λόγο αυτό είναι ζωτικής σημασίας για τα νεαρά φυτά.
- Potassium Content: Το κάλιο βοηθά στη ρύθμιση του ανοίγματος και του κλεισίματος των στομάτων, το οποίο ρυθμίζει την ανταλλαγή υδρατμών, οξυγόνου και διοξειδίου του άνθρακα. Εάν το Κάλιο είναι ελλιπές ή δεν παρέχεται σε επαρκείς ποσότητες, εμποδίζει την ανάπτυξη των φυτών και μειώνει την απόδοση τους.
- Salinity: Η αλατότητα του εδάφους είναι η συσσώρευση υδατοδιαλυτών αλάτων. Συνήθως, είναι επιτραπέζιο αλάτι NaCl. Ο κατάλογος είναι πολύ πιο εκτενής και περιλαμβάνει διάφορες ενώσεις νατρίου, καλίου, ασβεστίου, μαγνησίου, θειικών, χλωριδίων και υδατανθράκων. Γενικά, τα εδάφη αναλόγως την αλατότητα τους κατηγοριοποιούνται σε αλατούχα, νατριούχα και τον συνδυασμό των δυο, ανάλογα με την περιεκτικότητα.
- Total Dissolved Solids: Το TDS στο έδαφος αναφέρεται στη συνολική ποσότητα διαλυτών αλάτων σε ένα εκχύλισμα πάστας κορεσμένου εδάφους εκφρασμένο σε μέρη ανά εκατομμύριο ή χιλιοστόγραμμα ανά λίτρο (ppm ή mg/L).

Η επικοινωνία υλοποιείται στέλνοντας Modbus μήνυμα με συχνότητα επικοινωνίας στα 4Hz όπως φαίνεται στις παρακάτω φωτογραφίες καθώς ήταν επίσης απαραίτητο να γίνει ανάπτυξη βιβλιοθηκών για την σωστή επίτευξη της επικοινωνίας στον κώδικα της συσκευής που τρέχει στον STM32.

```
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  if (millis() - timeNow > 250) {
      timeNow = millis();
      uint8_t cmd[] =  {0x01, 0x03, 0x00, 0x00, 0x00 , 0x09,  0x85 , 0xCC};
      HAL_StatusTypeDef res = HAL_UART_Transmit(&NPK_UART, (uint8_t*)&cmd, sizeof(cmd), 1000);
      if (res == HAL_OK) {
          checkNPK();
      }
  }
```

Σχήμα 2.11 Δομή μηνύματος επικοινωνίας με τον αισθητήρα για την εξαγωγή των μετρήσεων

```
#include "npk.h"

void npkConversions(npkData* Sensor) {

    if ( abs( 65536 - Sensor->_Temperature.raw ) < 250 ) {
      Sensor->Temperature = - (65536 - Sensor->_Temperature.raw) * 0.1;
    }
    else {
      Sensor->Temperature = Sensor->_Temperature.raw * 0.1;
    }
    Sensor->Moisture = Sensor->_Moisture.raw * 0.1;
    Sensor->Conductivity = Sensor->_Conductivity.raw;
    Sensor->PH = Sensor->_PH.raw*0.1;
    Sensor->Nitrogen =  Sensor->_Nitrogen.raw;
    Sensor->Phosphorus =  Sensor->_Phosphorus.raw;
    Sensor->Potassium =  Sensor->_Potassium.raw;
    Sensor->Salinity =  Sensor->_Salinity.raw;
    Sensor->TDS =  Sensor->_TDS.raw;


}

void npkParse(npkData* Sensor , char* d , uint16_t len) {

    uint8_t b = 0;
    for(uint8_t i=3;i<len-2;i++){
      Sensor->bytes[b] = d[i];
      b++;
    }

    for(uint8_t i=0;i<sizeof(npkData);i=i+2){
      uint8_t temp = Sensor->bytes[i];
      Sensor->bytes[i] = Sensor->bytes[i+1];
      Sensor->bytes[i+1] = temp;
    }
    npkConversions(Sensor);

}
```

Σχήμα 2.12 Δείγμα ανάπτυξης βιβλιοθήκης για την Modbus επικοινωνία με τον αισθητήρια NPK

2.6 Lora επικοινωνία

Το LoRa (Long Range) είναι μια κατοχυρωμένη πατέντα τεχνολογία ψηφιακής ασύρματης επικοινωνίας δεδομένων που αναπτύχθηκε από την Cycleo της Γκρενόμπλ της Γαλλίας και αργότερα αποκτήθηκε από τη Semtech, προμηθευτή ημιαγωγών αναλογικών και ψηφιακών. Έχει υιοθετηθεί ευρέως για τα δίκτυα Internet of Things (IoT) παγκοσμίως λόγω του μοναδικού συνδυασμού χαρακτηριστικών του. Βασικά χαρακτηριστικά του LoRa:

Σχήμα 2.13 LoRa επικοινωνία λογότυπο

- Συνδεσιμότητα μεγάλης εμβέλειας: Το πιο καθοριστικό χαρακτηριστικό του LoRa είναι η ικανότητά του σε μεγάλη εμβέλεια. Μπορεί να συνδέσει συσκευές σε αποστάσεις έως και 20 km σε αγροτικές περιοχές και 2-5 km σε αστικές περιοχές, κάτι που είναι πολύ πιο μακριά από άλλα ασύρματα πρότυπα όπως το Wi-Fi ή το Bluetooth.
- Χαμηλή κατανάλωση ενέργειας: Οι συσκευές με δυνατότητα LoRa είναι γνωστές για τη χαμηλή κατανάλωση ενέργειας. Αυτή η δυνατότητα είναι ζωτικής σημασίας για εφαρμογές IoT όπου συσκευές, όπως αισθητήρες ή ιχνηλάτες, πρέπει να λειτουργούν με ισχύ μπαταρίας για εκτεταμένες περιόδους (συχνά αρκετά χρόνια).
- Μετάδοση χαμηλού ρυθμού δεδομένων: Το LoRa δεν έχει σχεδιαστεί για εφαρμογές υψηλής ταχύτητας δεδομένων. Αντίθετα, μεταδίδει μικρές ποσότητες δεδομένων με χαμηλό ρυθμό, γεγονός που βοηθά στη διατήρηση της διάρκειας ζωής της μπαταρίας και υποστηρίζει τη δυνατότητα μεγάλης εμβέλειας.
- Εύρος ζώνης και συντελεστής διάδοσης: Το LoRa επιτρέπει διαφορετικά εύρη ζώνης και παράγοντες διασποράς, τα οποία μπορούν να προσαρμοστούν για να ισορροπήσουν μεταξύ του εύρους επικοινωνίας και του ρυθμού δεδομένων. Ένας υψηλότερος συντελεστής διασποράς αυξάνει την εμβέλεια αλλά μειώνει τον ρυθμό μετάδοσης δεδομένων και αντίστροφα.
- Ζώνη συχνοτήτων Sub-GHz: Το LoRa λειτουργεί σε ζώνες συχνοτήτων υπό gigahertz όπως 868 MHz στην Ευρώπη και 915 MHz στη Βόρεια Αμερική. Αυτές οι ζώνες έχουν λιγότερο κόσμο από τα 2,4 GHz που χρησιμοποιούνται από τεχνολογίες όπως το Wi-Fi, μειώνοντας έτσι τις παρεμβολές.
- Τοπολογία Δικτύου Star-of-Stars: Το LoRa χρησιμοποιεί συνήθως μια τοπολογία star-of-stars, όπου πολλοί τερματικοί κόμβοι (αισθητήρες, συσκευές) επικοινωνούν με μια πύλη που προωθεί τα δεδομένα σε έναν κεντρικό διακομιστή δικτύου.

- Ασφάλεια: Το LoRa ενσωματώνει κρυπτογράφηση από άκρο σε άκρο για να διασφαλίσει την ασφάλεια των δεδομένων, χρησιμοποιώντας την κρυπτογράφηση AES-128 τόσο για το επίπεδο δικτύου όσο και για το επίπεδο εφαρμογής.

Σχήμα 2.14 LoRa επικοινωνία χρήσεις

Οι πιο συνήθεις εφαρμογές είναι :

- Έξυπνες πόλεις: Το LoRa χρησιμοποιείται σε εφαρμογές έξυπνων πόλεων για πράγματα όπως ο έλεγχος του φωτισμού του δρόμου, η διαχείριση απορριμμάτων και οι έξυπνες λύσεις στάθμευσης.
- Γεωργία: Στη γεωργία, χρησιμοποιείται για την παρακολούθηση της υγρασίας του εδάφους, την παρακολούθηση των ζώων και τη διαχείριση του θερμοκηπίου.
- Βιομηχανικό IoT: Σε βιομηχανικά περιβάλλοντα, το LoRa χρησιμοποιείται για την παρακολούθηση της υγείας του εξοπλισμού, συμμόρφωση με την ασφάλεια και υλικοτεχνική παρακολούθηση.
- Περιβαλλοντική Παρακολούθηση: Είναι επίσης ιδανικό για περιβαλλοντική παρακολούθηση, όπως η παρακολούθηση της ποιότητας του αέρα ή του νερού, λόγω της ικανότητάς του να προσεγγίζει απομακρυσμένες τοποθεσίες.

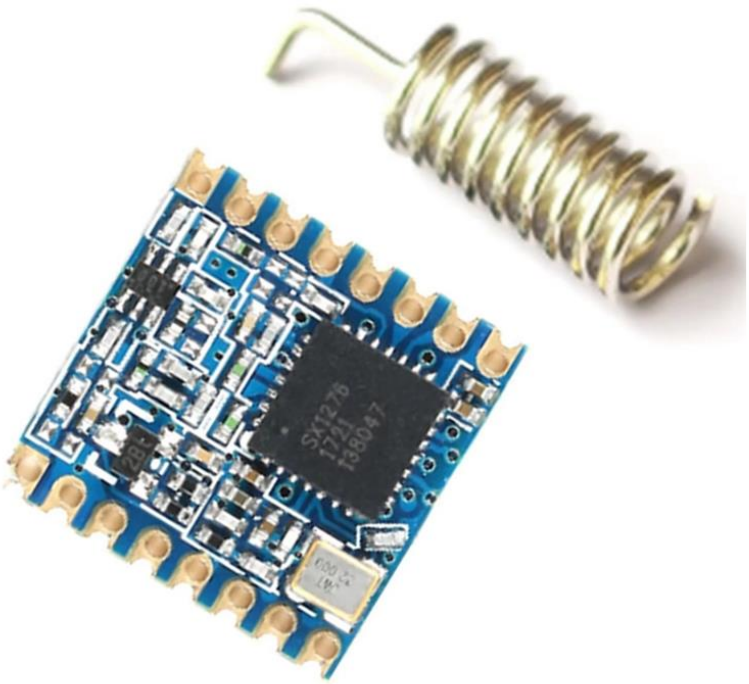
Πλεονεκτήματα:

- ✓ Δυνατότητα διείσδυσης: Τα σήματα LoRa μπορούν να διεισδύσουν μέσα από κτίρια και υπόγεια, καθιστώντας το κατάλληλο για αστικά περιβάλλοντα και εφαρμογές σε εσωτερικούς χώρους.
- ✓ Οικονομική: Η ανάπτυξη και η συντήρηση της τεχνολογίας είναι σχετικά φθηνή, κάτι που είναι επωφελές για μεγάλης κλίμακας αναπτύξεις IoT.

- ✓ Ευελιξία και επεκτασιμότητα: Η αρχιτεκτονική του δικτύου του επιτρέπει την εύκολη κλιμάκωση από μικρά σε μεγάλα δίκτυα χωρίς σημαντικές αλλαγές στην υποδομή.
- ✓ Χαμηλή απαίτηση υποδομής: Σε αντίθεση με τα κυψελωτά δίκτυα, τα δίκτυα LoRa μπορούν να λειτουργούν με λιγότερες πύλες, μειώνοντας το κόστος υποδομής.

Προκλήσεις και περιορισμοί:

- Περιορισμένο εύρος ζώνης: Λόγω του χαμηλού ρυθμού δεδομένων, το LoRa δεν είναι κατάλληλο για εφαρμογές που απαιτούν υψηλή απόδοση δεδομένων, όπως η ροή βίντεο.
- Περιορισμοί παρεμβολών και εύρους: Αν και λιγότερο επιρρεπής σε παρεμβολές από άλλες συχνότητες, το LoRa μπορεί να επηρεαστεί από εμπόδια σε αστικά περιβάλλοντα, επηρεάζοντας την εμβέλειά του.
- Κανονιστική συμμόρφωση: Η LoRa πρέπει να συμμορφώνεται με τους περιφερειακούς κανονισμούς σχετικά με τη χρήση ραδιοσυχνοτήτων, οι οποίοι μπορεί να διαφέρουν σημαντικά μεταξύ διαφορετικών χωρών.



Σχήμα 2.15 LoRa επικοινωνία SX1276

Συνοπτικά, το LoRa ξεχωρίζει στο τοπίο του IoT για την εξαιρετική του εμβέλεια, τη χαμηλή κατανάλωση ενέργειας και την ευκολία επεκτασιμότητας. Είναι ιδιαίτερα κατάλληλο για εφαρμογές όπου οι συσκευές πρέπει να στέλνουν μικρές ποσότητες δεδομένων σε μεγάλες αποστάσεις χωρίς να απαιτείται συχνή αντικατάσταση μπαταριών. Αυτό το καθιστά βασική τεχνολογία στον αναπτυσσόμενο κόσμο του IoT, συμβάλλοντας σε πιο έξυπνες πόλεις, αποτελεσματικές γεωργικές πρακτικές και προηγμένη περιβαλλοντική παρακολούθηση. Το μικροτσίπ που επιλέχτηκε για να διατελέσει την

επικοινωνία Lora στην συσκευή είναι το SX1276, πρόκειται για ένα πομποδέκτης ραδιοσυχνοτήτων (RF) που έχει σχεδιαστεί για ασύρματες εφαρμογές μεγάλης εμβέλειας και αποτελεί βασικό εξάρτημα σε πολλά συστήματα που βασίζονται σε LoRa (Μεγάλη εμβέλεια). Κατασκευασμένο από τη Semtech. Κάποια από τα βασικά του χαρακτηριστικά είναι τα παρακάτω:

- Εύρος συχνοτήτων: Λειτουργεί σε πολλαπλές ζώνες συχνοτήτων από 137 MHz έως 1020 MHz, καλύπτοντας τις συνήθως χρησιμοποιούμενες ζώνες υπό- GHz (όπως 433 MHz, 868 MHz στην Ευρώπη και 915 MHz στη Βόρεια Αμερική), οι οποίες είναι ιδανικές για επικοινωνία μεγάλης εμβέλειας.
- Διαμόρφωση LoRa και FSK/OOK: Το SX1276 είναι ικανό για διαμόρφωση LoRa και FSK (Frequency Shift Keying)/OOK (On-Off Keying).
- Υψηλή ευαισθησία και μεγάλη εμβέλεια: Προσφέρει υψηλή ευαισθησία έως και -148 dBm, επιτρέποντας επικοινωνία εξαιρετικά μεγάλης εμβέλειας. Με την τεχνολογία LoRa, μπορεί να επιτύχει αυτονομία πάνω από 15 KM σε αγροτικές περιοχές και 2-5 km σε αστικά περιβάλλοντα, ανάλογα με τις ρυθμίσεις και τις περιβαλλοντικές συνθήκες.
- Χαμηλή τάση: Λειτουργεί σε εύρος τάσης συνήθως μεταξύ 1,8 V και 3,7 V, καθιστώντας το συμβατό με μια ποικιλία μικροελεγκτών χαμηλής τάσης.
- Ενσωμάτωση: Ενσωματώνει πολλά χαρακτηριστικά όπως αισθητήρα θερμοκρασίας, δυνατότητα αναπήδησης συχνότητας και ανιχνευτή χαμηλής μπαταρίας, ενισχύοντας περαιτέρω τη χρησιμότητά του σε πολύπλοκα δίκτυα IoT.
- Πρωτόκολλο επικοινωνίας με μικροελεγκτή: SPI

Εξετάζοντας κάθε λεπτομέρεια για τις απαιτήσεις της συσκευής που αναπτύχθηκε και σύμφωνα με τα παραπάνω χαρακτηριστικά η επικοινωνία LoRa με την χρήση του SX1276 επιλέχτηκε ώστε να αποστέλλονται τα δεδομένα της μέτρησης της συσκευής στον δέκτη LoRa ο οποίος με την σειρά του τα μεταφέρει σε ένα ηλεκτρονικό υπολογιστή για περαιτέρω επεξεργασία. Η αρχιτεκτονική που επιλέχτηκε είναι η απλουστευμένη Node to Node απευθείας επικοινωνία καθώς οι απαιτήσεις της εφαρμογής είναι η απομακρυσμένη μετάδοση δεδομένων από την συσκευή σε κάποιον κοντινό υπολογιστή-λάπτοπ με οπτική επαφή.

2.7 Lora SX1276 και STM32

Η επικοινωνία του SX1276 με τον STM32 διαμορφώθηκε σε ταχύτητα 16 Mbits/s με 4-Wire SPI σαν πρωτόκολλο επικοινωνίας καθώς και δύο επιπλέων ποδιών (RESET pin, IO0 pin). Έγινε εισαγωγή βιβλιοθήκης ανοιχτού κώδικα για την οδήγηση του SX1276 , παρόλο που χρειάστηκε ειδική επεξεργασία και παραμετροποίηση για να είναι λειτουργική και να γίνει επιτυχής αποστολή δεδομένων. Ελέγχοντας την version που επιστρέφει το SX1276 καθώς και με εξαναγκασμένο RESET στην αρχή λύνεται το πρόβλημα του μη σωστής εκκίνησης του τσιπ. Επιλέχτηκε μέγιστη ισχύς εκπομπής , μέγιστος παράγοντας διασποράς όπως και συγχρονισμός λέξης έπειτα από πειραματισμό με τις παραμέτρους αναζητώντας την πιο αξιόπιστη επικοινωνία με τα λιγότερα σφάλματα και την καλύτερη εμβέλεια.

```
void loraInit(lora_sx1276* lora) {

    bool loraInit = false;
    HAL_GPIO_WritePin(LORA_RESET_GPIO_Port, LORA_RESET_Pin, LOW);
    HAL_Delay(10);
    HAL_GPIO_WritePin(LORA_RESET_GPIO_Port, LORA_RESET_Pin, HIGH);
    HAL_Delay(10);

    HAL_GPIO_WritePin(LORA_NSS_GPIO_Port, LORA_NSS_Pin, HIGH);

    uint8_t res = lora_init(lora, &hspi1, LORA_NSS_GPIO_Port, LORA_NSS_Pin, 868E6);
    if (res == LORA_ERROR) {
          DEBUGF("[Lora]Firstinit Failed");
          uint8_t ver = lora_version(lora);
          DEBUGF("[Lora]Checking again version for 0x12");
          while (ver != 0x12) {
              DEBUGF("[Lora]Wrong version:%d",ver);
              ver = lora_version(lora);
              HAL_Delay(1000);
          }
          res = lora_init(lora, &hspi1, LORA_NSS_GPIO_Port, LORA_NSS_Pin, LORA_BASE_FREQUENCY_EU);
          if (res == LORA_OK) {
              loraInit = true;
              DEBUGF("[Lora]Init successfully");
          }
          else {
              DEBUGF("[Lora]Init stuck");
              while (1);
          }
    }
    else {
          DEBUGF("[Lora]Init OK");
          loraInit = true;
    }

    if (loraInit) {
          //lora_set_crc(lora,1);
          //lora_set_signal_bandwidth(&lora,7);
          lora_set_tx_power(lora,20);
          lora_set_syncword(lora,0xA5);
          lora_set_spreading_factor(lora, 12);
    }

}
```

Σχήμα 2.16 Δείγμα κώδικα αρχικοποίησης επικοινωνίας με το SX1276

2.8 GPS επικοινωνία NEO-6M

Το NEO-6M είναι μια δημοφιλής μονάδα GPS που χρησιμοποιείται ευρέως σε διάφορες εφαρμογές, ιδιαίτερα στους τομείς της ηλεκτρονικής και της ρομποτικής. Κατασκευασμένο από την u-blox, μια ελβετική εταιρεία γνωστή για την τεχνολογία GPS και ασύρματων ημιαγωγών της, η μονάδα NEO-6M προσφέρει πολλά βασικά χαρακτηριστικά:

- Ακρίβεια και απόδοση: Το NEO-6M παρέχει υψηλή ευαισθησία και ισχυρή απόδοση σε αστικά περιβάλλοντα και πυκνό φύλλωμα, τα οποία είναι συνήθως προκλητικά για τη λήψη GPS. Προσφέρει ακριβείς πληροφορίες θέσης και ταχύτητας.
- Συμπαγής σχεδιασμός: Το μικρό του μέγεθος το καθιστά κατάλληλο για συμπαγείς συσκευές και έργα όπου ο χώρος είναι περιορισμένος.

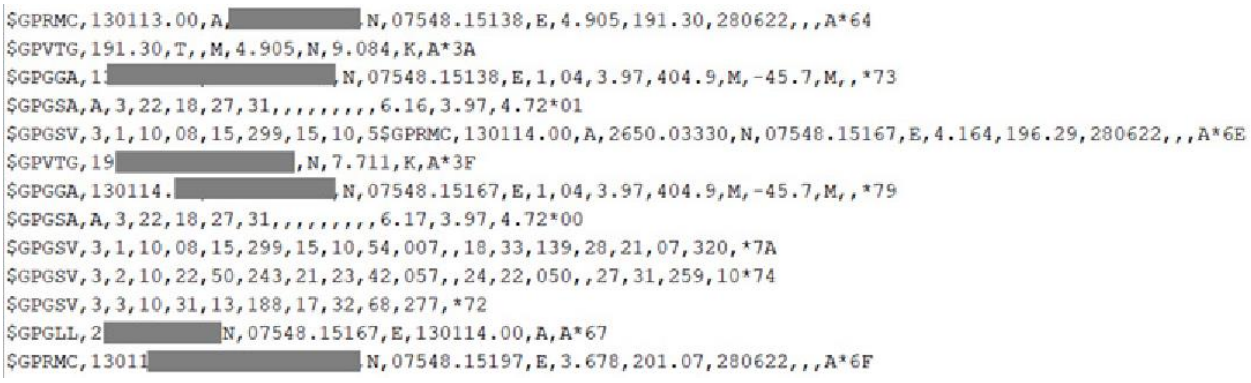
- Απόδοση ισχύος: Έχει σχεδιαστεί για χαμηλή κατανάλωση ενέργειας, η οποία είναι κρίσιμη για εφαρμογές που τροφοδοτούνται από μπαταρίες, όπως drones, φορητές μονάδες GPS και άλλες φορητές συσκευές.
- Ευελιξία: Η μονάδα είναι σε θέση να υποστηρίζει διάφορα δορυφορικά συστήματα όπως GPS (ΗΠΑ), GLONASS (Ρωσία) και Galileo (ΕΕ), αυξάνοντας την αξιοπιστία και την ακρίβειά της σε διάφορες περιοχές του κόσμου.
- Ακρίβεια: Αν και δεν είναι τόσο ακριβής όσο ορισμένες μονάδες GPS προηγμένης τεχνολογίας, παρέχει επαρκή ακρίβεια για πολλούς γενικούς σκοπούς, με τυπική ακρίβεια θέσης στο εύρος των μέτρων.
- Χρόνος πρώτου εντοπισμού θέσης: Διαθέτει αξιοσέβαστο χρόνο που χρειάζεται η μονάδα GPS για να αποκτήσει δορυφορικά σήματα και να καθορίσει την αρχική θέση, καθιστώντας την κατάλληλη για εφαρμογές όπου απαιτείται γρήγορη απόκτηση τοποθεσίας.

Συνολικά, η μονάδα GPS NEO-6M εκτιμάται για το συνδυασμό απόδοσης, απόδοσης ισχύος και προσαρμοστικότητας, καθιστώντας την μια δημοφιλή επιλογή για ένα ευρύ φάσμα έργων και εφαρμογών που βασίζονται σε GPS. Η επικοινωνία με τον μικροελεγκτή γίνεται μέσω UART και ο ρυθμός μετάδοσης είναι στα 9600 bits/sec. Τα δεδομένα έρχονται σε μορφή NMEA.

Οι προτάσεις NMEA είναι μια τυπική μορφή που χρησιμοποιείται από το GPS και άλλες συσκευές πλοήγησης για τη μετάδοση δεδομένων. Το NMEA σημαίνει National Marine Electronics Association, και είναι πρότυπο μετάδοσης δεδομένων. Οι προτάσεις NMEA είναι ASCII (βασισμένες σε κείμενο) και δομούνται ως μια σειρά τιμών διαχωρισμένων με κόμματα, γνωστές ως πεδία. Κάθε πρόταση ξεκινά με ένα σύμβολο «$» και τελειώνει με \n\r (αλλαγή γραμμής και carriage return). Ένα άθροισμα ελέγχου για τον έλεγχο σφαλμάτων ακολουθεί τον αστερίσκο στο τέλος της πρότασης.

- Τύποι προτάσεων: Υπάρχουν διάφοροι τύποι προτάσεων NMEA, καθένας από τους οποίους ξεκινά με διαφορετικά γράμματα που υποδεικνύουν τον σκοπό της πρότασης. Για παράδειγμα, οι προτάσεις που ξεκινούν με "GPGGA" παρέχουν βασικά δεδομένα επιδιόρθωσης, όπως γεωγραφικό πλάτος, γεωγραφικό μήκος, χρόνο και ποιότητα επιδιόρθωσης. Άλλοι συνηθισμένοι τύποι περιλαμβάνουν «GPGLL» (γεωγραφικό πλάτος και μήκος), «GPGSA» (κατάσταση δορυφόρου), «GPGSV» (δορυφόροι σε προβολή) και «GPRMC» (προτεινόμενες ελάχιστες πληροφορίες πλοήγησης).
- Χρήση: Αυτές οι προτάσεις χρησιμοποιούνται για την επικοινωνία δεδομένων όπως τοποθεσία, ταχύτητα, κατεύθυνση και χρόνος από δέκτες GPS σε υπολογιστές, αυτόματους πιλότους και άλλες συσκευές πλοήγησης. Είναι μια τυπική μορφή που αναγνωρίζεται από τα περισσότερα λογισμικά GPS και πλοήγησης.
- Συμβατότητα και ευελιξία: Λόγω της τυποποιημένης μορφής τους, οι προτάσεις NMEA μπορούν να χρησιμοποιηθούν σε ένα ευρύ φάσμα εξοπλισμού, ανεξάρτητα από τον κατασκευαστή. Αυτό τα καθιστά εξαιρετικά ευέλικτα σε θαλάσσιες, χερσαίες και εναέριες εφαρμογές πλοήγησης.
- Περιορισμοί: Ενώ οι προτάσεις NMEA χρησιμοποιούνται ευρέως, έχουν περιορισμούς στο εύρος ζώνης και στον όγκο των δεδομένων που μπορούν να μεταδώσουν. Αυτό τα καθιστά λιγότερο κατάλληλα για τη μετάδοση μεγάλων ποσοτήτων πολύπλοκων δεδομένων.
- Προσαρμοσμένες προτάσεις: Ορισμένοι κατασκευαστές επεκτείνουν το πρότυπο με προσαρμοσμένες προτάσεις NMEA για συγκεκριμένους σκοπούς ή για να παρέχουν πρόσθετα δεδομένα που δεν καλύπτονται από τις τυπικές προτάσεις.

Το NEO-6M σύμφωνα με τον κατασκευαστή έχει ακρίβεια 2.5 μέτρα σε ιδανικές συνθήκες. Σε επιτυχής επικοινωνία με το module ο ρυθμός μετάδοσης δεδομένων είναι στο 1 δευτερόλεπτο. Όταν ολοκληρωθεί η αναζήτηση δορυφόρων και έχει κλειδώσει η αναζήτηση τοποθεσίας αναβοσβήνει το μικρό μπλε LED που υπάρχει πάνω στην αναπτυξιακή πλακέτα. Σε αντίθετη περίπτωση αναζήτησης σήματος το μπλε LED παραμένει σβηστό.



