ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη Αυτόνομου Drone με Σύστημα Αισθητήρων και Εφαρμογή Android για Γεωργία Ακριβείας»

Φοιτητής

Λιολιόπουλος Κωνσταντίνος 515304

Επιβλέπων

Δρ. Κυριάκος Τσιακμάκης

Σεπτέμβριος 2025

Ανάπτυξη Αυτόνομου Drone με Σύστημα Αισθητήρων και Εφαρμογή Android για Γεωργία Ακριβείας

Κωδικός: 25144

Φοιτητής: Λιολιόπουλος Κωνσταντίνος

Εισηγητής: Δρ Κυριάκος Τσιακμάκης

Ημερομηνία ανάληψης Π.Ε. 06-03-2025

Ημερομηνία περάτωσης Π.Ε. 09-09-2025

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 πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Περίληψη

Η παρούσα πτυχιακή εργασία αφορά την ανάπτυξη και υλοποίηση ενός αυτόνομου τετρακόπτερου drone, σχεδιασμένου για εφαρμογές γεωργίας ακριβείας. Το drone είναι εξοπλισμένο με ενσωματωμένο σύστημα αισθητήρων, το οποίο συλλέγει δεδομένα θερμοκρασίας, υγρασίας, έντασης φωτός, ακτινοβολίας UV και βαρομετρικής πίεσης από προκαθορισμένα σημεία (waypoints). Τα δεδομένα αυτά, χρήσιμα για την ανάλυση της αποδοτικότητας των καλλιεργειών, καταγράφονται και αποθηκεύονται τοπικά, ενώ μεταφέρονται ασύρματα σε μια εφαρμογή Android, που παρέχει στον χρήστη μια εύχρηστη διεπαφή για την προβολή και διαχείρισή τους.

Ο χρήστης έχει τη δυνατότητα να προγραμματίζει προκαθορισμένες διαδρομές (waypoint mission), τις οποίες το drone ακολουθεί αυτόνομα, εκτελώντας μετρήσεις σε όλη την πορεία της αποστολής. Το drone είναι σχεδιασμένο να καλύπτει μεγάλες αποστάσεις αυτόνομα, ενσωματώνοντας δικλείδες ασφαλείας που εξασφαλίζουν την αξιοπιστία και τη σταθερότητα της λειτουργίας του κατά τη διάρκεια πτήσεων μεγάλης εμβέλειας. Για την επικοινωνία μεταξύ του drone και του συστήματος αισθητήρων, χρησιμοποιείται το πρωτόκολλο MAVLink, επιτρέποντας τη μετάδοση πληροφοριών σε πραγματικό χρόνο. Η υλοποίηση του ενσωματωμένου συστήματος βασίζεται σε έναν ESP32, ο οποίος διαχειρίζεται τους αισθητήρες, συλλέγει τα δεδομένα και τα αποστέλλει σε μία εφαρμογή Android, από όπου ο χρήστης μπορεί να τα διαχειριστεί.

Τα αποτελέσματα της εργασίας δείχνουν ότι το σύστημα μπορεί να συλλέξει και να μεταδώσει αξιόπιστα περιβαλλοντικά δεδομένα, συμβάλλοντας στη βελτίωση της παρακολούθησης των αγροτικών εκτάσεων. Η επιτυχής ολοκλήρωση της εργασίας αποδεικνύει ότι τέτοιες τεχνολογίες μπορούν να ενισχύσουν τη γεωργία ακριβείας, προσφέροντας στους αγρότες χρήσιμες πληροφορίες για τη βέλτιστη διαχείριση των καλλιεργειών.

«Development of an Autonomous Drone with Sensor System and Android Application for Precision Agriculture»

Abstract

This thesis focuses on the development and implementation of an autonomous quadcopter drone designed for precision agriculture applications. The drone is equipped with an integrated sensor system that collects data on temperature, humidity, ambient light, UV radiation, and barometric pressure from predefined waypoints. This data, valuable for analyzing crop performance, is recorded and stored locally before being wirelessly transmitted to an Android application, providing the user with an intuitive interface to view and manage the data.

The user can program predefined routes (waypoints), which the drone autonomously follows to perform measurements at specific locations. The drone is designed for long-range autonomous flights and incorporates safety mechanisms to ensure reliability and stability during extended operations. Communication between the drone and the sensor system is achieved using the MAVLink protocol, allowing real-time data transmission. The integrated system is based on an ESP32, which manages the sensors, collects the data, and sends it to an Android application, where the user can manage and view the data.

The results of the thesis demonstrate that the system is capable of reliably collecting and transmitting environmental data, contributing to improved monitoring of agricultural areas. The successful completion of this project shows that such technologies can enhance precision agriculture, providing farmers with valuable insights for optimal crop management.

Ευχαριστίες

Ευχαριστώ θερμά τον επιβλέποντα καθηγητή μου Τσιακμάκη Κυριάκο για την πολύτιμη καθοδήγηση, την υποστήριξη και τις χρήσιμες παρατηρήσεις καθ' όλη τη διάρκεια της εκπόνησης της παρούσας εργασίας. Επίσης, θα ήθελα να ευχαριστήσω τους συναδέλφους και φίλους μου για την πολύτιμη βοήθεια και τις συζητήσεις που είχαν ως αποτέλεσμα το κίνητρο και την όρεξη για την ανάπτυξη του έργου. Ευχαριστώ την οικογένειά μου για την αμέριστη υποστήριξη και την κατανόηση σε όλη τη διάρκεια των σπουδών μου. Τέλος, ευχαριστώ όλους όσους με ενθάρρυναν και συνέβαλαν στην ολοκλήρωση αυτής της προσπάθειας.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
6S1P	6 Serial 1 Parallel
BLS	BLheli_S
CAN	Controller Area Network
CWSI	Crop Water Stress Index
ESC	Electronic Speed Controller
FC	Flight Controller
GNDVI	Green Normalized Difference Vegetation Index
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HTTP	Hypertext Transfer Protocol
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
LOS	Line-of-Sight
LTE	Long Term Evolution
MSP	MultiWii Serial Protocol
MQTT	Message Queuing Telemetry Transport
NDVI	Normalized Difference Vegetation Index
NIR	Near Infrared
PPK	Post-Processed Kinematic
RTH	Return-to-Home
RTK	Real-Time Kinematic
SBUS	Serial Bus

UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UART	Universal Asynchronous Receiver-Transmitter
UV	Ultraviolet

Κεφάλαιο 1ο: Εισαγωγή

1.1 Γενική Επισκόπηση

Η ραγδαία ανάπτυξη της τεχνολογίας των μη επανδρωμένων αεροσκαφών (UAVs) ή drones έχει επιφέρει σημαντικές αλλαγές σε πλήθος τομέων, όπως η γεωργία, η επιτήρηση, οι μεταφορές προϊόντων, η παρακολούθηση περιβαλλοντικών δεδομένων και η κινηματογράφηση. Τα drones έχουν την ικανότητα να εκτελούν αυτόνομες πτήσεις και να συλλέγουν δεδομένα χωρίς την ανάγκη ανθρώπινης παρέμβασης, κάτι που τα καθιστά ιδανικά για αποστολές σε δύσβατες ή επικίνδυνες περιοχές. Στον τομέα της γεωργίας ακριβείας, συμβάλλουν στην αποδοτικότητα και στην ακρίβεια των μετρήσεων πάνω από καλλιέργειες, παρέχοντας στους αγρότες πολύτιμες πληροφορίες σχετικά με την κατάσταση των γεωργικών εκτάσεών τους [1].

Τα τελευταία χρόνια έχουν αναπτυχθεί συστήματα αισθητήρων που συνδυάζουν τη χρήση εφαρμογών και επικοινωνούν με τα drones προσφέροντας νέες δυνατότητες για την ακριβή παρακολούθηση των περιβαλλοντικών συνθηκών, την ανάλυση δεδομένων σε πραγματικό χρόνο και την υποστήριξη της λήψης αποφάσεων. Η συλλογή μετρήσεων, όπως θερμοκρασίας, υγρασίας και UV ακτινοβολίας σε συνδυασμό με τις δυνατότητες ασύρματης μετάδοσης, διευκολύνει την αποδοτική παρακολούθηση των καλλιεργειών, τη βελτιστοποίηση της άρδευσης και τη διατήρηση της παραγωγικότητας [2].

Η παρούσα διπλωματική εργασία εστιάζει στην ανάπτυξη και υλοποίηση ενός αυτόνομου συστήματος drone που ενσωματώνει αισθητήρες για τη συλλογή δεδομένων και την μετάδοσή τους σε μία εφαρμογή Android. Η εργασία έχει ως στόχο να προσφέρει μια ευρύτερη κατανόηση της σημασίας των UAVs στη γεωργία ακριβείας και να δείξει πώς η αξιοποίηση σύγχρονων αισθητήρων και λογισμικού μπορεί να βελτιώσει τη διαχείριση και την αποδοτικότητα των καλλιεργειών.

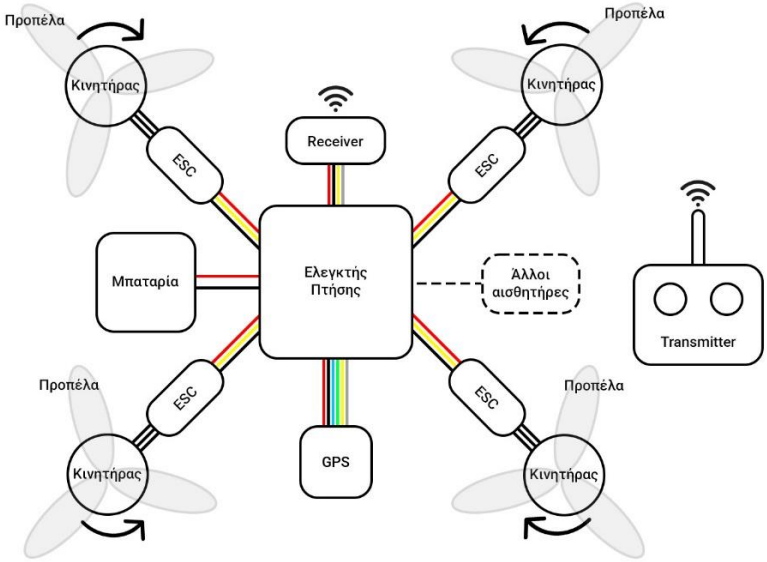
1.2 Drones, Ενσωμάτωση και Εφαρμογές

1.2.1 Δομή και Λειτουργία των Drones

Τα μη επανδρωμένα αεροσκάφη αποτελούν συστήματα που ενσωματώνουν μια ποικιλία αισθητήρων, προκειμένου να εξασφαλίζεται η ασφαλής και αποδοτική πτήση τους. Ανάλογα με την εφαρμογή τους, το μέγεθός τους ποικίλλει, από μικρότερα συστήματα που προορίζονται για επέμβαση σε δύσβατες περιοχές ή παρακολούθηση, έως μεγαλύτερα αεροσκάφη που χρησιμοποιούνται για τη μεταφορά βαρέων αντικειμένων, όπως δεξαμενές, επαγγελματικές κάμερες ή δέματα αποστολής. Παρά τις διαφορές στο μέγεθος και την εφαρμογή τους, όλα τα drones μοιράζονται κάποια κοινά συστήματα, τα οποία είναι απαραίτητα για τη σωστή λειτουργία τους και την επίτευξη ασφαλών πτήσεων.

Τα βασικά μέρη ενός drone είναι τα εξής:

1. **Ελεγκτής Πτήσης (Flight Controller)**: Είναι το κύριο υποσύστημα που διαχειρίζεται την πτήση του drone. Συντονίζει όλους τους αισθητήρες και τους κινητήρες, καθοδηγώντας το drone μέσω αλγορίθμων πλοήγησης. Συνήθως ενσωματώνει δικό του IMU σύστημα (Inertial Measurement Unit) με γυροσκόπιο, βαρόμετρο, επιταχυνσιόμετρο και σπανιότερα μαγνητόμετρο.
2. **Ηλεκτρονικοί Ελεγκτές Ταχύτητας (ESCs)**: Οι ESCs είναι υπεύθυνοι για τη διαχείριση της ταχύτητας των κινητήρων του drone. Ρυθμίζουν τη ροή του ρεύματος στους κινητήρες, επιτρέποντας την ακριβή ρύθμιση της ισχύος και της ταχύτητας των προπελών. Εξασφαλίζουν τη σωστή απόκριση του drone σε εντολές ελέγχου, όπως η επιτάχυνση, η επιβράδυνση ή η αλλαγή κατεύθυνσης.
3. **Κινητήρες Μόνιμου Μαγνήτη (Brushless Motors)**: Οι κινητήρες είναι υπεύθυνοι για την κίνηση και την ισχύ του drone. Καθορίζουν την ανύψωση και την κατεύθυνση του αεροσκάφους, ενώ συνεργάζονται με τις προπέλες για να παρέχουν τη δύναμη που απαιτείται για την πτήση.
4. **Προπέλες**: Οι προπέλες συνεργάζονται με τους κινητήρες για να παράγουν την αναγκαία ανύψωση και ώθηση, επιτρέποντας στο drone να πετάξει και να ελιχθεί στον αέρα.
5. **Μπαταρία**: Η μπαταρία παρέχει την απαιτούμενη ενέργεια για τη λειτουργία όλων των επιμέρους συστημάτων του drone.
6. **Σύστημα Επικοινωνίας (Transmitter/Receiver)**: Το σύστημα επικοινωνίας επιτρέπει τη σύνδεση του drone με το χειριστήριο ή με άλλες συσκευές για τον έλεγχο και την παρακολούθηση των δεδομένων του drone σε πραγματικό χρόνο.
7. **Σύστημα Πλοήγησης GPS**: Χρησιμοποιείται για την πλοήγηση του drone σε αυτόνομα συστήματα, καθώς και για τη σταθεροποίηση της πτήσης.
8. **Αισθητήρες**: Ανάλογα με τη χρήση του, ένα drone μπορεί να φέρει επιπλέον αισθητήρες, όπως κάμερες για βιντεοσκόπηση, αισθητήρες απόστασης για αποφυγή εμποδίων, γυροσκόπια και βαρομετρικούς αισθητήρες.
9. **Λογισμικό και Αλγόριθμοι Πλοήγησης (Flight Software)**: Το λογισμικό που ελέγχει τον τρόπο λειτουργίας του drone. Αυτό περιλαμβάνει τα συστήματα πλοήγησης, τους αλγόριθμους για την πλοήγηση σε προκαθορισμένα waypoints, καθώς και τα συστήματα ασφαλείας για την αποφυγή εμποδίων και την ασφαλή προσγείωση.



Εικόνα 1.1: Βασικά μέρη ενός drone

Η σωστή λειτουργία ενός drone βασίζεται στην αποδοτική επικοινωνία μεταξύ των υποσυστημάτων του. Ο flight controller διαχειρίζεται δεδομένα τόσο από αισθητήρες που ενσωματώνει εσωτερικά (IMU) όσο και από περιφερειακά υποσυστήματα με ενσύρματη ή ασύρματη επικοινωνία μέσω διαφόρων πρωτοκόλλων. Αυτή η επικοινωνία επιτρέπει στο drone να λαμβάνει δεδομένα από το περιβάλλον του, να προσαρμόζει την πτήση του και να εκτελεί αποστολές με ακρίβεια.

Ο flight controller χρησιμοποιεί ελεγκτές PID για να σταθεροποιήσει το drone και να διατηρήσει την ισορροπία του. Οι ελεγκτές PID αποτελούν τον πιο διαδεδομένο τρόπο ελέγχου σε συστήματα UAV , συλλέγοντας δεδομένα από αισθητήρες και υποσυστήματα (κινητήρες, γυροσκόπιο, βαρόμετρο κ.α.) για να καθορίσουν αν το σύστημα αποκλίνει από την επιθυμητή του θέση και προσαρμόζουν την ταχύτητα των κινητήρων μέσω των ESCs, διορθώνοντας το σφάλμα στην κίνηση. Οι ελεγκτές αυτοί χρησιμοποιούν τρία στοιχεία, το P (proportional), το I (integral) και το D (derivative), η ρύθμιση των οποίων είναι κρίσιμη για την επίτευξη της ομαλής πτήσης [3].

Η ροή δεδομένων μεταξύ των βασικών μερών του drone περιλαμβάνει τα εξής στάδια:

- **IMU προς Ελεγκτή Πτήσης:** Οι αισθητήρες, όπως το γυροσκόπιο, το επιταχυνσιόμετρο και το βαρόμετρο, παρέχουν δεδομένα στο flight controller μέσω των διαύλων I2C, SPI ή UART. Ο ελεγκτής πτήσης χρησιμοποιεί αυτά τα δεδομένα για να σταθεροποιήσει το drone και να υπολογίσει τις απαραίτητες διορθώσεις πτήσης.
- **Ελεγκτής Πτήσης προς Ηλεκτρονικούς Ελεγκτές Ταχύτητας (ESCs):** Ο flight controller, αφού επεξεργαστεί τις πληροφορίες από τους αισθητήρες και το χειριστήριο, στέλνει σήματα

στους ESCs μέσω των πρωτοκόλλων PWM (Pulse Width Modulation), OneShot, DShot, ή ProShot, καθορίζοντας την ταχύτητα περιστροφής κάθε κινητήρα.

- **GPS και Σύστημα Πλοήγησης:** Το GPS στέλνει δεδομένα θέσης στον FC μέσω UART ή CAN bus σε πιο προηγμένα συστήματα. Ο controller χρησιμοποιεί αυτά τα δεδομένα για τον καθορισμό συντεταγμένων πτήσης και την υλοποίηση αυτόνομων αποστολών.
- **Σύστημα Επικοινωνίας και Τηλεμετρία:** Ο δέκτης (Transmitter/Receiver) που ενσωματώνεται στο σύστημα του drone επιτρέπει τη σύνδεση του χειριστηρίου (Transmitter) με το drone μέσω πρωτοκόλλων όπως το CRSF (Crossfire), SBUS ή IBUS και χρησιμοποιείται για τον χειροκίνητο έλεγχο, ενώ παράλληλα υποστηρίζει τη μετάδοση δεδομένων μέσω τηλεμετρίας. Η τηλεμετρία επιτρέπει την αποστολή πληροφοριών σε πραγματικό χρόνο από το drone προς τον χειριστή μέσω πρωτοκόλλων όπως το MAVLink και το MSP, διευκολύνοντας την παρακολούθηση της πτήσης και των αισθητήρων μέσω εφαρμογών όπως το Mission Planner ή το QGroundControl.

1.2.2 Λογισμικά Πτήσης

Τα λογισμικά πτήσης είναι προγράμματα που ελέγχουν κάθε πτυχή της πτήσης και εξασφαλίζουν ότι το drone εκτελεί τις εντολές με ακρίβεια και ασφάλεια. Αυτά τα λογισμικά υποστηρίζονται από διάφορους κατασκευαστές ελεγκτών πτήσης, κάνοντας την παραμετροποίηση των drone πιο φιλική προς τον χρήστη. Συντονίζουν όλους τους αισθητήρες και τις συσκευές του drone, όπως τους κινητήρες, το GPS και τα συστήματα τηλεμετρίας.