Σχήμα 2.17 Παράδειγμα NMEA μνημάτων που έρχονται από το GPS Module

Για παράδειγμα σε μια πρόταση της μορφής
($GPGGA, 130113.00, 37XX.XXXX,N, 07XXX.XXXX, E,1,04,3.97,404.9,M,45.7,M,,*79)
τα δεδομένα χωρισμένα με κόμμα δηλώνουν τα παρακάτω:

1. 130113: Αντιπροσωπεύει την χρονική στιγμή που ήρθε το μήνυμα σε μορφή UTC , 13:01:13 UTC
2. 37XX.XXXX,N: Το γεωγραφικό πλάτος της μορφής , 37XX.XXXX
3. 07XXX.XXXX,E: Το γεωγραφικό μήκος της μορφής 07XXX.XXXX
4. Τύπος εντοπισμού:
   - 0 = Invalid
   - 1 = GPS fix
   - 2 = DGPS fix
   - 3 = PPS fix
   - 4 = Real Time Kinematic;
   - 5 = Float RTK
   - 6 = Estimated (Dead reckoning)
   - 7 = Manual Input Mode
   - 8 = Simulation Mode
5. 04: Αριθμός δορυφόρων που εντοπίστηκαν
6. 3.97 – HDOP (Horizontal Dilution Of Position) όσο περισσότεροι καλοί ορατοί δορυφόροι χαμηλά στον ουρανό, τόσο καλύτερο είναι το HDOP
7. 404.9, M: υψόμετρο, σε μέτρα από την επιφάνεια της θάλασσας
8. 45.7, M: Ύψος του γεωειδούς (μέση στάθμη της θάλασσας) πάνω από το ελλειψοειδές WGS84
9. Άδειο πεδίο: DGPS ώρα ενημέρωσης
10. Άδειο πεδίο: DGPS ταυτότητα σταθμού
11. *79: Δεδομένα αθροίσματος ελέγχου (Checksum)

2.9 GPS επικοινωνία NEO-6M και STM32

Έπειτα από αναζήτηση βιβλιοθήκης ανοιχτού κώδικα για να επιτευχθεί η επικοινωνία με STM32, έγιναν προγραμματιστικές δοκιμές για να επιβεβαιωθεί η λειτουργικότητα της. Για να γίνει λειτουργική η βιβλιοθήκη έπρεπε να γίνουν παραμετροποιήσεις στον τρόπο parsing των NMEA προτάσεων αφού επιβεβαιώθηκε ότι εκεί υπήρχε το πρόβλημα. Υλοποιήθηκαν οι μέθοδοι ανάγνωσης μηνυμάτων τύπου GPGGA, GPRMC, GPGLL, GPVTG που σαφώς καλύπτουν τις απαιτήσεις της συσκευής που τα ζητούμενα είναι το γεωγραφικό πλάτος και το γεωγραφικό μήκος. Κάθε NMEA πρόταση για να περάσει στο στάδιο parsing , πάντα ελέγχεται και υπολογίζεται το checksum για να υπάρχει αξιοπιστία στην σωστή μετάδοση του μηνύματος. Παράδειγμα τέτοιας εφαρμογής φαίνεται στην παρακάτω εικόνα.

```
void parseGPSString(char *gpsString , uint8_t type) {

    //DEBUGF("[parseGPSString] Type:%d",type);

    char *token = strtok(gpsString, ",");
    int num = 0;
    while (token != NULL) {
        //DEBUGF("[parseGPSString] Type:GPGGA Num:%d",num);
        if (type==GPGGA) {

            switch (num) {
                case 1:
                    GPS.utc_time = atof(token);
                break;
                case 2:
                    GPS.nmea_latitude = atof(token);
                break;
                case 3:
                    GPS.ns = token[0];
                break;
                case 4:
                    GPS.nmea_longitude = atof(token);
                break;
                case 5:
                    GPS.ew = token[0];
                break;
                case 6:
                    GPS.lock = token[0] > '0';
                break;
                case 7:
                    GPS.satelites = atoi(token);
                break;
                case 8:
                    GPS.hdop = atof(token);
                break;
                case 9:
                    GPS.msl_altitude = atof(token);
                break;
                case 10:
                    GPS.msl_units = token[0];
                break;
                case 11:
                    GPS.geoid = atof(token);
                break;
            }
            num++;
        }
```

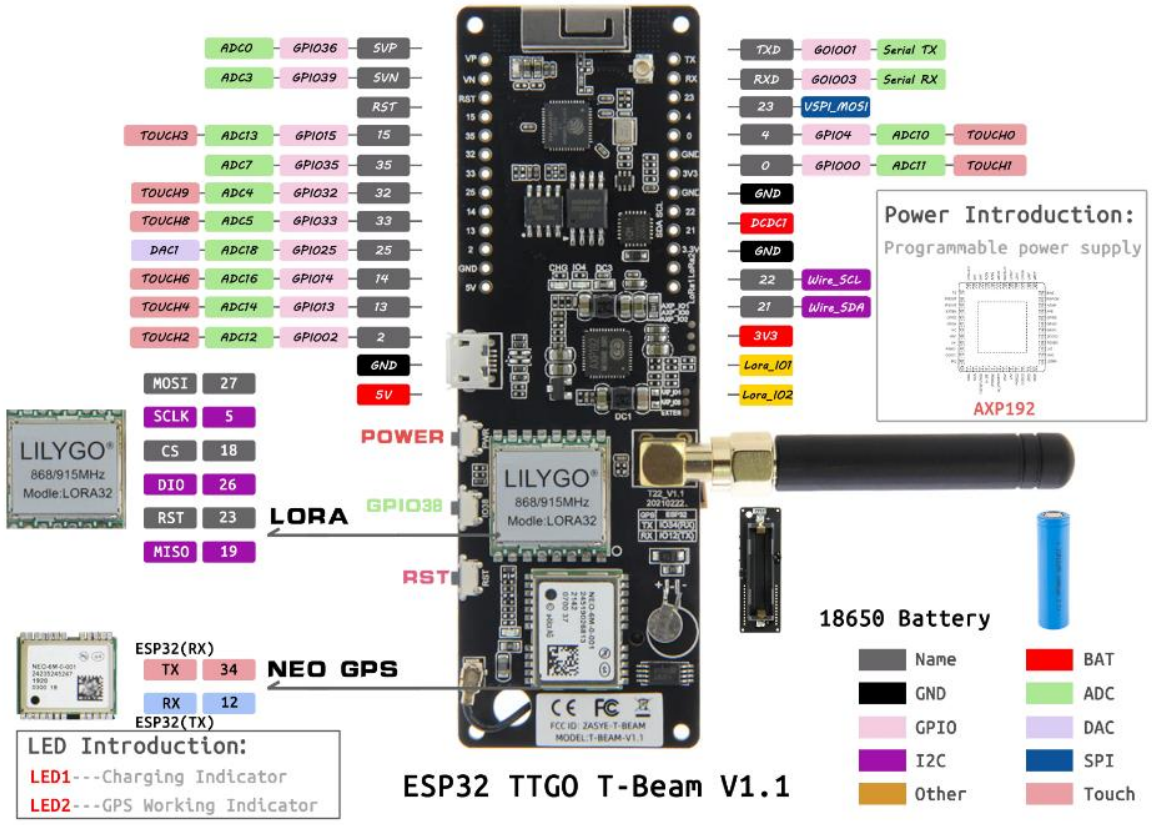
Σχήμα 2.18 Κώδικας επεξεργασίας και εκχώρησης δεδομένων από GPS NMEA προτάσεις

Κεφάλαιο 3ο: Application συσκευής και δέκτης LoRa

3.1 Εισαγωγή

Το κεφάλαιο αυτό αναφέρεται διεξοδικά στο λογισμικό υψηλού επιπέδου που κατασκευάστηκε να ολοκληρώσει την συσκευής μέτρησης , καθώς και στον δέκτη της μέτρησης μέσω LoRa. Το λογισμικό που αναπτύχθηκε χωρίζεται σε δύο μέρη , το πρώτο μέρος αναφέρεται στο application script που αναπτύχθηκε σε Python και το δεύτερο μέρος που αναφέρεται στην ανάπτυξη του Web application. Αρχικά αναλύεται η μεθοδολογία που ακολουθήθηκε ώστε να κατασκευαστεί το πρόγραμμα λήψης του δέκτη LoRa ESP32 , κάνοντας αναφορά στον τρόπο λήψης και μεταφοράς αυτών στον υπολογιστή. Μετέπειτα, αναλύονται όλες οι λειτουργίες που εκτελούνται στο application script όπως η αποθήκευση των δεδομένων σε πίνακα MySQL καθώς και η μεταφορά τους μέσω Websockets στο web application. Τέλος επεξηγείται το web application με την αποτύπωση των μετρήσεων σε χάρτη google maps καθώς και η λογική που λειτουργία το app και επικοινωνεί με το application script.

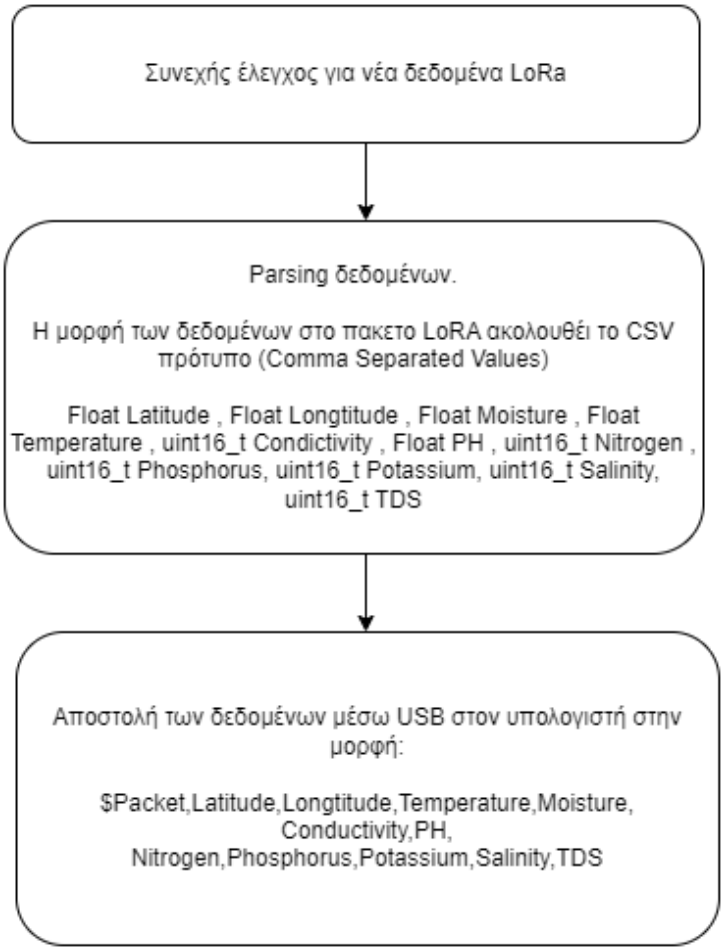
3.2 Δέκτης LoRa ESP32



Σχήμα 3.1 Δέκτης LoRa ESP32 αναπτυξιακή πλακέτα *LILYGO® TTGO T-Beam V1.1*

Ως δέκτης Lora χρησιμοποιήθηκε ένα ESP32 στην αναπτυξιακή πλακέτα *LILYGO® TTGO T-Beam V1.1* η οποία είναι κατασκευασμένη από την κινέζικη εταιρεία LilyGo. Στην συγκεκριμένη πλακέτα συμπεριλαμβάνεται ένας SX1276 πομποδέκτης για την επικοινωνία Lora (όπως χρησιμοποιεί και η συσκευή μέτρησης) , ένα AXP192 τσιπ διαχειριστής ισχύος τους συστήματος , ένα NEO-6M GPS Lora (όπως χρησιμοποιεί και η συσκευή μέτρησης) , στο πίσω μέρος της πλακέτας ένας 18650 υποδοχέας μπαταρίας , micro-usb για σειριακή επικοινωνία, τροφοδοσία και φόρτιση της μπαταρίας , SMA σύνδεσμος για την κεραία του SX1276, LED για ένδειξη διάφορων λειτουργιών, κουμπιά για επανεκκίνηση και ενεργοποίηση/απενεργοποίηση του ESP32 καθώς και κάποιες εκτεθειμένες ακίδες IO προς ελεύθερη χρήση. Πρόκειται για έναν πολύ καλό σχεδιασμό πλακέτας που καλύπτει γκάμα εφαρμογών για ασύρματη επικοινωνία LoRa και αυτοματισμούς. Για τις απαιτήσεις της εφαρμογής του δέκτη απαραίτητη είναι η σειριακή επικοινωνία για ανταλλαγή δεδομένων με τον υπολογιστή καθώς και η Lora επικοινωνία. Η τροφοδοσία της πλακέτας γίνεται μέσω micro-USB , οπότε η χρήση της μπαταρίας δεν είναι αναγκαία.

Το πρόγραμμα που τρέχει στο ESP32 ευθύνεται για την λήψη των δεδομένων από την συσκευή μέτρησης, το parsing αυτών σύμφωνα με το πρότυπος δομής τους , την αναπροσαρμογή από bytes σε string και την επαναποστολή αυτών μέσω του USB στον υπολογιστή. Το διάγραμμα ροής αυτής της διαδικασία παρουσιάζεται στο Σχ. 3.2.



Σχήμα 3.2 Διάγραμμα ροής προγράμματος του δέκτη LoRa ESP32.

3.3 Application script σε Python

Σχήμα 3.3 Λογότυπο Python

Η Python, γνωστή για τη σαφή και ευανάγνωστη σύνταξη της, είναι μια υψηλού επιπέδου, ερμηνευμένη γλώσσα προγραμματισμού που χαίρει ευρείας δημοτικότητας τόσο στους αρχάριους όσο και στους έμπειρους προγραμματιστές. Αναπτύχθηκε από τον Guido van Rossum και κυκλοφόρησε αρχικά το 1991, η Python έχει σχεδιαστεί για να δίνει έμφαση στην αναγνωσιμότητα κώδικα, χρησιμοποιώντας σημαντικό κενό χώρο για την επίτευξη αυτού του στόχου. Παρά την απλότητά της, η Python είναι ισχυρή και προσαρμόσιμη, καθιστώντας την κατάλληλη για ένα ευρύ φάσμα πολύπλοκων εφαρμογών. Στον τομέα της ανάπτυξης web εφαρμογών, η Python είναι καθοριστική, με framework όπως το Flask που επιτρέπουν την αποτελεσματική δημιουργία διαδικτυακών εφαρμογών. Η δύναμη της Python στην επιστήμη δεδομένων και τη μηχανική μάθηση είναι επίσης αξιοσημείωτη, με βιβλιοθήκες όπως που παρέχουν ισχυρά εργαλεία για ανάλυση δεδομένων και μηχανική μάθηση. Η εξέχουσα θέση της γλώσσας στην τεχνητή νοημοσύνη και τους επιστημονικούς υπολογιστές είναι σημαντική, με πολλούς ερευνητές και επιστήμονες να προτιμούν την Python για την ευελιξία και την ευκολία χρήσης της.

Ο ρόλος της Python στο Internet of Things (IoT) είναι ιδιαίτερα κρίσιμος. Η απλότητα και η επεκτασιμότητα του το καθιστούν κατάλληλο για τη διαχείριση πολυπλοκοτήτων του IoT, οι οποίες συχνά περιλαμβάνουν διαχείριση μεγάλου όγκου δεδομένων και επεξεργασία σε πραγματικό χρόνο. Η υποστήριξη της για σειριακή επικοινωνία είναι σημαντική για εργασίες όπως η συλλογή δεδομένων και η διασύνδεση με αισθητήρες και συσκευές. Η ενσωμάτωση της με τη MySQL αποτελεί παράδειγμα των δυνατοτήτων της στη διαχείριση βάσεων δεδομένων. Οι προγραμματιστές μπορούν εύκολα να συνδεθούν με βάσεις δεδομένων MySQL για ένα ευρύ φάσμα εφαρμογών, από υπηρεσίες web έως αποθήκευση και ανάκτηση δεδομένων. Επιπλέον, η υποστήριξη της για WebSockets είναι ένας άλλος τομέας στον οποίο υπερέχει, επιτρέποντας σε πραγματικό χρόνο, αμφίδρομη επικοινωνία μεταξύ των πελατών Ιστού και των διακομιστών. Αυτό είναι ιδιαίτερα χρήσιμο σε εφαρμογές που απαιτούν ζωντανές ενημερώσεις, όπως εφαρμογές συνομιλίας, ενημερώσεις ζωντανών αθλητικών ή συστήματα παρακολούθησης δεδομένων σε πραγματικό χρόνο.

Το εκτεταμένο οικοσύστημα και η υποστήριξή της για διάφορα παραδείγματα προγραμματισμού ενισχύουν την ευελιξία της. Αυτή η προσαρμοστικότητα, σε συνδυασμό με μια προσιτή καμπύλη μάθησης και μια υποστηρικτική κοινότητα, έχει θεμελιώσει την Python της ως μια ευρέως αποδεκτή γλώσσα προγραμματισμού. Τελικά, ο μοναδικός συνδυασμός απλότητας, ευχρηστίας και ευελιξίας, μαζί με το εκτεταμένο εύρος εφαρμογών από την ανάπτυξη ιστού έως το IoT, την καθιστούν μια

γλώσσα-κομβική στον τεχνολογικό τομέα. Συνεχίζει να αποτελεί βασικό πυλώνα σε διάφορους τομείς, από την ταχεία ανάπτυξη εφαρμογών έως την εξελιγμένη ανάλυση δεδομένων και πολλά ακόμη.

Έχοντας υπ' όψην όλα αυτά τα θετικά στοιχεία που αναφέρθηκαν, καθώς και μια εκτεταμένη έρευνα για μια γλώσσα προγραμματισμού που να εξυπηρετεί με τον καλύτερο τρόπο την συσκευή μέτρησης υλοποιήθηκε ένα backend script σε Python. Οι κύριες ανάγκες του application η σειριακή επικοινωνία με τον ESP32 , η αποθήκευση των μετρήσεων σε μια βάση δεδομένων , η υποστήριξη μιας real-time επικοινωνίας με web-clients καθώς και ένας logger που θα αποθηκευόντουσαν μύνηματα λειτουργίας σε κάποιο έγγραφο για μελλοντική επεξεργασία και debugging.

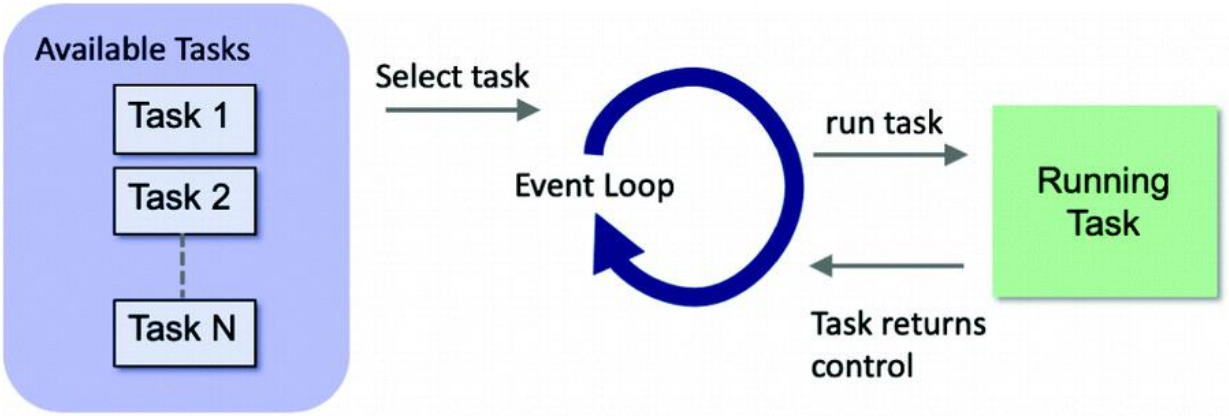
Για την επίτευξη της σύνδεσης σε πραγματικό χρόνο με του clients της ιστοσελίδας χρησιμοποιήθηκε η τεχνολογία Websocket. Το Websocket είναι μια επέκταση ενός HTTP αιτήματος. Το Πρωτόκολλο Μεταφοράς Υπερκειμένου (HyperText Transfer Protocol, HTTP) είναι ένα πρωτόκολλο επικοινωνίας. Αποτελεί το κύριο πρωτόκολλο που χρησιμοποιείται στους browsers του Παγκοσμίου Ιστού για να μεταφέρει δεδομένα ανάμεσα σε έναν διακομιστή (server) και έναν πελάτη (client). Το Websocket είναι μία μοναδική σύνδεση υποδοχής (socket connection) ή τερματική σύνδεση, που επιτρέπει ταυτόχρονη και αμφίδρομη (full-duplex, bi-directional) επικοινωνία πάνω σε μια TCP σύνδεση. Με το πρωτόκολλο WebSocket, το HTTP αίτημα, μετατρέπεται σε ένα μοναδικό αίτημα, για το άνοιγμα μια σύνδεσης υποδοχής, χρησιμοποιώντας την ίδια σύνδεση τόσο από τον πελάτη προς τον εξυπηρετητή, όσο και από τον εξυπηρετητή προς τον πελάτη. Με την τεχνολογία WebSocket, μειώνεται σημαντικά η καθυστέρηση επικοινωνίας και ανταλλαγής δεδομένων, διότι από την στιγμή που εγκατασταθεί μια σύνδεση WebSocket, ο εξυπηρετητής μπορεί και στέλνει πίσω στον πελάτη δεδομένα, όσο αυτά είναι διαθέσιμα. Σε αντίθεση με την τεχνική Polling, με το WebSocket γίνεται ένα μοναδικό αίτημα, ενώ στην συνέχεια, ο εξυπηρετητής δεν χρειάζεται να περιμένει αίτημα από τον πελάτη. Παρόμοια, ο πελάτης μπορεί να στέλνει μηνύματα στον εξυπηρετητή ανά πάσα στιγμή. Αυτό το μοντέλο αποστολής ενός μοναδικού αιτήματος μειώνει σημαντικά την καθυστέρηση, αντί της τεχνικής Polling, με την οποία στέλνεται ένα αίτημα ανά τακτά χρονικά διαστήματα, ανεξάρτητα των δεδομένων που είναι διαθέσιμα. Η TCP θύρα που χρησιμοποιείται από τα websockets είναι η 80 που είναι η καθιερωμένη θύρα του HTTP πρωτοκόλλου. Για να εγκατασταθεί μια σύνδεση πρέπει ο περιηγητής ιστού να στείλει ένα WebSocket handshake αίτημα και ο web server να απαντήσει με μία WebSocket handshake απάντηση. Παρόλο που αρχικά σχεδιάστηκε για χρήση από διακομιστές ιστού, και περιηγητές ιστού αντίστοιχα, μπορεί να χρησιμοποιηθεί σε οποιαδήποτε εφαρμογή που ακολουθεί το μοντέλο πελάτη-διακομιστή. Έτσι, προσφέρεται εύκολη, αμφίδρομη διαρκής επικοινωνία, για όσο η σύνδεση παραμένει ανοιχτή. Πολύ σημαντικό πλεονέκτημα των WebSockets είναι η ευκολία που προσφέρουν στον προγραμματιστή στην ανάπτυξη ενός προγράμματος βασισμένου σε αυτά. Πρακτικά χρειάζεται να υλοποιηθούν 4 βασικές συναρτήσεις. Σε διάφορες υλοποιήσεις υπάρχουν και πιο εξειδικευμένες συναρτήσεις αλλά οι 4 βασικές είναι οι εξής:

- onOpen(): Εκτελείται όταν το WebSocket ανοίγει.
- onClose(): Εκτελείται όταν το WebSocket κλείνει.
- onMessage(): Εκτελείται όταν το WebSocket δεχθεί μήνυμα.
- onError(): Εκτελείται όταν συμβεί κάποιο σφάλμα.

Για την αποθήκευση των δεδομένων χρησιμοποιήθηκε η βάση δεδομένων τύπου SQL. Η SQL (Structured Query Language) είναι μια τυποποιημένη γλώσσα προγραμματισμού που χρησιμοποιείται για τη διαχείριση σχεσιακών βάσεων δεδομένων και την εκτέλεση διαφόρων λειτουργιών στα δεδομένα

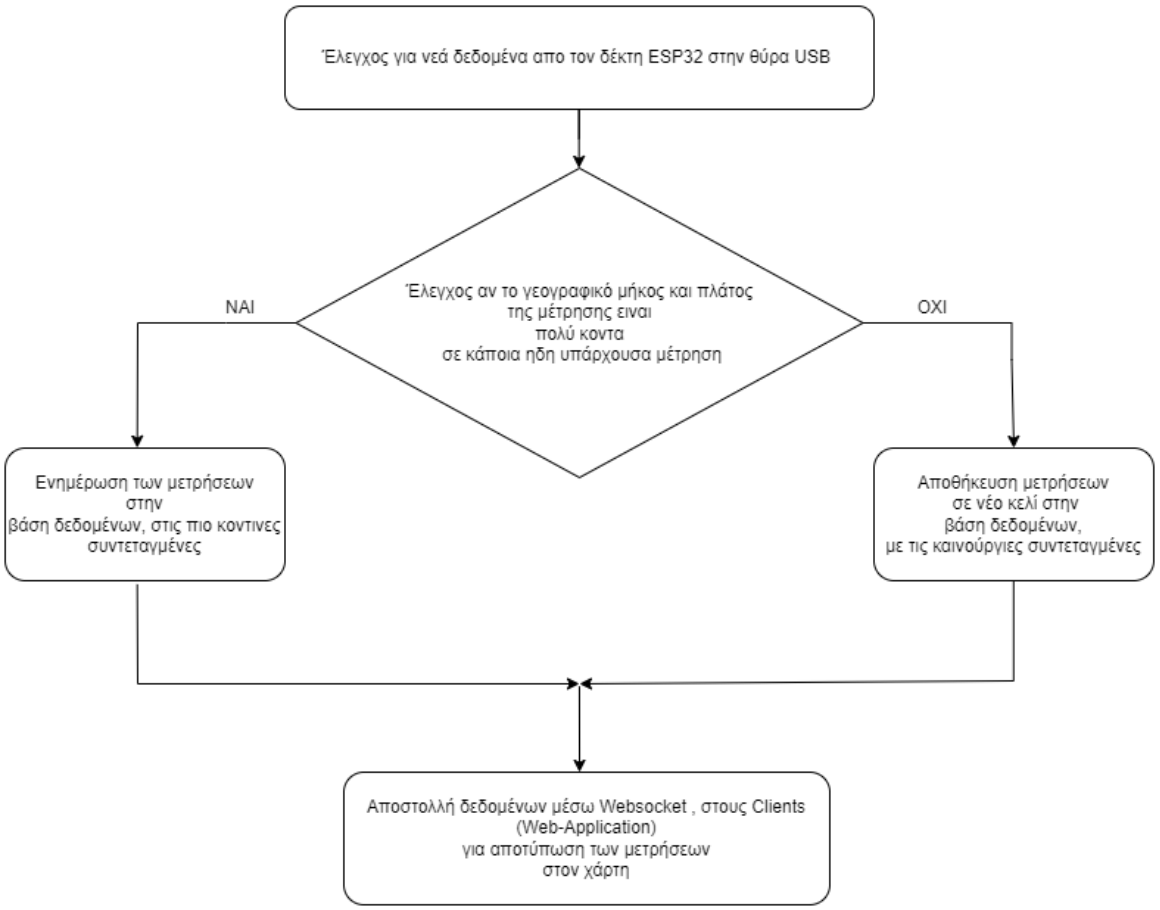
σε αυτές. Αρχικά δημιουργήθηκε τη δεκαετία του 1970, η SQL χρησιμοποιείται τακτικά όχι μόνο από διαχειριστές βάσεων δεδομένων, αλλά και από προγραμματιστές που γράφουν σενάρια ενοποίησης δεδομένων και αναλυτές δεδομένων που θέλουν να δημιουργήσουν και να εκτελέσουν αναλυτικά ερωτήματα. Οι χρήσεις της SQL περιλαμβάνουν την τροποποίηση δομών πινάκων βάσεων δεδομένων και ευρετηρίου. προσθήκη, ενημέρωση και διαγραφή σειρών δεδομένων · και ανάκτηση υποσυνόλων πληροφοριών από μια βάση δεδομένων για εφαρμογές επεξεργασίας συναλλαγών και αναλυτικών στοιχείων. Τα ερωτήματα και άλλες λειτουργίες SQL λαμβάνουν τη μορφή εντολών που γράφονται ως queries. Επίσης γνωστά στις βάσεις δεδομένων SQL, τα σχεσιακά συστήματα περιλαμβάνουν ένα σύνολο πινάκων που περιέχουν δεδομένα σε σειρές και στήλες. Κάθε στήλη σε έναν πίνακα αντιστοιχεί σε μια κατηγορία δεδομένων - για παράδειγμα, όνομα πελάτη ή διεύθυνση - ενώ κάθε σειρά περιέχει μια τιμή δεδομένων για τη διασταυρούμενη στήλη. Στην βάση που υλοποιήθηκε χρησιμοποιείται συγκεκριμένα ή MySQL. Επειδή είναι ανοικτού κώδικα (open source), οποιοσδήποτε μπορεί να κατεβάσει τη MySQL και να την διαμορφώσει με βάση τις ανάγκες του, σύμφωνα πάντα με την γενική άδεια χρήσης. Είναι γνωστή κυρίως για την ταχύτητα, την αξιοπιστία, και την ευελιξία που παρέχει. Η MySQL αυτή τη στιγμή μπορεί να λειτουργήσει σε περιβάλλον Linux, Unix και Windows.