Ορισμένα από τα πιο γνωστά λογισμικά πτήσης που χρησιμοποιούνται για τη λειτουργία των drones είναι τα εξής:

1. **Betaflight**: Πρόκειται για ένα από τα πιο δημοφιλή λογισμικά πτήσης, που χρησιμοποιείται κυρίως σε μικρότερα αγωνιστικά drones. Εστιάζει σε αγωνιστικά drone, με έμφαση στην ακρίβεια και στην γρήγορη απόκριση του ελέγχου, επιτρέποντας μεγάλη παραμετροποίηση στον PID controller του ελεγκτή πτήσης και τη χρήση προηγμένων λειτουργιών σταθεροποίησης [4].
2. **ArduPilot**: Είναι ένα από τα πρώτα λογισμικά ανοιχτού κώδικα που υποστηρίζει μια μεγάλη γκάμα από drones, από quadcopters μέχρι αεροπλάνα και υδροπλάνα. Παρέχει δυνατότητες αυτόνομης πτήσης, επιτρέποντας στα drones να ακολουθούν προκαθορισμένες διαδρομές και να εκτελούν αποστολές αυτόνομα χωρίς ανθρώπινη παρέμβαση [4].
3. **INAV**: Το INAV συνδυάζει χαρακτηριστικά από το Betaflight και το ArduPilot, δίνοντας έμφαση στην πλοήγηση μέσω GPS και δυνατότητα αυτόνομων αποστολών. Είναι ιδανικό για εφαρμογές που χρειάζεται σταθερότητα και ακρίβεια στον έλεγχο κατά τη διάρκεια της πτήσης

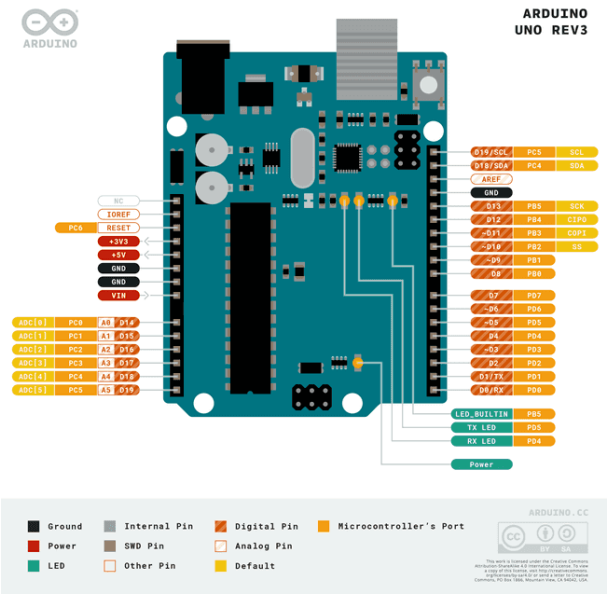
[4]. Ενσωματώνει επίσης αρκετούς σύγχρονους μηχανισμούς ασφαλείας (failsafe) βελτιώνοντας την συνολική ασφάλεια της πτήσης.

Η επιλογή του λογισμικού πτήσης επηρεάζεται από τη συμβατότητα με τον ελεγκτή πτήσης, καθώς το λογισμικό πρέπει να υποστηρίζει τη συγκεκριμένη αρχιτεκτονική και τις δυνατότητες του ελεγκτή πτήσης. Επιπλέον, ο τύπος και η εφαρμογή του drone (π.χ. γεωργία ακριβείας, φωτογράφιση ή επιτήρηση) επηρεάζουν την επιλογή, καθώς κάποια λογισμικά προσφέρουν εξειδικευμένες δυνατότητες όπως υποστήριξη GPS, RTK, ή κάμερες. Άλλοι παράγοντες περιλαμβάνουν την υποστήριξη για ενσωμάτωση αισθητήρων, το κόστος καθώς και την ευχρηστία του συστήματος παραμετροποίησης και της διεπαφής χρήστη.

1.2.3 Ενσωμάτωση Συστημάτων Αισθητήρων και Διαχείριση Δεδομένων

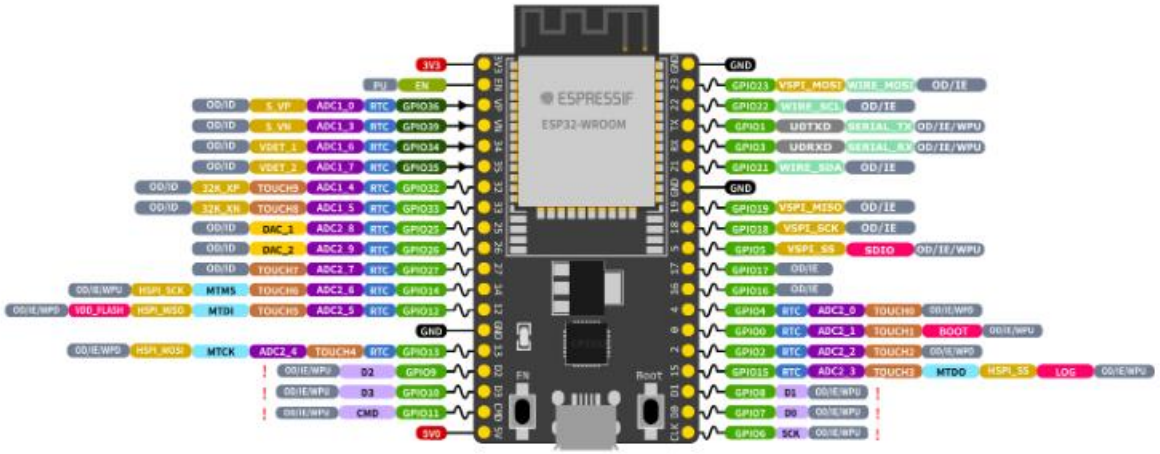
Στα drones, η ενσωμάτωση συστημάτων αισθητήρων μπορεί να επιτευχθεί μέσω μικροελεγκτών όπως ο ESP32, το Arduino και το Raspberry Pi, τα οποία προσφέρουν ευελιξία και αποτελεσματικότητα στη συλλογή και επεξεργασία δεδομένων. Αυτοί οι μικροελεγκτές, χρησιμοποιούνται για τη σύνδεση με διάφορους αισθητήρες, όπως αισθητήρες θερμοκρασίας, υγρασίας, απόστασης, ή ακόμα και κάμερες και άλλους εξειδικευμένους αισθητήρες, ανάλογα με τις απαιτήσεις της εφαρμογής.

Ο Arduino είναι ένας ευρέως χρησιμοποιούμενος μικροελεγκτής για έργα που απαιτούν απλή και οικονομική ενσωμάτωση αισθητήρων, λόγω των απλών του εισόδων/εξόδων και της ευρείας υποστήριξης από βιβλιοθήκες.



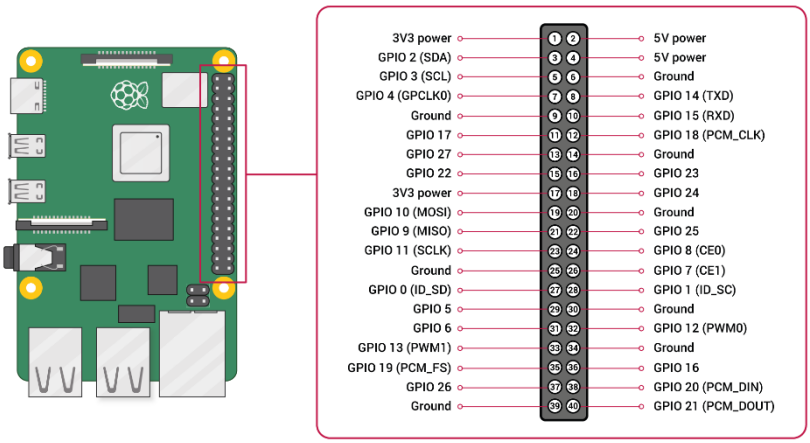
Εικόνα 1.2: Το pin layout της πλακέτας ανάπτυξης Arduino Uno 3
[https://docs.arduino.cc/static/6ec5e4c2a6c0e9e46389d4f6dc924073/a6d36/Pinout-UNOrev3_latest.png]

Ο ESP32 είναι ένας μικροελεγκτής της Espressif Systems που παρέχει μεγαλύτερη υπολογιστική ισχύ και υποστήριξη για Wi-Fi – Bluetooth, καθιστώντας το ιδανικό για εφαρμογές που απαιτούν ασύρματη επικοινωνία με τον ελεγκτή πτήσης ή με άλλες συσκευές.



Εικόνα 1.3: Το pin layout της πλακέτας ανάπτυξης Espressif ESP32-DevKitC
[https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp-dev-kits-en-master-esp32.pdf#page=6.17]

Ο μικροϋπολογιστής Raspberry Pi, με τη δυνατότητα εκτέλεσης πλήρους λειτουργικού συστήματος, είναι κατάλληλος για πιο απαιτητικές εφαρμογές που απαιτούν μεγαλύτερη υπολογιστική ισχύ και αποθήκευση δεδομένων, όπως η επεξεργασία εικόνας ή η ανάλυση μεγάλου όγκου δεδομένων.



Εικόνα 1.4: Το pin layout του μικροϋπολογιστή Raspberry Pi 5
[https://www.raspberrypi.com/documentation/computers/images/GPIO-Pinout-Diagram-2.png?hash=df7d7847c57a1ca6d5b2617695de6d46]

Η διαχείριση των δεδομένων από το drone, όπως τα δεδομένα από αισθητήρες ή η κατάσταση της πτήσης, μπορεί να πραγματοποιηθεί μέσω του κώδικα που τρέχει στον μικροελεγκτή. Ο μικροελεγκτής μπορεί να συλλέγει δεδομένα από τους αισθητήρες και να τα αποστέλλει είτε στον ελεγκτή πτήσης μέσω επικοινωνίας τύπου UART ή I2C, είτε σε ένα εξωτερικό σύστημα, όπως ένα smartphone ή ένας υπολογιστής μέσω ασύρματης σύνδεσης (Wi-Fi, Bluetooth). Μέσω αυτής της επικοινωνίας, τα δεδομένα μπορούν να παρακολουθούνται σε πραγματικό χρόνο, να αποθηκεύονται για μελλοντική ανάλυση ή να χρησιμοποιούνται για τον έλεγχο της πτήσης, με σκοπό τη βελτίωση της απόδοσης και της ασφάλειας του drone. Επιπλέον, είναι δυνατή η αποθήκευση των δεδομένων τοπικά μέσω ενός SD card module, το οποίο επιτρέπει την αποθήκευση των μετρήσεων του αισθητήρα, καθώς και την καταγραφή πληροφοριών πτήσης για μελλοντική ανάλυση [5][16][21].

1.2.4 Απεικόνιση Δεδομένων μέσω Διεπαφών Χρήστη

Η αξιοποίηση δεδομένων που προέρχονται από ενσωματωμένα συστήματα αισθητήρων στα μη επανδρωμένα εναέρια οχήματα προϋποθέτει την ύπαρξη κατάλληλων μηχανισμών απεικόνισης και αλληλεπίδρασης. Η διεπαφή χρήστη (User Interface – UI) συνιστά το τελικό στάδιο σε μια αλυσίδα που ξεκινά από τη συλλογή δεδομένων μέσω αισθητήρων, ακολουθείται από την επεξεργασία και αποθήκευση των πληροφοριών και ολοκληρώνεται με την παρουσίαση αυτών στον τελικό χρήστη. Η στενή σχέση ανάμεσα στα συστήματα αισθητήρων και στις διεπαφές χρήστη καθιστά την τελευταία απαραίτητη για την ορθή αξιολόγηση των παραμέτρων που παρακολουθούνται, καθώς δίχως κατάλληλη απεικόνιση, τα δεδομένα καθίστανται δυσνόητα ή μη αξιοποιήσιμα στην πράξη.

Οι διεπαφές χρήστη παρέχουν ένα ενδιάμεσο επίπεδο επικοινωνίας μεταξύ του υπολογιστικού συστήματος του drone και του ανθρώπινου παράγοντα, επιτρέποντας την αποδοτική παρουσίαση πληροφοριών με τρόπο κατανοητό και λειτουργικό. Σε αυτό το πλαίσιο, οι UI αναπτύσσονται με σκοπό την ενίσχυση της εμπειρίας χρήσης (User Experience – UX), προσφέροντας εύκολη πρόσβαση σε κρίσιμα δεδομένα, υποστήριξη για εποπτεία πτήσης, καθώς και δυνατότητες απομακρυσμένης ή τοπικής αλληλεπίδρασης [13].

Τεχνολογικά, οι διεπαφές αυτές μπορούν να υλοποιηθούν μέσω:

- Γραφικών περιβαλλόντων εργασίας (Graphical User Interfaces - GUIs) σε επιτραπέζιους ή φορητούς υπολογιστές,
- Εφαρμογών για φορητές συσκευές, όπως tablets και smartphones,
- Περιβαλλόντων βασισμένων στον παγκόσμιο ιστό (web-based dashboards), προσβάσιμων μέσω browser,
- Διαδραστικών γεωχωρικών απεικονίσεων, όπου τα δεδομένα απεικονίζονται πάνω σε χάρτες με βάση τις γεωγραφικές τους συντεταγμένες.

Η απεικόνιση μπορεί να περιλαμβάνει διαγράμματα, χρονικά γραφήματα, χάρτες θερμότητας (heatmaps), καθώς και ειδοποιήσεις για εξαιρετικές τιμές ή συμβάντα. Για παράδειγμα, το Veronte Pipe UI είναι ένα λογισμικό σχεδιασμένο να υποστηρίζει πολλαπλά map types, widgets, virtual joystick και λειτουργεί σε touch συσκευές, αυξάνοντας την ευελιξία του χειριστή [6]. Η σχεδίαση των διεπαφών πρέπει να λαμβάνει υπόψη τόσο την εργονομία όσο και την επιστημονική ακρίβεια, εξισορροπώντας τη σαφήνεια με την πληρότητα της πληροφορίας.

Η ύπαρξη κατάλληλης διεπαφής δεν είναι απλώς επιθυμητή, αλλά αναγκαία, ιδίως σε εφαρμογές όπου η ταχεία ερμηνεία και αντίδραση στα δεδομένα είναι ζωτικής σημασίας, όπως στη γεωργία ακριβείας, στην πολιτική προστασία και στην περιβαλλοντική παρακολούθηση. Μέσω της διεπαφής, τα ακατέργαστα δεδομένα μετατρέπονται σε χρήσιμη πληροφορία, ικανή να υποστηρίξει τη λήψη αποφάσεων σε πραγματικό χρόνο ή εκ των υστέρων.

Ωστόσο, η ανάπτυξη αποτελεσματικών διεπαφών χρήστη δεν είναι απαλλαγμένη προκλήσεων. Βασικά ζητήματα που ανακύπτουν περιλαμβάνουν τη διαχείριση μεγάλου όγκου δεδομένων αισθητήρων, τη διατήρηση της απόκρισης σε πραγματικό χρόνο, τη διασφάλιση της συμβατότητας με πολλαπλές συσκευές και την ανάγκη για παραμετροποιήσιμες και επεκτάσιμες λύσεις. Επιπλέον, σε εφαρμογές όπου ο τελικός χρήστης ενδέχεται να μην έχει τεχνικό υπόβαθρο, είναι απαραίτητο το περιβάλλον να είναι φιλικό και διαισθητικό, αποφεύγοντας περιττή πολυπλοκότητα [14].

Συνοψίζοντας, η διεπαφή χρήστη δεν αποτελεί απλώς ένα εργαλείο απεικόνισης, αλλά έναν κρίσιμο παράγοντα επιτυχίας στην αξιοποίηση UAV συστημάτων με αισθητήρες. Η αποτελεσματική σχεδίαση και υλοποίησή της μπορεί να αναβαθμίσει ουσιαστικά την πληροφοριακή αξία των δεδομένων και να υποστηρίξει την ταχεία και τεκμηριωμένη λήψη αποφάσεων.

1.3 Εφαρμογές των Drones

Τα drones έχουν βρει εφαρμογές σε πλήθος τομέων λόγω της ικανότητάς τους να πραγματοποιούν πτήσεις με ελάχιστη ανθρώπινη παρέμβαση και να συλλέγουν δεδομένα από περιοχές που είναι δύσκολα προσβάσιμες. Οι εφαρμογές των drones εκτείνονται πλέον σε πλήθος βιομηχανικών, επιστημονικών και κοινωνικών πεδίων. Η δυνατότητα ενσωμάτωσης προηγμένων αισθητήρων, η αυτόνομη πλοήγηση, καθώς και η δυνατότητα ανάλυσης και παρουσίασης των συλλεγμένων δεδομένων σε πραγματικό χρόνο μέσω διεπαφών χρήστη, καθιστούν τα drones ένα ισχυρό εργαλείο για την αντιμετώπιση σύγχρονων προκλήσεων.

Οι βασικές κατηγορίες εφαρμογών των drones περιλαμβάνουν:

- **Γεωργία ακριβείας**, με χρήση για επιθεώρηση καλλιεργειών, παρακολούθηση της υγείας των φυτών και λήψη αποφάσεων βασισμένων σε δεδομένα.
- **Επιτήρηση και ασφάλεια**, τόσο σε στρατιωτικό όσο και σε πολιτικό επίπεδο, με χρήση για συνοριακούς ελέγχους, παρακολούθηση πλήθους και καταστάσεις έκτακτης ανάγκης.

- **Αναγνώριση και χαρτογράφηση**, σε τομείς όπως η τοπογραφία, η αρχαιολογία και η πολεοδομία.
- **Ανίχνευση και διαχείριση φυσικών καταστροφών**, όπως πυρκαγιές, πλημμύρες και σεισμοί, μέσω ταχείας εναέριας επιθεώρησης και αποστολής σωστικών ενεργειών.
- **Μεταφορές και logistics**, με τη μεταφορά μικρών φορτίων ή ιατρικών προμηθειών σε απομακρυσμένες περιοχές.
- **Οπτικοακουστική παραγωγή και δημοσιογραφία**, παρέχοντας μοναδικές οπτικές γωνίες για φωτογράφηση και καταγραφή βίντεο.

1.4 Γεωργία Ακριβείας

Η γεωργία ακριβείας χρησιμοποιεί τεχνολογίες για τη βελτιστοποίηση των γεωργικών διαδικασιών και τη μείωση της σπατάλης πόρων. Τα drones, με την ικανότητα να συλλέγουν δεδομένα σε πραγματικό χρόνο από μεγάλες εκτάσεις, διαδραματίζουν καθοριστικό ρόλο στην παρακολούθηση και διαχείριση καλλιεργειών. Ενσωματώνοντας αισθητήρες και GPS, τα drones επιτρέπουν την ακριβή ανάλυση παραμέτρων όπως η υγεία των φυτών, οι ανάγκες σε νερό και η ποιότητα του εδάφους, ενισχύοντας την αποδοτικότητα και βιωσιμότητα της γεωργίας [7].

1.4.1 Παρακολούθηση Καλλιεργειών

Η παρακολούθηση των καλλιεργειών αποτελεί μια από τις σημαντικότερες διαγνωστικές εφαρμογές των UAVs στη γεωργία ακριβείας, καθώς επιτρέπει συστηματική και υψηλής ανάλυσης καταγραφή χωρικοχρονικών μεταβολών στα φυτά και στο έδαφος. Μέσω φορητών πολυφασματικών/υπερφασματικών και θερμικών αισθητήρων, τα UAVs υποστηρίζουν την εκτίμηση δεικτών βλάστησης (π.χ. NDVI, GNDVI) και μετρήσεων θερμοκρασίας, διευκολύνοντας την έγκαιρη ανίχνευση στρεσογόνων παραγόντων όπως τροφοπενίες, ασθένειες ή στρες και επιτρέποντας στοχευμένες παρεμβάσεις άρδευσης και προστασίας [9][10].

Η δυνατότητα σχεδόν σε πραγματικό χρόνο χαρτογράφησης μεγάλων εκτάσεων βελτιώνει τη λήψη αποφάσεων σε επίπεδο αγρού, μειώνοντας σπατάλες σε νερό, λιπάσματα και φυτοφάρμακα μέσω στοχευμένων εφαρμογών και προγραμματισμού εργασιών, ενώ έχει τεκμηριωθεί ότι συνδέεται με αυξήσεις αποδόσεων και βελτίωση της αποδοτικότητας πόρων. Επιπλέον, οι θερμικές λήψεις από UAVs επιτρέπουν την εκτίμηση δεικτών υδατικού στρες (π.χ. CWSI) και της διαπνοής, υποστηρίζοντας βελτιστοποίηση προγραμμάτων άρδευσης και ανθεκτικότητα σε ξηροθερμικές συνθήκες [11].Τέλος, η ενσωμάτωση των δεδομένων UAV με δορυφορικές παρατηρήσεις ή επίγειους αισθητήρες (IoT) ενισχύει τις μακροπρόθεσμες στρατηγικές διαχείρισης, επιτρέποντας μοντελοποίηση τάσεων και αξιολόγηση πρακτικών σε εποχιακές ή πολυετείς κλίμακες [12].

1.4.2 Αισθητήρες και Τεχνολογίες

Στη γεωργία ακριβείας, η αξιοποίηση προηγμένων αισθητήρων και τεχνολογιών είναι καίρια για την ακριβή παρακολούθηση των καλλιεργειών και την αποτελεσματική διαχείριση των γεωργικών δραστηριοτήτων. Οι αισθητήρες που ενσωματώνονται στα drones επιτρέπουν τη λήψη δεδομένων σε πραγματικό χρόνο για διάφορες παραμέτρους, όπως η υγεία των φυτών, η ποιότητα του εδάφους και οι περιβαλλοντικές συνθήκες.

Οι κυριότεροι τύποι αισθητήρων που χρησιμοποιούνται στη γεωργία ακριβείας περιλαμβάνουν:

- Οπτικοί αισθητήρες (RGB, NIR, multispectral): Χρησιμοποιούνται για τη λήψη εικόνας των καλλιεργειών σε διάφορους φασματικούς δείκτες του φωτός (ορατό, υπέρυθρο, και άλλες περιοχές του φάσματος), επιτρέποντας την παρακολούθηση της υγείας των φυτών και την ανίχνευση διαφορών στη χλωροφύλλη, που μπορεί να υποδεικνύουν ασθένειες ή ελλείψεις θρεπτικών στοιχείων. Τα δεδομένα που συλλέγονται μέσω πολυφασματικών εικόνων επιτρέπουν την εκτίμηση της καταπονημένης κατάστασης των καλλιεργειών και τη λήψη μέτρων για τη βελτίωση της υγείας τους [8].
- Αισθητήρες μετρήσεων περιβαλλοντικών δεδομένων: Αυτοί οι αισθητήρες περιλαμβάνουν μια ποικιλία συσκευών για τη μέτρηση της θερμοκρασίας, της υγρασίας, της βαρύτητας του αέρα, της ηλιακής ακτινοβολίας και άλλων παραμέτρων του περιβάλλοντος. Με τη χρήση αυτών των αισθητήρων, τα drones μπορούν να παρέχουν πληροφορίες για τις κλιματικές συνθήκες που επηρεάζουν την ανάπτυξη των φυτών, όπως η σχετική υγρασία και η θερμοκρασία του αέρα. Τα δεδομένα αυτά επιτρέπουν στους γεωργούς να λάβουν πιο ακριβείς αποφάσεις για την άρδευση, την προστασία από τις ασθένειες και τη διαχείριση των φυτών γενικότερα.
- Αισθητήρες ποιότητας αέρα: Αυτοί οι αισθητήρες χρησιμοποιούνται για τη μέτρηση των ατμοσφαιρικών συνθηκών γύρω από τις καλλιέργειες, όπως η παρουσία σωματιδίων ή επικίνδυνων αερίων (π.χ. διοξείδιο του άνθρακα, όζον). Η παρακολούθηση αυτών των παραμέτρων είναι σημαντική για την εκτίμηση της υγείας του οικοσυστήματος γύρω από τις καλλιέργειες και την αντιμετώπιση περιβαλλοντικών πιέσεων.
- Θερμικοί αισθητήρες (IR): Επιτρέπουν την παρακολούθηση των θερμικών χαρακτηριστικών των καλλιεργειών, παρέχοντας πληροφορίες για την υγρασία του εδάφους και τις συνθήκες αρδεύσεως [11].
- Αισθητήρες υγρασίας εδάφους: Χρησιμοποιούνται για τη μέτρηση της υγρασίας του εδάφους, επιτρέποντας στους γεωργούς να βελτιστοποιήσουν τα συστήματα άρδευσης και να εξοικονομήσουν νερό.

Επιπλέον, η τεχνολογία GPS επιτρέπει στους γεωργούς να ακριβώς προσδιορίζουν τη θέση του drone κατά την πτήση, παρέχοντας αξιόπιστα δεδομένα γεωαναφοράς για την παρακολούθηση των καλλιεργειών και τη σχεδίαση στρατηγικών άρδευσης ή λίπανσης.

1.4.3 Συστήματα Πλοήγησης και Ανάλυσης Δεδομένων

Η ακρίβεια στην πλοήγηση και η ικανότητα επεξεργασίας και ανάλυσης των συλλεχθέντων δεδομένων αποτελούν βασικούς πυλώνες της γεωργίας ακριβείας με τη χρήση drones. Ο συνδυασμός προηγμένων συστημάτων πλοήγησης με τεχνολογίες ανάλυσης δεδομένων επιτρέπει τη βελτιστοποίηση των γεωργικών πρακτικών, την αποδοτικότερη χρήση των πόρων και τη μείωση του κόστους παραγωγής.

Τα συστήματα πλοήγησης των drones βασίζονται κυρίως σε δέκτες **GNSS (Global Navigation Satellite Systems)**, όπως GPS, GLONASS, Galileo και BeiDou. Η χρήση **RTK** και **PPK** τεχνολογιών επιτρέπει την επίτευξη εξαιρετικά υψηλής ακρίβειας στη γεωαναφορά των δεδομένων, συχνά σε επίπεδο εκατοστού. Αυτό είναι κρίσιμο για την καταγραφή επαναλαμβανόμενων μετρήσεων στις ίδιες ακριβώς περιοχές και τη δημιουργία χαρτών μεταβλητότητας [17][18].

Παράλληλα, η ανάλυση των δεδομένων που συλλέγονται από τους αισθητήρες γίνεται με τη χρήση ειδικού λογισμικού ή αλγορίθμων υπολογιστικής όρασης και μηχανικής μάθησης. Οι εφαρμογές αυτές μετατρέπουν τις εικόνες και τις μετρήσεις σε χρήσιμες πληροφορίες, όπως χάρτες NDVI, χάρτες υγρασίας, θερμικές απεικονίσεις ή μοντέλα πρόβλεψης απόδοσης. Η ανάλυση μπορεί να γίνει είτε τοπικά σε υπολογιστικά συστήματα, είτε μέσω cloud πλατφορμών, όπου τα δεδομένα ανεβαίνουν αυτόματα και επεξεργάζονται εξ αποστάσεως [9][15].

Ο συνδυασμός πλοήγησης υψηλής ακρίβειας και ανάλυσης δεδομένων επιτρέπει στους γεωργούς να λαμβάνουν τεκμηριωμένες αποφάσεις, να εντοπίζουν εγκαίρως προβλήματα στις καλλιέργειες και να προσαρμόζουν τις γεωργικές παρεμβάσεις με στόχο τη μέγιστη αποδοτικότητα και βιωσιμότητα [15].

1.4.4 DJI Agras T50

Το DJI Agras T50 είναι ένα προηγμένο drone γεωργίας ακριβείας που κατασκευάζεται από την DJI, μια κορυφαία εταιρεία στην ανάπτυξη και κατασκευή drones και τεχνολογιών αεροναυτικής. Η DJI ιδρύθηκε το 2006 και από τότε έχει ηγηθεί στον τομέα των drones, με μια σειρά από εξαιρετικά επιτυχημένα μοντέλα που καλύπτουν ένα φάσμα εφαρμογών, από τη φωτογραφία και βιντεοσκόπηση έως τη γεωργία και τη βιομηχανία [19].

Εικόνα 1.5: Το DJI Agras T50
[https://www.topomarket.gr/8253-large_default/dji-agras-t50-drone.jpg]

Το Agras T50 ανήκει στη σειρά Agras, η οποία ξεκίνησε το 2015 και επικεντρώνεται στη παραγωγή drones ειδικά για γεωργικές εφαρμογές, με δυνατότητες όπως ο ψεκασμός φυτοφαρμάκων σε καλλιέργειες και η παρακολούθηση της υγείας των φυτών. Το Agras T50, το οποίο παρουσιάστηκε το 2021, αποτελεί μια από τις πιο προηγμένες και καινοτόμες προτάσεις στον τομέα της γεωργίας ακριβείας, συνδυάζοντας τεχνολογίες αιχμής και αυτονομία για την αποτελεσματική και ακριβή διαχείριση των γεωργικών εργασιών [20].

Βασικά Χαρακτηριστικά:

- Ακρίβεια και RTK Τεχνολογία: Το DJI Agras T50 εξοπλίζεται με τεχνολογία RTK, επιτρέποντας τη λήψη δεδομένων με εξαιρετική ακρίβεια. Αυτό είναι ιδιαίτερα χρήσιμο για εφαρμογές όπως η ακριβής παρακολούθηση της ανάπτυξης των φυτών και η ακριβής εφαρμογή λιπασμάτων ή φυτοφαρμάκων.
- Αυτονομία και Χωρητικότητα Φορτίου: Το T50 μπορεί να πετάει για περίπου 20-30 λεπτά με πλήρες φορτίο, κάτι που το καθιστά αποτελεσματικό για μεγάλες εκτάσεις καλλιεργειών. Η δυνατότητα μεταφοράς μεγάλων ποσοτήτων ψεκαστικού υγρού ή λιπάσματος επιτρέπει την ταχύτατη κάλυψη μεγάλων γεωργικών περιοχών με ελάχιστο χρόνο εκτέλεσης.
- Αυτόνομοι Έλεγχοι και Σχεδιασμός Πτήσεων: Το drone υποστηρίζει αυτόνομες πτήσεις μέσω του λογισμικού DJI Agras, το οποίο επιτρέπει την προγραμματισμένη διαδρομή του drone και την εκτέλεση εργασιών όπως ο ψεκασμός και η επιθεώρηση καλλιεργειών χωρίς ανθρώπινη παρέμβαση. Ο χρήστης μπορεί να χαρτογραφήσει τις περιοχές, να ρυθμίσει τις παραμέτρους ψεκασμού και να παρακολουθήσει την πρόοδο μέσω της εφαρμογής.
- Αναλυτικά Συστήματα Αισθητήρων: Το DJI Agras T50 μπορεί να φέρει πολυάριθμους αισθητήρες για τη μέτρηση και ανάλυση παραμέτρων όπως η υγρασία του εδάφους, η

θερμοκρασία, και η κατάσταση των φυτών. Επιπλέον, η υποστήριξη αισθητήρων εικόνας, όπως οι θερμικές και πολυφασματικές κάμερες, βοηθά στην παρακολούθηση της υγείας των καλλιεργειών και στον εντοπισμό πιθανών προβλημάτων όπως ασθένειες ή ελλείψεις σε θρεπτικά συστατικά.

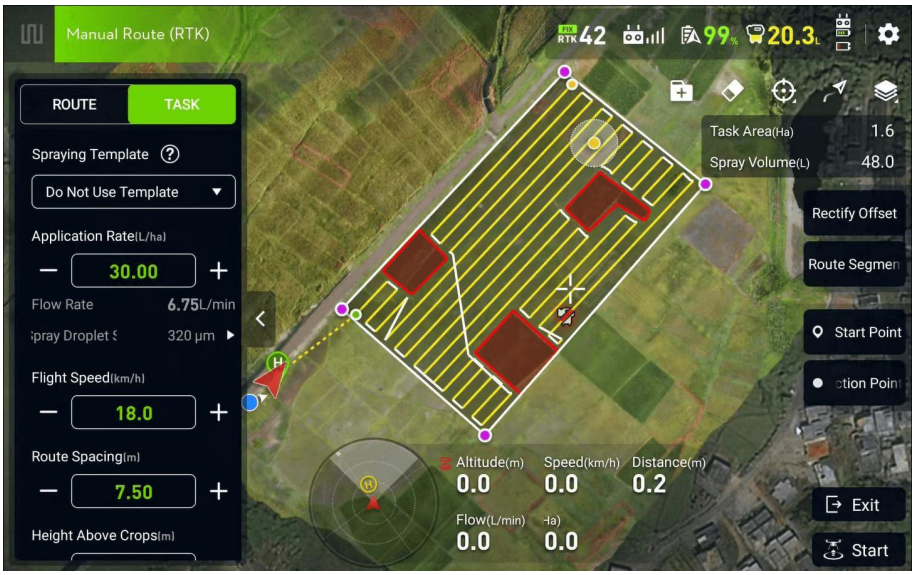
Εφαρμογές στη Γεωργία Ακριβείας:

Το DJI Agras T50 είναι ιδανικό για εφαρμογές που απαιτούν υψηλή ακρίβεια και αυτονομία, όπως:

- **Ψεκασμός καλλιεργειών:** Με τη δυνατότητα εφαρμογής υγρών φαρμάκων ή λιπασμάτων, το drone μειώνει τη σπατάλη και βελτιστοποιεί τη χρήση των χημικών ουσιών.

Εικόνα 1.6: Ψεκασμός φυτοφαρμάκων με την χρήση του DJI Agras T50 [https://nuwayag.com/cdn/shop/files/0S3B9250-Square.jpg?v=1716326559&width=2048]

- **Παρακολούθηση φυτών και εδάφους:** Με τη χρήση αισθητήρων εικόνας και δεδομένων σε πραγματικό χρόνο, οι αγρότες μπορούν να παρακολουθούν την κατάσταση των καλλιεργειών και να εντοπίζουν προβλήματα πριν αυτά γίνουν σοβαρά.
- **Δημιουργία γεωχωρικών χαρτών:** Χρησιμοποιώντας δεδομένα από το drone, οι γεωργοί μπορούν να δημιουργήσουν αναλυτικούς χάρτες για την καλύτερη κατανομή των πόρων και την καλύτερη διαχείριση της καλλιέργειας.



Εικόνα 1.7: Προγραμματισμός και παρακολούθηση πτήσης μέσω του λογισμικού της DJI [https://www1.djicdn.com/cms/uploads/c8e58a9dfc6fdb14a49beed7a95b81bf.png]

1.5 Παραδοτέο Τελικό Σύστημα

Το παραδοτέο της παρούσας πτυχιακής εργασίας είναι ένα αυτόνομο μη επανδρωμένο τετρακόπτερο drone σχεδιασμένο να συλλέγει περιβαλλοντικά δεδομένα και να τα απεικονίζει μέσω μίας εφαρμογής android στον χρήστη. Ο σχεδιασμός και η υλοποίηση στοχεύουν στην χρήση του drone σε εφαρμογές πάνω στην γεωργία και τον περιβαλλοντικό έλεγχο μεγάλων εκτάσεων. Το σύστημα ενσωματώνει αισθητήρες, όπως GPS υψηλής ακρίβειας για τον προσανατολισμό του στον χώρο, αισθητήρα θερμοκρασίας και υγρασίας αέρα για την συλλογή δεδομένων από το περιβάλλον. Μέσω προγραμματισμένων αυτόνομων πτήσεων που ορίζει ο χρήστης, το drone είναι σε θέση να συλλέγει περιβαλλοντικά δεδομένα από αισθητήρες κατά τη διάρκεια της πτήσης και να τα αποθηκεύει. Τα δεδομένα αυτά, συγχρονίζονται με δεδομένα πτήσης που συλλέγονται από το drone και αποστέλλονται στο δίκτυο μέσω ενώς τοπικού web server. Η απεικόνιση των δεδομένων πραγματοποιείται μέσω εφαρμογής android, με την οποία ο χρήστης μπορεί να δει τα δεδομένα που έχει συλλέξει το drone, καθώς και πολλά κρίσιμα δεδομένα για την κατάσταση του drone σε πραγματικό χρόνο, όπως η τάση της μπαταρίας, η ακριβή θέση του drone, το υψόμετρο κ.α.

Το συνολικό έργο στοχεύει σε ένα ολοκληρωμένο σύστημα συλλογής και απεικόνισης δεδομένων που ξεκινά από την ασφαλή αυτόνομη πτήση έως και τον τελικό χρήστη προσφέροντας μία αποδοτική, προσιτή και φιλική προς το περιβάλλον λύση για σύγχρονους αγρότες.

1.6 Δομή της Εργασίας

Η παρούσα Π.Ε. οργανώνεται σε πέντε κεφάλαια. Τα κεφάλαια που την συνθέτουν περιγράφονται παρακάτω:

1ο Κεφάλαιο: Το πρώτο κεφάλαιο παρουσιάζει το θεωρητικό υπόβαθρο, καλύπτοντας το βασικά υλικά και λογισμικά που χρησιμοποιούν τα μη επανδρωμένα drone, την συλλογή και απεικόνιση δεδομένων και τις εφαρμογές τους στην σύγχρονη εποχή.

2ο Κεφάλαιο: Το δεύτερο κεφάλαιο περιγράφει την αρχιτεκτονική και μεθοδολογία που χρησιμοποιήθηκε για την επιλογή και σύνθεση του βασικού συστήματος πτήσης του drone, συμπεριλαμβανομένων των υλικών, των αισθητήρων και του λογισμικού πλοήγησης.

3ο Κεφάλαιο: Το τρίτο κεφάλαιο αναλύει την ανάπτυξη και υλοποίηση του ενσωματωμένου συστήματος αισθητήρων που χρησιμοποιείται για την συλλογή δεδομένων και την ορθή αποστολή τους μέσω δικτύου. Παρουσιάζονται τα βήματα που ακολουθήθηκαν για την κατασκευή και ενσωμάτωση του συστήματος με το βασικό σύστημα πτήσης, αναλύεται ο κώδικας ανάπτυξης και τα πρωτόκολλα που χρησιμοποιήθηκαν για την ανάγνωση και αποστολή των δεδομένων .

4ο Κεφάλαιο: Το τέταρτο κεφάλαιο ολοκληρώνει το σύστημα παρουσιάζοντας την ανάπτυξη της εφαρμογής android που είναι υπεύθυνη για την απεικόνιση των δεδομένων στον τελικό χρήστη. Αναλύεται η διαδικασία ανάκτησης των δεδομένων από το ενσωματωμένο σύστημα αισθητήρων, οι τεχνικές που χρησιμοποιήθηκαν για την ομαλή επικοινωνία και σύνδεση της εφαρμογής με το υπόλοιπο σύστημα και η κατάλληλη διαχείριση των δεδομένων για την εμφάνισή τους στη διεπαφή χρήστη.