Το κεντρικό framework που χρησιμοποιήθηκε για την διαχείριση όλων των διεργασιών του application είναι το asyncio. Το asyncio σου επιτρέπει να χρησιμοποιήσεις ταυτόχρονες λειτουργίες σε πραγματικό χρόνο , που από φύσης κάποιες λειτουργίες είναι blocking και σε πραγματικά σενάρια θα δημιουργούσαν καθυστερήσεις στην εφαρμογή. Στηρίζεται σε έναν ατέρμονα βρόχο συμβάντων (Event Loop) που τρέχει συνεχώς και χειρίζεται αποδοτικά τα tasks που εκχωρήθηκαν προς εκτέλεση. Πιο λεπτομερώς κάποια βασικά χαρακτηριστικά είναι ότι:



Σχήμα 3.4 Επεξηγητικό γράφημα για την ασύγχρονη εκτέλεση εργασιών μέσω Event Loop , με χρήση asyncio

- Ο βρόχος συμβάντων διατηρεί μια ουρά εργασιών και τις εκτελεί τη μία μετά την άλλη. Κυκλοφορεί συνεχώς μέσα σε αυτήν την ουρά με εργασίες, διαχειρίζοντας την εκτέλεση αυτών.
- Όταν μια εργασία πρέπει να περιμένει να ολοκληρωθεί μια εξωτερική λειτουργία, όπως μια λειτουργία εισόδου/εξόδου (π.χ. ανάγνωση ενός αρχείου, αναμονή για δεδομένα δικτύου, ανάγνωση σειριακής επικοινωνίας), ο βρόχος συμβάντων μπορεί να διακόψει αυτήν την εργασία και να συνεχίσει μια άλλη. Αυτή η ικανότητα επιτρέπει την ταυτόχρονη εκτέλεση χωρίς την ανάγκη πολλαπλών thread.



Σχήμα 3.5 Διάγραμμα ροής προγράμματος του κεντρικού python script για την διαχείριση των μετρήσεων της συσκευής

Ο κώδικας του προγράμματος σπάει σε τρεις κύριες κλάσεις που η κάθε μια δημιουργήθηκε ώστε να έχει απομονωμένη λογική και να είναι υπεύθυνη για μια λειτουργιά μόνο. Η κλάση Serial έχει μεθόδους:

- Για να δημιουργήσει μια νέα σύνδεση μέσω USB με το ESP32.
- Να εκτελέσει επανεκκίνηση στο ESP32.
- Να στείλει δεδομένα προς το ESP32 , να λάβει νέα δεδομένα και να στείλει προς επεξεργασία.

Η κλάση Websockets έχει λειτουργίες αμφίδρομης επικοινωνία με τους clients της web σελίδας και πιο συγκεκριμένα:

- ✓ Αναζήτηση της IP που πρόκειται να ξεκινήσει ο Websocket server.
- ✓ Λειτουργία αναμετάδοσης μηνύματος για συγχρονισμό των δεδομένων σε όλους τους clients.
- ✓ Λειτουργία για στείλει δεδομένα ή να λάβει δεδομένα.

Η τρίτη κλάση Mysql έχει λειτουργίες επικοινωνία με την βάση δεδομένων για την αποθήκευση και ανάκτηση των μετρήσεων της συσκευής αναλυτικότερα:

- Λειτουργία σύνδεσης με βάση δεδομένων τύπου MySQL
- Δυνατότητα εισαγωγής , διαγραφής , και ενημέρωσης κελιού

#	Name	Datatype	Length/Set	Unsigned	Allow N...	Zerofill	Default
1	**id**	**INT**	**10**	☐	☐	☐	**AUTO_INCREMENT**
2	latitude	VARCHAR	50	☐	☑	☐	NULL
3	longtitude	VARCHAR	50	☐	☑	☐	NULL
4	data	VARCHAR	512	☐	☑	☐	NULL
5	last_updated	DATETIME		☐	☑	☐	NULL ON UPDATE CU
6	description	VARCHAR	255	☐	☑	☐	'Sample'

Πίνακας 3.1 Ο μορφή του MySQL πίνακα soil-measurements για την αποθήκευση όλων των στοιχείων της μέτρησης των στοιχείων του εδάφους

Για την συγκεκριμένη εφαρμογή υλοποιήθηκε ένας ενιαίος πίνακας που συμπεριλαμβάνει όλα τα απαραίτητα κελιά για την κατηγοριοποίηση των δεδομένων. Το κελί *id* υποδηλώνει ένα μοναδικό ακέραιο αριθμό που αυτός αυξάνεται κάθε φορά που εκχωρείται ένα νέο στοιχείο στον πίνακα. Το κελί *latitude* που αναφέρεται στο γεωγραφικό πλάτος από τις συντεταγμένες που πάρθηκε η μέτρηση. Το κελί *longtitude* που αναφέρεται στο γεωγραφικό πλάτος από τις συντεταγμένες που πάρθηκε η μέτρηση. Στο κελί *data* αποθηκεύονται όλες οι μετρήσεις σε μορφή JSON. Το last_updated κελί περιέχει την ημερομηνία που ενημερώθηκε η μέτρηση και είναι της μορφής "2000-12-31 14:58:31". Στο *description* κελί αποθηκεύεται σε ελεύθερο κείμενο μια περιγραφή για την μέτρηση για παράδειγμα ένα όνομα του τύπου "μέτρηση κοντά στο δέντρο χρώμα εδάφους καφέ".

Παράδειγμα μέτρησης μορφής JSON στο κελί *data:*

- *{*
- *"Temperature": "Temperature:25.00 u00b0C",*
- *"Moisture": "Moisture:25.40 %",*
- *"Conductivity": "Conductivity:296 us/cm",*
- *"PH": "PH:7.00 PH",*
- *"Nitrogen": "Nitrogen:21 mg/kg",*
- *"Phosphorus": "Phosphorus:28 mg/kg",*
- *"Potassium": "Potassium:71 mg/kg",*
- *"Salinity": "Salinity:162 ppt",*
- *"TDS": "TDS:148 ppm"*
- *}*

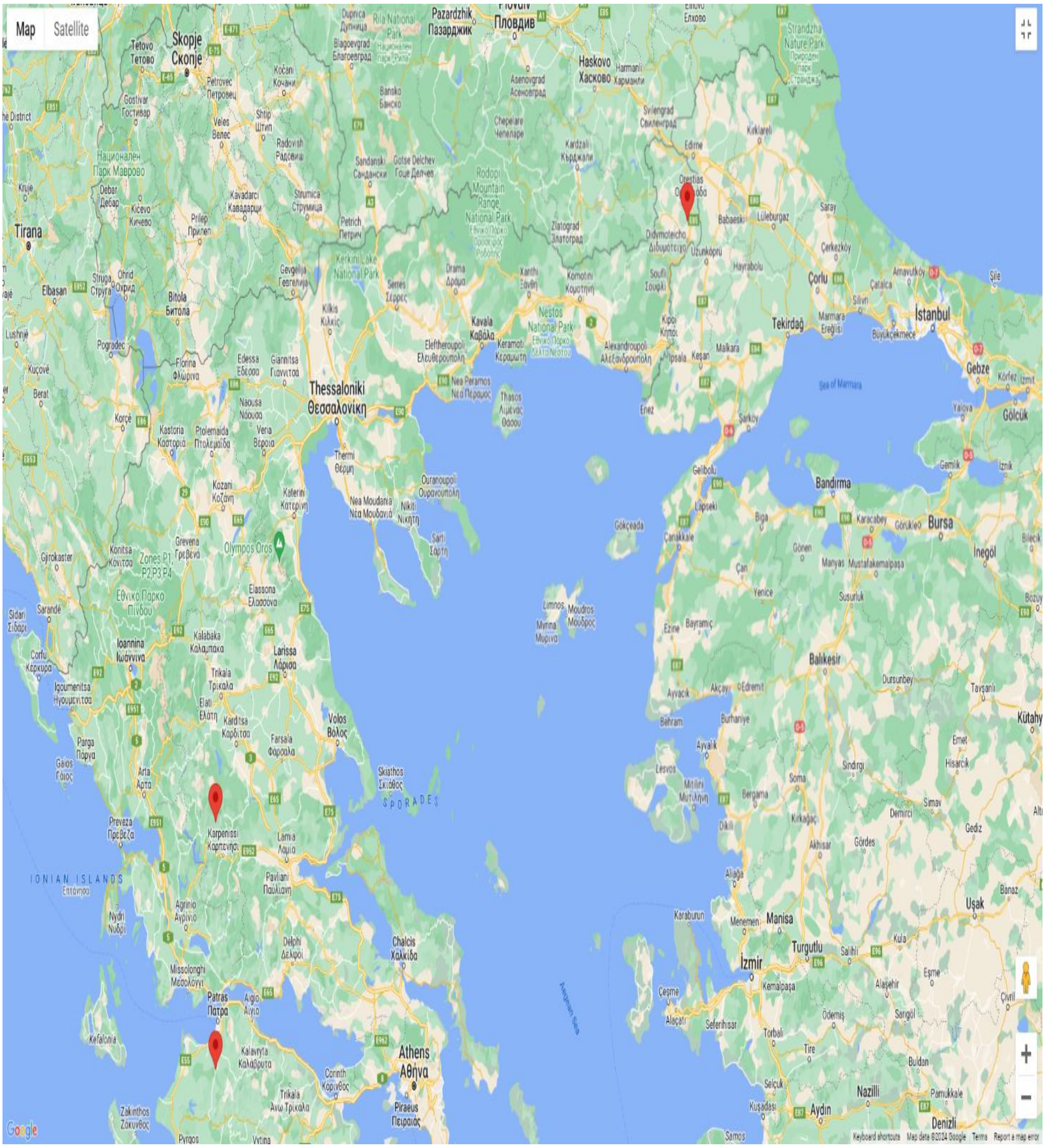
Η γενική λειτουργία της εφαρμογής περιγράφεται στο διάγραμμα ροής Σχ. 3.5. Αρχικά γίνεται συνεχής έλεγχος για νέα δεδομένα στην θύρα USB. Εφόσον υπάρχουν νέα δεδομένα , ελέγχονται οι συντεταγμένες αυτών. Αν οι συντεταγμένες είναι πολύ κοντινές από μια ήδη αποθηκευμένη μέτρηση στην βάση δεδομένων τότε γίνεται ενημέρωση των μετρήσεων στο κελί *data*. Αλλιώς γίνεται νέα εκχώρηση με όλες τι καινούργιες μετρήσεις. Τελικά τα δεδομένα στέλνονται και αποτυπώνονται στον χάρτη, που αναφέρεται και στα παρακάτω κεφάλαια διεξοδικά. Η επιλογή της μεθόδου στο να μην αποθηκεύονται πολύ κοντινές τιμές ακολουθείται για να μην υπάρχει θορυβώδης εισαγωγή ανεξέλεγκτων τιμών καθώς το NEO-6M GPS παρουσιάζει θόρυβο στον προσδιορισμό των συντεταγμένων με κάποιο μέγιστο σφάλμα απόκλισης. Η ακριβής τιμής του σφάλματος θα υπολογιστεί έπειτα από πειραματικές μετρήσεις και παρατήρηση αυτών.

3.4 Web εφαρμογή

Μια web εφαρμογή (εφαρμογή ιστού) είναι ένα λογισμικό εφαρμογών που εκτελείται σε έναν διακομιστή ιστού, σε αντίθεση με τα προγράμματα λογισμικού που βασίζονται σε υπολογιστή και αποθηκεύονται τοπικά στο Λειτουργικό Σύστημα (OS) της συσκευής. Η πρόσβαση στις εφαρμογές Ιστού γίνεται από τον χρήστη μέσω ενός προγράμματος περιήγησης ιστού με ενεργή σύνδεση σε ένα τοπικό δίκτυο (ή το Διαδίκτυο). Τα βασικά στοιχεία που απαρτίζουν μια web εφαρμογή είναι αρχεία HTML , CSS και JavaScript.

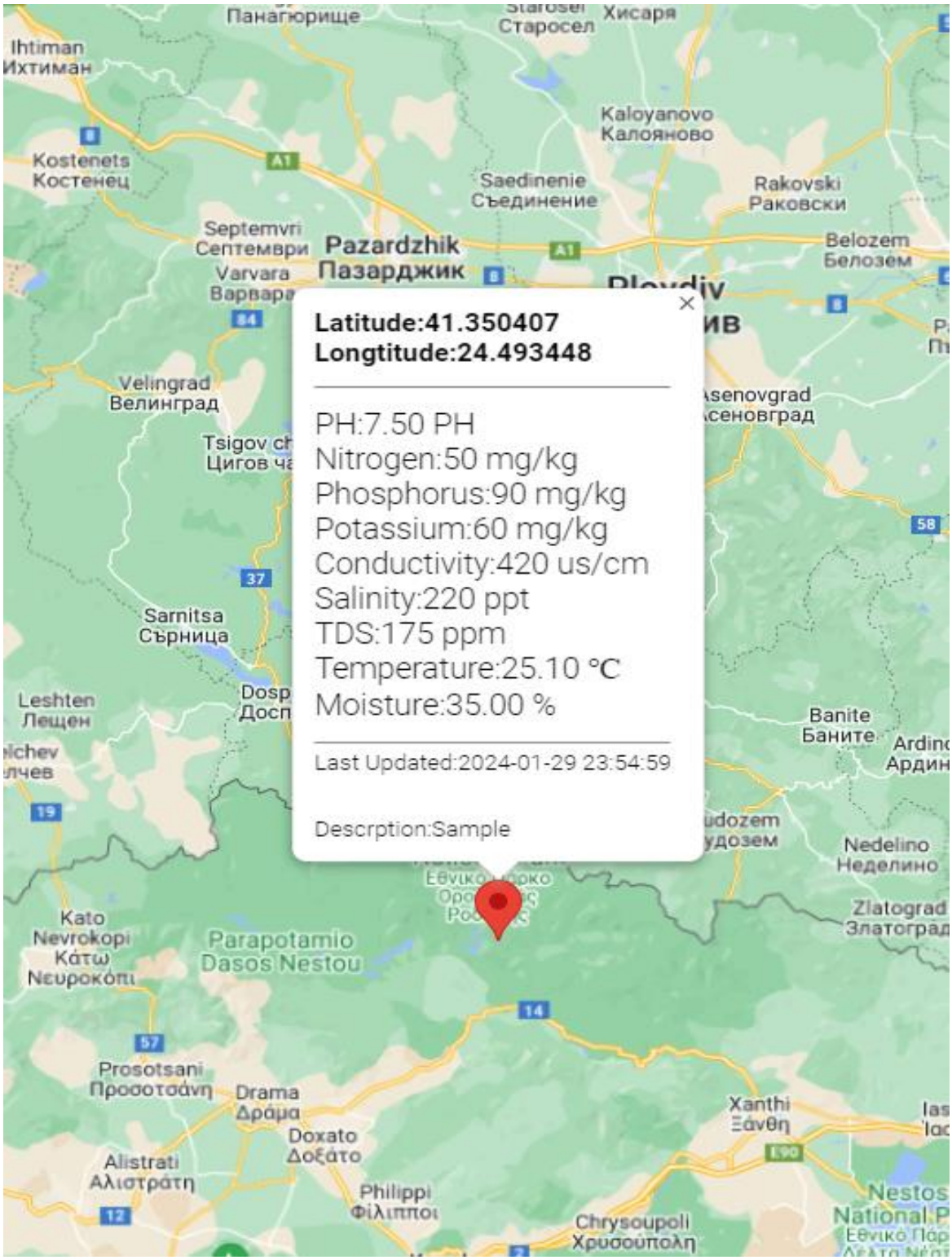
Η HTML (HyperText Markup Language) Γλώσσα Σήμανσης Υπερκειμένου είναι η κύρια γλώσσα σήμανσης για τις ιστοσελίδες, και τα στοιχεία της είναι τα βασικά δομικά στοιχεία των ιστοσελίδων. Η HTML γράφεται υπό μορφή elements (στοιχείων) HTML τα οποία αποτελούνται από ετικέτες (tags), οι οποίες περικλείονται μέσα σε σύμβολα «μεγαλύτερο από» και «μικρότερο από» (για παράδειγμα <html>), μέσα στο περιεχόμενο της ιστοσελίδας. Οι ετικέτες HTML συνήθως λειτουργούν ανά ζεύγη (για παράδειγμα <h1> και </h1>), με την πρώτη να ονομάζεται ετικέτα έναρξης και τη δεύτερη ετικέτα λήξης (ή σε άλλες περιπτώσεις ετικέτα ανοίγματος και ετικέτα κλεισίματος αντίστοιχα). Ανάμεσα στις ετικέτες, οι σχεδιαστές ιστοσελίδων μπορούν να τοποθετήσουν κείμενο, πίνακες, εικόνες κλπ. Ο σκοπός ενός web browser είναι να διαβάζει τα έγγραφα HTML και να τα συνθέσει σε σελίδες που μπορεί κανείς να διαβάσει ή να ακούσει. Ο browser δεν εμφανίζει τις ετικέτες HTML, αλλά τις χρησιμοποιεί για να παρουσιάσει το περιεχόμενο της σελίδας. Η CSS (Cascading Style Sheets – διαδοχικά φύλλα ύφους ή επάλληλα φύλλα ύφους) είναι μια γλώσσα υπολογιστή που ανήκει στην κατηγορία των γλωσσών φύλλων ύφους που χρησιμοποιείται για τον έλεγχο της εμφάνισης ενός εγγράφου που έχει γραφτεί με μια γλώσσα σήμανσης. Χρησιμοποιείται δηλαδή για τον έλεγχο της εμφάνισης ενός εγγράφου που γράφτηκε σε γλώσσα HTML, δηλαδή για τον έλεγχο της εμφάνισης μιας ιστοσελίδας και γενικότερα ενός ιστοτόπου. Η CSS είναι μια γλώσσα υπολογιστή προορισμένη να αναπτύσσει στιλιστικά μια ιστοσελίδα δηλαδή να διαμορφώνει περισσότερα χαρακτηριστικά, χρώματα, στοίχιση και δίνει περισσότερες δυνατότητες σε σχέση με την html. Για μια όμορφη και καλοσχεδιασμένη ιστοσελίδα η χρήση της CSS κρίνεται ως απαραίτητη. Η JavaScript (JS) είναι διερμηνευμένη γλώσσα προγραμματισμού για ηλεκτρονικούς υπολογιστές. Αρχικά αποτέλεσε μέρος της υλοποίησης των φυλλομετρητών Ιστού, ώστε τα σενάρια από την πλευρά του πελάτη (client-side scripts) να μπορούν να επικοινωνούν με τον χρήστη, να ανταλλάσσουν δεδομένα ασύγχρονα και να αλλάζουν δυναμικά το περιεχόμενο του εγγράφου που εμφανίζεται. Η JavaScript είναι μια γλώσσα σεναρίων που βασίζεται στα πρωτότυπα (prototype-based), είναι δυναμική, με ασθενείς τύπους και έχει συναρτήσεις ως αντικείμενα πρώτης τάξης. Η σύνταξή της είναι επηρεασμένη από τη C. Η JavaScript αντιγράφει πολλά ονόματα και συμβάσεις ονοματοδοσίας από τη Java, αλλά γενικά οι δύο αυτές γλώσσες δε σχετίζονται και έχουν πολύ διαφορετική σημασιολογία. Είναι γλώσσα βασισμένη σε διαφορετικά προγραμματιστικά παραδείγματα (multi-paradigm), υποστηρίζοντας αντικειμενοστρεφές, προστακτικό και συναρτησιακό στυλ προγραμματισμού.

Βασίζοντας στις τρεις γλώσσες ανάπτυξης ιστοσελίδας αναπτύχθηκε η web εφαρμογή όπου το ζητούμενο ήταν ναι μπορούν να εμφανιστούν οι τιμές των μετρήσεων του εδάφους σε έναν χάρτη. Για την ανάπτυξη, δημιουργία και αλληλεπίδραση με τον χάρτη ήταν αναγκαίο να χρησιμοποιηθεί ένα API χαρτών καθώς για να γίνει κάτι τέτοιο από την αρχή θα ήταν πολύ δύσκολο και χρονοβόρο. Επιλέχθηκε να χρησιμοποιηθεί το Google Maps API μέσω JavaScript διότι παρέχονται πολλές μέθοδοι παραμετροποίησης και γραφημάτων στον ενημερωμένο παγκόσμιο χάρτη που διατηρεί η Google.



Σχήμα 3.6 Γενική εικόνα χάρτη της web εφαρμογής με τρεις markers σαν παράδειγμα επίδειξης

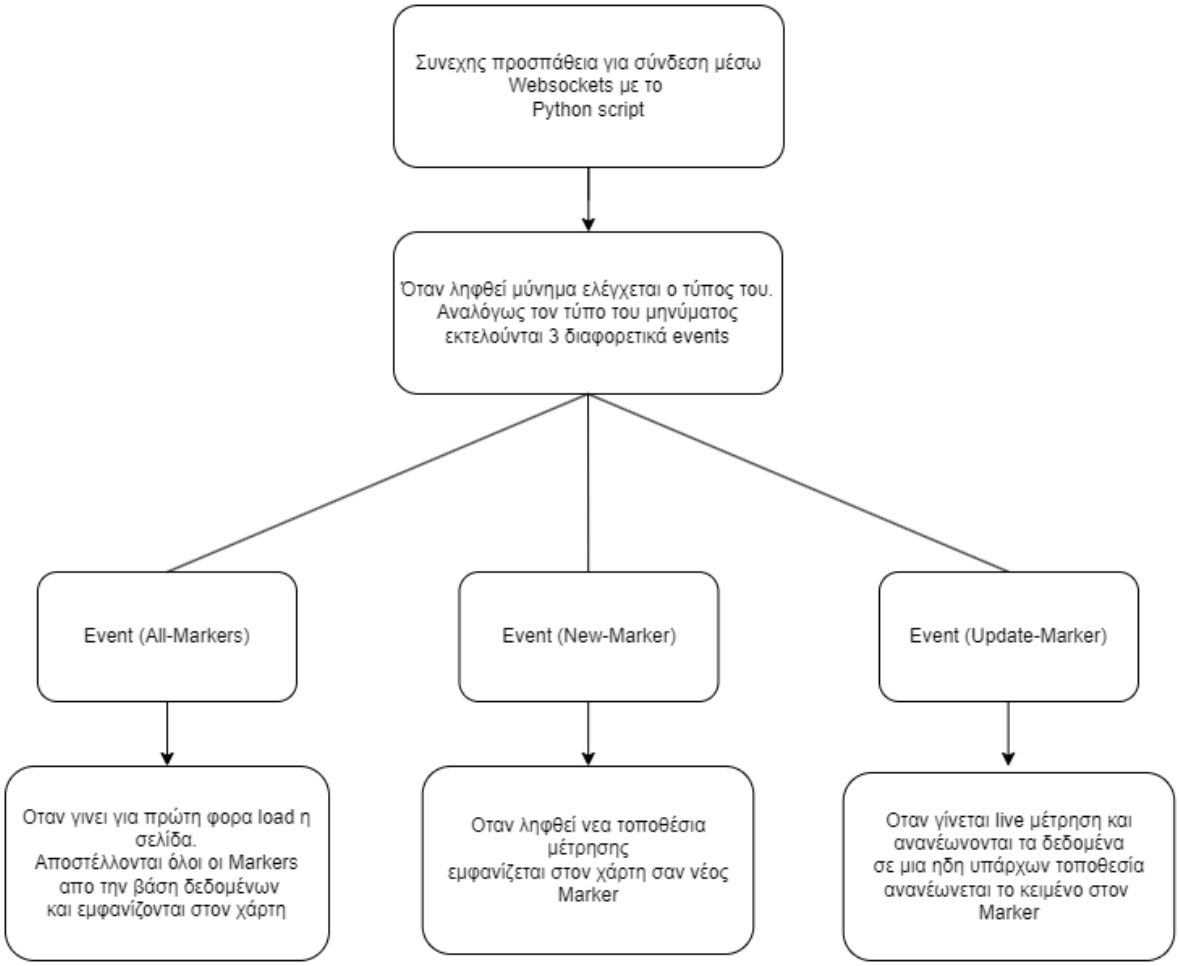
Η γενική ιδέα της εφαρμογής είναι η απεικόνιση όλων των διαφορετικών μετρήσεων που έχουν παρθεί σαν κόκκινες πινέζες (markers) στις συντεταγμένες που πραγματοποιήθηκαν. Ο χρήστης θα μπορεί να πατήσει πάνω σε κάθε marker και θα βγαίνει ένα αναδυόμενο παράθυρο με τις τιμές των μετρήσεων και την ημερομηνία από την τελευταίας ενημέρωσης τους. Σε περίπτωση που κάποια μέτρηση πραγματοποιείται σε πραγματικό χρόνο , ο κόκκινος marker έχει ένα animation αναπήδησης για να δώσει την πληροφορία ότι πρόκειται για live μέτρηση που ανανεώνει τα δεδομένα της συνεχώς.



Σχήμα 3.7 Παράδειγμα απεικόνισής μετρήσεων σε τυχαίο σημείο στον χάρτη

Όπως απεικονίζεται και στο διάγραμμα ροής της JavaScript μόλις φορτωθεί η σελίδα η πρώτη ενέργεια που εκτελείται είναι η σύνδεση μέσω Websockets με το Python script. Μόλις η σύνδεση είναι επιτυχής δημιουργούνται handlers για το άνοιγμα και κλείσιμο της σύνδεσης καθώς και την λήψη μηνύματος. Το μήνυμα που εισέρχεται έχει μορφή JSON έχοντας κάθε φορά ένα πεδίο "type" που καθορίζει αναλόγως την τιμή του το event που θα εκτελεστεί από το πρόγραμμα. Τα τρία διαφορετικά events είναι τα ("all-markers","new-marker","update-marker") που το ονόματα τους δηλώνουν και την σημασία

τους. Το “all-markers” event εκτελείται στην πρώτη φόρτωση της σελίδας ώστε να έρθουν όλα τα Markers που έχουν αποθηκευτεί στην βάση δεδομένων μαζί με όλες τις μετρήσεις του εδάφους. Το “new-marker” event χρησιμοποιείται όταν υπάρχει μια μέτρηση σε νέα διαφορετική τοποθεσία που δεν υπάρχει στην βάση. Το “update-marker” event τρέχει κάθε φορά που έρχονται νέα δεδομένα μέτρησης για μια ήδη υπάρχων τοποθεσία καταχωρημένη στην βάση.



Σχήμα 3.8 Διάγραμμα ροής JavaScript κώδικα που τρέχει στο Web application και απεικονίζει τις μετρήσεις στον χάρτη.

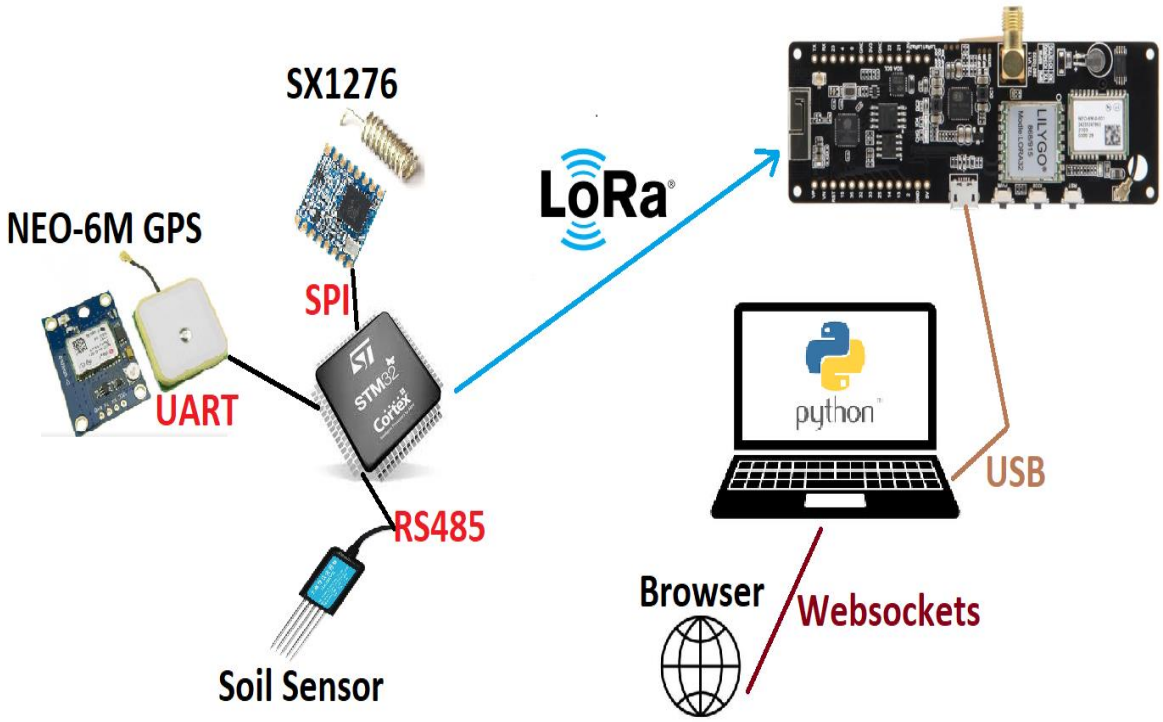
Κάθε Marker έχει μοναδικά δεδομένα εκτός από τις μετρήσεις των συστατικών τους εδάφους ένα κελί τελευταίας ενημέρωσης (Last Updated) και μια περιγραφή (Description). Το κελί (Last Updated) επιλέχθηκε να χρησιμοποιηθεί για να δώσει ένα νόημα στην μέτρηση που πάρθηκε καθώς μετά από ένα μεγάλο χρονικό διάστημα το συστατικά στο έδαφος υπάρχει μεγάλή πιθανότητα να έχουν αλλάξει. Το κελί (Description) χρησιμοποιείται από τον χρήστη για να δώσει ένα περιγραφικό όνομα στην μέτρηση για να του θυμίζει που ακριβώς έγινε η τι ακριβώς μετρήθηκε για παράδειγμα «μέτρηση ένα μέτρο αριστερά από το δέντρο πορτοκαλιάς» , μια περιγραφή που να δίνει πληροφορία για την ακριβής τοποθεσία.