5ο Κεφάλαιο: Το πέμπτο κεφάλαιο περιλαμβάνει τα συμπεράσματα από την ολοκλήρωση του έργου, τα τελικά χαρακτηριστικά του συστήματος και το κόστος τωνυλικών. Αναφέρονται επίσης οι προκλίσεις κατά την υλοποίηση, οι περιορισμοί του συστήματος και προτάσεις για μελλοντική ανάπτυξη και έρευνα.

Στο κείμενο της Π.Ε. χρησιμοποιήθηκε το εργαλείο τεχνητής νοημοσύνης ChatGPT για υποβοήθηση στον ορθογραφικό έλεγχο και στη βελτίωση της συνοχής των προτάσεων. Η ουσία της εργασίας, ο σχεδιασμός, η υλοποίηση και τα τεχνικά αποτελέσματα αποτελούν προϊόν της προσωπικής μου προσπάθειας.

Κεφάλαιο 2ο: Σχεδιασμός και Σύνθεση του Συστήματος Πτήσης

2.1 Εισαγωγή

Το παρόν κεφάλαιο παρουσιάζει τη διαδικασία σύνθεσης και διαμόρφωσης ενός αυτόνομου drone, σχεδιασμένου για τη συλλογή δεδομένων στη γεωργία ακριβείας. Αναλύονται τα κριτήρια επιλογής του υλικού και λογισμικού, καθώς και οι τεχνικές προκλήσεις και προβλήματα που αντιμετωπίστηκαν. Το κεφάλαιο θέτει τη βάση για την συλλογή και απεικόνιση των δεδομένων που αναλύονται στα επόμενα κεφάλαια.

2.2 Εξαρτήματα Υλικού

Η επιλογή των εξαρτημάτων βασίστηκε σε τέσσερα κριτήρια που θα το έκαναν ανταγωνιστικό και αποδοτικό:

- Χαμηλό κόστος
- Μεγάλη αυτονομία
- Σταθερότητα και ισχύς πτήσης
- Περιθώρια ενσωμάτωσης επιπλέον συστημάτων

Ακολουθεί συνοπτικός πίνακας με τα εξαρτήματα που χρησιμοποιήθηκαν:

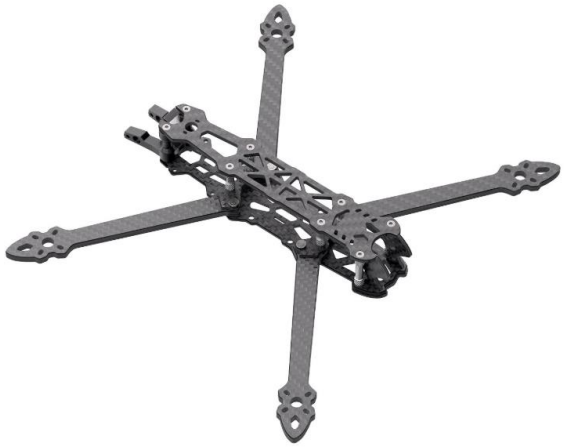
Πίνακας 2.1: Συνοπτικός πίνακας εξαρτημάτων

Α/Α	**Εξάρτημα**	**Μοντέλο**	**Λεπτομέρειες**	**Λειτουργία**
1	Πλαίσιο	GEPRC Mark4 7”	7 ιντσών, carbon fiber, H type	Υποστήριξη εξαρτημάτων
2	Κινητήρες	YSIDO 2807 1.300kV	6-7 ιντσών προπέλες, 6S, 1400W	Σταθερότητα και ισχύς πτήσης
3	FC – ESC Stack	SpeedyBee F405 v.4 BLS 60A	STM32F405, 60A ESC, Bluetooth	Υποστήριξη λογισμικού, ενσωμάτωση
4	Μπαταρία	Li-ion 6S 4.500mAh	Molicel P45B, 6S1P, XT60	Μεγάλη αυτονομία, υψηλό ρεύμα εκφόρτισης
5	GPS	GEPRC M10Q	u-blox M10, QMC5883L	Πλοήγηση με ακρίβεια
6	Προπέλες	HQProp 7040 7x4x3mm	7 ιντσών, τρίπτερες	Ισορροπία ισχύος-κατανάλωσης
7	Δέκτης	RadioMaster RP1	ExpressLRS, T κεραία	Μικρό βάρος, μεγάλη εμβέλεια
8	Πομπός (Χειριστήριο)	RadioMaster Pocket (LBT)	ExpressLRS, EdgeTX, Hall-effect gimbals	Τηλεμετρία, χειροκίνητος έλεγχος

2.2.1 Πλαίσιο

Το πλαίσιο GEPRC Mark4 7 ιντσών επιλέχθηκε για την κατασκευή του drone λόγω της ελαφριάς κατασκευής του από υψηλής ποιότητας 3K ανθρακονήματα (carbon fiber) και της αντοχής του σε κραδασμούς και πτώσεις.

Με διαστάσεις 226x226mm, wheelbase 295mm και πλαίσιο τύπου H, το Mark4-7 προσφέρει επαρκή χώρο για να ενσωματώσει όλα τα εξαρτήματα και τους αισθητήρες, καθώς και μία μεγάλη μπαταρία στο πάνω μέρος του πλαισίου. Επιπλέον, οι οπές στερέωσης διευκολύνουν την εγκατάσταση ηλεκτρονικών εξαρτημάτων.



Εικόνα 2.1: Το πλαίσιο Mark4 7 ιντσών
[https://geprc.com/wp-content/uploads/2023/11/6_Main_0000-6.jpg]

2.2.2 Κινητήρες

Οι τέσσερεις κινητήρες YSIDO 2807 1.300kV επιλέχθηκαν για την υψηλή ισχύ και αποδοτικότητά τους. Με μέγιστη ώθηση από 1,5 έως 2 κιλά ανά κινητήρα και συμβατότητα με έλικες 7 ιντσών, οι κινητήρες εξασφαλίζουν σταθερότητα πτήσης ακόμα και με επιπλέον φορτίο από πρόσθετα συστήματα, όπως το ενσωματωμένο σύστημα αισθητήρων.

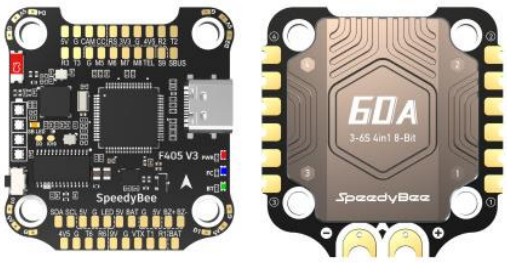
Εικόνα 2.2: Ο κινητήρας 2807 1.300kV της YSIDO
[https://ae01.alicdn.com/kf/S423de7c03b2b4f468dd17bf6fbcc3c97W.jpg]

Με χρήση μίας μπαταρίας 6S, οι κινητήρες αυτοί μπορούν να παρέχουν μέγιστη ισχύ 1.400W και το μέγιστο ρεύμα που μπορούν να υποστηρίξουν είναι 55A.

2.2.3 FC – ESC

Για λόγους οικονομίας χώρου και υποστήριξης λογισμικού, για τον ελεγκτή πτήσης και τους ηλ. ελεγκτές ταχύτητας, επιλέχθηκε μία ολοκληρωμένη λύση που προσφέρει η SpeedyBee.

Ο ελεγκτής πτήσης SpeedyBee F405 v.4 και ο τέσσερεις-σε-έναν ηλ. ελεγκτής ταχύτητας BLS 60A της SpeedyBee είναι σχεδιασμένοι να στοιβάζονται ο ένας πάνω στον άλλο (stack), ώστε να φαίνονται σαν ένα εξάρτημα. Αυτό δίνει χώρο στο drone να μπορεί να ενσωματώσει περισσότερα εξαρτήματα χωρίς να περιορίζεται.



Εικόνα 2.3: Ο FC F405 v.4 και ο ESC BLS 60A
[https://cdn11.bigcommerce.com/s-fhxxhuiq8q/images/stencil/1280x1280/products/246/1432/SB_F405V4-60A-5__77967.1707209775.jpg?c=2]

Ο ελεγκτής πτήσης χρησιμοποιεί επεξεργαστή STM32F405 και γυροσκόπιο ICM42688P, εξασφαλίζοντας στο drone ακριβή έλεγχο πτήσης και υποστήριξη αυτόνομων πτήσεων μέσω του

λογισμικού πτήσης INAV. Επιπλέον, υποστηρίζεται η Blackbox καταγραφή σε microSD, η οποία αποτελεί χρήσιμο εργαλείο για την ανάλυση δεδομένων και συμπεριφοράς πτήσης του drone.

Ο BLS 60A ενσωματώνει τέσσερεις ESC, ένα για κάθε κινητήρα, υποστηρίζοντας συνεχές ρεύμα έως 60, ενώ μπορεί να φτάσει και τα 80A (όριο ασφαλείας 10 δευτερολέπτων) αν το απαιτεί η πτήση. Η μεγάλη ψύκτρα στο πάνω μέρος του προστατεύει τα κυκλώματα από υψηλές θερμοκρασίες. Ο συγκεκριμένος ESC είναι ιδανικός για τους ισχυρούς 2807 κινητήρες και μέσω μίας μπαταρίας 6S μπορεί να υποστηρίξει απαιτητικές πτήσεις χωρίς να δουλεύει στα όριά του.

Εικόνα 2.4: Το SpeedyBee F405 v.4 Stack
[https://cdn11.bigcommerce.com/s-fhxxhuiq8q/images/stencil/1280x1280/products/246/1428/SB_F405V4-60A-3__16385.1707208047.jpg?c=2]

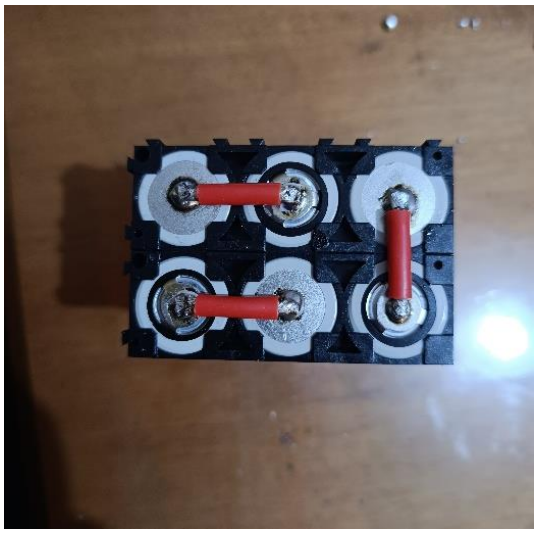
2.2.4 Μπαταρία

Η επιλογή της μπαταρίας ήταν κρίσιμη για την αυτονομία του συστήματος και την υποστήριξη των επιμέρους συστημάτων του drone. Οι Li-ion μπαταρίες έχουν υψηλότερη ενεργειακή πυκνότητα έναντι των Li-Po (έως 250 Wh/kg έναντι 150-200Wh/kg για Lipo), το οποίο τις κάνει πιο αποδοτικές για μακρινές πτήσεις ή μεγάλες γεωργικές εκτάσεις. Η μπαταρία τροφοδοτεί αρχικά το ESC και αυτός με την σειρά του τους κινητήρες και τον ελεγκτή πτήσης που διανέμει ύστερα τη τάση στα υπόλοιπα συστήματα.

Για την κατασκευή της μπαταρίας, χρησιμοποιήθηκαν έξι κυψέλες Molicel P45B μεγέθους 21700, λόγω χαμηλής εσωτερικής αντίστασης και υψηλού ρεύματος εκφόρτισης (45A), το οποίο εξασφαλίζει σταθερή τροφοδοσία για όλο το σύστημα. Η κατασκευή έγινε χειροκίνητα με συγκόλληση των κυψελών σε διάταξη 6S1P και η σύνδεση με το stack επιτυγχάνεται μέσω βύσματος XT60, ενώ η φόρτιση γίνεται μεσω βύσματος JST.

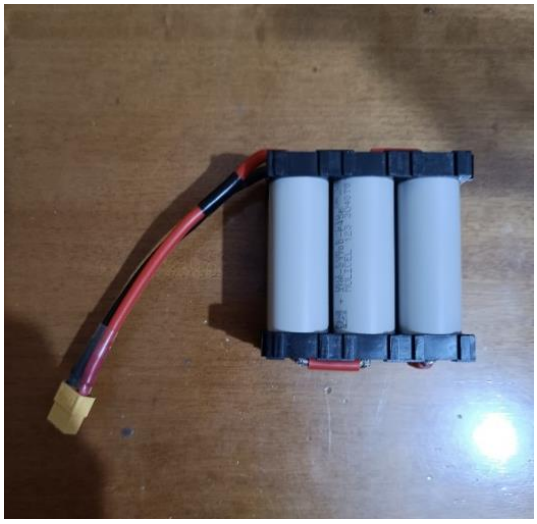
Εικόνα 2.5: Οι κυψέλες Molicel P45B
[https://marsen.com.au/wp-content/uploads/2025/05/Molicel-P45B-2-1.jpg]

Αρχικά οι έξι κυψέλες τοποθετήθηκαν σε ειδικό αποστάτη που τις κρατάει σε μικρή απόσταση μεταξύ τους. Στην συνέχεια έγινε η συγκόλληση μεταξύ των πόλων μέσω καλωδίου σιλικόνης (12AWG - χαλκός), έτσι ώστε οι κυψέλες να είναι σε σειρά μεταξύ τους, όπως φαίνεται στην εικόνα 2.6.



Εικόνα 2.6: Συγκόλληση των κυψελών

Αξίζει να σημειωθεί πως δόθηκε ιδιαίτερη προσοχή κατά την συγκόλληση ώστε να μην προκληθεί βλάβη στις κυψέλες λόγω αυξημένης θερμοκρασίας. Στην συνέχεια έγινε συγκόλληση του βύσματος τροφοδοσίας XT60 και ενός βύσματος JST για την ισορροπημένη φόρτιση όλων των κυψελών, όπως φαίνεται στην εικόνα 2.7.



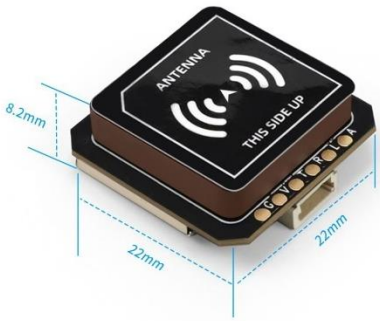
Εικόνα 2.7: Συγκόλληση του XT60

Τέλος, για την μόνωση της μπαταρίας, χρησιμοποιήθηκε μονωτικό υλικό για μπαταρίες στους πόλους και θερμοσυστελλόμενο που καλύπτει όλη την μπαταρία για προστασία. Η τελική κατασκευή που φαίνεται στην εικόνα 2.8 είναι μία μπαταρία 4.500mAh μέγιστης τάσης τροφοδοσίας 25,2V που μπορεί να τροφοδοτήσει το σύστημα με μέγιστο συνεχές ρεύμα 45A.

Εικόνα 2.8: Η μπαταρία του drone

2.2.5 GPS

Το M10Q GPS module της GEPRC αποτελεί το πιο πρόσφατο μοντέλο της εταιρίας, ενσωματώνοντας το u-blox M10 chip και το μαγνητόμετρο QMC5883L, τα οποία προσφέρουν μεγάλη ακρίβεια θέσης και κατεύθυνσης κατά την αυτόνομη πλοήγηση του drone πάνω από αγροτικές καλλιέργειες.



Εικόνα 2.9: Το M10Q GPS module της GEPRC
[https://thefpvproject.com/wp-content/uploads/2024/03/M1025Qb.jpg]

Το GPS υποστηρίζει πολλά δορυφορικά συστήματα, όπως GPS, GLONASS, Galileo, BeiDou, QZSS και SBAS, ενώ μπορεί να υποστηρίξει ταυτόχρονη σύνδεση με έως και 32 δορυφόρους ταυτόχρονα. Έχει ακρίβεια θέσης ±1,5μ. οριζόντια και ±2μ. κάθετα, αρκετά ακριβή για τη χρήση του στη γεωργία.

2.2.6 Προπέλες

Οι τέσσερεις τρίπτερες προπέλες της HQProp με διάμετρο 7 ιντσών και pitch 4 ενσωματώνονται στους κινητήρες και είναι ικανές να εξασφαλίσουν αποδοτική ώθηση και σταθερότητα στο drone. Οι δύο προπέλες είναι σχεδιασμένες να περιστρέφονται με την φορά του ρολογιού (CW), ενώ οι άλλες δύο αντίθετα από την φορά του ρολογιού (CCW).

Κατασκευασμένες από ανθεκτικό πολυανθρακικό πλαστικό, προσφέρουν ισορροπία μεταξύ ισχύος και αυτονομίας, υποστηρίζοντας πτήσεις διάρκειας άνω των 20 λεπτών. Οι οπές στήριξής τους είναι μεγέθους M5 και για την ασφαλή τοποθέτησή τους χρησιμοποιείται παξιμάδι ασφαλείας.

Εικόνα 2.10: Οι τέσσερεις προπέλες HQProp 7040
[https://ueeshop.ly200-cdn.com/u_file/UPAJ/UPAJ828/2112/products/08/4bb21dfe63.jpg.640x640.jpg?x-oss-process=image/format,webp]

2.2.7 Πομπός - Δέκτης

Η ανάγκη ύπαρξης χειροκίνητου ελέγχου αποτελεί κρίσιμο παράγοντα για την ορθή και ασφαλή χρήση του drone. Με την χρήση ενός ραδιοχειριστηρίου (πομπός) μπορούμε να ελέγξουμε το drone χειροκίνητα, να πάρουμε πληροφορίες για την κατάστασή του μέσω τηλεμετρίας, ενώ αποτελεί και μία δικλείδα ασφαλείας σε περίπτωση που το drone δεν αντιδρά επιθυμητά. Η επικοινωνία με το drone γίνεται μέσω ενός δέκτη που είναι ενσωματωμένος επάνω στο drone. Και τα δύο εξαρτήματα παράγονται από την RadioMaster και επιλέχθηκαν με γνώμονα την χαμηλή καθυστέρηση της ασύρματης επικοινωνίας και την μεγάλη εμβέλεια των κεραιών.

Ο πομπός RadioMaster Pocket και ο δέκτης RadioMaster RP1 χρησιμοποιούν το πρωτόκολλο ExpressLRS 2,4GHz για την μεταξύ τους επικοινωνία. Η χρήση αυτού του πρωτοκόλλου παρέχει αξιόπιστη, χαμηλής καθυστέρησης επικοινωνία σε μεγάλες αποστάσεις.

Ο RadioMaster Pocket, έχει προεγκατεστημένο το EdgeTX firmware και χρησιμοποιεί μηχανισμούς ελέγχου (joysticks) με ενσωματωμένους hall-effect αισθητήρες για τον ακριβή έλεγχο του drone. Η ειδική έκδοση LBT, περιορίζει την ισχύ του στα 100mW, ώστε να εξασφαλίζει συμμόρφωση με τους ευρωπαϊκούς κανονισμούς.

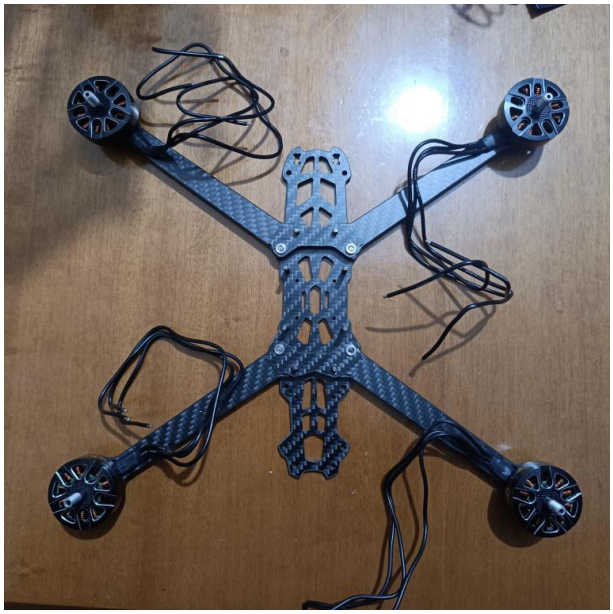
Εικόνα 2.11: Ο πομπός RadioMaster Pocket 2,4Ghz (LBT)
[https://www.ariesrc.gr/33718-large_default/radiomaster-pocket-24ghz-16ch-elrscc2500-hall-gimbals-edgetx-system-radio-controller-built-in-led-lights.jpg]

Ο δέκτης RadioMaster RP1 βασίζεται στο ESP8285 MCU και χρησιμοποιεί μία UFL κεραία τύπου T για εμβέλεια έως 2 χιλιόμετρα, ιδανικό για να υποστηρίξει την εφαρμογή του drone στην γεωργία ακριβείας.

Εικόνα 2.12: Ο δέκτης RadioMaster RP1 2,4GHz
[https://radiomasterrc.com/cdn/shop/files/RP1-V240123.jpg?v=1750061953]

2.3 Συναρμολόγηση του Συστήματος Πτήσης

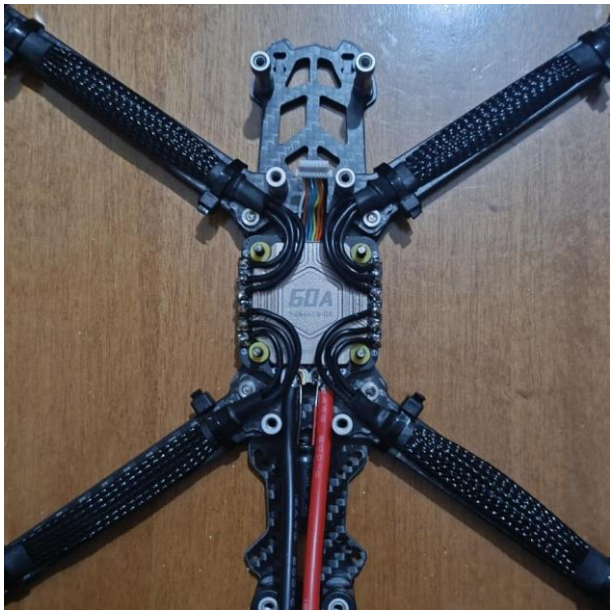
Η συναρμολόγηση του συστήματος πτήσης, όπως βλέπουμε στην εικόνα 2.13, ξεκίνησε με την τοποθέτηση των κινητήρων στους τέσσερεις βραχίονες του πλαισίου, χρησιμοποιώντας βίδες M4.



Εικόνα 2.13: Εγκατάσταση κινητήρων στο πλαίσιο

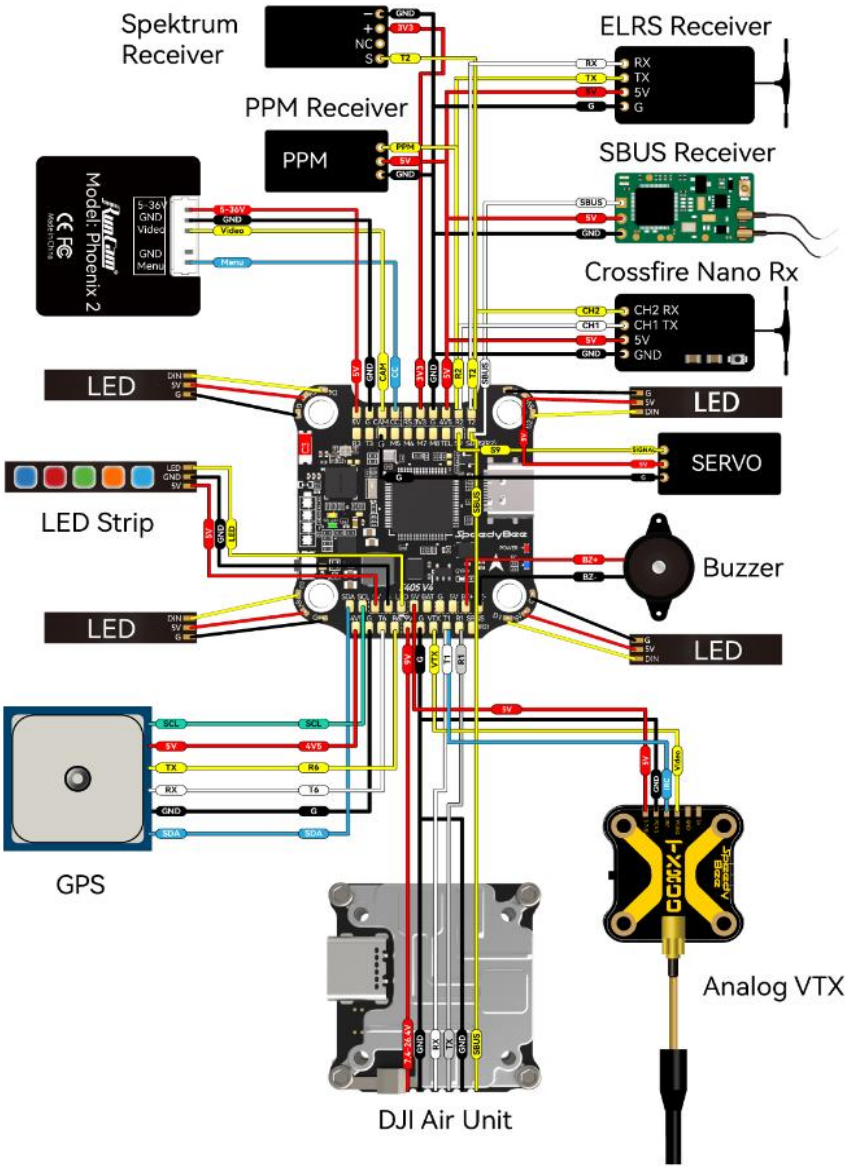
Στην συνέχεια, τοποθετήθηκε ο ESC στο κέντρο του πλαισίου και έγινε συγκόλληση των καλωδίων των κινητήρων στο ESC καθώς και του θηλυκού βύσματος τροφοδοσίας XT60. Παράλληλα με την τροφοδοσία, τοποθετήθηκε ένας low ESR πυκνωτής (1000μF 35V) για σταθεροποίηση τάσης και μείωση του ηλεκτρικού θορύβου. Τα καλώδια των κινητήρων καλύφθηκαν με ειδικό πλαστικό πλέγμα

προστασίας καλωδίων, όπως φαίνεται στην εικόνα 2.14 και στερεώθηκαν στους βραχίονες του πλαισίου με πλαστικά δεματικά (tire-ups).



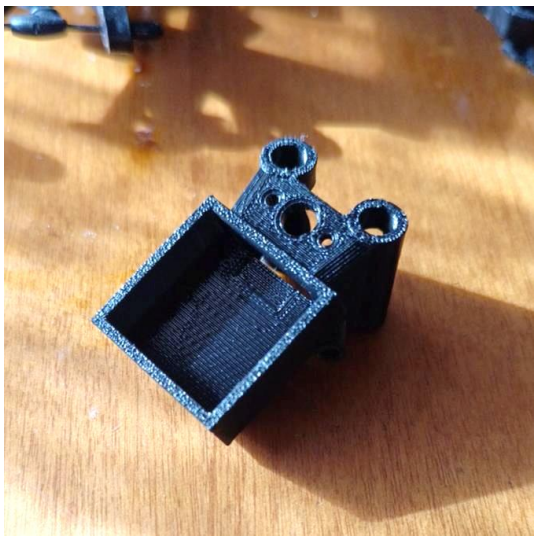
Εικόνα 2.14: Σύνδεση κινητήρων με το ESC

Έπειτα, συνδέθηκε ο ελεγκτής πτήσης στο ESC και στερεώθηκε ακριβώς από πάνω του (stack). Όλα τα υπόλοιπα περιφερειακά και υποσυστήματα συνδέθηκαν στον FC μέσω των θυρών UART που διαθέτει και διαχειρίζονται από αυτόν. Στην παρακάτω εικόνα φαίνονται όλα τα περιφερειακά που μπορεί να υποστηρίξει ο SpeedyBee F405 v.4 και η σύνδεσή τους με αυτόν:



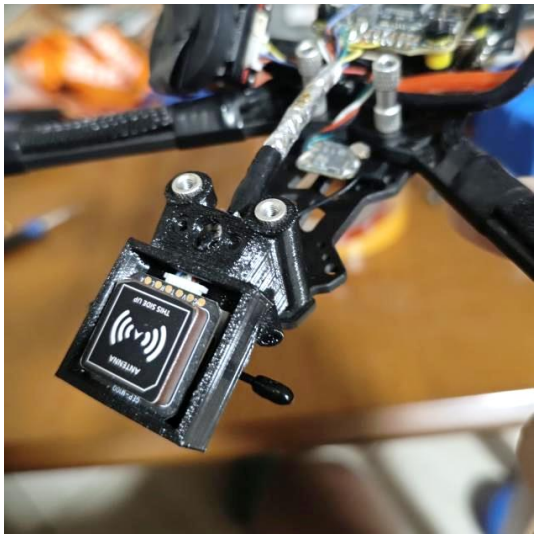
Εικόνα 2.15: Περιφερειακά που υποστηρίζονται από τον ελεγκτή πτήσης SpeedyBee F405 [https://www.costruzionedroni.it/img/cms/speedybee/FC/f405-v4/speedybee-f405-v4-flight-controller_8.jpg]

Για την στήριξη του GPS και της κεραίας του δέκτη μακριά από ηλεκτρομαγνητικές παρεμβολές, χρησιμοποιήθηκε ένα πλαστικό εξάρτημα 3D εκτύπωσης που ενσωματώνεται στο πίσω μέρος του πλαισίου, όπως φαίνεται στην εικόνα 2.16.

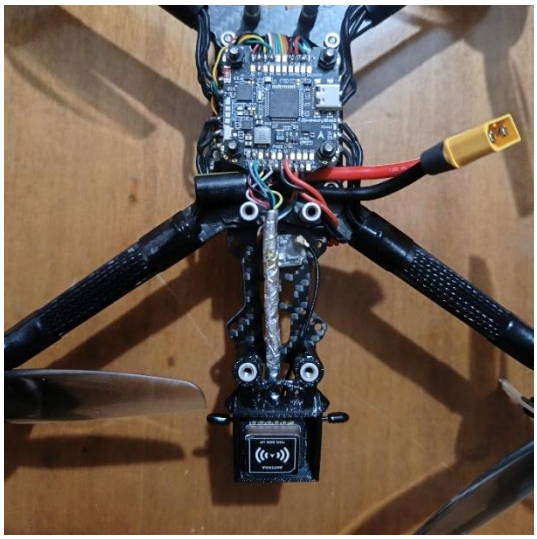


Εικόνα 2.16: Το εξάρτημα στήριξης του GPS και του δέκτη

Στη συνέχεια, το GPS συνδέθηκε στη UART 6 θύρα και ο δέκτης στην UART 2 θύρα του ελεγκτή πτήσης. Στις εικόνες 2.17 και 2.18 φαίνεται η τοποθέτηση και σύνδεση τους στον ελεγκτή πτήσης, όπως ορίζει ο κατασκευαστής:



Εικόνα 2.17: Η ενσωμάτωση του GPS και του δέκτη



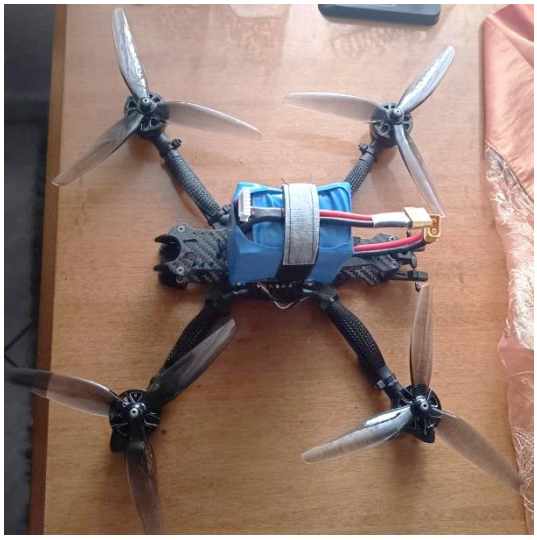
Εικόνα 2.18: Σύνδεση GPS και δέκτη στον ελεγκτή πτήσης

Η τοποθέτηση του GPS μακριά από καλώδια τροφοδοσίας που μπορεί να προκαλέσουν ηλεκτρομαγνητικές παρεμβολές, όπως της μπαταρίας, είναι κρίσιμη για την ορθή λειτουργία της πυξίδας (μαγνητόμετρο GPS). Για το λόγο αυτό, τα καλώδια σύνδεσης του GPS με τον ελεγκτή πτήσης τυλίχθηκαν με μεταλλικό μονωτικό υλικό, όπως φαίνεται στην εικόνα 2.19.

Εικόνα 2.19: Μόνωση καλωδίων GPS

Τέλος, βιδώθηκε το κάλυμμα του πλαισίου και τοποθετήθηκε η μπαταρία στο επάνω μέρος. Στις εικόνες 2.20 και 2.21 φαίνεται το τελικό βασικό σύστημα του drone μετά την συναρμολόγηση:

Εικόνα 2.20: Η τοποθέτηση του πάνω καλύμματος του πλαισίου



Εικόνα 2.21: Το drone μετά την συναρμολόγηση

2.4 Παραμετροποίηση Συστήματος

Ο ελεγκτής πτήσης δεν μπορεί να αναγνωρίσει από μόνος του τα επιμέρους εξαρτήματα που συνδέονται σε αυτόν και για αυτόν το λόγο χρησιμοποιείται το λογισμικό πτήσης, από το οποίο μπορούμε να

ρυθμίσουμε πολλές παραμέτρους του συστήματος ώστε να βεβαιωθούμε ότι το drone είναι ασφαλή για την πρώτη του πτήση.

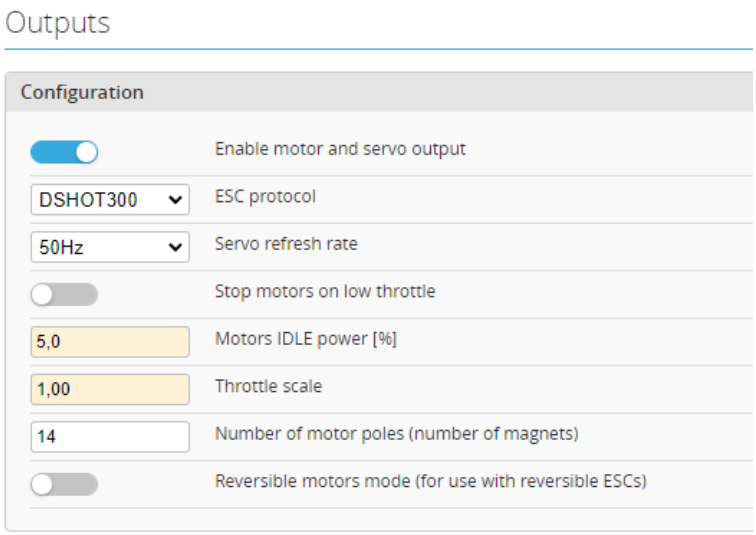
2.4.1 Επιλογή Λογισμικού Πτήσης

Η επιλογή του κατάλληλου firmware για τον ελεγκτή πτήσης ενός drone είναι κρίσιμη για την υποστήριξη αυτόνομων πτήσεων, ιδιαίτερα σε εφαρμογές γεωργίας ακριβείας όπως η παρακολούθηση αγροτικών εκτάσεων. Για τον σκοπό αυτό, επιλέχθηκε το ανοιχτού κώδικα λογισμικό πτήσης INAV Configurator, το οποίο υποστηρίζεται από τον ελεγκτή πτήσης SpeedyBee F405 v.4. Το INAV προσφέρει εξειδικευμένες λειτουργίες για αυτόνομα συστήματα, όπως η λειτουργία Return-to-Home (RTH) και το Mission Planner για προγραμματισμένες αποστολές σε waypoints, οι οποίες είναι απαραίτητες για την συλλογή ή χαρτογράφηση μεγάλων αγροτικών περιοχών. Δίνοντας προτεραιότητα στην ακρίβεια πλοήγησης, το INAV αξιοποιεί τα δεδομένα θέσης που διαβάζει από το GPS και μέσω αλγορίθμων μπορεί να πραγματοποιήσει αυτόνομες πτήσεις με επιτυχία και ασφάλεια. Η εγκατάσταση του λογισμικού στον ελεγκτή πτήσης και η παραμετροποίηση του συστήματος γίνονται μέσω του λογισμικού INAV Configurator 8.1.

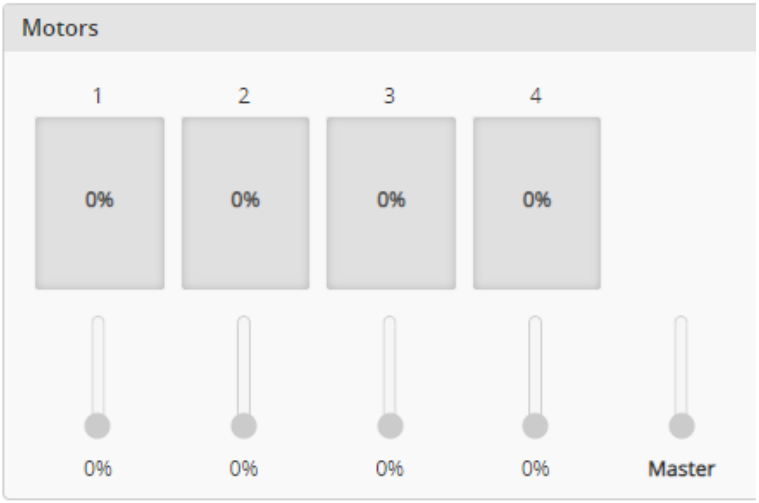
2.4.2 Βασικές Ρυθμίσεις Συστήματος

Για την ρύθμιση του συστήματος όλες οι προπέλες αφαιρέθηκαν και συνδέθηκε η μπαταρία στο drone.