Κεφάλαιο 4ο : Ανάλυση συσκευής πειράματα και μετρήσεις

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο αυτό αναφέρεται συνολικά η αναπαράσταση του συστήματος που αναπτύχθηκε σε αυτήν την εργασία , η κατασκευή της συσκευής μέτρησης σε στάδιο πρωτότυπου , καθώς και οι μετρήσεις και τα πειράματα που πραγματοποιήθηκαν. Αρχικά, γίνεται λόγος για την διαδικασία κατασκευής της πλακέτας με κάποιες φωτογραφίες και περιγραφές καθώς και στα αναπτυξιακά-περιφερειακά που επιλέχθηκαν να χρησιμοποιηθούν. Εκτελούνται πειράματα για την αξιοπιστία των μετρήσεων του αισθητήρα και αποτυπώνονται με φωτογραφίες. Τα πειράματα που εκτελούνται ελέγχουν την αγωγιμότητα και το pH καθώς η θερμοκρασία και η υγρασία διακρίνονται εύκολα ότι ανταποκρίνονται εύκολα στις πραγματικές συνθήκες. Τέλος γίνεται αναφορά σε δείγματα εδάφους που μαζεύτηκαν από διαφορετικές περιοχές με κριτήριο επιλογής το χρώμα τους , οι μετρήσεις αυτών από γεωπόνο καθώς και από την συσκευή.

4.2 Αναπαράσταση του συστήματος



Σχήμα 4.1 Συνολική αναπαράσταση του συστήματος μέτρησης συστατικών εδάφους ως προ την ανταλλαγή/εξαγωγή της πληροφορίας.

Η ιδέα ανάπτυξης του συστήματος στηρίζεται στο παρακάτω σενάριο λειτουργίας.

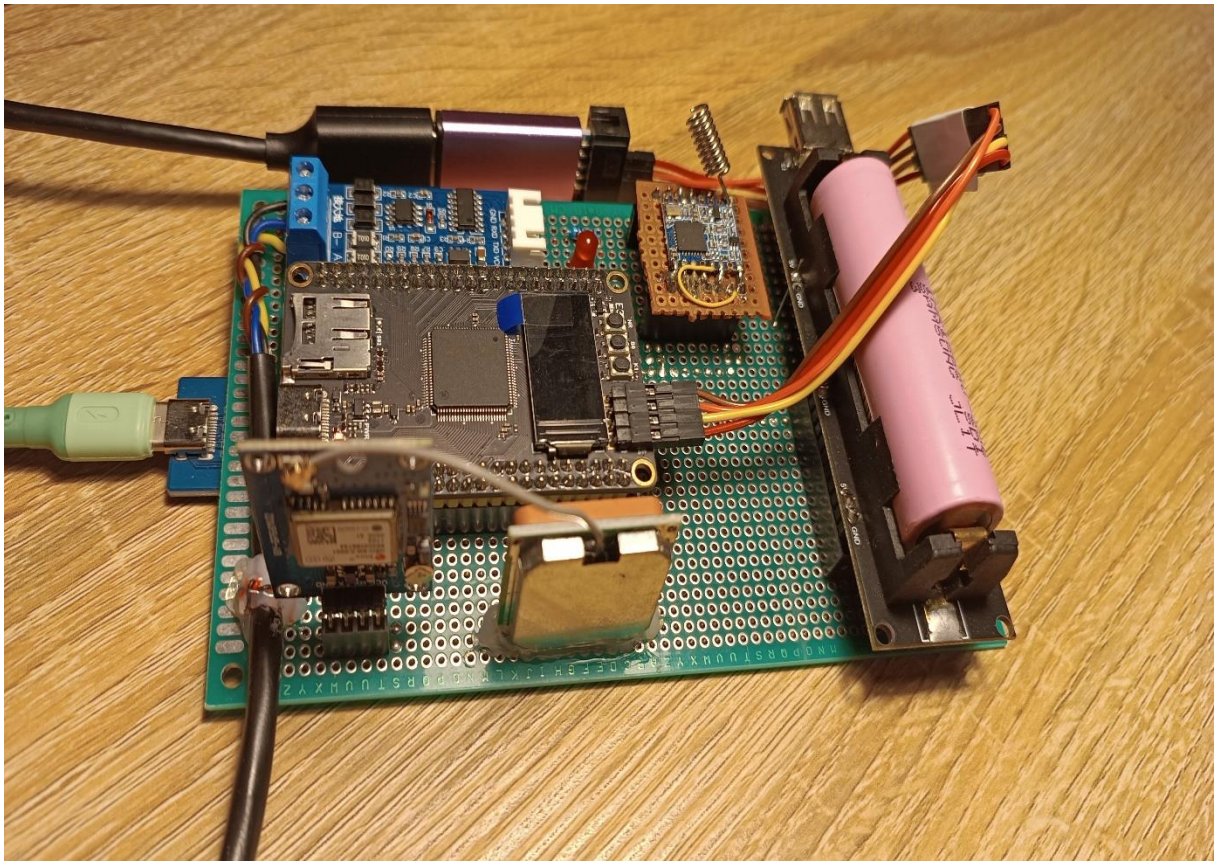
- Ο χρήστης τοποθετεί τον αισθητήρα μέτρησης μέσα στο έδαφος.

- Η συσκευή μέτρησης αποστέλλει τα δεδομένα στον δέκτη LoRa ESP32 (σε μια μέγιστη απόσταση 2-3 χιλιομέτρων με line of sight). Η διαδικασία αυτή εκτελείται με απαραίτητη προϋπόθεση ότι υπάρχει επικοινωνία με δορυφόρο και έχει γίνει προσδιορισμός γεωγραφικών συντεταγμένων στην συσκευή.
- Ο δέκτης LoRa λαμβάνει τα δεδομένα μέτρησης τα επεξεργάζεται και τα στέλνει στον ηλεκτρονικό υπολογιστή μέσω της θύρας USB.
- Ο υπολογιστής τρέχει διαρκώς έναν python script υποστηρίζοντας την high level εφαρμογή του συστήματος.
- Τα δεδομένα επεξεργάζονται , αποθηκεύονται και εμφανίζονται σε Χάρτη σε ιστοσελίδα.

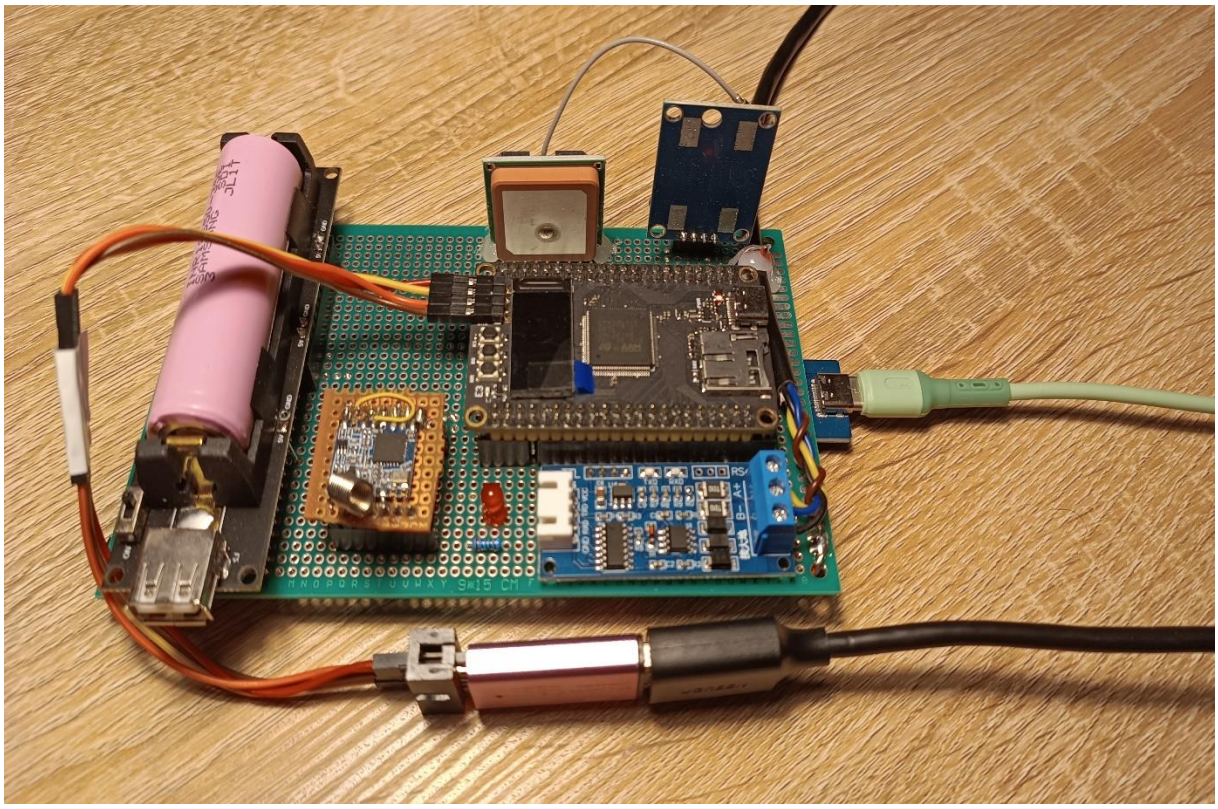
Η επικοινωνία LoRa μέσω εκπομπού και δέκτη είναι μονόδρομη καθώς δεν δημιουργήθηκαν ανάγκες να αποστέλλονται δεδομένα πίσω στην συσκευή μέτρησης.

4.3 Κατασκευή συσκευής

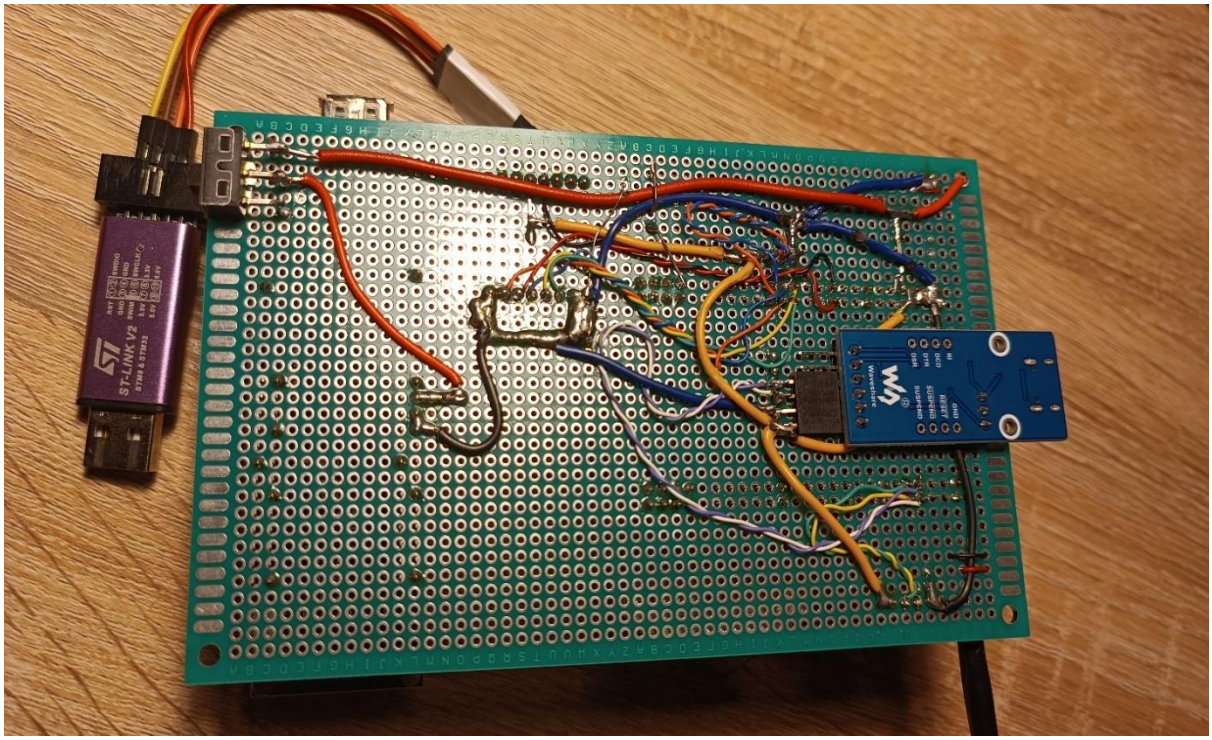
Η κατασκευή της συσκευής έγινε σε πλακέτα prototyping 90x150mm διπλής όψης. Αυτή ή επιλογή έγινε καθαρά για λόγους ανάπτυξης αφού δεν υπήρχαν ανάγκες σημάτων μεγάλων συχνοτήτων , γεγονός που δεν θέτει αναγκαίο το σχεδιασμό και εκτύπωση PCB.



Σχήμα 4.2 Πρωτότυπο συσκευής μέτρησης συνολική μπροστινή εικόνα δεξιάς όψης



Σχήμα 4.3 Πρωτότυπο συσκευής μέτρησης συνολική μπροστινή εικόνα αριστερής όψης



Σχήμα 4.4 Πρωτότυπο συσκευής μέτρησης συνολική οπίσθια εικόνα

Για την κατασκευή της πρωτότυπης συσκευής προτιμήθηκε να χρησιμοποιηθούν υλικά τα οποία είναι διαθέσιμα στο εμπόριο και σε χαμηλό κόστος. Τα περισσότερα υλικά έχουν τοποθετηθεί σε θηλυκά headers να γίνει ευκολότερη η αντικατάσταση τους σε περίπτωση ενδεχομένης βλάβης. Κατασκευάστηκε ένα βοηθητικό πλακετάκι για να κολληθεί πάνω του το SX1276 LoRa module καθώς

δεν είναι συμβατό με τις τρύπες 2.56mm της prototyping πλακέτας. Χρησιμοποιήθηκε ένα κόκκινο LED για την ένδειξη της μεταφοράς των δεδομένων μέσω LoRa. Το LED αναβοσβήνει αναλόγως τον ρυθμό μετάδοσης. Για την ανάγκη φορητότητας της συσκευής χρησιμοποιήθηκε μια μπαταρία ιόντων λιθίου τύπου 18650.

Κάποια βασικά χαρακτηριστικά της μπαταρίας είναι :

1. Βασικό ρεύμα φόρτισης: 1.7A , προτείνεται 1,020A για μεγάλη ζωή μπαταρίας
2. Μέγιστο ρεύμα φόρτισης: 2A
3. Μέγιστο ρεύμα εκ φόρτισής : 8A (σε συνεχή χρήση) , 13A (στιγμιαίο ρεύμα)
4. Τάση αποκοπής : 2.65V (Πιο κάτω από αυτό το κατώφλι δεν θα πρέπει να χρησιμοποιείται)

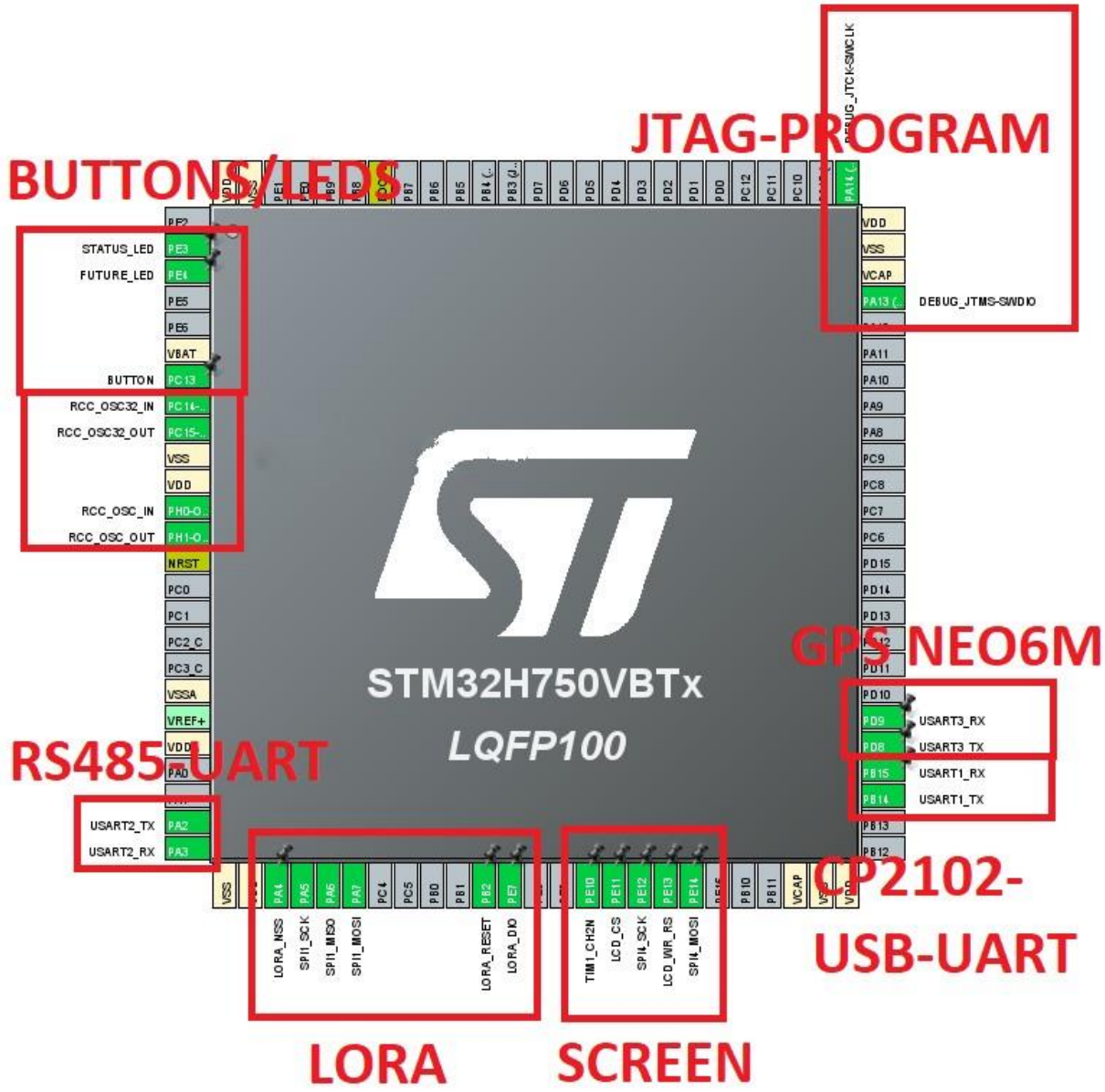
Σχήμα 4.5 Μπαταρία ιόντων λιθίου 18650 με χωρητικότητα 3500 mAh.

Η μπαταρία ιόντων λιθίου από μόνη της μπορεί να δώσει τάση από 2.7V έως 4.2V. Για να παραχθεί 5V στην συσκευή για την τροφοδοσία του αισθητήρα μέτρησης εδάφους θα πρέπει να χρησιμοποιηθεί step up μετατροπέας. Υπάρχουν τέτοιοι μετατροπείς στο εμπόριο σε χαμηλό κόστος σε πλακέτες απλής λειτουργίας ή και συνδυάζοντας παραπάνω λειτουργίες όπως φόρτιση μπαταρίας κ.α. Έγινε επιλογή πλακέτας όπως φαίνεται στο Σχ4.6 που συνδυάζει φόρτιση μπαταρίας , σταθεροποίηση τάσεων στα 5V, 3.3V και θέση τοποθέτησης μπαταρίας 18650.

Για λόγους αποσφαλμάτωσης και απεικόνισης μηνυμάτων προτιμήθηκε να προστεθεί στην συσκευή ένας μετατροπέας από USB σε UART πρωτόκολλο όπως φαίνεται στο Σχ4.7. Ο μετατροπέας μπορεί να τροφοδοτηθεί με 3,3V και με 5V. Προτιμήθηκε να τροφοδοτηθεί με 3.3V ώστε να μην υπάρχουν πολλές είσοδοι για κεντρική τροφοδοσία των 5V στην συσκευή (πολλαπλά μονοπάτια κεντρικής τροφοδοσίας).

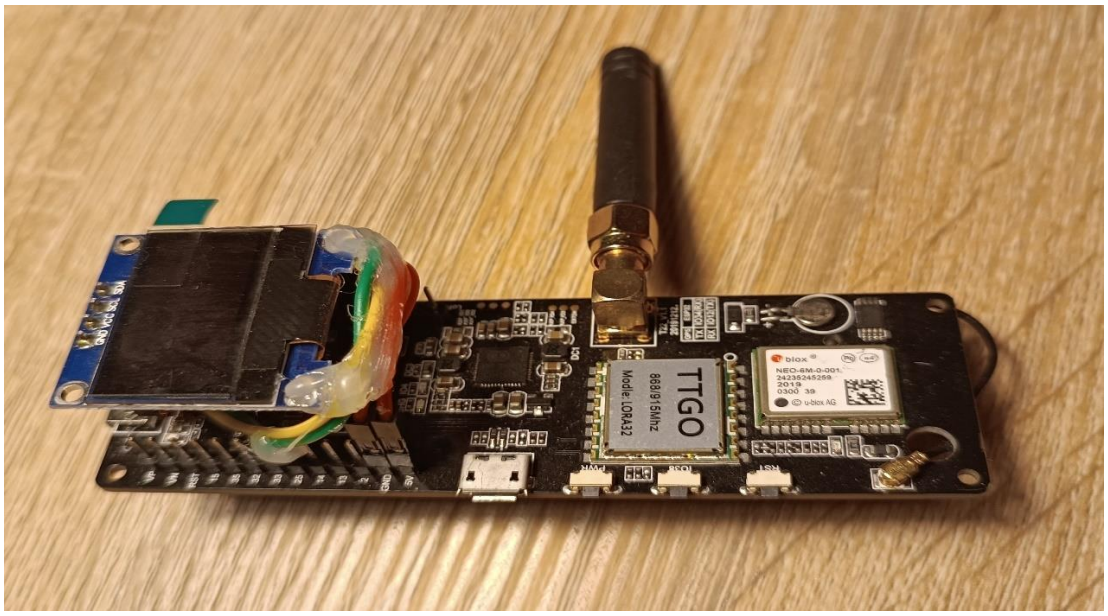
Σχήμα 4.6 Φορτιστής μπαταρίας 18650 και step up μετατροπέας στα 5V

Σχήμα 4.7 USB σε UART μετατροπέας CP2102



Σχήμα 4.8 Διάγραμμα συνδέσεων στον STM32H750 όπως επιλέχθηκαν μέσα στο STM32CUBE-IDE

- PA2, PA3 (TX, RX) πόδια για το RS485 module
- PD9, PD8 (TX, RX) πόδια για τον μετατροπέα από USB σε UART (CP2102)
- PA4, PA5, PA6, PA7 (SS, SCK, MISO, MOSI) πόδια SPI για το SX1276
- PB2, PE7 (RESET, DIO) έξτρα πόδια για την επικοινωνία με το SX1276
- PE10, PE11, PE12, PE13, PE14 (LCD-PWM, SS, SCK, RS/DC, MOSI) ποδιά SPI και ελέγχου για την IPS lCD οθόνη 0.96 ιντσών.
- PC13, PE3 (Κουμπί γενικής χρήσης, LED γενικής χρήσης)
- PA13, PA14 (JTAG-Debug/Προγραμματισμός) του STM32
- PC14, PC15 (Κρυσταλλικός ταλαντωτής χαμηλών συχνοτήτων 32.768KHz)
- PH0-OSC-IN, PH1-OSC-OUT (Κρυσταλλικός ταλαντωτής υψηλών συχνοτήτων 25MHz)
- PD9, PD8 (TX, RX) πόδια για την UART επικοινωνία με το NEO-6M GPS module



Σχήμα 4.9 Έτοιμη πλακέτα ESP32 LoRa δέκτη *LILYGO® TTGO T-Beam V1.1*

4.4 Πείραμα αξιολόγησης μέτρησης ηλεκτρικής αγωγιμότητας και pH

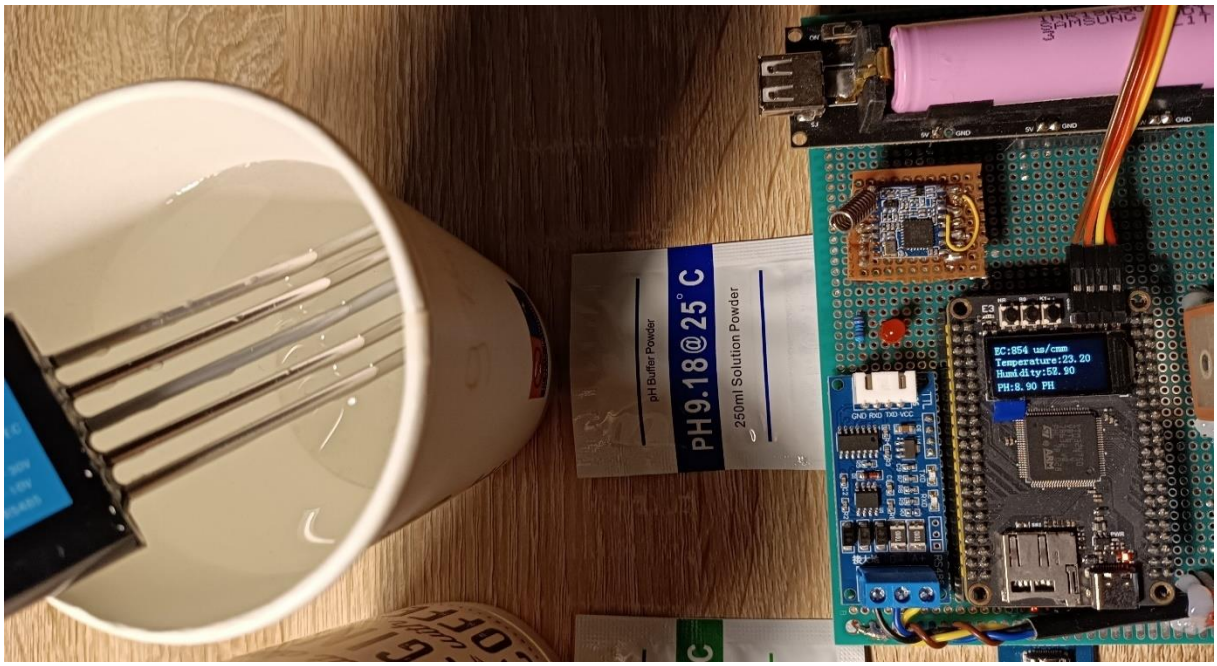
Για την αξιολόγηση των μετρήσεων του αισθητήρα στο pH του εδάφους , πάρθηκαν τρία φακελάκια ειδική σκόνης με τρείς συγκεκριμένες τιμές pH που μετρήθηκαν από τον κατασκευαστή στους 25 βαθμούς. Τα συγκεκριμένα φακελάκια χρησιμοποιούνται για βαθμονόμηση οργάνων που μετρούν το pH. Το πρώτο βήμα της διαδικασίας ήταν να φτιαχτούν τα τρία διαλείμματα. Το νερό που θα

Σχήμα 4.10 Πείραμα μέτρησης pH σε γνωστό υδατικό διάλειμμα με τιμή pH 6.86

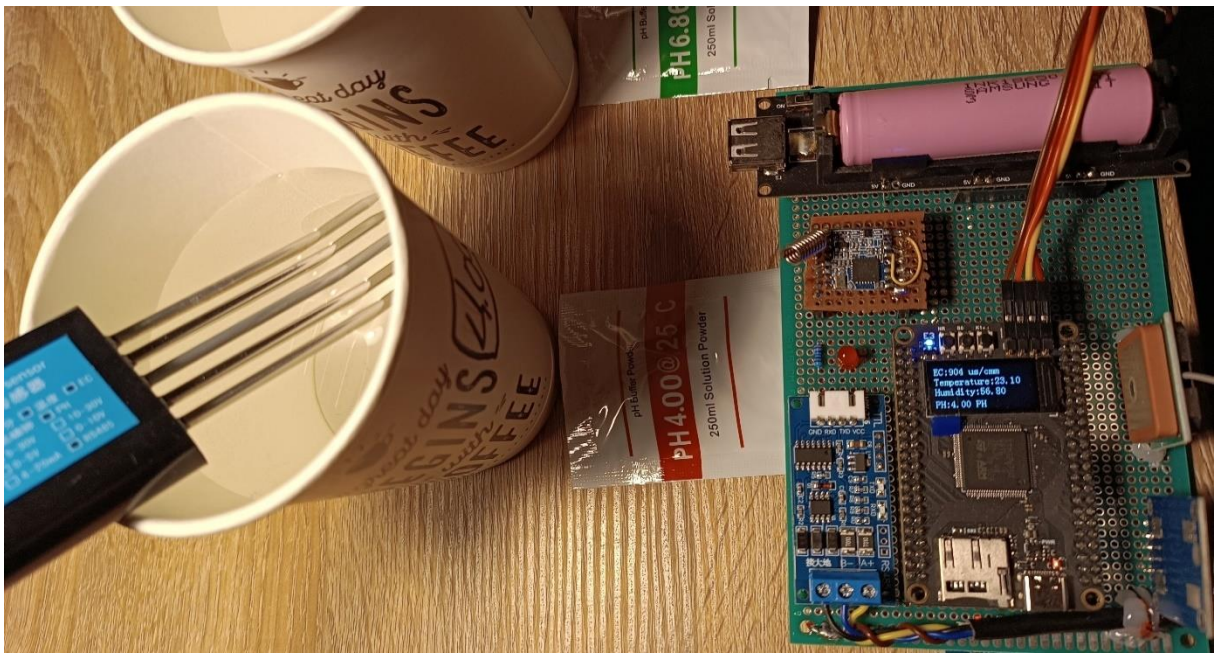
ανακατευτεί το κάθε διάλειμμα θα πρέπει να είναι επίσης στους 25 βαθμούς. Το ανακατεύεις μέχρι να μην υπάρχουν καθόλου κόκκοι. Κάθε φορά που γίνεται αλλαγή μέτρησης για διαφορετικό pH θα πρέπει