Αρχικά, για την ρύθμιση των κινητήρων, μέσω της καρτέλας «Outputs» ορίστηκε σαν πρωτόκολλο επικοινωνίας του ESC με τον ελεγκτή πτήσης το DSHOT300, το οποίο αποτελεί ένα ψηφιακό πρωτόκολλο χαμηλής καθυστέρησης που στέλνει εντολές ταχύτητας από τον ελεγκτή πτήσης στο ESC με ταχύτητα επικοινωνίας 300kbits/s. Η χρήση ψηφιακού πρωτοκόλλου αποτρέπει τους κινητήρες να επηρεάζονται από παρεμβολές των αναλογικών κυκλωμάτων του συστήματος. Ορίστηκε ο ρυθμός ανανέωσης του σήματος ελέγχου του σερβοκινητήρα στα 50Hz και ο αριθμός των μαγνητών των κινητήρων στους 14, όπως αναφέρει ο κατασκευαστής. Μετά την αποθήκευση των ρυθμίσεων έγινε δοκιμή των κινητήρων σε χαμηλή ταχύτητα ώστε να ελεγχθεί η συμπεριφορά τους και η φορά περιστροφής τους.

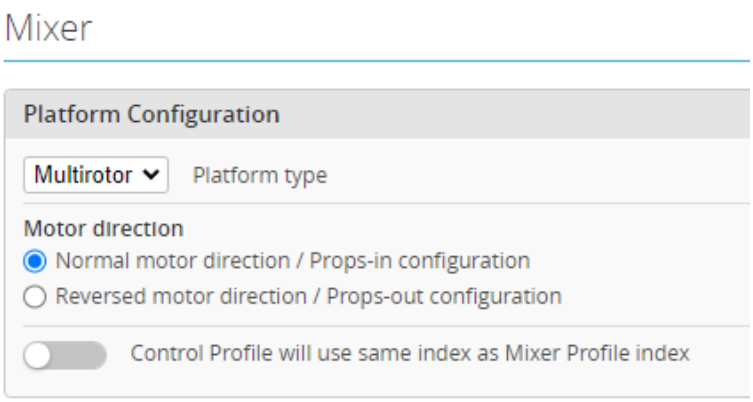


Εικόνα 2.22: Ρυθμίσεις στην καρτέλα Outputs

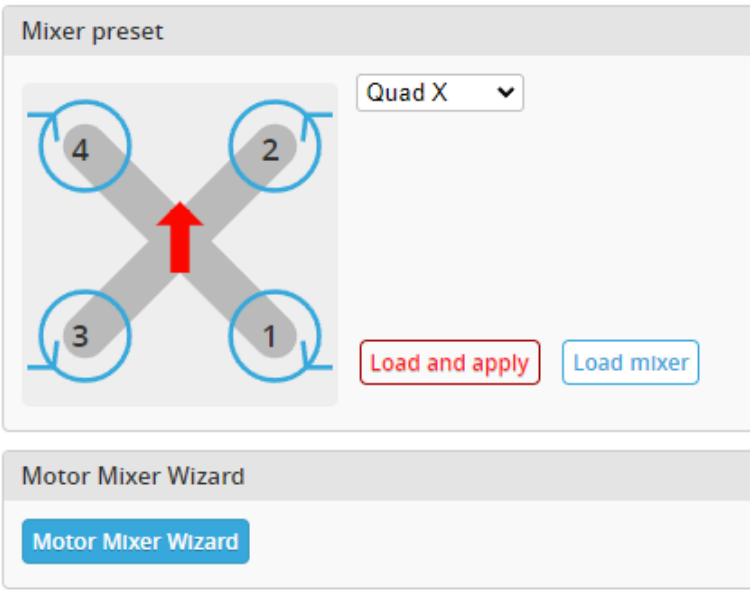


Εικόνα 2.23: Εργαλείο δοκιμής κινητήρων στο INAV

Στην συνέχεια, μέσα από την καρτέλα «Mixer», ορίστηκε το είδος της πλατφόρμας σε «Multirotor», επιλέχθηκε η κανονική φορά περιστροφής των κινητήρων και σαν τύπος πλαισίου ορίστηκε το «Quad X». Μέσα από την επιλογή «Mixer wizard» που παρέχει το INAV, ρυθμίστηκε η ορθή φορά περιστροφής των κινητήρων και δοκιμάστηκε ξανά η συμπεριφορά τους μέσα από την καρτέλα «Outputs».



Εικόνα 2.24: Ρυθμίσεις στην καρτέλα Mixer (1)



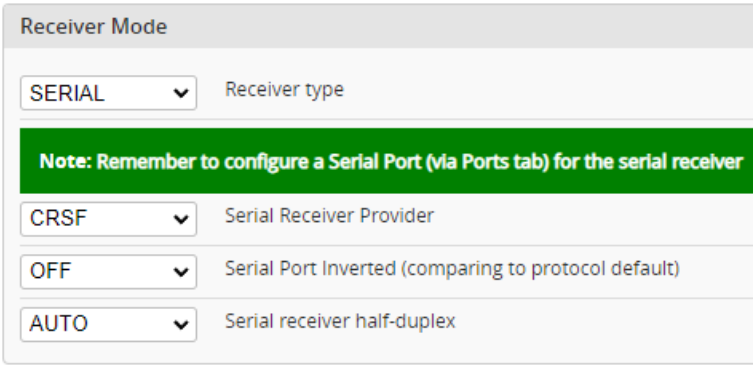
Εικόνα 2.25: Ρυθμίσεις στην καρτέλα Mixer (2)

Η περιγραφή και ο ορισμός των περιφερειακών υποσυστημάτων που συνδέονται στις θύρες του ελεγκτή πτήσης γίνονται μέσα από την καρτέλα «Ports». Ενεργοποιήθηκε η UART 2 που είναι εγκατεστημένος ο δέκτης RadioMaster RP1 ενεργοποιώντας στην στήλη RX το «Serial RX». Για το GPS, ενεργοποιήθηκε το UART 6 επιλέγοντας από την στήλη Sensors την επιλογή GPS με baud rate 115200.

Identifier	Data	Telemetry	RX	Sensors
UART1	MSP 115200	Disabled AUTO	Serial RX	Disabled 115200
UART2	MSP 115200	Disabled AUTO	Serial RX	Disabled 115200
UART3	MSP 115200	Disabled AUTO	Serial RX	Disabled 115200
UART4	MSP 115200	Disabled AUTO	Serial RX	Disabled 115200
UART5	MSP 115200	Disabled AUTO	Serial RX	Disabled 115200
UART6	MSP 115200	Disabled AUTO	Serial RX	GPS 115200

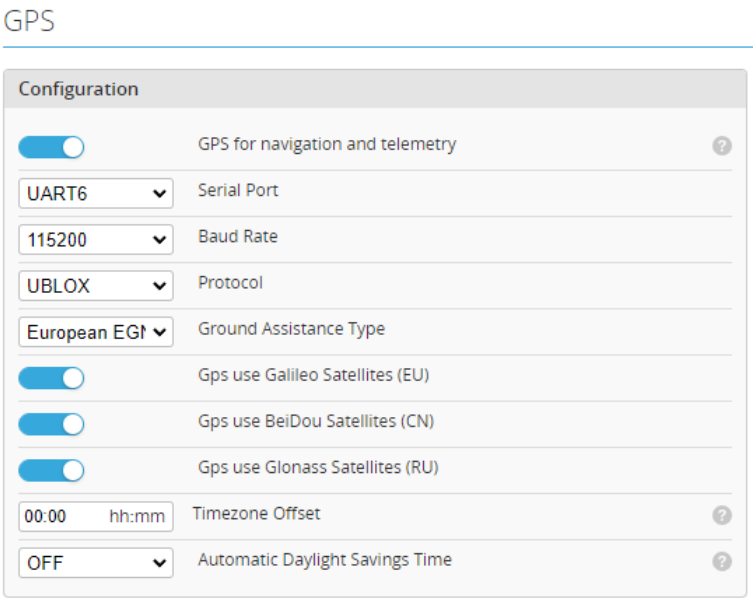
Εικόνα 2.26: Ρυθμίσεις στην καρτέλα Ports

Για να ολοκληρωθεί η ρύθμιση του δέκτη χρησιμοποιούμε την καρτέλα «Receiver», στην οποία επιλέχθηκε ο τύπος του δέκτη σε «Serial» και το πρωτόκολλο επικοινωνίας με τον ελεγκτή πτήσης σε «CRSF» (Crossfire), το οποίο είναι ένα πρωτόκολλο χαμηλής καθυστέρησης που αποστέλλει εντολές RC και δεδομένα τηλεμετρίας σε συστήματα επικοινωνίας που χρησιμοποιούν ExpressLRS.



Εικόνα 2.27: Ρυθμίσεις στην καρτέλα Receiver

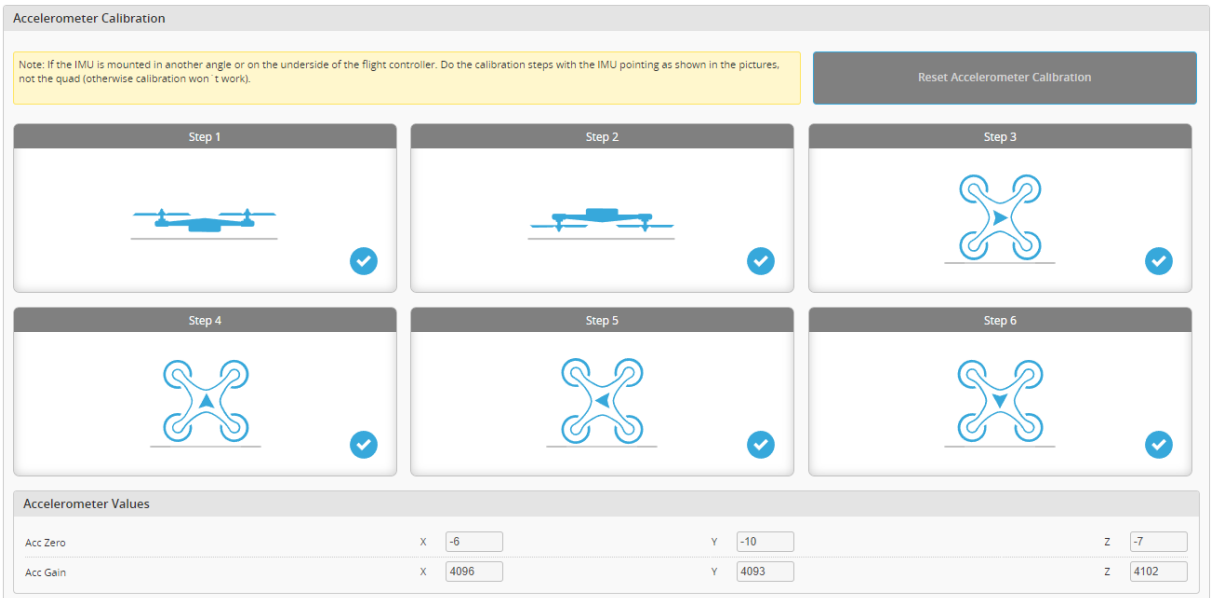
Επίσης, για την ολοκλήρωση ρύθμισης του GPS, μέσω της καρτέλας «GPS» ενεργοποιήθηκε ο αισθητήρας για πλοήγηση και τηλεμετρία. Επιλέχθηκε το πρωτόκολλο του κατασκευαστή (UBLOX) και ενεργοποιήθηκαν οι επιλογές για εύρεση δορυφόρων Galileo, BeiDou και Glonass για μέγιστη απόδοση του GPS.



Εικόνα 2.28: Ρυθμίσεις στην καρτέλα GPS

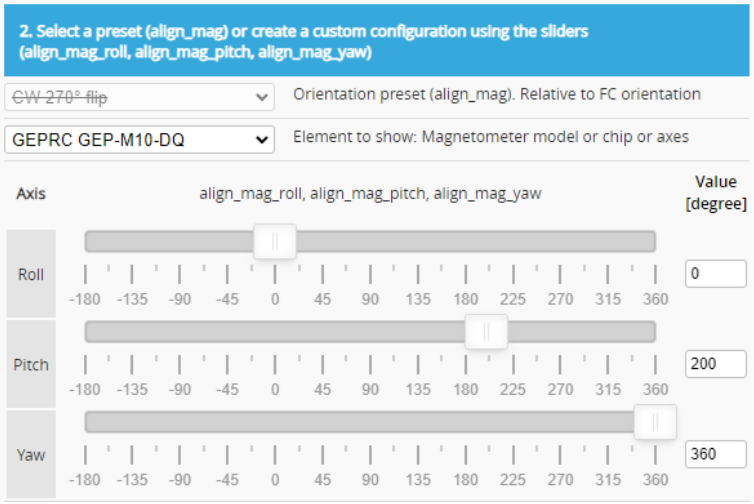
Για την ορθή πτήση του drone χρειάζεται να γίνει βαθμονόμηση του επιταχυνσιόμετρου στον ελεγκτή πτήσης και της πυξίδας στο GPS module. Το INAV μέσω της καρτέλας «Calibration» προσφέρει αυτήν την επιλογή.

Για την βαθμονόμηση του επιταχυνσιόμετρου συλλέχθηκαν οι τιμές του αισθητήρα από διάφορες κλίσεις του drone ενώ είναι ακίνητο πάνω σε μία επίπεδη επιφάνεια, όπως φαίνεται στην εικόνα 2.29.

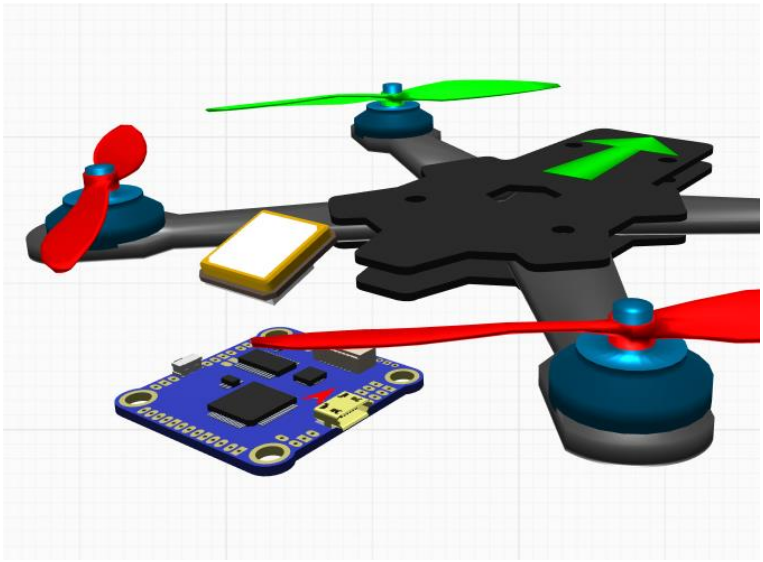


Εικόνα 2.29: Καλιμπράρισμα επιταχυνσιόμετρου του drone

Για το GPS, θα πρέπει πρώτα να ορίσουμε το σημείο που είναι τοποθετημένο το GPS πάνω στο drone καθώς και την κλίση σε μοίρες που έχει σε σχέση με τον ελεγκτή πτήσης. Μέσω της καρτέλας «Alignment Tool» επιλέχθηκε το μοντέλο του GPS module (GEPRC M10) και τοποθετήθηκε το γραφικό εξάρτημα όπως έχουμε τοποθετήσει το φυσικό επάνω στο drone, όπως φαίνεται στις εικόνες 2.30 και 2.31. Η γωνία του GPS module σε σχέση με τον ελεγκτή πτήσης υπολογίστηκε στις 20 μοίρες.



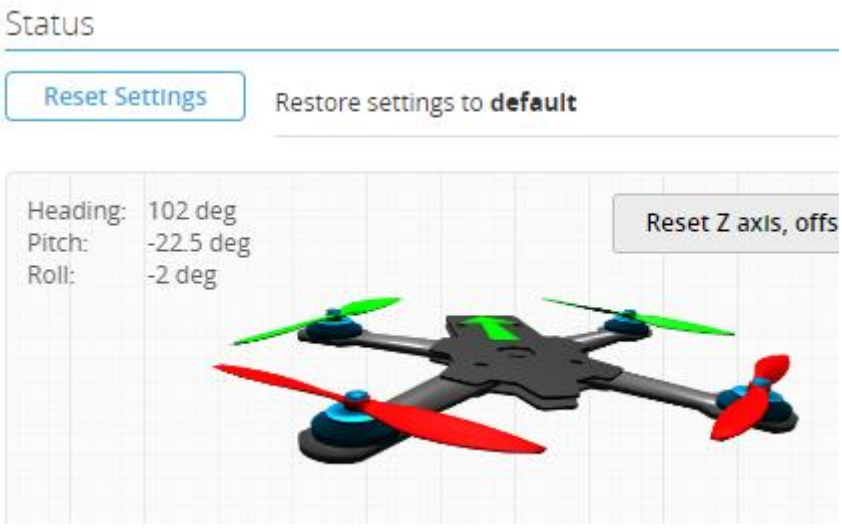
Εικόνα 2.30: Ρυθμίσεις στην καρτέλα Alignment Tool



Εικόνα 2.31: Η τοποθέτηση του γραφικού εξαρτήματος του GPS

Για τη βαθμονόμηση του μαγνητόμετρου, μέσω της καρτέλας «Calibration», το drone περιστράφηκε γύρω από τους τρεις άξονές του, ώστε κάθε πλευρά να τοποθετηθεί διαδοχικά σε θέση στραμμένη προς

το έδαφος, εξασφαλίζοντας πλήρη καταγραφή των μαγνητικών συνιστωσών σε όλες τις κατευθύνσεις. Με την ολοκλήρωση της διαδικασίας, στην καρτέλα «Status» του INAV, επιβεβαιώθηκε η σωστή κατεύθυνση (heading) του drone σε μοίρες.



Εικόνα 2.32: Η κατεύθυνση και κλίση του drone μέσα από το περιβάλλον του INAV

Μετά την ολοκλήρωση όλων των παραπάνω παραμετροποιήσεων, το drone ήταν έτοιμο για την πρώτη του δοκιμαστική πτήση.

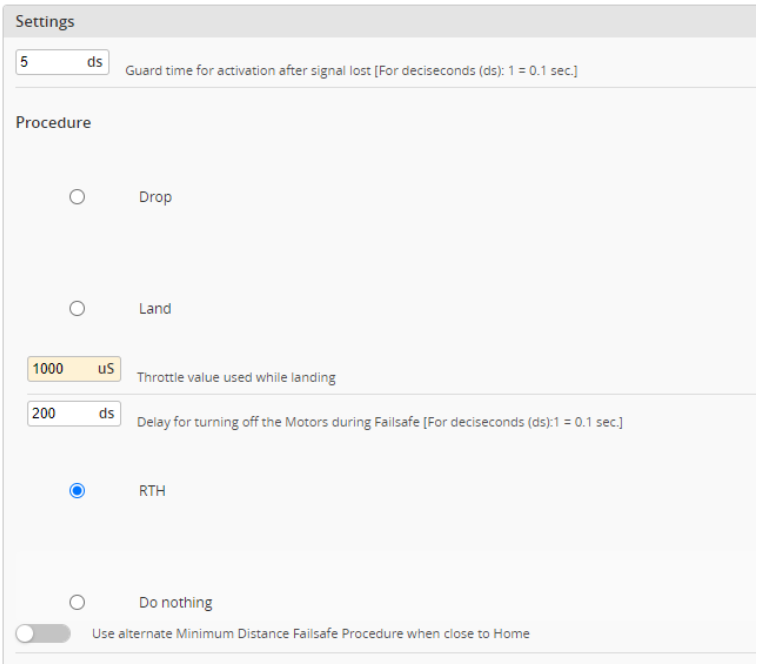
2.5 Δοκιμαστικές Πτήσεις

Στην πρώτη πτήση, δοκιμάστηκε η σταθερότητα του drone και η απόκρισή του στις αλλαγές ταχύτητας των κινητήρων (throttle) από τον χρήστη. Επίσης, έγινε έλεγχος της ικανότητας του drone να εκτελεί όλες τις διαθέσιμες κινήσεις (roll, pitch, yaw) μεμονωμένα και σε συνδυασμό μεταξύ τους.