ο αισθητήρας/όργανο να πλένεται καλά με απιονισμένο νερό και να θα καθαρίζεται καλά με ένα χαρτί/πανάκι. Οι μετρήσεις όπως φαίνεται στις φωτογραφίες Σχ. 4.10, 4.11, 4.12 είναι αρκετά κοντά στα αναγραφόμενα pH. Πιο συγκεκριμένα:

- Διάλειμμα τιμής pH 4 , η συσκευή μέτρησε 4 , συνολική απόκλιση 0 μονάδες
- Διάλειμμα τιμής pH 6.86 , η συσκευή μέτρησε 7.4 , συνολική απόκλιση 0.54 μονάδες
- Διάλειμμα τιμής pH 9.18 , η συσκευή μέτρησε 8.9 , συνολική απόκλιση 0.28 μονάδες

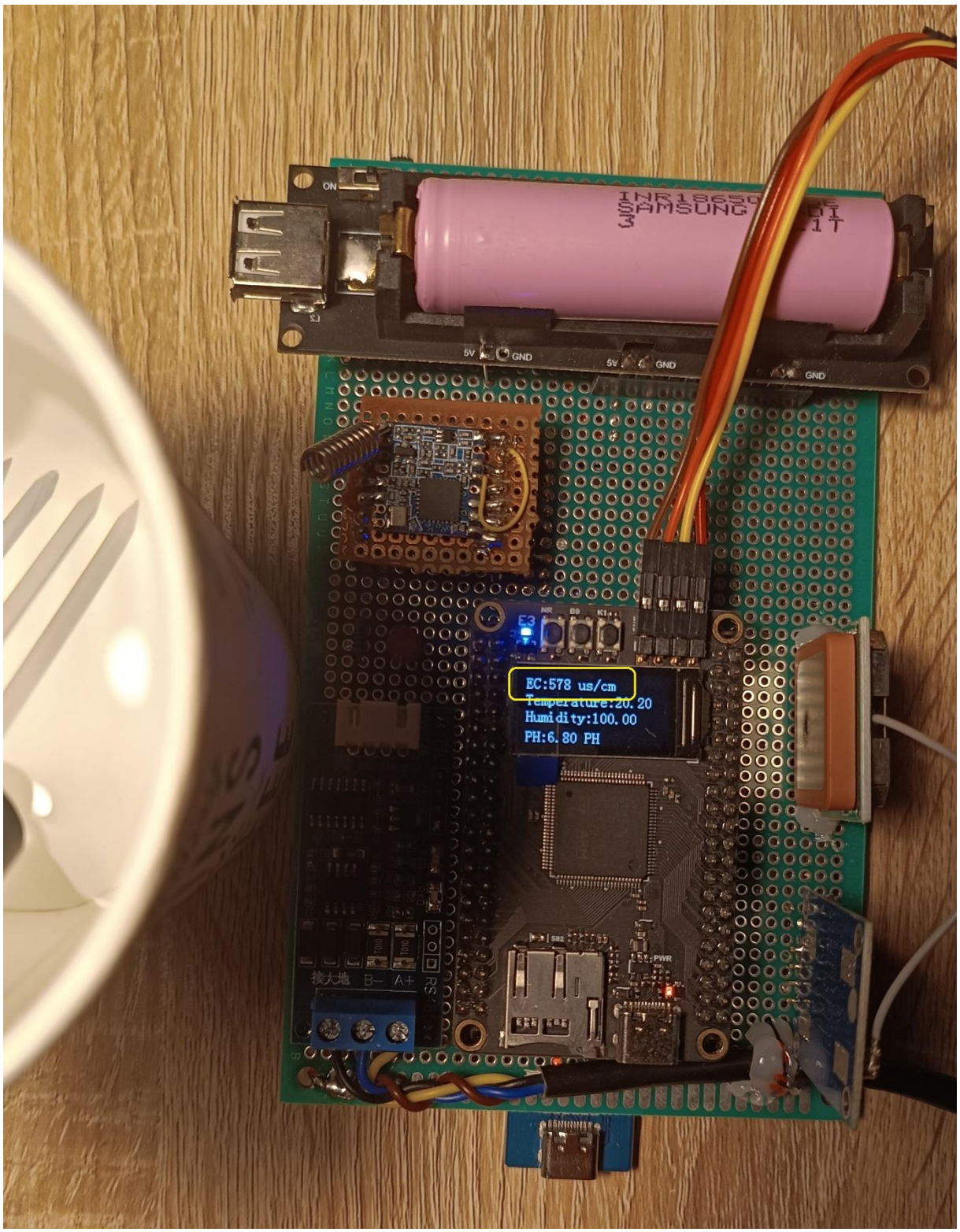


Σχήμα 4.11 Πείραμα μέτρησης pH σε γνωστό υδατικό διάλειμμα με τιμή pH 9.18



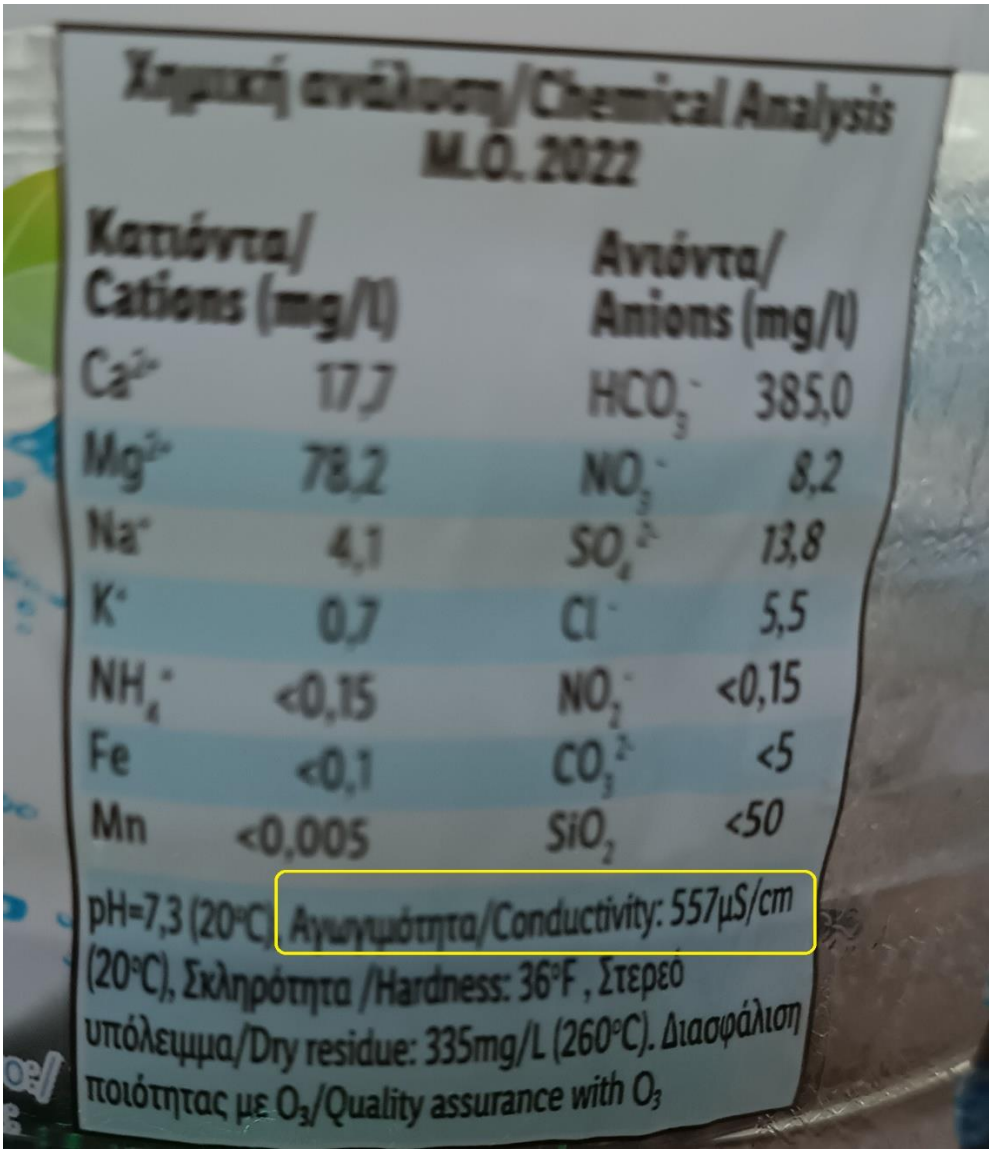
Σχήμα 4.12 Πείραμα μέτρησης pH σε γνωστό υδατικό διάλειμμα με τιμή pH 4.00

Σύμφωνα με τα αποτελέσματα από τις μετρήσεις παρατηρείται ότι η απόκλιση στα ουδέτερα διαλείμματα είναι μεγαλύτερη , από ότι στα βασικά διαλλείματα , ενώ παράλληλα η ακρίβεια βελτιώνεται στα όξινα διαλείμματα.



Σχήμα 4.13 Πείραμα μέτρησης αγωγιμότητας σε γνωστό υδατικό διάλειμμα με τιμή 557 μS/cm

Για την αξιολόγηση της μέτρησης της αγωγιμότητας , αγοράσθηκε ένα εμφιαλωμένο νερό από το εμπόριο στο οποίο αναγράφεται όλα τα χημικά στοιχεία που περιέχει καθώς και η αγωγιμότητα που μας ενδιαφέρει.



Σχήμα 4.14 Πείραμα μέτρησης αγωγιμότητας εμφιαλωμένο νερό με τιμή αναφοράς στην αγωγιμότητα

Όπως φαίνεται και από τα Σχ.4.13 , 4.14 προκύπτει μια μικρή διαφορά στην μέτρηση στα 21 μS/cm. Για να γίνει η μέτρηση ο αισθητήρας χρειάστηκε να βρίσκεται σε υγρασία 100% δηλαδή τελείως βυθισμένος στο διάλειμμα.

Όσο αναφορά τις μετρήσεις της θερμοκρασίας και της υγρασίας φαίνεται να είναι αρκετά αξιόπιστα σύμφωνα με τα πειράματα που έγιναν. Η υγρασία ανταποκρίνεται γραμμικά (αυξάνεται μέχρι το 100%) όσο βυθίζονται οι ακίδες του αισθητήρα στο νερό. Η θερμοκρασία που διαβάζεται από τον αισθητήρα συγκρίθηκε με ένα κοινό θερμόμετρο υδραργύρου και τα αποτελέσματα βρίσκονται πολύ κοντά μεταξύ τους.

4.5 Πείραμα μέτρησης εδάφους από τον αισθητήρα και εδαφική ανάλυση από το εργαστήριο

Σχήμα 4.15 Δείγμα εδάφους ερυθρού χρώματος προς μέτρηση (Δείγμα 1)

Το δείγμα 1 εξάχθηκε από χωράφι χωρίς υπάρχουσα καλλιέργεια την χρονική στιγμή της συλλογής. Οι διπλανές καλλιέργειες ήταν ελιές και πατάτες. Επιλέχτηκε ως δείγμα για το κοκκινωπό χρώμα του, καθώς κέντρισε το ενδιαφέρον ότι δεν βρίσκεται τόσο συχνά στην Ελλάδα.

Σχήμα 4.16 Δείγμα εδάφους χρώματος σκούρου καφέ προς μέτρηση (Δείγμα 2)

Το δείγμα 2 εξάχθηκε από περιοχή ανώμαλου εδάφους με απότομες κλήσεις εκτός κατοικήσιμης περιοχής που καλλιεργούνται ελιές, το χρώμα το εδάφους είναι το πιο κοινότυπο στην Ελλάδα.

Σχήμα 4.17 Δείγμα εδάφους χρώματος ανοιχτού καφέ προς μέτρηση (Δείγμα 3)

Το δείγμα 3 συλλέχθηκε εντός κατοικήσιμης περιοχής και πάνω σε αυτό είχαν αναπτυχθεί καλαμιές. Γενικώς όλα τα δείγματα πάρθηκαν από το νομό Μεσσηνίας της Πελοποννήσου. Ο νομός φημίζεται για τη χλωρίδα, την πανίδα του όπως επίσης και για το μεσογειακό του κλίμα το οποίο έχει ήπιους χειμώνες και δροσερά καλοκαίρια λόγω της ύπαρξης της θάλασσας. Η θερμοκρασία σπάνια κατέρχεται υπό του μηδενός το χειμώνα, ενώ το καλοκαίρι μπορεί να ξεπεράσει του 40 βαθμούς κελσίου στα πεδινά τμήματα.

Παρακάτω στον Πίνακα 4.1 παρουσιάζονται οι μετρήσεις που έγιναν στα τρία δείγματα εδάφους από την εδαφική ανάλυση που πραγματοποιήθηκε στο τμήμα εδαφολογίας στο Διεθνές πανεπιστήμιο της Ελλάδας στην περιοχή της Σίνδου.

	Δείγμα 1	Δείγμα 2	Δείγμα 3
pH	7,87	8,08	7,95
EC (μS/cm)	1472	463	1658
N (mg/kg)	3,91	29,25	6,21
P (mg/kg)	19,42	2,05	3,50
K (mg/kg)	260	66	140

Πίνακας 4.1 Αποτελέσματα μετρήσεων εδαφικής ανάλυσης από γεωπόνο στα τρία δείγματα

Οι μετρήσεις του pH και της ηλεκτρικής αγωγιμότητας (EC) έγιναν στην μορφή της πάστας κορεσμού. Επίσης οι μετρήσεις του Αζώτου, Φωσφόρου και Καλίου αναφέρονται στις τιμές που είναι αφομοιώσιμες από το φυτό και όχι αυτές που υπάρχουν στο έδαφος.

	Δείγμα 1	Δείγμα 2	Δείγμα 3
pH	8,1	8,3	7,5
EC (μS/cm)	1356	423	1598
N (mg/kg)	21	55	40
P (mg/kg)	96	23	31
K (mg/kg)	214	49	128

Πίνακας 4.2 Αποτελέσματα μετρήσεων των δειγμάτων με τον αισθητήρα

4.6 Παρατηρήσεις πειραμάτων

Έπειτα από παρατήρηση των μετρήσεων της συσκευής και διαφορών πειραμάτων που πραγματοποιήθηκαν εξάχθηκαν κάποια αποτελέσματα. Ο υπολογισμός της αλατότητας του εδάφους είναι άμεσα συνδεδεμένος με την μέτρηση της ηλεκτρικής αγωγιμότητας και με μεγαλύτερη ακρίβεια κατά παράγοντα 0.55.

Ο υπολογισμός του TDS στο έδαφος είναι άμεσα συνδεδεμένος με την ηλεκτρική αγωγιμότητα κατά παράγοντα 0.5. Έπειτα από παρατήρηση στο datasheet αυτοί οι παράγοντες μπορούν να αλλαχθούν και από τον χρήστη κάτι που σημαίνει ότι η αλατότητα και το TDS βγαίνουν με μαθηματική σχέση.

Οι μετρήσεις των NPK δεν μετριούνται με κάποιο χημικό η ηλεκτρικό τρόπο, αλλά αντιθέτως προκύπτουν από μαθηματική σχέση και η τιμή τους είναι προσεγγιστική και σαφώς διαφορετική από την μέτρηση του γεωπόνου καθώς ο γεωπόνος μετράει τις απορροφήσεις των ρίζων. Η μέθοδος TDR έχει μειονέκτημα ως προς την μικρή ζώνη επιρροής δηλαδή δεν μπορεί να μετρήσει μεγάλο όγκο και μεγάλο βάθος στο χώμα. Επίσης οι μετρήσεις επηρεάζονται από πιθανή επίδραση της αλατότητας του εδάφους και στα κενά αέρος που μπορούν να δημιουργηθούν κατά την τοποθέτηση του αισθητήρα στο έδαφος.

Προτείνεται στις μετρήσεις που γίνονται ο αισθητήρας να μένει σταθεροποιημένος και να παραμένει στο ίδιο σημείο για τουλάχιστον δεκαπέντε λεπτά. Ταυτόχρονα το έδαφος να έχει ποσοστό υγρασίας πάνω από 40%.

Τα δείγματα προς μέτρηση θα πρέπει να είναι όσο το δυνατών καθαρότερα από μεγάλα πετρώματα τα οποία μπορούν να τραυματίσουν τα ηλεκτρόδια του αισθητήρα κατά την τοποθέτηση. Το κοσκίνισμα του δείγματος μπορεί να βοηθήσει σε αυτό το πρόβλημα πριν την μέτρηση. Η μέθοδος του TDR φαίνεται αρκετά υποσχόμενη για την γρήγορη απεικόνιση του εδάφους σε μικρή κλίμακα καθώς θα βοηθήσει αρκετά τους αγρότες και του γεωπόνους.

Κεφάλαιο 5ο: Συμπεράσματα

Ο σχεδιασμός, ο προγραμματισμός και η υλοποίηση της εργασίας για την κατασκευή μιας συσκευής για την μέτρηση θρεπτικών συστατικών στο έδαφος ολοκληρώθηκε με επιτυχία. Οι κύριοι στόχοι που τέθηκαν από την αρχή ήταν η εύρεση ενός αισθητήρα του εμπορίου με ποικιλομορφία μετρήσεων στις παραμέτρους του εδάφους , η σωστή ανάπτυξη της επικοινωνία μέσω πρωτοκόλλου Modbus RS485, η απόκτηση των γεωγραφικών συντεταγμένων των μετρήσεων της συσκευής , η υλοποίηση και η ανάπτυξη της ασύρματης επικοινωνίας μέσω LoRa, η ενσύρματη επικοινωνία με έναν ηλεκτρονικό υπολογιστή για την μεταφορά των δεδομένων και την απεικόνιση τους , η αποθήκευση των δεδομένων. Χρειάστηκαν πολλά εργαλεία για να συνδράμουν στην δημιουργία αυτού του αποτελέσματος και οι προκλήσεις ήταν μεγάλες.

Το κομμάτι του προγραμματισμού ήταν πολύ απαιτητικό και ειδικά για τον κώδικα που γράφτηκε στον STM32 , ο οποίος θα έπρεπε να συνδυάσει πολλές λειτουργείες ταυτόχρονα και όλες αυτά να λειτουργήσουν άρτια μεταξύ τους. Έγινε χρήση βοηθητικών βιβλιοθηκών με αρκετές παρεμβάσεις για την επιτυχής λειτουργία τους, καθώς και ανάπτυξη βιβλιοθήκης από την αρχή για την επικοινωνία με τον αισθητήρα. Για αυτόν στον στόχο απαιτούσε την εν βάθος ενασχόληση και κατανόηση του κώδικα ανάπτυξης και στον STM32 .

Το κόστος των εξαρτημάτων που χρησιμοποιήθηκαν ήταν ικανοποιητικά χαμηλό σε κόστος εκτός του αισθητήρα που ήταν το κύριο συστατικό της εργασίας. Στο θέμα των πλακετών της συσκευής, για λόγους επερχόμενης ανάπτυξης δεν κατασκευάστηκε μια ενιαία πλακέτα επαγγελματικού τύπου για την σύνδεση όλων των στοιχείων. Η εξέλιξη της συσκευής σε θέματα λογισμικού η υλικών θα συνεχίζεται εφόσον υπάρχει η απαραίτητη ζήτηση.

Πάντα υπάρχει χώρος για βελτιώσεις και οι προτάσεις τρίτων είναι επιθυμητές. Κάποιες βελτιώσεις για μελλοντική αναβάθμιση που εξετάστηκαν μετά το πέρας της ολοκλήρωσης της κατασκευής παρατίθενται παρακάτω.

- Το λογισμικό να μπορεί να θέσει πρόταση για συγκεκριμένες καλλιεργείς που μπορούν να ευδοκιμήσουν αναλόγως τις μετρήσεις που γίναν στο έδαφος
- Το λογισμικό να μπορεί να προτείνει συγκεκριμένα λιπάσματα αναλόγως τις έλλειψες σε μικροστοιχεία που έχει το έδαφος.
- Μεγαλύτερη γκάμα πειραμάτων όσο αναφορά την αξιοπιστία των μετρήσεων , σε πολλά διαφορετικά δείγματα χωμάτων
- Πειραματισμός με άλλες μεθόδους μέτρησης στα συστατικά του εδάφους και σύγκριση αποτελεσμάτων
- Σε περίπτωση αδράνειας , ενεργοποίηση της δυνατότητας βαθύ ύπνου στον STM32 και το ESP32 με σκοπό την μείωση της άσκοπης κατανάλωσης ενέργειας.
- Απομακρυσμένη αποστολή δεδομένων μέσω Cloud Service, ώστε η συσκευή και ο υπολογιστής να μην βρίσκονται στο ίδιο τοπικό δίκτυο όπως συμβαίνει στον τρόπο που υλοποιήθηκε.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Jignyasa Sanghavi, R. Rina Damdoo, and Kanak Kalyani, “Agribot: Iot Based Farmbot for Smart Farming”, Biosc.Biotech.Res.Comm. Special Issue Vol 13 No 14, 2020, pp. 86-89.

[2] Saiful Muhammad, Sebastian Saa, Sat Darshan S Khalsa, Steve Weinbaum and Patrick H. Brown “Almonds: Botany, Production and Uses (Agriculture)”, Chapter 14 -1 - 14 Almond Tree Nutrition CABI Press, 2017, pp. 291-89

[3] Northeast Region Certified Crop Adviser (NRCCA) Study Resources, “Basic Concepts of Plant Nutrition” [Online]. Available: https://nrcca.cals.cornell.edu/soilFertilityCA/CA1/CA1_print.html

[4] USDA Natural Resources Conservation Service, “Soil Quality Indicators” [Online]. Available: https://www.nrcs.usda.gov/sites/default/files/2022-10/Soil%20Electrical%20Conductivity.pdf

[5] Barber T.E. and B.C. Matthew, “Release of non-exchangeable Potassium by resin equilibrium and significance for crop growth “, Canadian Journal of Soil Science, 1962

[6] Nair. P.K.R. and O. Talibudeem, “Dynamics of K and NO3 concentrations in the root zone of winter wheat at broadbalk using specific-ion electrodes”, Journal of Agriculture Science, 1973

[7] Nvent, “Soil Resistivity & Measurement – Ground Electrode Design Principles and Testing” [Online]. Available: https://blog.nvent.com/soil-resistivity-measurement-ground-electrode-design-principles-and-testing

[8] Dennis L. Corwin and Kevin Yemoto, “Salinity: Electrical Conductivity and Total Dissolved Solids”, Soil Science Society of America Journal, Volume 84, Issue 5, Sep. 2020, pp. 1353-1795

[9] Deepa V. Ramane, Supriya S. Patil and A. D. Shaligram “Detection of NPK nutrients of soil using Fiber Optic Sensor”, International Journal of Research in Advent Technology (E-ISSN: 2321-9637) Special Issue National Conference ACGT 2015, Feb. 2015

[10] GitHub STM32H750 Development Board, “STM32H7xx Core Board” [Online]. Available: https://github.com/WeActStudio/MiniSTM32H7xx

[11] STMicroelectronics, “STM32H750VB STM32H750ZB STM32H750IB STM32H750XB”, STM32H750 datasheet, Mar.2023

[12] Espressif Systems, “This document provides the specifications of ESP32 family of chips”, ESP32 datasheet , 2020

[13] AliExpress , “Renke soil integrated transmitter temperature humidity EC npk ph 7 in 1 soil sensor for agriculture” , [Online]. Available: https://www.aliexpress.com/item/1005004682923021.html?gatewayAdapt=glo2bra

[14] Wikipedia, “RS-485 Protocol” [Online]. Available: https://en.wikipedia.org/wiki/RS-485

[15] Modbus Tools, “Modbus Protocol Description” [Online]. Available: https://www.modbustools.com/modbus.html

[16] LoRa, “LoRa Developer Portal Documentation” [Online]. Available: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/

[17] uBlox, “NEO-6 u-blox 6 GPS Modules” , GPS NEO-6M datasheet, Dec.2011

[18] Wikipedia, “Internet protocol suite” [Online]. Available: https://en.wikipedia.org/wiki/Internet_protocol_suite.

[19] Wikipedia, “Websocket” [Online]. Available: https://en.wikipedia.org/wiki/WebSocket.

[20] Wikipedia, “JavaScript” [Online]. Available: https://en.wikipedia.org/wiki/JavaScript.

[21] Wikipedia, “HTML” [Online]. Available: https://el.wikipedia.org/wiki/HTML.

[22] Wikipedia, “CSS” [Online]. Available: https://el.wikipedia.org/wiki/CSS.

[23] SΛMSUNG, “for Lithium-ion rechargeable cell Model name : INR18650-35E” , Battery INR18650 , 2015.

ΠΑΡΑΡΤΗΜΑ Α : Κώδικας STM32H750

```
##################################### setup.h #####################################
#ifndef __SETUP_H
#define __SETUP_H

#include "main.h"
#include <stdbool.h>

#define DEBUG_VIA_UART

#define DEBUG_UART huart1
#define NPK_UART   huart2
#define GPS_UART   huart3
#define MILLIS_TIMER htim7

extern UART_HandleTypeDef DEBUG_UART;
extern UART_HandleTypeDef NPK_UART;
extern UART_HandleTypeDef GPS_UART;
extern TIM_HandleTypeDef MILLIS_TIMER;

enum {
	UART_DEBUG = 0,
	UART_NPK,
	UART_GPS,
};

typedef union {
	float floatVal;
	uint8_t bytes[4];
} floatToBytes;

#define HIGH GPIO_PIN_SET
#define LOW  GPIO_PIN_RESET

extern volatile uint64_t _millis;
extern uint64_t millis();
extern void cbkMillis(TIM_HandleTypeDef *htim);

#define SERIAL_BUFFER 512
#define MAX_SERIALS 3
extern const UART_HandleTypeDef *Uarts[MAX_SERIALS];
typedef struct {
	char incomingData[SERIAL_BUFFER];
```

```
	char receivedData;
	uint16_t uIndex;
	bool huartInterrupt;
	volatile uint64_t uartTime;
	volatile uint64_t uartTimeout;
	bool completeTransfer;
} structSerial;
extern structSerial Serial[MAX_SERIALS];
extern uint16_t readUart (char * buf , uint8_t posSerial);
extern void cbkUart(UART_HandleTypeDef *huart);

//extern bool inLoop;
#endif
```

###################################### setup.h ######################################

###################################### setup.c ######################################

```
#include "setup.h"

bool inLoop=false;

structSerial Serial[3];
const UART_HandleTypeDef * Uarts[MAX_SERIALS] = {
		&DEBUG_UART,
		&NPK_UART,
		&GPS_UART
};

volatile uint64_t _millis = 0;
uint64_t millis() {
	return _millis;
}

void initMillis() {
	 HAL_TIM_Base_Start_IT(&MILLIS_TIMER);
}

void cbkMillis(TIM_HandleTypeDef *htim) {
	if (htim->Instance == MILLIS_TIMER.Instance) {
		 _millis++;
	}
}

void initUarts() {
	for (uint8_t i=0;i<MAX_SERIALS;i++){
		HAL_StatusTypeDef res = HAL_UART_Receive_IT( Uarts[i] ,(uint8_t *
)&Serial[i].receivedData, 1);
		if (!res==HAL_OK) {
			DEBUGF("[UART] Init Error:%d At USART%d",res,i+1);
		}
		Serial[i].uartTimeout = 5;
		Serial[i].huartInterrupt = false;
		Serial[i].uIndex = 0;
```

```
                Serial[i].completeTransfer = false;
        }
}
uint16_t readUart (char * buf , uint8_t posSerial) {
        uint8_t lenRead = 0;
        uint8_t pos = posSerial;
        if (Serial[pos].huartInterrupt) {
                bool condition = millis() - Serial[pos].uartTime > Serial[pos].uartTimeout ||
Serial[pos].completeTransfer;

                if (  condition  ) {
                        HAL_GPIO_TogglePin(STATUS_LED_GPIO_Port,STATUS_LED_Pin);
                        lenRead = Serial[pos].uIndex;
                        memcpy(buf,Serial[pos].incomingData,Serial[pos].uIndex);
                        Serial[pos].huartInterrupt = false;
                        memset(Serial[pos].incomingData,0,Serial[pos].uIndex);
                        Serial[pos].uIndex = 0;
                        Serial[pos].completeTransfer = false;
                }
        }
        return lenRead;
}

void cbkUart(UART_HandleTypeDef *huart) {

        uint8_t typeUart = 0;
        if (huart->Instance == DEBUG_UART.Instance) {
                typeUart = UART_DEBUG;
        }
        if (huart->Instance == NPK_UART.Instance) {
                typeUart = UART_NPK;
        }
        if (huart->Instance == GPS_UART.Instance) {
                typeUart = UART_GPS;
        }

        HAL_StatusTypeDef res = HAL_UART_Receive_IT(huart,(uint8_t *
)&Serial[typeUart].receivedData,1);
//      if (!inLoop) {
//              //DEBUGF("Ignoring RX for USART%d",typeUart+1);
//              return ;
//      }
        if (res==HAL_OK){
          Serial[typeUart].uartTime = millis();
          if ( Serial[typeUart].uIndex< SERIAL_BUFFER - 1 ) {  //LEAVE Space for the Special
Terminated Character '/0' for a Char Array  ---> -1 FOR THAT REASON
                Serial[typeUart].incomingData[Serial[typeUart].uIndex++]=
Serial[typeUart].receivedData;
          }
          Serial[typeUart].huartInterrupt=true;
          if (Serial[typeUart].uIndex == SERIAL_BUFFER - 1 ){
                DEBUGF("[Overflow] at USART%d",typeUart+1);
                Serial[typeUart].uIndex = 0;
          }
```

```
            if (Serial[typeUart].incomingData[Serial[typeUart].uIndex-1] == '\n' && typeUart ==
UART_GPS) {
                        //DEBUGF("[Complete] at Index:%d",Serial[typeUart].uIndex);
                        Serial[typeUart].completeTransfer = true;
             }

          }
          else {
                    DEBUGF("[Error] Code:%d" , res);
          }

}

#ifdef DEBUG_VIA_UART
          #include <stdarg.h>
          extern void DEBUGF(const char *fmt, ...) {
                    char buf[128];
             va_list args;
             va_start(args, fmt);
             vsprintf(buf, fmt, args);
             va_end(args);
             char newBuf[256];
             sprintf(newBuf,"%s\n",buf);
             //CDC_Transmit_FS((uint8_t*)&newBuf, strlen(newBuf));
             HAL_UART_Transmit(&DEBUG_UART, (uint8_t*)&newBuf, strlen(newBuf), 1000);
          }
          extern void PRINT(const char *fmt, ...) {
                    char buf[128];
             va_list args;
             va_start(args, fmt);
             vsprintf(buf, fmt, args);
             va_end(args);
             char newBuf[256];
             sprintf(newBuf,"%s",buf);
             //CDC_Transmit_FS((uint8_t*)&newBuf, strlen(newBuf));
             HAL_UART_Transmit(&DEBUG_UART, (uint8_t*)&newBuf, strlen(newBuf), 1000);
          }


          #else
          #define DEBUGF(msg, ...)
#endif

###################################### setup.c ######################################

###################################### npk.h ######################################
#ifndef __NPK_H
#define __NPK_H

#include "main.h"
#include <stdio.h>

typedef union Val {
```

```
  uint16_t raw;
  uint8_t bytes[2];
} Val;

#define NPK_DATA_BYTES_LEN 9*2

typedef struct npkData {
  union {
    struct {
      Val _Moisture;
      Val _Temperature;
      Val _Conductivity;
      Val _PH;
      Val _Nitrogen;
      Val _Phosphorus;
      Val _Potassium;
      Val _Salinity;
      Val _TDS;
    };
    uint8_t bytes[NPK_DATA_BYTES_LEN]; //total 18 bytes
  };

  float Moisture;
  float Temperature;
  uint16_t Conductivity;
  float PH;
  uint16_t Nitrogen;
  uint16_t Phosphorus;
  uint16_t Potassium;
  uint16_t Salinity;
  uint16_t TDS;

} npkData;

void npkParse(npkData* Sensor , char* d , uint16_t len);
void npkDebug(npkData* Sensor);
void npkParseBytes(npkData* Sensor , char* d , uint16_t len);
void npkConversions(npkData* Sensor);

#endif

###################################### npk.h ######################################

###################################### npk.c ######################################
#include "npk.h"
```

```
void npkConversions(npkData* Sensor) {

   if ( abs( 65536 - Sensor->_Temperature.raw ) < 250 ) {
    Sensor->Temperature = - (65536 - Sensor->_Temperature.raw) * 0.1;
   }
   else {
    Sensor->Temperature = Sensor->_Temperature.raw * 0.1;
   }
   Sensor->Moisture = Sensor->_Moisture.raw * 0.1;
   Sensor->Conductivity = Sensor->_Conductivity.raw;
   Sensor->PH = Sensor->_PH.raw*0.1;
   Sensor->Nitrogen =  Sensor->_Nitrogen.raw;
   Sensor->Phosphorus =  Sensor->_Phosphorus.raw;
   Sensor->Potassium =  Sensor->_Potassium.raw;
   Sensor->Salinity =  Sensor->_Salinity.raw;
   Sensor->TDS =  Sensor->_TDS.raw;

}

void npkParse(npkData* Sensor , char* d , uint16_t len) {

   uint8_t b = 0;
   for(uint8_t i=3;i<len-2;i++){
    Sensor->bytes[b] = d[i];
    b++;
   }

   for(uint8_t i=0;i<sizeof(npkData);i=i+2){
    uint8_t temp = Sensor->bytes[i];
    Sensor->bytes[i] = Sensor->bytes[i+1];
    Sensor->bytes[i+1] = temp;
   }
   npkConversions(Sensor);

}

void npkParseBytes(npkData* Sensor  , char* d , uint16_t len) {
        for(uint8_t i=0;i<len;i++) {
          if (i<NPK_DATA_BYTES_LEN) {
            Sensor->bytes[i] = d[i];
          }
        }
        npkConversions(Sensor);

}
```

```
void npkDebug(npkData* Sensor) {

   char buf[64];
   sprintf(buf,"Temperature:%.2f °C\r",Sensor->Temperature);
   DEBUGF(buf);
   sprintf(buf,"Moisture:%.2f %%\r",Sensor->Moisture);
   DEBUGF(buf);
   sprintf(buf,"Conductivity:%d us/cm\r",Sensor->Conductivity);
   DEBUGF(buf);
   sprintf(buf,"PH:%.2f PH\r",Sensor->PH);
   DEBUGF(buf);
   sprintf(buf,"Nitrogen:%d mg/kg\r",Sensor->Nitrogen);
   DEBUGF(buf);
   sprintf(buf,"Phosphorus:%d mg/kg\r",Sensor->Phosphorus);
   DEBUGF(buf);
   sprintf(buf,"Potassium:%d mg/kg\r",Sensor->Potassium);
   DEBUGF(buf);
   sprintf(buf,"Salinity:%d ppt\r",Sensor->Salinity);
   DEBUGF(buf);
   sprintf(buf,"TDS:%d ppm\r",Sensor->TDS);
   DEBUGF(buf);

}
```


lora1276.h

```
// Copyright (c) Konstantin Belyalov. All rights reserved.
// Licensed under the MIT license.

#ifndef __LORA_H
#define __LORA_H

#include "main.h"


#define LORA_MAX_PACKET_SIZE            128

// Operational frequency
#define MHZ                             1000000LLU
#define LORA_BASE_FREQUENCY_US          (915LLU*MHZ)
#define LORA_BASE_FREQUENCY_EU          (860LLU*MHZ)

// Default settings
#define LORA_DEFAULT_TX_POWER           17
#define LORA_DEFAULT_SF                 7
#define LORA_DEFAULT_PREAMBLE_LEN       10
```

```
#define LORA_DEFAULT_RX_ADDR            0
#define LORA_DEFAULT_TX_ADDR            0
#define LORA_DEFAULT_SPI_TIMEOUT        1000 // ms

#define LORA_COMPATIBLE_VERSION         0x12U

// LORA return codes
#define LORA_OK                     0
#define LORA_CRC_ERROR              1
#define LORA_TIMEOUT                2
#define LORA_INVALID_HEADER         3
#define LORA_ERROR                  4
#define LORA_BUSY                   5
#define LORA_EMPTY                  6

// TX power mode select
#define LORA_PA_OUTPUT_RFO          0
#define LORA_PA_OUTPUT_PA_BOOST     1

// Coding rate
#define LORA_CODING_RATE_4_5        0x08
#define LORA_CODING_RATE_4_6        0x10
#define LORA_CODING_RATE_4_7        0x18
#define LORA_CODING_RATE_4_8        0x20

// Signal bandwidth ("spread factor")
enum {
  LORA_BANDWIDTH_7_8_KHZ = 0,
  LORA_BANDWIDTH_10_4_KHZ,
  LORA_BANDWIDTH_15_6_KHZ,
  LORA_BANDWIDTH_20_8_KHZ,
  LORA_BANDWIDTH_31_25_KHZ,
  LORA_BANDWIDTH_41_7_KHZ,
  LORA_BANDWIDTH_62_5_KHZ,
  LORA_BANDWIDTH_125_KHZ,  // default SF - 7
  LORA_BANDWIDTH_250_KHZ,
  LORA_BANDWIDTH_500_KHZ,
  LORA_BW_LAST,
};

// LORA definition
typedef struct {
  // SPI parameters
  SPI_HandleTypeDef  *spi;
  GPIO_TypeDef       *nss_port;
  uint32_t           spi_timeout;
```

```
  // Operating frequency, in Hz
  uint32_t          frequency;
  // Output PIN (module internal, not related to your design)
  // Can be one of PA_OUTPUT_PA_BOOST / PA_OUTPUT_RFO
  uint32_t          pa_mode;
  // Base FIFO addresses for RX/TX
  uint8_t           tx_base_addr;
  uint8_t           rx_base_addr;

  uint16_t          nss_pin;
} lora_sx1276;


// LORA Module setup //

// Initialize LORA with default parameters:
// Params:
//  - `lora` LoRa definition to be initialized
//  - `spi` SPI HAL bus (`hspi1`, `hspi2`, etc)
//  - `nss_port` - GPIO port where `NSS` pin connected to
//  - `nss_pin` - GPIO pin number in `nss_port`
//  - `freq` - operating frequency. In Hz
// Returns:
//  - `LORA_OK` - modem initialized successfully
//  - `LORA_ERROR` - initialization failed (e.g. no modem present on SPI bus / wrong NSS port/pin)
uint8_t  lora_init(lora_sx1276 *lora, SPI_HandleTypeDef *spi, GPIO_TypeDef *nss_port,
                uint16_t nss_pin, uint64_t freq);

// Returns LoRa modem version number (usually 0x12)
uint8_t  lora_version(lora_sx1276 *lora);


// LORA mode selection //

// Put radio into SLEEP mode:
// In this mode only SPI and configuration registers are accessible.
// LoRa FIFO is not accessible.
void     lora_mode_sleep(lora_sx1276 *lora);

// Put radio into standby (idle) mode:
// Both Crystal Oscillator and LoRa baseband blocks are turned on.
// RF part and PLLs are disabled.
void     lora_mode_standby(lora_sx1276 *lora);

// Put radio into continuous receive mode:
// When activated the RFM95/96/97/98(W) powers all remaining blocks required for reception,
```

```
// processing all received data until a new user request is made to change operating mode.
void     lora_mode_receive_continuous(lora_sx1276 *lora);

// Put radio into single receive mode:
// When activated the RFM95/96/97/98(W) powers all remaining blocks required for reception, remains
in
// this state until a valid packet has been received and then returns to Standby mode.
void     lora_mode_receive_single(lora_sx1276 *lora);


// LORA signal / transmission parameters //

// Sets LoRa transmit power.
// Params:
//  - `level` - TX power in dBm. Valid range from 2dBm to 20dBm
void     lora_set_tx_power(lora_sx1276 *lora, uint8_t level);

// Set operational frequency.
// Params:
//  - `freq` - frequency in Hz
void     lora_set_frequency(lora_sx1276 *lora, uint64_t freq);

// Set signal bandwidth.
// Params:
//  - `bw` - desired bandwidth, from LORA_BANDWIDTH_7_8_KHZ to
LORA_BANDWIDTH_500_KHZ
// For more information refer to section 4.1 of datasheet.
void     lora_set_signal_bandwidth(lora_sx1276 *lora, uint64_t bw);

// Set signal spreading factor.
// Params:
//  - `sf` - spreading factor. Value from 6 to 12
// For more information refer to section 4.1 of datasheet.
void     lora_set_spreading_factor(lora_sx1276 *lora, uint8_t sf);

// Set coding rate.
//  - `rate` - coding rate. Use any of LORA_CODING_RATE* constants.
// For more information refer to section 4.1 of datasheet.
void     lora_set_coding_rate(lora_sx1276 *lora, uint8_t rate);

// Enable / disable CRC
// Params:
//  - `enable` - set to 0 to disable CRC, any other value enables CRC.
void     lora_set_crc(lora_sx1276 *lora, uint8_t enable);

// Set length of packet preamble.
```

```
// Params:
//  - `len` - length of packet preamble
// For more information refer to section 4.1.1.6 of datasheet
void     lora_set_preamble_length(lora_sx1276 *lora, uint16_t len);

// Set "implicit header" mode, meaning no packet header at all.
// Refer to section 4.1.1.6 of datasheet
void     lora_set_implicit_header_mode(lora_sx1276 *lora);

// Set "explicit", i.e. always add packet header with various system information.
// Refer to section 4.1.1.6 of datasheet
void     lora_set_explicit_header_mode(lora_sx1276 *lora);


// Received packet information //

// Returns RSSI of last received packet
int8_t  lora_packet_rssi(lora_sx1276 *lora);

// Returns SNR of last received packet
uint8_t  lora_packet_snr(lora_sx1276 *lora);


// SEND packet routines //

// Query modem for any ongoing packet transmission.
// Returns 0 if no active transmission present
uint8_t  lora_is_transmitting(lora_sx1276 *lora);

// Send packet in non-blocking mode
// Params:
//  - `data` - pointer to buffer to be transmitted
//  - `data_len` - how many bytes to transmit from `data` buffer.
// Returns:
//  - `LORA_BUSY` in case of active transmission ongoing
//  - `LORA_OK` packet scheduled to be sent.
//     Check state with `lora_is_transmitting()` or by interrupt.
uint8_t  lora_send_packet(lora_sx1276 *lora, uint8_t *data, uint8_t data_len);

// Send packet using DMA mode.
// You must call lora_send_packet_dma_complete() from DMA transfer complete callback
// Params:
//  - `data` - pointer to buffer to be transmitted
//  - `data_len` - how many bytes to transmit from `data` buffer.
// Returns:
//  - `LORA_BUSY` in case of active transmission ongoing
```

```
//  - `LORA_TIMEOUT` packet wasn't transmitted in given time frame.
//  - `LORA_OK` packet scheduled to be sent.
uint8_t  lora_send_packet_dma_start(lora_sx1276 *lora, uint8_t *data, uint8_t data_len);

// Finish packet send initiated by lora_send_packet_dma_start()
void     lora_send_packet_dma_complete(lora_sx1276 *lora);

// Send packet and returns only when packet sent / error occurred (blocking mode).
// Params:
//  - `data` - pointer to buffer to be transmitted
//  - `data_len` - how many bytes to transmit from `data` buffer.
//  - `timeout` - maximum wait time to finish transmission.
// Returns:
//  - `LORA_BUSY` in case of active transmission ongoing
//  - `LORA_TIMEOUT` packet wasn't transmitted in given time frame.
//  - `LORA_OK` packet scheduled to be sent.
uint8_t    lora_send_packet_blocking(lora_sx1276 *lora, uint8_t *data, uint8_t data_len, uint32_t timeout);


// RECEIVE packet routines //

// Checks if packet modem has packet awaiting to be received
// Returns 0 if no packet is available, or any positive integer in case packet is ready
uint8_t  lora_is_packet_available(lora_sx1276 *lora);

// If modem has packet awaiting to be received - returns it's length.
uint8_t  lora_pending_packet_length(lora_sx1276 *lora);

// Receives packet from LoRa modem
// Params:
//  - `buffer` - pointer to buffer where copy packet to.
//  - `buffer_len` - length of `buffer`. If incoming packet greater than `buffer_len` it will be truncated
//    to fit `buffer_len`.
//  - `error` - pointer to `uint8_t` to store error. Can be NULL - so no error information will be stored.
// Returns actual packet length stored into `buffer`.
//
// `error` is one of:
//  - `LORA_OK` - packet successfully received.
//    - `LORA_EMPTY` - no packet received at the moment (check for packet by `lora_is_packet_available()` before).
//  - `LORA_TIMEOUT` - timeout while receiving packet (only for single receive mode).
//  - `LORA_INVALID_HEADER` - packet with malformed header received.
//  - `LORA_CRC_ERROR` - malformed packet received (CRC failed). Please note that you need to enable
//    this functionality explicitly, it is disabled by default.
```

```
uint8_t  lora_receive_packet(lora_sx1276 *lora, uint8_t *buffer, uint8_t buffer_len, uint8_t *error);

// Start receiving packet from LoRa modem in DMA mode.
// 1. In case of single receive mode: when packet arrived or timeout occurred.
//    Please note that it is not enough to set `timeout` to something positive -
//    you also need to set timeout in LoRa modem by calling `lora_set_rx_symbol_timeout` before.
// 2. For continuous receiving mode will wait until packet arrived / timeout occurred.
// Params:
//  - `buffer` - pointer to buffer where copy packet to.
//  - `buffer_len` - length of `buffer`. If incoming packet greater than `buffer_len` it will be truncated
//    to fit `buffer_len`.
//  - `error` - pointer to `uint8_t` to store error. Can be NULL - so no error information will be stored.
// Returns actual packet length stored into `buffer`.
//
// `error` is one of:
//  - `LORA_OK` - packet successfully received.
//     - `LORA_EMPTY` - no packet received at the moment (check for packet by
`lora_is_packet_available()` before).
//  - `LORA_TIMEOUT` - timeout while receiving packet (only for single receive mode).
//  - `LORA_INVALID_HEADER` - packet with malformed header received.
//  - `LORA_CRC_ERROR` - malformed packet received (CRC failed). Please note that you need to
enable
//    this functionality explicitly, it is disabled by default.
uint8_t  lora_receive_packet_dma_start(lora_sx1276 *lora, uint8_t *buffer, uint8_t buffer_len,
                                       uint8_t *error);

// Finish receive packet in DMA dome
void     lora_receive_packet_dma_complete(lora_sx1276 *lora);


// Receive packet in "blocking mode" i.e. function return only when packet:
// 1. In case of single receive mode: when packet arrived or timeout occurred.
//    Please note that it is not enough to set `timeout` to something positive -
//    you also need to set timeout in LoRa modem by calling `lora_set_rx_symbol_timeout` before.
// 2. For continuous receiving mode will wait until packet arrived / timeout occurred.
// Params:
//  - `buffer` - pointer to buffer where copy packet to.
//  - `buffer_len` - length of `buffer`. If incoming packet greater than `buffer_len` it will be truncated
//    to fit `buffer_len`.
//  - `error` - pointer to `uint8_t` to store error. Can be NULL - so no error information will be stored.
// Returns actual packet length stored into `buffer`.
//
// `error` is one of:
//  - `LORA_OK` - packet successfully received.
//     - `LORA_EMPTY` - no packet received at the moment (check for packet by
`lora_is_packet_available()` before).
```

```
//  - `LORA_TIMEOUT` - timeout while receiving packet (only for single receive mode).
//  - `LORA_INVALID_HEADER` - packet with malformed header received.
//  - `LORA_CRC_ERROR` - malformed packet received (CRC failed). Please note that you need to enable
//    this functionality explicitly, it is disabled by default.
uint8_t  lora_receive_packet_blocking(lora_sx1276 *lora, uint8_t *buffer, uint8_t buffer_len,
                                      uint32_t timeout, uint8_t *error);

// Sets timeout for `lora_mode_receive_single()` in symbols.
// Params:
//  - `symbols` - timeout value. Valid from `4` to `1024` symbols.
// For more information refer to datasheet section 4.1.5
void     lora_set_rx_symbol_timeout(lora_sx1276 *lora, uint16_t symbols);


// Enables interrupt on DIO0 when packet received
// SX1276 module will pull DIO0 line high
void     lora_enable_interrupt_rx_done(lora_sx1276 *lora);

// Enables interrupt on DIO0 when transmission is done
// SX1276 module will pull DIO0 line high
void     lora_enable_interrupt_tx_done(lora_sx1276 *lora);

// Clears all RX interrupts on DIO0 (done, timeout, crc, etc)
void      lora_clear_interrupt_rx_all(lora_sx1276 *lora);

// Clears TX interrupt on DIO0
void      lora_clear_interrupt_tx_done(lora_sx1276 *lora);

void      lora_set_syncword(lora_sx1276 *lora, uint8_t sw);

#endif
```