Εικόνα 2.33: Η πρώτη πτήση του drone

Ακολούθησαν επιπλέον δοκιμαστικές πτήσεις και ελέγχθηκε το drone ως προς την απόδοση σε απότομες αλλαγές ταχύτητας, τη συμπεριφορά του σε πτήσεις με ήπιο άνεμο, τη κατανάλωση ρεύματος των κινητήρων σε διαφορετικές ταχύτητες πτήσης, καθώς και στην μέτρηση θερμοκρασίας των κινητήρων. Δεν παρατηρήθηκε σε κανέναν έλεγχο περίεργη συμπεριφορά του συστήματος. Η κινητήρες σε ήπια πτήση, καταναλώνουν 5-10A συνολικά, αρκετά κάτω από τα 45A που μπορεί να παρέχει η μπαταρία, ενώ σε γρήγορες ομαλές πτήσεις μπορεί να φτάσουν τα 25A κατανάλωση. Η μικρή κατανάλωση ρεύματος των κινητήρων έπαιξε πολύ σημαντικό ρόλο στην αυτονομία πτήσης του drone, με όλες τις δοκιμαστικές πτήσεις να κυμαίνονται στα 20-25 λεπτά.

Στην πορεία, ενσωματώθηκε μέσω του INAV η λειτουργία RTH (Return-to-Home), όπου με το πάτημα ενός κουμπιού στο χειριστήριο, το drone μπαίνει σε λειτουργία αυτόνομης πλοήγησης και επιστρέφει στο σημείο που απογειώθηκε κάνοντας αυτόνομη προσγείωση. Η λειτουργία RTH επιλέχθηκε και σαν δικλείδα ασφαλείας (καρτέλα Failsafe) στο INAV, ώστε αν το drone παρουσιάσει πρόβλημα στην επικοινωνία με τον χρήστη, να επιστρέψει μόνο του στο σημείο απογείωσης.



Εικόνα 2.34: Επιλογή του RTH ως Failsafe κατάσταση

Κατά την δοκιμή επιβεβαιώθηκε η ακρίβεια πλοήγησης του GPS καθώς και η κατεύθυνση της πυξίδας. Το drone κατάφερε σε όλες τις πτήσεις να επιστρέψει στο σημείο απογείωσης με μεγάλη επιτυχία, πετυχαίνοντας ομαλές προσγειώσεις με μέγιστη απόκλιση 50cm από το σημείο που είχε απογειωθεί. Το σύστημα πλοήγησης σε ανοιχτό πεδίο καταφέρνει να συνδεθεί με 20-25 δορυφόρους εντός 1 λεπτού από την ενεργοποίηση του drone.

Τέλος, μέσω του Mission Planner που παρέχει το INAV φορτώθηκε μία αυτόνομη αποστολή στον ελεγκτή πτήσης με 3 σημεία στον χάρτη (waypoints) και επιστροφή στο σημείο απογείωσης (RTH). Η απόσταση πτήσης του drone από το έδαφος ορίστηκε στα 40 μέτρα για λόγους ασφαλείας, ενώ πριν την πτήση ελέγχθηκε το πεδίο για τυχόν εμπόδια ή ανθρώπινη παρουσία.

Το drone εκτέλεσε την αυτόνομη πτήση με επιτυχία επιστρέφοντας στο αρχικό σημείο εκκίνησης. Η απόσταση του από το έδαφος κυμαινόταν από 39 έως 41 μέτρα καθ’ όλη την διάρκεια της πτήσης με τις διαφορές στο υψόμετρο να προέρχονται κυρίως όταν χρειαζόταν να αλλάξει κατεύθυνση (άφιξη σε waypoint).

Κεφάλαιο 3ο: Σχεδίαση και Υλοποίηση Ενσωματωμένου Συστήματος Αισθητήρων

3.1 Εισαγωγή

Η ενσωμάτωση συστημάτων αισθητήρων σε μη επανδρωμένα εναέρια οχήματα (drones) έχει αποτελέσει σημαντικό βήμα προς την κατεύθυνση της αυτοματοποιημένης συλλογής δεδομένων σε εφαρμογές γεωργίας ακριβείας. Η δυνατότητα καταγραφής και αποστολής μετεωρολογικών και πτητικών δεδομένων σε πραγματικό χρόνο, συμβάλλει ουσιαστικά στη λήψη πιο τεκμηριωμένων αποφάσεων σχετικά με την καλλιεργητική διαδικασία.

Το παρόν κεφάλαιο περιγράφει τον σχεδιασμό και την υλοποίηση ενός ενσωματωμένου συστήματος αισθητήρων, το οποίο βασίζεται στον μικροελεγκτή ESP32. Το σύστημα αυτό επικοινωνεί με τον ελεγκτή πτήσης του κυρίως συστήματος του drone που αναλύθηκε στο 2ο κεφάλαιο μέσω του πρωτοκόλλου MAVlink, συλλέγει δεδομένα θερμοκρασίας και υγρασίας από έναν αισθητήρα DHT22 και διαθέτει δυνατότητα αποστολής των δεδομένων μέσω ενσωματωμένου web server. Σκοπός του είναι η αυτόνομη λειτουργία κατά την αυτόνομη πτήση του drone για συλλογή και αποθήκευση των δεδομένων, καθώς και η μετάδοση αυτών για την μετέπειτα απεικόνισή τους.

Η υλοποίηση αυτού του συστήματος αποτελεί βασικό τμήμα της συνολικής αρχιτεκτονικής του συστήματος, καθώς καλύπτει το κρίσιμο κομμάτι της περιβαλλοντικής μέτρησης και της αξιοποίησης των δεδομένων στο πεδίο. Η αλληλεπίδραση του υποσυστήματος με το υπόλοιπο σύστημα του drone είναι καθοριστική για την επιτυχή εφαρμογή του στην παρακολούθηση συνθηκών καλλιέργειας και θέτει τα θεμέλια για μελλοντική επεκτασιμότητα σε πιο σύνθετες μορφές ανάλυσης και λήψης αποφάσεων.

3.2 Γενική Αρχιτεκτονική Συστήματος

Το ενσωματωμένο σύστημα αισθητήρων σχεδιάστηκε με σκοπό να λειτουργεί ως αυτόνομη μονάδα συλλογής και μετάδοσης δεδομένων, συνδεδεμένη λειτουργικά με τον αυτόματο πιλότο του drone. Η αρχιτεκτονική του συστήματος εστιάζει στη συνεργασία μεταξύ του μικροελεγκτή ESP32, των περιβαλλοντικών αισθητήρων και της διεπαφής με τον ελεγκτή πτήσης, προσφέροντας μία ολοκληρωμένη λύση για την παρακολούθηση συνθηκών καλλιέργειας κατά τη διάρκεια της πτήσης.

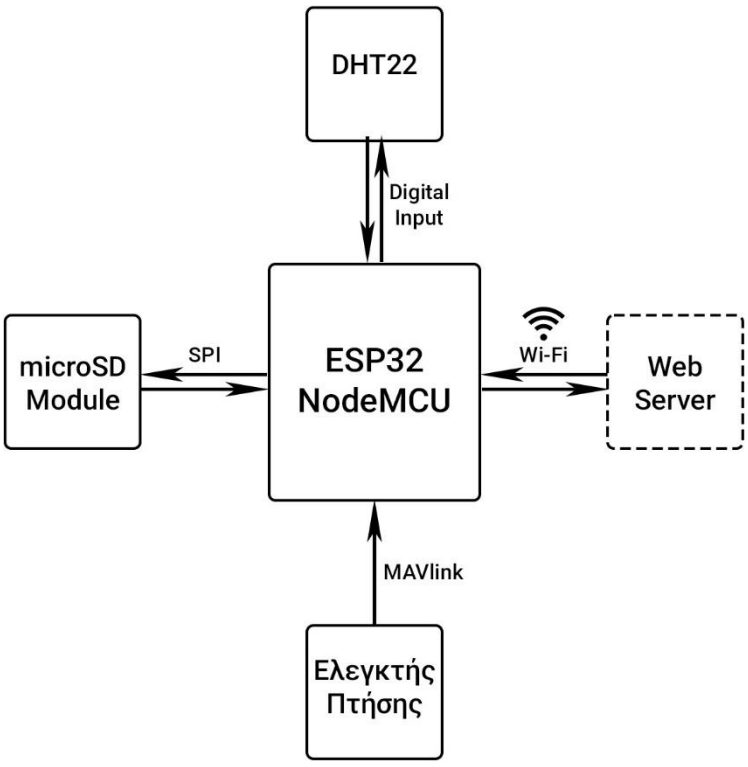
Στον πυρήνα της αρχιτεκτονικής βρίσκεται ο ESP32, ο οποίος αναλαμβάνει πολλαπλούς ρόλους:

- Διασύνδεση μέσω σειριακής θύρας UART με τον ελεγκτή πτήσης για τη λήψη δεδομένων MAVlink.
- Ανάγνωση περιβαλλοντικών μετρήσεων από τον αισθητήρα θερμοκρασίας και υγρασίας αέρα DHT22.

- Ανίχνευση αυτόνομης αποστολής και αποθήκευση δεδομένων μετρήσεων σε microSD.
- Ενσωμάτωση web server για τοπική απεικόνιση των δεδομένων μέσω Wi-Fi.

Η επικοινωνία με τον αυτόματο πιλότο του ελεγκτή πτήσης επιτυγχάνεται μέσω ενός απλού πρωτοκόλλου UART, επιτρέποντας την ανάγνωση βασικών πτητικών δεδομένων, όπως γεωγραφική θέση, ταχύτητα, υψόμετρο και τάση μπαταρίας. Οι περιβαλλοντικές μετρήσεις λαμβάνονται σε τακτά χρονικά διαστήματα και συγχρονίζονται με δεδομένα πτήσης, δημιουργώντας ένα ολοκληρωμένο σύνολο πληροφορίας που μπορεί να αξιοποιηθεί για γεωργικές εφαρμογές.

Η δομή του συστήματος είναι διαμορφωμένη έτσι ώστε να επιτρέπει την εύκολη ενσωμάτωσή του στο σκάφος, με διακριτό διαχωρισμό μεταξύ της συλλογής δεδομένων και της επικοινωνίας. Η υλοποίηση αυτή διασφαλίζει την αξιοπιστία, την επεκτασιμότητα και την προσαρμοστικότητα του συστήματος σε μελλοντικές βελτιώσεις, όπως η προσθήκη επιπλέον αισθητήρων ή η απομακρυσμένη μετάδοση δεδομένων μέσω άλλων πρωτοκόλλων.



Σχήμα 3.1: Μπλοκ διάγραμμα σύνδεσης του ενσωματωμένου συστήματος αισθητήρων

3.3 Επιλογή Υλικού (Hardware)

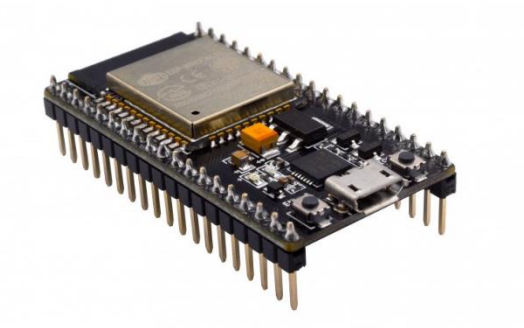
Η επιλογή των επιμέρους υλικών του συστήματος αισθητήρων έγινε με βάση την αξιοπιστία, τη διαθεσιμότητα, την ενεργειακή απόδοση και την ικανότητα των εξαρτημάτων να λειτουργούν σε εξωτερικές συνθήκες. Οι βασικές μονάδες εξαρτημάτων που χρησιμοποιήθηκαν είναι οι εξής:

3.3.1 NodeMCU-32S Module

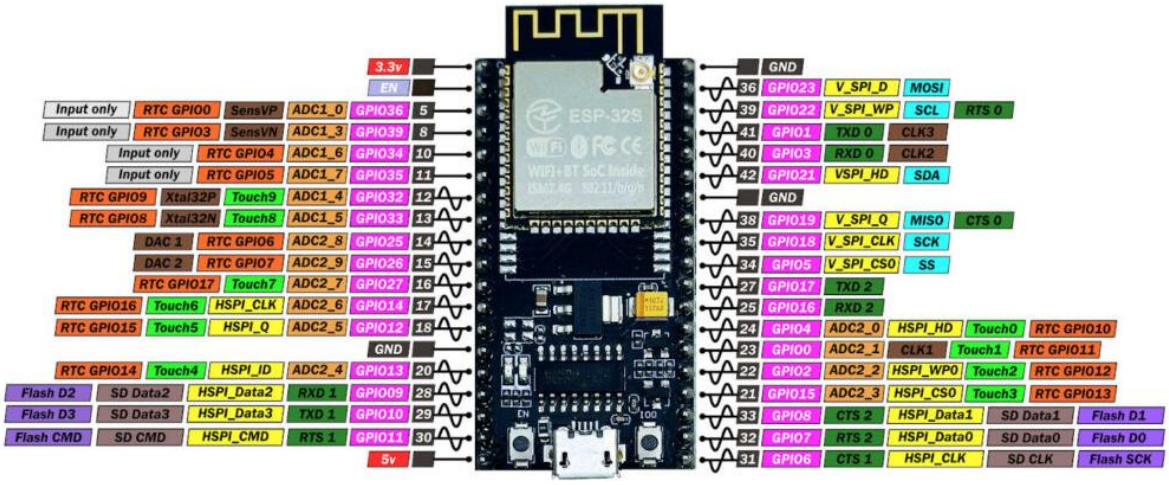
Ως κεντρική μονάδα επεξεργασίας επιλέχθηκε το NodeMCU-32S Module, μία πλακέτα ανάπτυξης που βασίζεται στον μικροελεγκτή ESP32 της Espressif. Η επιλογή αυτή έγινε λόγω της ευκολίας ενσωμάτωσης, της υποστήριξης Wi-Fi/Bluetooth και της πληθώρας διαθέσιμων θυρών εισόδου/εξόδου.

Τα βασικά χαρακτηριστικά της πλακέτας είναι:

- Διπύρηνος επεξεργαστής Tensilica Xtensa LX6, έως 240MHz.
- Ενσωματωμένες διεπαφές Wi-Fi και Bluetooth για ασύρματη επικοινωνία.
- Υποστήριξη διαύλων UART, SPI και I2C, που επιτρέπουν διασύνδεση με αισθητήρες και άλλα περιφερειακά.
- Ενσωματωμένος μετατροπέας USB-to-Serial, που διευκολύνει τον προγραμματισμό και την τροφοδοσία.
- Σχεδίαση με pins κατάλληλα για πειραματισμό σε breadboard και τοποθέτηση σε διάτρητες πλακέτες.



Εικόνα 3.1: Το ESP32 NodeMCU-32S DevKit
[https://www.smart-prototyping.com/image/cache/data/9_Modules/101839%20NodeMCU-32S%20Lua%20WiFi%20ESP32%20module/nodemcu-32s-lua-esp32-module-wifi-44305-750x750.jpg]



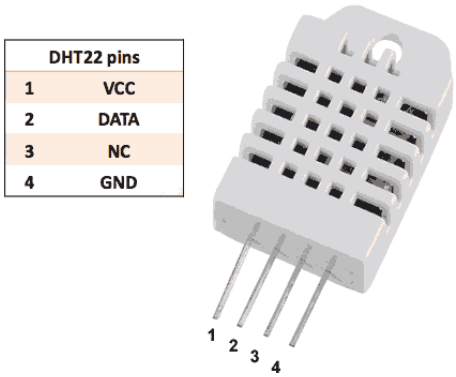
Εικόνα 3.2: Το διάγραμμα ακροδεκτών του NodeMCU-32S Module [https://mischianti.org/wp-content/uploads/2024/02/ESP32-NODEMCU-ESP-32S-Kit-pinout-low-res-mischianti-1024x599.jpg]

Ο συγκεκριμένος τύπος ESP32 προσφέρει τον απαραίτητο υπολογιστικό πυρήνα για την ταυτόχρονη συλλογή δεδομένων, την τοπική αποθήκευση, καθώς και την εξυπηρέτηση περιεχομένου μέσω web server κατά τη διάρκεια της πτήσης με ελάχιστη καθυστέρηση.

3.3.2 Αισθητήρας DHT22

Ο αισθητήρας DHT22 (AM2302) χρησιμοποιήθηκε για την καταγραφή της θερμοκρασίας και της σχετικής υγρασίας στον αγρό. Αποτελεί αξιόπιστη επιλογή για αγροτικές εφαρμογές, με τα εξής τεχνικά χαρακτηριστικά:

- Εύρος θερμοκρασίας: -40 έως 80°C, ακρίβεια ±0,5°C.
- Εύρος υγρασίας: 0-100% RH, ακρίβεια ±2-5%.
- Απλοποιημένη συνδεσμολογία με μία ψηφιακή έξοδο
- Χαμηλή κατανάλωση ισχύος και σταθερή λειτουργία σε εξωτερικές συνθήκες



Εικόνα 3.3: Ο αισθητήρας DHT22
[https://grobotronics.com/images/companies/1/DHT22-PinOut.png?1612948488234]

Η επιλογή του DHT22 εξασφαλίζει ικανοποιητική ακρίβεια για εφαρμογές γεωργίας ακριβείας, ιδιαίτερα σε αρχικά στάδια υλοποίησης.

3.3.3 MicroSD Card Module

Για την τοπική αποθήκευση δεδομένων κατά την διάρκεια πτήσης, χρησιμοποιήθηκε μονάδα κάρτας μνήμης microSD, η οποία συνδέθηκε με τον ESP32 μέσω διαύλου SPI. Η υλοποίηση αυτή εξυπηρετεί περιπτώσεις όπου:

- Δεν υπάρχει διαθέσιμο δίκτυο Wi-Fi
- Απαιτείται δημιουργία αρχείων με χρονική σήμανση (timestamped JSON)
- Το drone λειτουργεί αυτόνομα χωρίς άμεση σύνδεση με android smartphone ή web server

Εικόνα 3.4: Το microSD card module
[https://hubtronics.in/image/cache/catalog/sagar/Mini-SD-TF-Memory-Card-Module-1-550x550.jpg]

Η microSD επιτρέπει την μετέπειτα μεταφορά δεδομένων για απεικόνιση όταν το ζητήσει ο χρήστης, ενώ τα δεδομένα οργανώνονται με τρόπο κατάλληλο για μελλοντική επεξεργασία ή εισαγωγή σε βάσεις δεδομένων.