lora1276.h

lora1276.c

```
// Copyright (c) Konstantin Belyalov. All rights reserved.
// Licensed under the MIT license.
#include "lora_sx1276.h"

// sx1276 registers
#define REG_FIFO              0x00
#define REG_OP_MODE           0x01
#define REG_FRF_MSB           0x06
#define REG_FRF_MID           0x07
#define REG_FRF_LSB           0x08
```

```
#define REG_PA_CONFIG           0x09
#define REG_OCP                 0x0b
#define REG_LNA                 0x0c
#define REG_FIFO_ADDR_PTR       0x0d
#define REG_FIFO_TX_BASE_ADDR   0x0e
#define REG_FIFO_RX_BASE_ADDR   0x0f
#define REG_FIFO_RX_CURRENT_ADDR 0x10
#define REG_IRQ_FLAGS           0x12
#define REG_RX_NB_BYTES         0x13
#define REG_PKT_SNR_VALUE       0x19
#define REG_PKT_RSSI_VALUE      0x1a
#define REG_MODEM_CONFIG_1      0x1d
#define REG_MODEM_CONFIG_2      0x1e
#define REG_SYMB_TIMEOUT_LSB    0x1f
#define REG_PREAMBLE_MSB        0x20
#define REG_PREAMBLE_LSB        0x21
#define REG_PAYLOAD_LENGTH      0x22
#define REG_MODEM_CONFIG_3      0x26
#define REG_DETECTION_OPTIMIZE  0x31
#define REG_DETECTION_THRESHOLD 0x37
#define REG_DIO_MAPPING_1       0x40
#define REG_VERSION             0x42
#define REG_PA_DAC              0x4d
#define REG_SYNC_WORD                   0x39

// modes
#define OPMODE_SLEEP            0x00
#define OPMODE_STDBY            0x01
#define OPMODE_TX               0x03
#define OPMODE_RX_CONTINUOUS    0x05
#define OPMODE_RX_SINGLE        0x06
#define OPMODE_LONG_RANGE_MODE  0x80  // (1 << 7)

// Power Amplifier (PA_DAC) settings
#define PA_DAC_HIGH_POWER       0x87
#define PA_DAC_HALF_POWER       0x84

// Over Current Protection (OCP) config
#define OCP_ON                  (1 << 5)

// Modem config register parameters
#define MC1_IMPLICIT_HEADER_MODE    (1 << 0)

#define MC2_CRC_ON              (1 << 2)

#define MC3_AGCAUTO             (1 << 2)
#define MC3_MOBILE_NODE         (1 << 3)

// IRQs
#define IRQ_FLAGS_RX_TIMEOUT        (1 << 7)
#define IRQ_FLAGS_RX_DONE           (1 << 6)
#define IRQ_FLAGS_PAYLOAD_CRC_ERROR (1 << 5)
#define IRQ_FLAGS_VALID_HEADER      (1 << 4)
#define IRQ_FLAGS_TX_DONE           (1 << 3)
#define IRQ_FLAGS_CAD_DONE          (1 << 2)
```

```
#define IRQ_FLAGS_FHSSCHANGECHANNEL (1 << 1)
#define IRQ_FLAGS_CAD_DETECTED      (1 << 0)
#define IRQ_FLAGS_RX_ALL            0xf0

// Just to make it readable
#define BIT_7                       (1 << 7)

#define TRANSFER_MODE_DMA           1
#define TRANSFER_MODE_BLOCKING      2

// Debugging support
// To enable debug information add
// #define LORA_DEBUG
// to main.h

// SPI helpers //

// Reads single register
static uint8_t read_register(lora_sx1276 *lora, uint8_t address)
{
  uint8_t value = 0;

  // 7bit controls read/write mode
  CLEAR_BIT(address, BIT_7);

  // Start SPI transaction
  HAL_GPIO_WritePin(lora->nss_port, lora->nss_pin, GPIO_PIN_RESET);
  // Transmit reg address, then receive it value
  uint32_t res1 = HAL_SPI_Transmit(lora->spi, &address, 1, lora->spi_timeout);
  uint32_t res2 = HAL_SPI_Receive(lora->spi, &value, 1, lora->spi_timeout);
  // End SPI transaction
  HAL_GPIO_WritePin(lora->nss_port, lora->nss_pin, GPIO_PIN_SET);

  if (res1 != HAL_OK || res2 != HAL_OK || value==0) {
    DEBUGF("SPI transmit/receive failed (%d %d)", res1, res2);
  }

  return value;
}

// Writes single register
static void write_register(lora_sx1276 *lora, uint8_t address, uint8_t value)
{
  // 7bit controls read/write mode
  SET_BIT(address, BIT_7);

  // Reg address + its new value
  uint16_t payload = (value << 8) | address;

  // Start SPI transaction, send address + value
  HAL_GPIO_WritePin(lora->nss_port, lora->nss_pin, GPIO_PIN_RESET);
  uint32_t res = HAL_SPI_Transmit(lora->spi, (uint8_t*)&payload, 2, lora->spi_timeout);
  // End SPI transaction
  HAL_GPIO_WritePin(lora->nss_port, lora->nss_pin, GPIO_PIN_SET);
```

```
  if (res != HAL_OK) {
    DEBUGF("SPI transmit failed: %d", res);
  }
}

// Copies bytes from buffer into radio FIFO given len length
static void write_fifo(lora_sx1276 *lora, uint8_t *buffer, uint8_t len, uint8_t mode)
{
  uint8_t address = REG_FIFO | BIT_7;

  // Start SPI transaction, send address
  HAL_GPIO_WritePin(lora->nss_port, lora->nss_pin, GPIO_PIN_RESET);
  uint32_t res1 = HAL_SPI_Transmit(lora->spi, &address, 1, lora->spi_timeout);
  if (mode == TRANSFER_MODE_DMA) {
    HAL_SPI_Transmit_DMA(lora->spi, buffer, len);
    // Intentionally leave SPI active - let DMA finish transfer
    return;
  }
  uint32_t res2 = HAL_SPI_Transmit(lora->spi, buffer, len, lora->spi_timeout);
  // End SPI transaction
  HAL_GPIO_WritePin(lora->nss_port, lora->nss_pin, GPIO_PIN_SET);

  if (res1 != HAL_OK || res2 != HAL_OK) {
    DEBUGF("SPI transmit failed");
  }
}

// Reads data "len" size from FIFO into buffer
static void read_fifo(lora_sx1276 *lora, uint8_t *buffer, uint8_t len, uint8_t mode)
{
  uint8_t address = REG_FIFO;

  // Start SPI transaction, send address
  HAL_GPIO_WritePin(lora->nss_port, lora->nss_pin, GPIO_PIN_RESET);
  uint32_t res1 = HAL_SPI_Transmit(lora->spi, &address, 1, lora->spi_timeout);
  uint32_t res2;
  if (mode == TRANSFER_MODE_DMA) {
    res2 = HAL_SPI_Receive_DMA(lora->spi, buffer, len);
    // Do not end SPI here - must be done externally when DMA done
  } else {
    res2 = HAL_SPI_Receive(lora->spi, buffer, len, lora->spi_timeout);
    // End SPI transaction
    HAL_GPIO_WritePin(lora->nss_port, lora->nss_pin, GPIO_PIN_SET);
  }

  if (res1 != HAL_OK || res2 != HAL_OK) {
    DEBUGF("SPI receive/transmit failed");
  }
}

static void set_mode(lora_sx1276 *lora, uint8_t mode)
{
  write_register(lora, REG_OP_MODE, OPMODE_LONG_RANGE_MODE | mode);
}
```

```
// Set Overload Current Protection
static void set_OCP(lora_sx1276 *lora, uint8_t imax)
{
  uint8_t value;

  // Minimum available current is 45mA, maximum 240mA
  // As per page 80 of datasheet
  if (imax < 45) {
    imax = 45;
  }
  if (imax > 240) {
    imax = 240;
  }

  if (imax < 130) {
    value = (imax - 45) / 5;
  } else {
    value = (imax + 30) / 10;
  }

  write_register(lora, REG_OCP, OCP_ON | value);
}

static void set_low_data_rate_optimization(lora_sx1276 *lora)
{
  assert_param(lora);

  // Read current signal bandwidth
  uint64_t bandwidth = read_register(lora, REG_MODEM_CONFIG_1) >> 4;
  // Read current spreading factor
  uint8_t  sf = read_register(lora, REG_MODEM_CONFIG_2) >> 4;

  uint8_t  mc3 = MC3_AGCAUTO;

  if (sf >= 11 && bandwidth == LORA_BANDWIDTH_125_KHZ) {
    mc3 |= MC3_MOBILE_NODE;
  }

  write_register(lora, REG_MODEM_CONFIG_3, mc3);
}

void lora_mode_sleep(lora_sx1276 *lora)
{
  assert_param(lora);

  set_mode(lora, OPMODE_SLEEP);
}

void lora_mode_receive_continuous(lora_sx1276 *lora)
{
  assert_param(lora);

  // Update base FIFO address for incoming packets
  write_register(lora, REG_FIFO_RX_BASE_ADDR, lora->rx_base_addr);
  // Clear all RX related IRQs
```

```
  write_register(lora, REG_IRQ_FLAGS, IRQ_FLAGS_RX_ALL);

  set_mode(lora, OPMODE_RX_CONTINUOUS);
}

void lora_mode_receive_single(lora_sx1276 *lora)
{
  assert_param(lora);

  // Update base FIFO address for incoming packets
  write_register(lora, REG_FIFO_RX_BASE_ADDR, lora->rx_base_addr);
  // Clear all RX related IRQs
  write_register(lora, REG_IRQ_FLAGS, IRQ_FLAGS_RX_ALL);

  set_mode(lora, OPMODE_RX_SINGLE);
}

void lora_mode_standby(lora_sx1276 *lora)
{
  assert_param(lora);

  set_mode(lora, OPMODE_STDBY);
}

void lora_set_implicit_header_mode(lora_sx1276 *lora)
{
  assert_param(lora);

  uint8_t mc1 = read_register(lora, REG_MODEM_CONFIG_1);
  mc1 |= MC1_IMPLICIT_HEADER_MODE;
  write_register(lora, REG_MODEM_CONFIG_1, mc1);
}

void lora_set_explicit_header_mode(lora_sx1276 *lora)
{
  assert_param(lora);

  uint8_t mc1 = read_register(lora, REG_MODEM_CONFIG_1);
  mc1 &= ~MC1_IMPLICIT_HEADER_MODE;
  write_register(lora, REG_MODEM_CONFIG_1, mc1);
}

void lora_set_tx_power(lora_sx1276 *lora, uint8_t level)
{
  assert_param(lora);

  if (lora->pa_mode == LORA_PA_OUTPUT_RFO) {
    // RFO pin
    assert_param(level <= 15);
    if (level > 15) {
      level = 15;
    }
    // 7 bit -> PaSelect: 0 for RFO    --- = 0x70
    // 6-4 bits -> MaxPower (select all) --^
    // 3-0 bits -> Output power, dB (max 15)
```

```
    write_register(lora, REG_PA_CONFIG, 0x70 | level);
  } else {
    // PA BOOST pin, from datasheet (Power Amplifier):
    //   Pout=17-(15-OutputPower)
    assert_param(level <= 20 && level >= 2);
    if (level > 20) {
      level = 20;
    }
    if (level < 2) {
      level = 2;
    }
    // Module power consumption from datasheet:
    // RFOP = +20 dBm, on PA_BOOST -> 120mA
    // RFOP = +17 dBm, on PA_BOOST -> 87mA
    if (level > 17) {
      // PA_DAC_HIGH_POWER operation changes last 3 OutputPower modes to:
      // 13 -> 18dB, 14 -> 19dB, 15 -> 20dB
      // So subtract 3 from level
      level -= 3;
      // Enable High Power mode
      write_register(lora, REG_PA_DAC, PA_DAC_HIGH_POWER);
      // Limit maximum current to 140mA (+20mA to datasheet value to be sure)
      set_OCP(lora, 140);
    } else {
      // Enable half power mode (default)
      write_register(lora, REG_PA_DAC, PA_DAC_HALF_POWER);
      // Limit maximum current to 97mA (+10mA to datasheet value to be sure)
      set_OCP(lora, 97);
    }
    // Minimum power level is 2 which is 0 for chip
    level -= 2;
    // 7 bit -> PaSelect: 1 for PA_BOOST
    write_register(lora, REG_PA_CONFIG, BIT_7 | level);
  }
}

void lora_set_frequency(lora_sx1276 *lora, uint64_t freq)
{
  assert_param(lora);

  // From datasheet: FREQ = (FRF * 32 Mhz) / (2 ^ 19)
  uint64_t frf = (freq << 19) / (32 * MHZ);

  write_register(lora, REG_FRF_MSB, frf >> 16);
  write_register(lora, REG_FRF_MID, (frf & 0xff00) >> 8);
  write_register(lora, REG_FRF_LSB, frf & 0xff);
}

void lora_set_syncword(lora_sx1276 *lora, uint8_t sw) {
          write_register(lora, REG_SYNC_WORD, sw);
}

int8_t lora_packet_rssi(lora_sx1276 *lora)
{
  assert_param(lora);
```

```
  uint8_t rssi = read_register(lora, REG_PKT_RSSI_VALUE);

  return lora->frequency < (868 * MHZ) ? rssi - 164 : rssi - 157;
}

uint8_t lora_packet_snr(lora_sx1276 *lora)
{
  assert_param(lora);

  uint8_t snr = read_register(lora, REG_PKT_SNR_VALUE);

  return snr / 5;
}

void lora_set_signal_bandwidth(lora_sx1276 *lora, uint64_t bw)
{
  assert_param(lora && bw < LORA_BW_LAST);

  // REG_MODEM_CONFIG_1 has 2 more parameters:
  // Coding rate / Header mode, so read them before set bandwidth
  uint8_t mc1 = read_register(lora, REG_MODEM_CONFIG_1);
  // Signal bandwidth uses 4-7 bits of config
  mc1 = (bw << 4) | (mc1 & 0x0F);
  write_register(lora, REG_MODEM_CONFIG_1, mc1);

  set_low_data_rate_optimization(lora);
}

void lora_set_spreading_factor(lora_sx1276 *lora, uint8_t sf)
{
  assert_param(lora && sf <= 12 && sf >=6);

  if (sf < 6) {
    sf = 6;
  } else if (sf > 12) {
    sf = 12;
  }

  if (sf == 6) {
    write_register(lora, REG_DETECTION_OPTIMIZE, 0xc5);
    write_register(lora, REG_DETECTION_THRESHOLD, 0x0c);
  } else {
    write_register(lora, REG_DETECTION_OPTIMIZE, 0xc3);
    write_register(lora, REG_DETECTION_THRESHOLD, 0x0a);
  }
  // Set new spread factor
  uint8_t mc2 = read_register(lora, REG_MODEM_CONFIG_2);
  mc2 = (mc2 & 0x0F) | (sf << 4);
  // uint8_t new_config = (current_config & 0x0f) | ((sf << 4) & 0xf0);
  write_register(lora, REG_MODEM_CONFIG_2, mc2);

  set_low_data_rate_optimization(lora);
}
```

```
void lora_set_crc(lora_sx1276 *lora, uint8_t enable)
{
  assert_param(lora);

  uint8_t mc2 = read_register(lora, REG_MODEM_CONFIG_2);

  if (enable) {
    mc2 |= MC2_CRC_ON;
  } else {
    mc2 &= ~MC2_CRC_ON;
  }

  write_register(lora, REG_MODEM_CONFIG_2, mc2);
}

void lora_set_coding_rate(lora_sx1276 *lora, uint8_t rate)
{
  assert_param(lora);

  uint8_t mc1 = read_register(lora, REG_MODEM_CONFIG_1);

  // coding rate bits are 1-3 in modem config 1 register
  mc1 |= rate << 1;
  write_register(lora, REG_MODEM_CONFIG_1, mc1);
}

void lora_set_preamble_length(lora_sx1276 *lora, uint16_t len)
{
  assert_param(lora);

  write_register(lora, REG_PREAMBLE_MSB, len >> 8);
  write_register(lora, REG_PREAMBLE_LSB, len & 0xf);
}

uint8_t lora_version(lora_sx1276 *lora)
{
  assert_param(lora);

  return read_register(lora, REG_VERSION);
}

uint8_t lora_is_transmitting(lora_sx1276 *lora)
{
  assert_param(lora);

  uint8_t opmode = read_register(lora, REG_OP_MODE);

  return (opmode & OPMODE_TX) == OPMODE_TX ? LORA_BUSY : LORA_OK;
}

static uint8_t lora_send_packet_base(lora_sx1276 *lora, uint8_t *data, uint8_t data_len, uint8_t mode)
{
  assert_param(lora && data && data_len > 0);

  if (lora_is_transmitting(lora)) {
```

```
    return LORA_BUSY;
  }

  // Wakeup radio because of FIFO is only available in STANDBY mode
  set_mode(lora, OPMODE_STDBY);

  // Clear TX IRQ flag, to be sure
  lora_clear_interrupt_tx_done(lora);

  // Set FIFO pointer to the beginning of the buffer
  write_register(lora, REG_FIFO_ADDR_PTR, lora->tx_base_addr);
  write_register(lora, REG_FIFO_TX_BASE_ADDR, lora->tx_base_addr);
  write_register(lora, REG_PAYLOAD_LENGTH, data_len);

  // Copy packet into radio FIFO
  write_fifo(lora, data, data_len, mode);
  if (mode == TRANSFER_MODE_DMA) {
    return LORA_OK;
  }

  // Put radio in TX mode - packet will be transmitted ASAP
  set_mode(lora, OPMODE_TX);
  return LORA_OK;
}

uint8_t lora_send_packet(lora_sx1276 *lora, uint8_t *data, uint8_t data_len)
{
  return lora_send_packet_base(lora, data, data_len, TRANSFER_MODE_BLOCKING);
}

uint8_t lora_send_packet_dma_start(lora_sx1276 *lora, uint8_t *data, uint8_t data_len)
{
  return lora_send_packet_base(lora, data, data_len, TRANSFER_MODE_DMA);
}

// Finish packet send initiated by lora_send_packet_dma_start()
void  lora_send_packet_dma_complete(lora_sx1276 *lora)
{
  // End transfer
  HAL_GPIO_WritePin(lora->nss_port, lora->nss_pin, GPIO_PIN_SET);
  // Send packet
  set_mode(lora, OPMODE_TX);
}

uint8_t lora_send_packet_blocking(lora_sx1276 *lora, uint8_t *data, uint8_t data_len, uint32_t
timeout)
{
  assert_param(lora && data && data_len > 0 && timeout > 0);

  uint8_t res = lora_send_packet(lora, data, data_len);

  if (res == LORA_OK) {
    // Wait until packet gets transmitted
    uint32_t elapsed = 0;
    while (elapsed < timeout) {
```

```
      uint8_t state = read_register(lora, REG_IRQ_FLAGS);
      if (state & IRQ_FLAGS_TX_DONE) {
        // Packet sent
        write_register(lora, REG_IRQ_FLAGS, IRQ_FLAGS_TX_DONE);
        return LORA_OK;
      }
      HAL_Delay(1);
      elapsed++;
    }
  }

  return LORA_TIMEOUT;
}

void lora_set_rx_symbol_timeout(lora_sx1276 *lora, uint16_t symbols)
{
  assert_param(lora && symbols <= 1024 && symbols >= 4);

  if (symbols < 4) {
    symbols = 4;
  }
  if (symbols > 1023) {
    symbols = 1024;
  }

  write_register(lora, REG_SYMB_TIMEOUT_LSB, symbols & 0xf);
  if (symbols > 255) {
    // MSB (2 first bits of config2)
    uint8_t mc2 = read_register(lora, REG_MODEM_CONFIG_2);
    mc2 |= symbols >> 8;
    write_register(lora, REG_MODEM_CONFIG_2, mc2);
  }
}

uint8_t lora_is_packet_available(lora_sx1276 *lora)
{
  assert_param(lora);

  uint8_t irqs = read_register(lora, REG_IRQ_FLAGS);

  // In case of Single receive mode RX_TIMEOUT will be issued
  return  irqs & (IRQ_FLAGS_RX_DONE | IRQ_FLAGS_RX_TIMEOUT);
}

uint8_t lora_pending_packet_length(lora_sx1276 *lora)
{
  uint8_t len;

  // Query for current header mode - implicit / explicit
  uint8_t implicit = read_register(lora, REG_MODEM_CONFIG_1) &
MC1_IMPLICIT_HEADER_MODE;
  if (implicit) {
    len = read_register(lora, REG_PAYLOAD_LENGTH);
  } else {
    len = read_register(lora, REG_RX_NB_BYTES);
```

```
  }

  return len;
}


static uint8_t lora_receive_packet_base(lora_sx1276 *lora, uint8_t *buffer, uint8_t buffer_len, uint8_t
*error, uint8_t mode)
{
  assert_param(lora && buffer && buffer_len > 0);

  uint8_t res = LORA_EMPTY;
  uint8_t len = 0;

  // Read/Reset IRQs
  uint8_t state = read_register(lora, REG_IRQ_FLAGS);
  write_register(lora, REG_IRQ_FLAGS, IRQ_FLAGS_RX_ALL);

  if (state & IRQ_FLAGS_RX_TIMEOUT) {
    DEBUGF("timeout");
    res = LORA_TIMEOUT;
    goto done;
  }

  if (state & IRQ_FLAGS_RX_DONE) {
    if (!(state & IRQ_FLAGS_VALID_HEADER)) {
      DEBUGF("invalid header");
      res = LORA_INVALID_HEADER;
      goto done;
    }
    // Packet has been received
    if (state & IRQ_FLAGS_PAYLOAD_CRC_ERROR) {
      DEBUGF("CRC error");
      res = LORA_CRC_ERROR;
      goto done;
    }
    // Query for current header mode - implicit / explicit
    len = lora_pending_packet_length(lora);
    // Set FIFO to beginning of the packet
    uint8_t offset = read_register(lora, REG_FIFO_RX_CURRENT_ADDR);
    write_register(lora, REG_FIFO_ADDR_PTR, offset);
    // Read payload
    read_fifo(lora, buffer, len, mode);
    res = LORA_OK;
  }

done:
  if (error) {
    *error = res;
  }

  return len;
}

uint8_t lora_receive_packet(lora_sx1276 *lora, uint8_t *buffer, uint8_t buffer_len, uint8_t *error)
```

```
{
  return lora_receive_packet_base(lora, buffer, buffer_len, error, TRANSFER_MODE_BLOCKING);
}

uint8_t lora_receive_packet_dma_start(lora_sx1276 *lora, uint8_t *buffer, uint8_t buffer_len, uint8_t
*error)
{
  return lora_receive_packet_base(lora, buffer, buffer_len, error, TRANSFER_MODE_DMA);
}

void lora_receive_packet_dma_complete(lora_sx1276 *lora)
{
  // Nothing to do expect - just end SPI transaction
  HAL_GPIO_WritePin(lora->nss_port, lora->nss_pin, GPIO_PIN_SET);
}

uint8_t lora_receive_packet_blocking(lora_sx1276 *lora, uint8_t *buffer, uint8_t buffer_len,
                uint32_t timeout, uint8_t *error)
{
  assert_param(lora && buffer && buffer_len > 0);

  uint32_t elapsed = 0;

  // Wait up to timeout for packet
  while (elapsed < timeout) {
    if (lora_is_packet_available(lora)) {
      break;
    }
    HAL_Delay(1);
    elapsed++;
  }

  return lora_receive_packet(lora, buffer, buffer_len, error);
}

void lora_enable_interrupt_rx_done(lora_sx1276 *lora)
{
  // Table 63 DIO Mapping LoRaTM Mode:
  // 00 -> (DIO0 rx_done)
  // DIO0 uses 6-7 bits of DIO_MAPPING_1
  write_register(lora, REG_DIO_MAPPING_1, 0x00);
}

void lora_enable_interrupt_tx_done(lora_sx1276 *lora)
{
  // Table 63 DIO Mapping LoRaTM Mode:
  // 01 -> (DIO0 tx_done)
  // DIO0 uses 6-7 bits of DIO_MAPPING_1
  write_register(lora, REG_DIO_MAPPING_1, 0x40);
}

void lora_clear_interrupt_tx_done(lora_sx1276 *lora)
{
  write_register(lora, REG_IRQ_FLAGS, IRQ_FLAGS_TX_DONE);
}
```

```
void lora_clear_interrupt_rx_all(lora_sx1276 *lora)
{
  write_register(lora, REG_IRQ_FLAGS, IRQ_FLAGS_RX_ALL);
}


uint8_t lora_init(lora_sx1276 *lora, SPI_HandleTypeDef *spi, GPIO_TypeDef *nss_port,
    uint16_t nss_pin, uint64_t freq)
{
  assert_param(lora && spi);

  // Init params with default values
  lora->spi = spi;
  lora->nss_port = nss_port;
  lora->nss_pin = nss_pin;
  lora->frequency = freq;
  lora->pa_mode = LORA_PA_OUTPUT_PA_BOOST;
  lora->tx_base_addr = LORA_DEFAULT_TX_ADDR;
  lora->rx_base_addr = LORA_DEFAULT_RX_ADDR;
  lora->spi_timeout = LORA_DEFAULT_SPI_TIMEOUT;

  // Check version
  uint8_t ver = lora_version(lora);
  if (ver != LORA_COMPATIBLE_VERSION) {
    DEBUGF("Got wrong radio version 0x%x, expected 0x12", ver);
    return LORA_ERROR;
  }

  // Modem parameters (freq, mode, etc) must be done in SLEEP mode.
  lora_mode_sleep(lora);
  // Enable LoRa mode (since it can be switched on only in sleep)
  lora_mode_sleep(lora);

  // Set frequency
  lora_set_frequency(lora, freq);
  lora_set_spreading_factor(lora, LORA_DEFAULT_SF);
  lora_set_preamble_length(lora, LORA_DEFAULT_PREAMBLE_LEN);

  // By default - explicit header mode
  lora_set_explicit_header_mode(lora);

  // Set LNA boost
  uint8_t current_lna = read_register(lora, REG_LNA);
  write_register(lora, REG_LNA,  current_lna | 0x03);
  // Set auto AGC
  write_register(lora, REG_MODEM_CONFIG_3, 0x04);
  // Set default output power
  lora_set_tx_power(lora, LORA_DEFAULT_TX_POWER);
  // Set default mode
  lora_mode_standby(lora);

  return LORA_OK;
}
```

################################### lora1276.c ######################################

################################### gps.h ###

```
#ifndef __GPS_H
#define __GPS_H

#include "main.h"

#define GPS_DEBUG   1
#define GPS_USART   &huart1

#define GPGGA 1
#define GPRMC 2
#define GPGLL 3
#define GPVTG 4

void  GPS_Init();
void  GPS_UART_CallBack();
void  GPS_Loop();

typedef struct{

   // calculated values
   float dec_longitude;
   float dec_latitude;
   float altitude_ft;

   // GGA - Global Positioning System Fixed Data
   float nmea_longitude;
   float nmea_latitude;
   float utc_time;
   char ns, ew;
   bool lock;
   int satelites;
   float hdop;
   float msl_altitude;
   char msl_units;
   float geoid;

   // RMC - Recommended Minimmum Specific GNS Data
   char rmc_status;
   float speed_k;
   float course_d;
   int date;
```

```
    // GLL
    char gll_status;

    // VTG - Course over ground, ground speed
    float course_t; // ground speed true
    char course_t_unit;
    float course_m; // magnetic
    char course_m_unit;
    char speed_k_unit;
    float speed_km; // speek km/hr
    char speed_km_unit;
} GPS_t;

int GPS_validate(char *nmeastr);
void GPS_parse(char *GPSstrParse);
float GPS_nmea_to_dec(float deg_coord, char nsew);

extern GPS_t GPS;

#endif
```

###################################### gps.h ##

###################################### gps.c ##

```
#include <stdio.h>
#include <string.h>
#include "gps.h"
#include "stdint.h"
#include "stdlib.h"
#include "math.h"
#include "stdbool.h"

GPS_t GPS;

int GPS_validate(char *nmeastr) {
    char check[3];
    char checkcalcstr[3];
    int i;
    int calculated_check;
    int maxStringLength = 75;

    i=0;
    calculated_check=0;

    // check to ensure that the string starts with a $
    if(nmeastr[i] == '$')
        i++;
    else
        return 0;
```

```
    //No NULL reached, 75 char largest possible NMEA message, no '*' reached
    while((nmeastr[i] != 0) && (nmeastr[i] != '*') && (i < maxStringLength)){
      calculated_check ^= nmeastr[i];// calculate the checksum
      i++;
    }

    if(i >= maxStringLength){
      return 0;// the string was too long so return an error
    }

    if (nmeastr[i] == '*'){
      check[0] = nmeastr[i+1];    //put hex chars in check string
      check[1] = nmeastr[i+2];
      check[2] = 0;
    }
    else
      return 0;// no checksum separator found there for invalid

    sprintf(checkcalcstr,"%02X",calculated_check);
    return((checkcalcstr[0] == check[0])
      && (checkcalcstr[1] == check[1])) ? 1 : 0 ;
}

void parseGPSString(char *gpsString , uint8_t type) {

        //DEBUGF("[parseGPSString] Type:%d",type);

        char *token = strtok(gpsString, ",");
        int num = 0;
        while (token != NULL) {
                //DEBUGF("[parseGPSString] Type:GPGGA Num:%d",num);
                if (type==GPGGA) {

                        switch (num) {
                                case 1:
                                        GPS.utc_time = atof(token);
                                break;
                                case 2:
                                        GPS.nmea_latitude = atof(token);
                                break;
                                case 3:
                                        GPS.ns = token[0];
                                break;
                                case 4:
                                        GPS.nmea_longitude = atof(token);
                                break;
                                case 5:
                                        GPS.ew = token[0];
                                break;
                                case 6:
                                        GPS.lock = token[0] > '0';
                                break;
                                case 7:
                                        GPS.satelites = atoi(token);
```

```
                        break;
                        case 8:
                                GPS.hdop = atof(token);
                        break;
                        case 9:
                                GPS.msl_altitude = atof(token);
                        break;
                        case 10:
                                GPS.msl_units = token[0];
                        break;
                        case 11:
                                GPS.geoid = atof(token);
                        break;
                }
                num++;
        }
        if (type==GPRMC) {
                //&GPS.utc_time, &GPS.nmea_latitude, &GPS.ns, &GPS.nmea_longitude,
&GPS.ew, &GPS.speed_k, &GPS.course_d, &GPS.date
                switch (num) {
                        case 1:
                                GPS.utc_time = atof(token);
                        break;
                        case 2:
                                GPS.nmea_latitude = atof(token);
                        break;
                        case 3:
                                GPS.ns = token[0];
                        break;
                        case 4:
                                GPS.nmea_longitude = atof(token);
                        break;
                        case 5:
                                GPS.ew = token[0];
                        break;
                        case 6:
                                GPS.speed_k = atof(token);
                        break;
                        case 7:
                                GPS.course_d = atof(token);
                        break;
                        case 8:
                                GPS.date = atoi(token);
                        break;
                }
                num++;
        }
        if (type==GPGLL) {
                //"$GPGLL,%f,%c,%f,%c,%f,%c", &GPS.nmea_latitude, &GPS.ns,
&GPS.nmea_longitude, &GPS.ew, &GPS.utc_time, &GPS.gll_status)
                switch (num) {
                        case 1:
                                GPS.nmea_latitude = atof(token);
                        break;
                        case 2:
```

```
                                        GPS.ns = token[0];
                                break;
                                case 3:
                                        GPS.nmea_longitude = atof(token);
                                break;
                                case 4:
                                        GPS.nmea_longitude = atof(token);
                                break;
                                case 5:
                                        GPS.ew = token[0];
                                break;
                                case 6:
                                        GPS.utc_time = atof(token);
                                break;
                                case 7:
                                        GPS.gll_status = token[0];
                                break;
                        }
                        num++;
                }
                if (type==GPVTG) {
                        //&GPS.course_t, &GPS.course_t_unit, &GPS.course_m,
&GPS.course_m_unit, &GPS.speed_k, &GPS.speed_k_unit, &GPS.speed_km,
&GPS.speed_km_unit)
                        switch (num) {
                                case 1:
                                        GPS.course_t = atof(token);
                                break;
                                case 2:
                                        GPS.course_t_unit = token[0];
                                break;
                                case 3:
                                        GPS.course_m = atof(token);
                                break;
                                case 4:
                                        GPS.course_m_unit = token[0];
                                break;
                                case 5:
                                        GPS.speed_k = atof(token);
                                break;
                                case 6:
                                        GPS.speed_k_unit = token[0];
                                break;
                                case 7:
                                        GPS.speed_km = atof(token);
                                break;
                                case 8:
                                        GPS.speed_km_unit = token[0];
                                break;
                        }
                        num++;
                }

                token = strtok(NULL, ",");
```

```
        }
        if (type==GPGGA) {
                GPS.dec_latitude  = GPS_nmea_to_dec(GPS.nmea_latitude,  GPS.ns);
                GPS.dec_longitude = GPS_nmea_to_dec(GPS.nmea_longitude, GPS.ew);
        }

}

void GPS_parse(char *GPSstrParse){
   if(!strncmp(GPSstrParse, "$GPGGA", 6))       {
        parseGPSString(GPSstrParse,GPGGA);
   }
   else if (!strncmp(GPSstrParse, "$GPRMC", 6)) {
        parseGPSString(GPSstrParse,GPRMC);
   }
   else if (!strncmp(GPSstrParse, "$GPGLL", 6)) {
        parseGPSString(GPSstrParse,GPGLL);
   }
   else if (!strncmp(GPSstrParse, "$GPVTG", 6)) {
        parseGPSString(GPSstrParse,GPVTG);
   }
}

float GPS_nmea_to_dec(float deg_coord, char nsew) {
   int degree = (int)(deg_coord/100);
   float minutes = deg_coord - degree*100;
   float dec_deg = minutes / 60;
   float decimal = degree + dec_deg;
   if (nsew == 'S' || nsew == 'W') { // return negative
      decimal *= -1;
   }
   return decimal;
}
```

###################################### gps.c ######################################

###################################### main.c ######################################

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
```

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "rtc.h"
#include "spi.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>

#include "lcd.h"
#include "lora_sx1276.h"
#include "npk.h"
#include "gps.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
lora_sx1276 lora;
npkData npk;
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

/*UART-SERIAL CUSTOM */
uint64_t timeNow;
uint8_t pageCounter=0;
bool pageChanged=false;

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
	cbkMillis(htim);
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
	cbkUart(huart);
}
```

```
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart) {

	uint8_t typeUart = 0;
	if (huart->Instance == DEBUG_UART.Instance) {
		typeUart = UART_DEBUG;
	}
	if (huart->Instance == NPK_UART.Instance) {
		typeUart = UART_NPK;
	}
	if (huart->Instance == GPS_UART.Instance) {
		typeUart = UART_GPS;
	}

	DEBUGF("[UART] USART%d ErrorCallback :%d", typeUart+1, huart->ErrorCode);

}

void loraInit(lora_sx1276* lora) {

	bool loraInit = false;
	HAL_GPIO_WritePin(LORA_RESET_GPIO_Port, LORA_RESET_Pin, LOW);
	HAL_Delay(10);
	HAL_GPIO_WritePin(LORA_RESET_GPIO_Port, LORA_RESET_Pin, HIGH);
	HAL_Delay(10);

	HAL_GPIO_WritePin(LORA_NSS_GPIO_Port, LORA_NSS_Pin, HIGH);

	uint8_t res = lora_init(lora, &hspi1, LORA_NSS_GPIO_Port, LORA_NSS_Pin, 868E6);
	if (res == LORA_ERROR) {
		DEBUGF("[Lora]Firstinit Failed");
		uint8_t ver = lora_version(lora);
		DEBUGF("[Lora]Checking again version for 0x12");
		while (ver != 0x12) {
			DEBUGF("[Lora]Wrong version:%d",ver);
			ver = lora_version(lora);
			HAL_Delay(1000);
		}
		res = lora_init(lora, &hspi1, LORA_NSS_GPIO_Port, LORA_NSS_Pin,
LORA_BASE_FREQUENCY_EU);
		if (res == LORA_OK) {
			loraInit = true;
			DEBUGF("[Lora]Init successfully");
		}
		else {
			DEBUGF("[Lora]Init stuck");
			while (1);
		}
	}
	else {
		DEBUGF("[Lora]Init OK");
		loraInit = true;
	}

	if (loraInit) {
		//lora_set_crc(lora,1);
```

```
                //lora_set_signal_bandwidth(&lora,7);
                lora_set_tx_power(lora,20);
                lora_set_syncword(lora,0xA5);
                lora_set_spreading_factor(lora, 12);
        }

}
void loraLoop(lora_sx1276* lora) {

        char buf[128];

        //float demoLat  = 44.350422;
        //float demoLng  = 26.493383;

        //12345678901234567890-12345678901234567890-12345678901234567890
        //sprintf(buf, "%.9f,%.9f,hmmmmmmmmmmm", GPS.dec_latitude, GPS.dec_longitude);


        if (GPS.dec_latitude!=0 && GPS.dec_longitude!=0) {

                HAL_GPIO_TogglePin(LORA_TX_GPIO_Port, LORA_TX_Pin);

                uint8_t bytes[64];
                uint8_t bytesIndex = 0;

                floatToBytes longBytes;
                floatToBytes latBytes;
                latBytes.floatVal  = GPS.dec_latitude;
                longBytes.floatVal = GPS.dec_longitude;

                for (uint8_t i=0;i<sizeof(floatToBytes);i++){
                        bytes[bytesIndex++] = latBytes.bytes[i];
                }
                bytes[bytesIndex++] = ',';
                for (uint8_t i=0;i<sizeof(floatToBytes);i++){
                        bytes[bytesIndex++] = longBytes.bytes[i];
                }
                bytes[bytesIndex++] = ',';
                for (uint8_t i=0;i<NPK_DATA_BYTES_LEN;i++){
                        bytes[bytesIndex++] = npk.bytes[i];
                }
                bytes[bytesIndex++] = '\0';

                lora_send_packet(lora , (uint8_t*)&bytes, bytesIndex );

//              char npkBytes[52];
//              uint8_t npkIndex=0;
//              for (uint16l_t i=10;i<bytesIndex-1;i++){
//                              npkBytes[npkIndex++] = bytes[i];
//              }
//              npkParseBytes(&npk, (char*)&npkBytes, npkIndex);
//              DEBUGF("======================AFTER
PARSE======================");
//              npkDebug(&npk);
```

```
                        //lora_send_packet_dma_start(lora , (uint8_t*)&buf, strlen(buf) );
//                      for (uint8_t i=0;i<bytesIndex;i++){
//                              PRINT("%d ", bytes[i]);
//                      }
//                      PRINT("\n");

//                      char deb[128];
//                      sprintf(deb,"[Lora]Sending ->%s with Length:%d",bytes,bytesIndex);
//                      DEBUGF((uint8_t*)&deb);

//lora_send_packet_dma_complete(lora);

                }

}

void checkNPK () {

  char d[512];
  uint16_t len = readUart( (char*)&d , UART_NPK);
  if ( len > 0 ) {
                //DEBUGF("[RS485]Got Some answer Len:%d\n",len);
                npkParse(&npk, (char*)&d,len);
                npkDebug(&npk);
  }

}

void checkDEBUG () {
  char d[512];
  uint16_t len = readUart( (char*)&d , UART_DEBUG);
  if ( len > 0 ) {
                DEBUGF("[DEBUG]Got Some answer DATA:%d\n",len);
                if (d[0] == 'R') {
                        NVIC_SystemReset();
                }
  }
}

void checkGPS () {
  char d[512];
  uint16_t len = readUart( (char*)&d , UART_GPS);
  if ( len > 0 ) {
//              for (uint16_t i=0;i<len;i++){
//                      PRINT("%c ",d[i]);
//              }
//              DEBUGF("===");
//              d[len-1] = '\0';
                if ( GPS_validate((char*)&d) ) {
                        GPS_parse( (char*)&d );
                        if ( GPS.satelites>0) DEBUGF("Location:
%.6f,%.6f",GPS.dec_latitude,GPS.dec_longitude);

                }
  }
```

```
}
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void PeriphCommonClock_Config(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
static void LED_Blink(uint32_t Hdelay,uint32_t Ldelay)
{
	HAL_GPIO_WritePin(STATUS_LED_GPIO_Port,STATUS_LED_Pin,GPIO_PIN_SET);
	HAL_Delay(Hdelay - 1);
	HAL_GPIO_WritePin(STATUS_LED_GPIO_Port,STATUS_LED_Pin,GPIO_PIN_SET);
	HAL_Delay(Ldelay-1);
}

void clearScreen() {
//	char text[64];
//	sprintf(text, "                          ");
//	LCD_ShowString(4, 4,  ST7735Ctx.Width, 16, 16, text);
//	LCD_ShowString(4, 22, ST7735Ctx.Width, 16, 16, text);
//	LCD_ShowString(4, 40, ST7735Ctx.Width, 16, 16, text);
//	LCD_ShowString(4, 56, ST7735Ctx.Width, 16, 16, text);
	ST7735_FillRect(&st7735_pObj , 0 , 0 , ST7735Ctx.Width , ST7735Ctx.Height , BLACK);
}

void lcdMeasurements() {

	char measurement[32];

	if (pageChanged) {
		clearScreen();
		pageChanged=false;
	}

	if (pageCounter==0) {

		sprintf(measurement,"EC:%d us/cm",npk.Conductivity);
		LCD_ShowString(4, 4, ST7735Ctx.Width, 16, 16, measurement);

		sprintf(measurement,"Temperature:%.2f °C",npk.Temperature);
		LCD_ShowString(4, 22, ST7735Ctx.Width, 16, 16, measurement);

		sprintf(measurement,"Humidity:%.2f",npk.Moisture);
		LCD_ShowString(4, 40, ST7735Ctx.Width, 16, 16, measurement);

		sprintf(measurement,"PH:%.2f PH",npk.PH);
		LCD_ShowString(4, 62, ST7735Ctx.Width, 16, 16, measurement);

	}
```

```
        if (pageCounter==1) {

                sprintf(measurement,"Nitrogen:%d mg/kg",npk.Nitrogen);
                LCD_ShowString(4, 4, ST7735Ctx.Width, 16, 16, measurement);

                sprintf(measurement,"Phosphorus:%d mg/kg",npk.Phosphorus);
                LCD_ShowString(4, 22, ST7735Ctx.Width, 16, 16, measurement);

                sprintf(measurement,"Potassium:%d mg/kg",npk.Potassium);
                LCD_ShowString(4, 40, ST7735Ctx.Width, 16, 16, measurement);

                sprintf(measurement,"TDS:%d ppm\r",npk.TDS);
                LCD_ShowString(4, 62, ST7735Ctx.Width, 16, 16, measurement);

        }


}

// EXTI Line9 External Interrupt ISR Handler CallBackFun
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
   if(GPIO_Pin == BUTTON_Pin)
   {
        HAL_GPIO_TogglePin(LORA_TX_GPIO_Port, LORA_TX_Pin);
        pageCounter = !pageCounter;
        pageChanged=true;
   }


}


/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
```

```
  SystemClock_Config();

/* Configure the peripherals common clocks */
  PeriphCommonClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_RTC_Init();
  MX_SPI4_Init();
  MX_TIM1_Init();
  MX_SPI1_Init();
  MX_USART1_UART_Init();
  MX_USART2_UART_Init();
  MX_USART3_UART_Init();
  MX_TIM7_Init();
  /* USER CODE BEGIN 2 */

  LCD_Test();
  initMillis();
  DEBUGF("Pre UARTS");
  initUarts();
  DEBUGF("AFTER Uarts");

  loraInit(&lora);



  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
          //  inLoop = true;
          //        char text[128];
          //
          //        sprintf(text, "This a Measurement Device");
          //        LCD_ShowString(4, 4, ST7735Ctx.Width, 16, 16, text);
          //        sprintf(text, "STM32H7xx 0x%X", HAL_GetDEVID());
          //        LCD_ShowString(4, 22, ST7735Ctx.Width, 16, 16, text);
          //        sprintf(text, "LCD ID:0x%X", st7735_id);
          //        LCD_ShowString(4, 40, ST7735Ctx.Width, 16, 16, text);
          //        sprintf(text, "123456789123456789123456789", st7735_id);

          //  LED_Blink(150,500);
          //  clearScreen();

          if (millis() - timeNow > 250) {
```

```
                timeNow = millis();
                uint8_t cmd[] =  {0x01, 0x03, 0x00, 0x00, 0x00 , 0x09,  0x85 , 0xCC};
                HAL_StatusTypeDef res = HAL_UART_Transmit(&NPK_UART, (uint8_t*)&cmd,
sizeof(cmd), 1000);
                if (res == HAL_OK) {
                        checkNPK();
                }
                lcdMeasurements();
        }
        checkDEBUG();
        checkGPS();
        static uint64_t loraSend = 0;
        if (millis() - loraSend > 800) {
                loraSend = millis();
                loraLoop(&lora);
        }
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Supply configuration update enable
  */
  HAL_PWREx_ConfigSupply(PWR_LDO_SUPPLY);

  /** Configure the main internal regulator output voltage
  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);

  while(!__HAL_PWR_GET_FLAG(PWR_FLAG_VOSRDY)) {}

  /** Configure LSE Drive Capability
  */
  HAL_PWR_EnableBkUpAccess();
  __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_HSE
                      |RCC_OSCILLATORTYPE_LSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.HSIState = RCC_HSI_DIV1;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
```

```
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 5;
  RCC_OscInitStruct.PLL.PLLN = 96;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  RCC_OscInitStruct.PLL.PLLRGE = RCC_PLL1VCIRANGE_2;
  RCC_OscInitStruct.PLL.PLLVCOSEL = RCC_PLL1VCOWIDE;
  RCC_OscInitStruct.PLL.PLLFRACN = 0;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2
                              |RCC_CLOCKTYPE_D3PCLK1|RCC_CLOCKTYPE_D1PCLK1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.SYSCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB3CLKDivider = RCC_APB3_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_APB1_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_APB2_DIV1;
  RCC_ClkInitStruct.APB4CLKDivider = RCC_APB4_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief Peripherals Common Clock Configuration
  * @retval None
  */
void PeriphCommonClock_Config(void)
{
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};

  /** Initializes the peripherals clock
  */
  PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_CKPER;
  PeriphClkInitStruct.CkperClockSelection = RCC_CLKPSOURCE_HSI;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */
```

```
/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

ΠΑΡΑΡΤΗΜΑ Β : Κώδικας ESP32

```
#include <LoRa.h>
#include "boards.h"

#define DEBUGF Serial.println

typedef union {
  float floatVal;
  uint8_t bytes[4];
} floatToBytes;

typedef union Val {
  uint16_t raw;
  uint8_t bytes[2];
} Val;

#define NPK_DATA_BYTES_LEN 9*2
```

```
typedef struct npkData {
  union {
    struct {
      Val _Moisture;
      Val _Temperature;
      Val _Conductivity;
      Val _PH;
      Val _Nitrogen;
      Val _Phosphorus;
      Val _Potassium;
      Val _Salinity;
      Val _TDS;
    };
    uint8_t bytes[NPK_DATA_BYTES_LEN]; //total 18 bytes
  };

  float Moisture;
  float Temperature;
  uint16_t Conductivity;
  float PH;
  uint16_t Nitrogen;
  uint16_t Phosphorus;
  uint16_t Potassium;
  uint16_t Salinity;
  uint16_t TDS;

} npkData;

npkData npk;

void npkParse(npkData* Sensor , char* d , uint16_t len);
void npkDebug(npkData* Sensor);

void npkConversions(npkData* Sensor) {

    if ( abs( 65536 - Sensor->_Temperature.raw ) < 250 ) {
      Sensor->Temperature = - (65536 - Sensor->_Temperature.raw) * 0.1;
    }
    else {
      Sensor->Temperature = Sensor->_Temperature.raw * 0.1;
    }
    Sensor->Moisture = Sensor->_Moisture.raw * 0.1;
    Sensor->Conductivity = Sensor->_Conductivity.raw;
    Sensor->PH = Sensor->_PH.raw*0.1;
    Sensor->Nitrogen =  Sensor->_Nitrogen.raw;
    Sensor->Phosphorus =  Sensor->_Phosphorus.raw;
    Sensor->Potassium =  Sensor->_Potassium.raw;
    Sensor->Salinity =  Sensor->_Salinity.raw;
    Sensor->TDS =  Sensor->_TDS.raw;

}

void npkParse(npkData* Sensor , char* d , uint16_t len) {

    uint8_t b = 0;
```

```
    for(uint8_t i=3;i<len-2;i++){
      Sensor->bytes[b] = d[i];
      b++;
    }

    for(uint8_t i=0;i<sizeof(npkData);i=i+2){
      uint8_t temp = Sensor->bytes[i];
      Sensor->bytes[i] = Sensor->bytes[i+1];
      Sensor->bytes[i+1] = temp;
    }

    npkConversions(Sensor);

}

void npkParseBytes(npkData* Sensor  , char* d , uint16_t len) {

  for(uint8_t i=0;i<len;i++) {

    if (i<NPK_DATA_BYTES_LEN) {
      Sensor->bytes[i] = d[i];
    }
  }

  npkConversions(Sensor);

}

void npkDebug(npkData* Sensor) {

    char buf[64];
    sprintf(buf,"Temperature:%.2f °C\r",Sensor->Temperature);
    DEBUGF(buf);
    sprintf(buf,"Moisture:%.2f %%\r",Sensor->Moisture);
    DEBUGF(buf);
    sprintf(buf,"Conductivity:%d us/cm\r",Sensor->Conductivity);
    DEBUGF(buf);
    sprintf(buf,"PH:%.2f PH\r",Sensor->PH);
    DEBUGF(buf);
    sprintf(buf,"Nitrogen:%d mg/kg\r",Sensor->Nitrogen);
    DEBUGF(buf);
    sprintf(buf,"Phosphorus:%d mg/kg\r",Sensor->Phosphorus);
    DEBUGF(buf);
    sprintf(buf,"Potassium:%d mg/kg\r",Sensor->Potassium);
    DEBUGF(buf);
    sprintf(buf,"Salinity:%d ppt\r",Sensor->Salinity);
    DEBUGF(buf);
    sprintf(buf,"TDS:%d ppm\r",Sensor->TDS);
    DEBUGF(buf);

}

void setup()
{
```

```
    initBoard();
    // When the power is turned on, a delay is required.
    delay(1500);
    Serial.println("LoRa Receiver");
    LoRa.setPins(RADIO_CS_PIN, RADIO_RST_PIN, RADIO_DIO0_PIN);
    if (!LoRa.begin(LoRa_frequency)) {
        Serial.println("Starting LoRa failed!");
        while (1);
    }
    LoRa.setSyncWord(0xA5);
    //LoRa.enableCrc();
    LoRa.setSpreadingFactor(12);

    pinMode(BOARD_LED,OUTPUT);

}

void loop()
{
    static bool state=false;

    if (LoRa.parsePacket()) {

        char readByte;
        byte recBytes[128];
        uint16_t recIndex  = 0;
        uint16_t recParser = 0;

        uint8_t npkBytes[64];
        uint8_t npkIndex=0;

        while (LoRa.available()) {
            readByte = (char)LoRa.read();
            recBytes[recIndex++] = readByte;
        }

        floatToBytes gpsLat;
        floatToBytes gpsLong;

        for (uint8_t i=0;i<sizeof(floatToBytes);i++){
            gpsLat.bytes[i] = recBytes[recParser++];
        }
        recParser++; //skip Comma
        for (uint8_t i=0;i<sizeof(floatToBytes);i++){
            gpsLong.bytes[i] = recBytes[recParser++];
        }
        recParser++; //skip Comma
        for (uint16_t i=recParser;i<recIndex-1;i++){
          npkBytes[npkIndex++] = recBytes[i];
        }
//        for (uint16_t i=0;i<npkIndex;i++){
//          Serial.print(npkBytes[i]);
//          Serial.print(" ");
//        }
//        Serial.println();
```

```
        npkParseBytes(&npk, (char*)&npkBytes, npkIndex);

        //npkDebug(&npk);

        //Serial.println(gpsLat.floatVal,6);
        //Serial.println(gpsLong.floatVal,6);

      sprintf((char*)&recBytes,"$Packet,%.6f,%.6f,%.2f,%.2f,%d,%.2f,%d,%d,%d,%d,%d",
        gpsLat.floatVal,
        gpsLong.floatVal,
        npk.Temperature,
        npk.Moisture,
        npk.Conductivity,
        npk.PH,
        npk.Nitrogen,
        npk.Phosphorus,
        npk.Potassium,
        npk.Salinity,
        npk.TDS
      );
      Serial.println((char*)&recBytes);

      //Serial.print("' with RSSI ");
      //Serial.println(LoRa.packetRssi());

#ifdef HAS_DISPLAY
        if (u8g2) {
          u8g2->clearBuffer();
          char buf[256];
          u8g2->drawStr(0, 12, "Received OK!");
          u8g2->drawStr(0, 26, (char*)&recBytes);
          snprintf(buf, sizeof(buf), "RSSI:%i", LoRa.packetRssi());
          u8g2->drawStr(0, 40, buf);
          snprintf(buf, sizeof(buf), "SNR:%.1f", LoRa.packetSnr());
          u8g2->drawStr(0, 56, buf);
          u8g2->sendBuffer();
        }
#endif
      state=!state;
      digitalWrite(BOARD_LED,state);
    }
}
```

ΠΑΡΑΡΤΗΜΑ Γ : Κώδικας Python

```
import asyncio
import datetime
import os
import logging
import json
```

```
path = os.path.abspath(os.path.join(__file__, "../")); config = f"{path}/config.ini"

from libs.myLogger import myLogger
from libs.myDB import myDB

from libs.myQueue import myQueue
from libs.myWebsockets import myWebsockets
from libs.mySerial import mySerial

class myMain(
    myLogger,
    myDB
    ):

    def __init__(self , loop):
        self.path = path
        self.loop = loop

    SESSION_LOGGING = True
    DEV = True
    Flag = True

    def startLogger(self):
        if self.DEV:
            DEBUG_LEVEL = logging.DEBUG
        else:
            DEBUG_LEVEL = logging.INFO
        myLogger.__init__(self , DEBUG_LEVEL)
        if self.SESSION_LOGGING:
            now = datetime.datetime.now().strftime('%Y_%m_%d_%H_%M_%S')
            date = datetime.datetime.now().strftime('%Y_%m_%d')
            subFolder = f"{__name__}_Session_{date}"
            subFolderPath = f"{path}/logs/{subFolder}"
            isExist = os.path.exists(subFolderPath)
            if not isExist:
                os.makedirs(subFolderPath)
            filename = f"{subFolderPath}/{__name__}_{now}.txt"
        else:
            filename = f"{path}/logs/{__name__}.txt"
        self.initLogger(filename)

    async def worker(self):
        while True:
            try:
                item = await self.Queue.get()

                job  = item[0]; data = item[1]

                if job == "serial-data":
                    if "$Packet" in data:
                        self.handleSoil(data)

                if job == "ws-receive":
                    self.logger.info(f"[WS-Data]: Data:{data['msg']} Client:{data['client']} ")
```

```
            # if job == "ws-send":
            #     self.logger.info(f"[WS-Send]: Data:{data['msg']} to Client:{data['client']} ")

            # if job == "ws-broadcast":
            #     self.logger.info("[WS-Data]: Data:{data}")


            self.Queue.task_done()
        except KeyError as e:
            print(e)

    def main(self):
        try:
            self.startLogger()

            #myDB.__init__(self)

            myDB.__init__(self , self.logger)
            self.Queue       = myQueue()
            self.Serial      = mySerial     (self.Queue,self.logger)
            self.Websockets  = myWebsockets (self.Queue,self.logger)

            startSerial      = self.loop.create_task(  self.Serial.start()  )
            startWebsockets  = self.loop.create_task(  self.Websockets.start()        )
            startWorker      = self.loop.create_task(  self.worker()                  )

            self.Websockets.onLoadWsCbk = self.onLoadWsCbk


            #self.loop.create_task(self.dummy())

            self.logger.info('[Main]Services runnning.')
            if self.SESSION_LOGGING: self.logger.info('[Main]This script is using
SESSION_LOGGING')

            try:
                self.loop.run_forever()
            except Exception as e:
                print(f'[myMain] Error {e}')
            finally:
                print(f'[myMain] Error Finally')
                self.loop.close()

        except Exception as e:
            self.logger.error(e)


    def onLoadWsCbk(self,websocket):
        try:
            history = self.getJson('soil')
            for x in history:
                x["last_updated"] = x["last_updated"].strftime('%Y-%m-%d %H:%M:%S')
            self.sendWsAllSoil(websocket,history)
        except Exception as e:
            self.logger.error(e)
```

```
    def sendWsAllSoil(self,websocket,packet):
        wsMsg = {
            "type": "all-markers",
            "payload": packet
        }

asyncio.run_coroutine_threadsafe(self.Websockets.send_client(websocket,json.dumps(wsMsg)),self.lo
op)

    def sendWsLiveSoil(self,packet,type):
        wsMsg = {
            "type": f"{type}-marker",
            "payload": packet
        }
        asyncio.run_coroutine_threadsafe(self.Websockets.broadcast(json.dumps(wsMsg)),self.loop)

    def handleSoil(self,data):

        extracted = data.split(',')
        GPS_NOISE = float(10e-4)
        history = self.getJson('soil')
        findSimilar = False
        sLang = ''
        sLong = ''
        typeEvent = 'update'
        packet = {}

        for x in history:

            sLang = x["latitude"]
            curLang = extracted[1]

            sLong = x["longtitude"]
            curLong = extracted[2]

            diffLang = abs(float(sLang)-float(curLang))
            diffLong = abs(float(sLong)-float(curLong))

            #print(f"[Diff in Lang]:{'{:.2E}'.format(diffLang)}")
            #print(f"[Diff in Long]:{'{:.2E}'.format(diffLong)}")

            if diffLang < GPS_NOISE and diffLong < GPS_NOISE :
                findSimilar = True
                break

        #$Packet,41.350586,26.493124,17.10,73.90,591,7.30,42,57,143,325,295
        #Soil conductivity depends on the Moisture or water in the Soil and the Minerals (N,P,K)
        #https://www.agric.wa.gov.au/soil-salinity/measuring-soil-salinity
        # Measuring EC at less than 25°C underestimates salinity; measuring EC above 25°C
overestimates salinity. The equation to correct to the 25°C standard is:
        # EC at 25°C = EC of sample ÷ (1 + (0.02 × (temperature of sample °C – 25))).
        # TDS can expressed as parts per million (ppm), milligrams per litre (mg/L), and molarity
(millimoles per litre, mmol/L).
```

```
        try:
            measurements =  {
                "Temperature": f"Temperature:{extracted[3]} °C",
                "Moisture": f"Moisture:{extracted[4]} %",
                "Conductivity": f"Conductivity:{extracted[5]} us/cm",
                "PH": f"PH:{extracted[6]} PH",
                "Nitrogen": f"Nitrogen:{extracted[7]} mg/kg",
                "Phosphorus": f"Phosphorus:{extracted[8]} mg/kg",
                "Potassium": f"Potassium:{extracted[9]} mg/kg",
                "Salinity": f"Salinity:{extracted[10]} ppt",
                "TDS": f"TDS:{extracted[11]} ppm"
            }
            packet = {
                "latitude":extracted[1],
                "longtitude":extracted[2],
                "data":json.dumps(measurements),
                "last_updated":datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
                "description":"Sample"
            }
            if not findSimilar:
                self.insertJson("measurements",packet)
                typeEvent = 'insert'
            else:
                packet["latitude"] = sLang
                packet["longtitude"] = sLong
                try:
                    typeEvent = 'update'
                    uniqueKeys = {
                        "latitude":sLang,
                        "longtitude":sLong
                    }
                    self.updateJson("measurements",packet,uniqueKeys)
                    print(packet)
                except Exception as e:
                    self.logger.error(e)

            self.sendWsLiveSoil(packet,typeEvent)

        except Exception as e:
            self.logger.error(f"[Error] {e}")

if __name__ == "__main__":
    try:
        loop = asyncio.get_event_loop()
        service = myMain(loop)
        service.main()
    except Exception as e:
        print(e)
```

ΠΑΡΑΡΤΗΜΑ Δ : Κώδικας Javascript

```
var markers = []
var map;

function initMap() {
  const center = { lat: 40.350, lng: 26.493 };
  map = new google.maps.Map(document.getElementById("map"), {
    zoom: 9,
    center,
  });
}

function makeMeasurementsText(lat,lng,packet,description,last_update) {

    const target = (`M-${lat}-${lng}`).replace(/\./g,'-');
    elem = `
        <h3 class='m-header'>Latitude:${lat}<br>Longtitude:${lng}</h3>

        <div class='${target} measurements'>
            ${updateMeasurementsText(packet,lat,lng)}
        </div>
        <div class="last-updated" >Last Updated:<span class="${target}
lastupdated">${last_update}</span></div>


        <br>
        <div >Descrption:<span contenteditable="true" >${description}</span></div>
    `
    return elem;

}

function updateMeasurementsText(packet,latitude,longtitude) {

  var cur;
  for (var i=0;i<markers.length;i++){
        const lat = markers[i].getPosition().lat();
        const lng = markers[i].getPosition().lng();
        if ( latitude == lat && longtitude == lng ) {
            cur = i;
        }
  }
  try {
    markers[cur].setAnimation(google.maps.Animation.BOUNCE);
    clearInterval(markers[cur].clearFutureAnimation);
    markers[cur].clearFutureAnimation = setTimeout ( ()=> {
        markers[cur].setAnimation(null);
    } , 4000);
  } catch (error) {
    console.log(error);
  }
```

```
  data = JSON.parse(packet);

  const elem = `
   <div>
    <div>${data.PH}</div>
    <div>${data.Nitrogen}</div>
    <div>${data.Phosphorus}</div>
    <div>${data.Potassium}</div>
    <div>${data.Conductivity}</div>
    <div>${data.Salinity}</div>
    <div>${data.TDS}</div>
    <div>${data.Temperature.replace("u00b0C","&#x2103;")}</div>
    <div>${data.Moisture}</div>
   </div>
  `;

  return elem;

}

function getDatetime() {
    var date = new Date();
    var dateStr =
      ("00" + (date.getMonth() + 1)).slice(-2) + "/" +
      ("00" + date.getDate()).slice(-2) + "/" +
      date.getFullYear() + " " +
      ("00" + date.getHours()).slice(-2) + ":" +
      ("00" + date.getMinutes()).slice(-2) + ":" +
      ("00" + date.getSeconds()).slice(-2);
    return dateStr;
}

function clearMarkers () {
    for (let i = 0; i < markers.length; i++) {
        markers[i].setMap(null);
      }
    markers = [];
}

function addMarker(position,text) {

  const infowindow = new google.maps.InfoWindow({
    content: text,
    maxWidth: 350,
  });
  const marker = new google.maps.Marker({
    position: position,
    map,
  });

  markers.push(marker);

  marker.addListener("click", () => {
    infowindow.open({
```

```
    anchor: marker,
    map,
  });
});


}

window.initMap = initMap;


var websock;
var websockConnected=false;
var path;
var clientID  = Math.random().toString(36).substring(7);
var connectionRetries=0;
var myConnnection=0;

function handleDisconnect() {
	console.log("Reload Page");
	setTimeout( () => {
		location.reload();
	}, 3000)
}
function handleMessages(msg) {
	try {
		var data = JSON.parse(msg);
		console.log(data);
		if (data.type=="all-markers") {
			clearMarkers();
			$(data.payload).each( (i,v) => {
				const marker = {
					lat:parseFloat(v.latitude),
					lng:parseFloat(v.longtitude),
				}
				addMarker(marker, makeMeasurementsText( v.latitude, v.longtitude,
v.data , v.description , v.last_updated));
			})
		}

		if (data.type=="insert-marker") {
			const v = data.payload;
			const marker = {
				lat:parseFloat(v.latitude),
				lng:parseFloat(v.longtitude),
			}
			addMarker(marker, makeMeasurementsText( v.latitude, v.longtitude, v.data ,
v.description , v.last_updated));
		}
		if (data.type=="update-marker") {
			const v = data.payload;

			//console.log(v)

			for (var i=0;i<markers.length;i++){
```

```
                                const lat = markers[i].getPosition().lat();
                                const lng = markers[i].getPosition().lng();

                                console.log(`[Marker] Who:${i} Lat:${lat} Long:${lng}
Vlatitutde:${v.latitude} Vlongtitude:${v.longtitude}`)

                                if ( parseFloat(v.latitude) == lat && parseFloat(v.longtitude) == lng )
{
                                        const target = (`M-${lat}-${lng}`).replace(/\./g,'-');
                                        console.log(`Parsed:${target}`)
                                        $(`.${target}.measurements`).html(
updateMeasurementsText(v.data,lat,lng) );
                                        $(`.${target}.lastupdated`).html( getDatetime() );
                                        break;
                                }
                        }
                }

        }
        catch (error) {
                console.log(msg);
                console.log(error);
        }
}

function start() {
        console.log("Recreating connection...");
        if (connectionRetries>=5){
                connectionRetries=0;
                handleDisconnect();
        }
        path = "";
        if (window.location.port != "" || window.location.port != 80 || window.location.port != 443) {
                path = "ws://" + window.location.hostname + ":" + window.location.port + "/ws";
        } else {
                path = "ws://" + window.location.hostname + "/ws";
        }
        try {
                ip = '192.168.1.100:6789'
                path = `ws://${ip}/ws`;
                websock = new WebSocket(path);
                websock.onopen = function(evt) {
                        connectionRetries=0;
                        clearInterval(myConnnection);
                        console.log("Websock open");
                        websockConnected = true;
                };
                websock.onclose = function(evt) {
                        console.log("Trying to reconnect");
                        myConnnection = setTimeout(function() {
                                start();
                        }, 1000);
                };
                websock.onerror = function(evt) {
```

```
                        console.log(evt);
                        handleDisconnect();
                }
                websock.onmessage = function(evt) {
                        console.log(evt.data);
                        handleMessages(evt.data);
                }
        }
        catch (error) {
                console.log(error);
        }
}

$(document).ready(function() {
        console.log("Ready!");
        setTimeout(() => {
                start();
        }, 50);
});
```