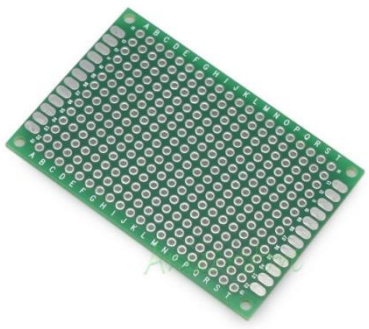
3.3.4 Επιπλέον Περιφερειακά

Πέρα από τα βασικά εξαρτήματα, χρησιμοποιήθηκαν και τα εξής περιφερειακά για την υποστήριξη της λειτουργίας του συστήματος:

- Diatone Mamba BEC 2A (5V): Χρησιμοποιήθηκε για την υποβίβαση της τάσης μπαταρίας του drone ώστε να τροφοδοτηθεί με αξιόπιστη και σταθερή τάση 5V το σύστημα αισθητήρων. Ο BEC εξασφαλίζει ότι ο ESP32 και οι περιφερειακές μονάδες τροφοδοτούνται με επαρκές ρεύμα χωρίς παρεμβολές ή διακυμάνσεις.

Εικόνα 3.5: Το Diatone Mamba BEC 2A (5V/9V)
[https://www.diatone.us/cdn/shop/products/DSC08454_800x.jpg?v=1602666038]

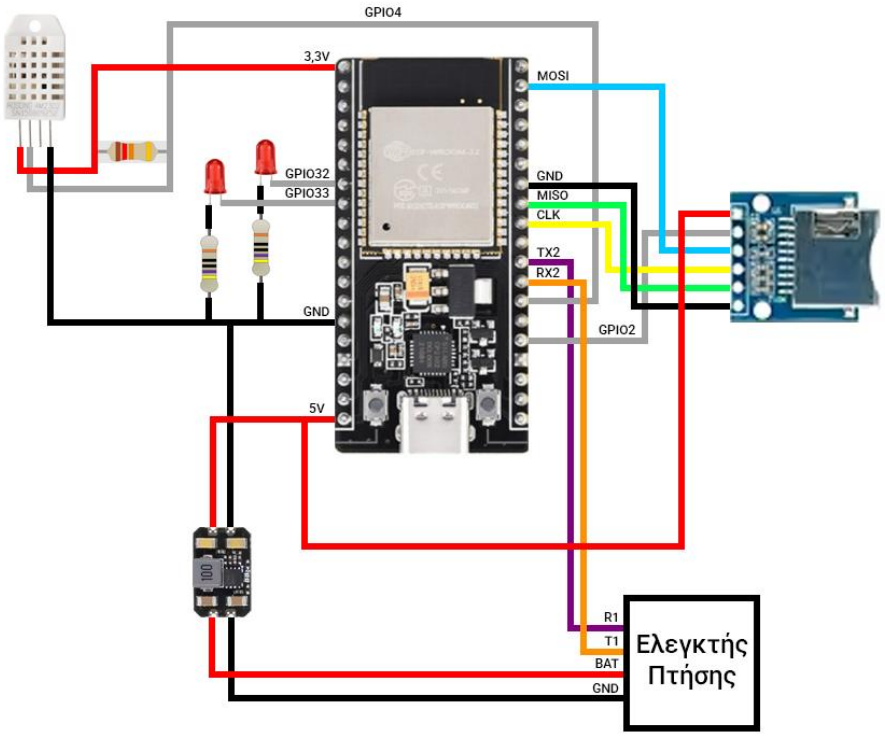
- LED ενδείξεων: Ενσωματώθηκαν για την ένδειξη της κατάστασης του συστήματος (σύνδεση με Wi-Fi, έλεγχος λειτουργίας περιφερειακών).
- Καλώδια σιλικόνης 24AWG: Χρησιμοποιήθηκαν για τις συνδέσεις του ESP32 με τα περιφερειακά, την τροφοδοσία και την επικοινωνία με τον ελεγκτή πτήσης.
- Διάτρητη πλακέτα PCB: Συγκολλήθηκαν όλα τα στοιχεία επάνω της και έγιναν οι κατάλληλες συνδέσεις, ώστε να εξασφαλιστεί σταθερότητα και ευκολία συντήρησης.
- Κουτί ηλεκτρονικών κυκλωμάτων: Χρησιμοποιήθηκε για την προστασία όλων των κυκλωμάτων.



Εικόνα 3.6: Τυπικές διάτρητες πλακέτες
[https://i.ebayimg.com/images/g/2O4AAOSwwNVTsBLx/s-l1200.jpg]

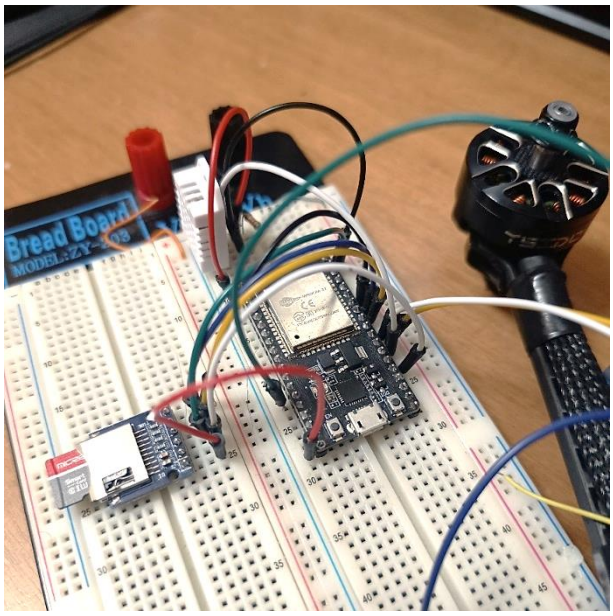
3.4 Διαδικασία Υλοποίησης του Ενσωματωμένου Συστήματος

Η σχεδίαση του κυκλώματος έγινε με γνώμονα την ευκολία ενσωμάτωσης στο πλαίσιο του drone, το χαμηλό βάρος και την εύκολη συντήρησή του. Επίσης, η επιλογή των pins και του τροπου σύνδεσης όλης της διάταξης ήταν κρίσιμη ώστε το σύστημα να έχει μικρές διαστάσεις που να διευκολύνουν την ενσωμάτωση στο drone. Στην εικόνα 3.7 παρουσιάζεται η σύνδεση – κύκλωμα του συστήματος με τα γύρω περιφεριακά:



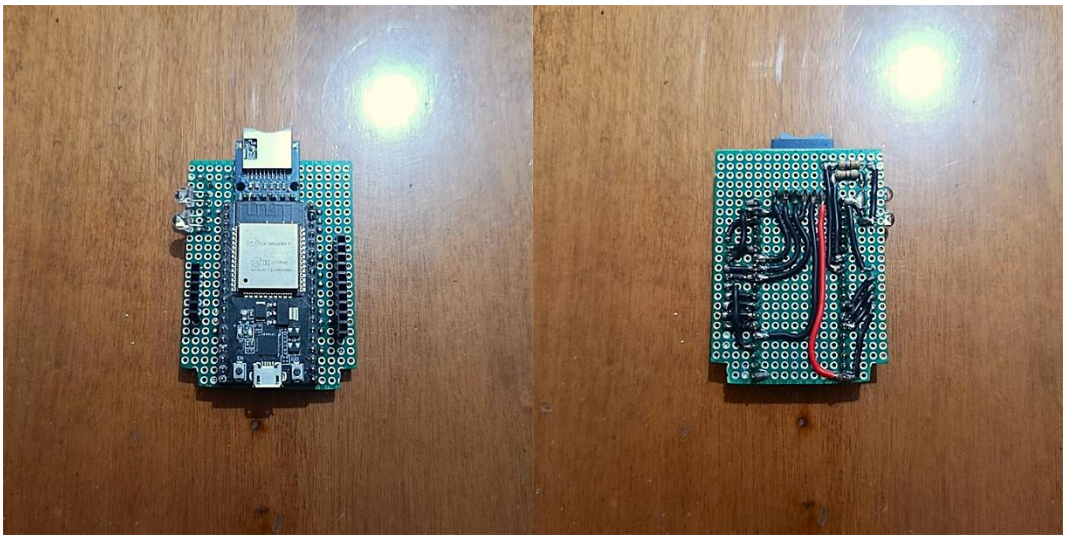
Εικόνα 3.7: Διάγραμμα συνδέσεων του ESP32 με τα περιφερειακά και τον ελεγκτή πτήσης

Αρχικά, το κύκλωμα σχεδιάστηκε σε breadboard, όπως φαίνεται στην εικόνα 3.8, για τον έλεγχο της ορθής λειτουργίας μεταξύ του ESP32 και των περιφερειακών του (SD module, DHT22, επικοινωνία με ελεγκτή πτήσης).



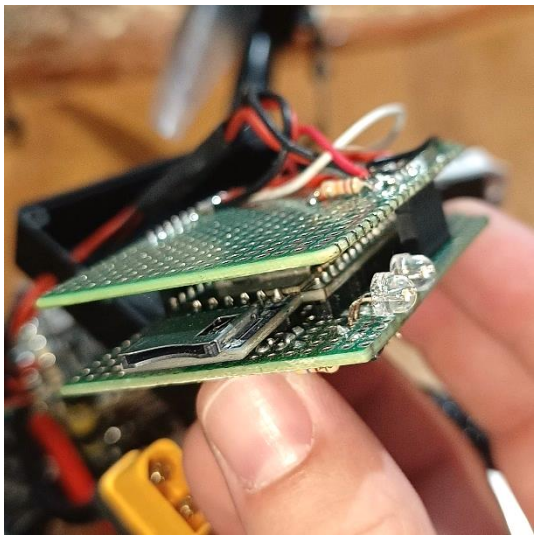
Εικόνα 3.8: Σχεδιασμός του κυκλώματος σε breadboard

Στην συνέχεια, το κύκλωμα έπρεπε να μετατραπεί σε μία πιο μόνιμη και στιβαρή κατασκευή. Για την εύκολη μελλοντική επέκταση του συστήματος, το κύκλωμα χωρίστηκε σε δύο μέρη. Το πρώτο μέρος αφορά τον μικροελεγκτή και το σύστημα αποθήκευσης στην κάρτα SD, ενώ το δεύτερο μέρος αφορά την ενσωμάτωση αισθητήρων και άλλων περιφερειακών. Για το λόγο αυτό, χρησιμοποιήθηκαν δύο διάτρητες πλακέτες, οι οποίες κόπηκαν στο σχήμα του κουτιού προστασίας με σκοπό να στοιβαχθούν η μία πάνω στην άλλη (stack). Η πρώτη, περιλαμβάνει τον ESP32, το microSD card module, ενδείκτες LED και θυληκά headers για τη σύνδεση της δεύτερης πλακέτας, όπως φαίνεται στην εικόνα 3.9:



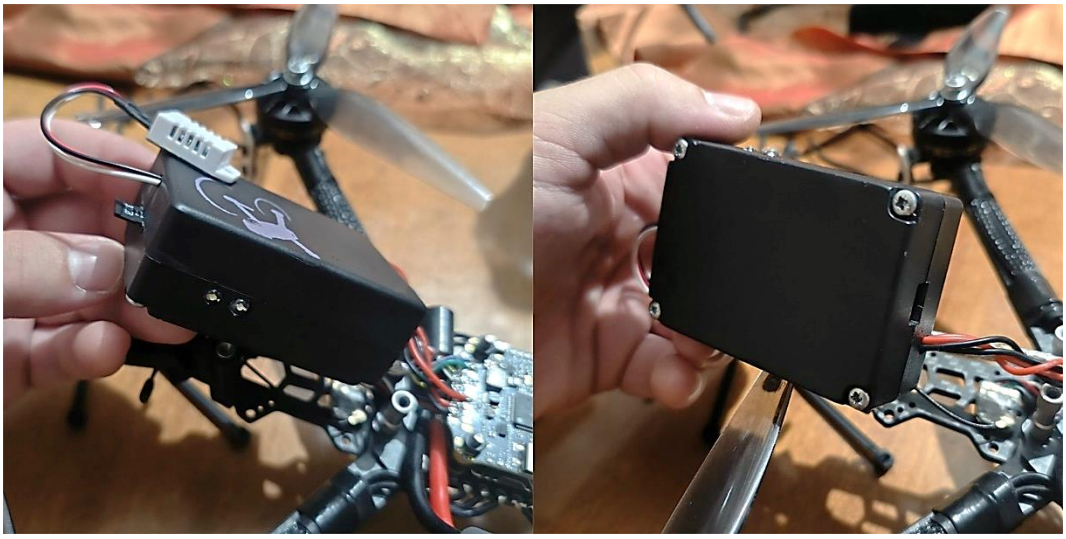
Εικόνα 3.9: Η σχεδίαση της πλακέτας με το βασικό σύστημα του μικροελεγκτή και τοπικής αποθήκευσης

Η δεύρερη πλακέτα συνδέθηκε με την βοήθεια αρσενικών pin headers πάνω στην άλλη. Τα pins συνδέονται με θύρες εξόδου τροφοδοσίας και γείωσης του ESP32, διαθέσιμα GPIO pins, γραμμές I2C και SPI για την εύκολη μελλοντική επέκταση του συστήματος με νέους αισθητήρες και υποσυστήματα. Στην εικόνα 3.10 φαίνεται το συνολικό ενσωματωμένο σύστημα με τις δύο πλακέτες ενωμένες.



Εικόνα 3.10: Το stack του συστήματος αισθητήρων

Έπειτα, όλο το κύκλωμα τοποθετήθηκε μέσα στο κουτί ηλεκτρονικών κυκλωμάτων μεγέθους 40x60mm, για την προστασία του. Το κουτί διαμορφώθηκε με ωπές, τέτοιες ώστε να υπάρχει εξωτερική πρόσβαση στην θύρα USB του ESP32 και την κάρτα SD, όπως επίσης τρυπήθηκαν ωπές για την εμφάνιση των LED και για διαχείριση των καλωδίων. Ο αισθητήρας DHT22 συνδέθηκε με καλώδια εκτός του κουτιού ώστε η μέτρησή του να είναι σωστή. Στην εικόνα 3.11 φαίνεται το τελικό σύστημα:



Εικόνα 3.11: Το τελικό ενσωματωμένο σύστημα αισθητήρων

Για την ορθή τοποθέτηση του συστήματος στο πλαίσιο, κρίθηκε απαραίτητο να τοποθετηθεί όσο το δυνατόν πιο κοντά στο κέντρο βάρους του drone, ώστε να μην επηρεάζεται η σταθερότητα και η απόδοση της πτήσης. Το κατάλληλο σημείο βρέθηκε στην κάτω πλευρά του πλαισίου, η οποία ήταν ελεύθερη. Ωστόσο, για να καταστεί εφικτή η τοποθέτηση σε αυτό το σημείο, ήταν αναγκαίο το drone να διαθέτει απόσταση περίπου 5 cm από το έδαφος κατά την προσγείωση. Για τον σκοπό αυτό, κατασκευάστηκαν τέσσερα σκέλη προσγείωσης, τα οποία βιδώθηκαν στο πλαίσιο κάτω από τους τέσσερις κινητήρες, όπως φαίνεται στην εικόνα 3.12. Το σύστημα τοποθετήθηκε σταθερά στο πλαίσιο με την βοήθεια πλαστικών δεματικών (tire-ups), ενώ ο αισθητήρας DHT22 τοποθετήθηκε με ταινία διπλής όψεος στην μπροστινή μεριά του πλαισίου, όπως φαίνεται στην εικόνα 3.13. Τέλος, συνδέθηκε το σύστημα στον ελεγκτή πτήσης μέσω της θύρας UART1. Η τροφοδοσία του ESP32 γίνεται μέσω της μπαταρίας του drone, από το αντίστοιχο pad (BAT) που διαθέτει ο SpeedyBee F405.

Εικόνα 3.12: Τα σκέλη προσγείωσης του drone

Εικόνα 3.13: Το σημείο τοποθέτησης του ενσωματωμένου συστήματος αισθητήρων

3.5 Επικοινωνία με το Drone μέσω MAVlink

Η ενσωμάτωση του αισθητηριακού συστήματος στο drone απαιτεί μία σταθερή και αμφίδρομη επικοινωνία μεταξύ του ESP32 και του ελεγκτή πτήσης SpeedyBee F405 v.4. Για το σκοπό αυτό, επιλέχθηκε το πρωτόκολλο MAVlink (Micro Air Vehicle Link), το οποίο αποτελεί το πιο διαδεδομένο πρότυπο για την επικοινωνία μεταξύ αυτόνομων αεροσκαφών και εξωτερικών συσκευών ή σταθμών βάσης.

3.5.1 MAVlink Protocol

Το MAVlink είναι ένα ελαφρύ πρωτόκολλο μηνυμάτων σχεδιασμένο για την ανταλλαγή δεδομένων μεταξύ UAVs και συσκευών εδάφους ή πρόσθετων υποσυστημάτων.

Η επικοινωνία γίνεται μέσω δυαδικών μηνυμάτων για αυξημένη ταχύτητα και μειωμένο όγκο δεδομένων. Το πρωτόκολλο υποστηρίζει τηλεμετρία, στέλνοντας δεδομένα πλοήγησης, κατάστασης συστήματος, αισθητήρων και χειρισμού. Το MAVlink είναι συμβατό με πλατφόρμες όπως ArduPilot και PX4, ενώ το INAV ξεκίνησε να υποστηρίζει το πρωτόκολλο από την έκδοση 8.0 και μετά.

3.5.2 Διασύνδεση ESP32 με τον Ελεγκτή Πτήσης

Ο ESP32 NodeMCU συνδέθηκε με τον ελεγκτή πτήσης (FC) του drone μέσω σειριακής θύρας UART, όπως παρακάτω:

- TX2 (ESP32) στο R1 (FC)
- RX2 (ESP32) στο T1 (FC)

- GND (ESP32) στο GND (FC) μέσω του BEC 5V/2A
- 5V (ESP32) στο BAT (FC) μέσω του BEC 5V/2A

Η τάση της μπαταρίας μέσω του BAT pad στον ελεγκτή πτήσης τροφοδοτεί το BEC, το οποίο υποβιβάζει την τάση στα 5V για να μπορέσει να τροφοδοτήσει τον ESP32.

Μέσα στο INAV Configurator 8.0.1, στην καρτέλα Ports θα πρέπει να ενεργοποιηθεί η θύρα UART1 επιλέγοντας από την στήλη Telemetry το MAVlink. Η επικοινωνία λειτουργεί στα 57600 baud rate, που αποτελεί συχνή επιλογή για τηλεμετρία MAVlink, ισορροπώντας ταχύτητα και αξιοπιστία.

Εικόνα 3.14: Ρύθμιση της θύρας UART1 για τηλεμετρία MAVlink

Η κατεύθυνση της επικοινωνίας είναι μονόδρομη (read-only) από τον ελεγκτή πτήσης προς τον ESP32 λόγω περιορισμένης υποστήριξης του πρωτοκόλλου MAVlink από το INAV. Ωστόσο, η αρχιτεκτονική του λογισμικού επιτρέπει μελλοντικά την υποστήριξη και εντολών χειρισμού (αλλαγή τρόπου πτήσης, δημιουργία αυτόνομων ελιγμών για αποφυγές εμποδίων).

3.6 Συλλογή και Επεξεργασία Περιβαλλοντικών Δεδομένων

Η μέτρηση περιβαλλοντικών παραμέτρων, όπως η θερμοκρασία και η σχετική υγρασία, αποτελεί ουσιώδες μέρος της εφαρμογής του drone στη γεωργία ακριβείας. Οι πληροφορίες αυτές παρέχουν στους αγρότες σημαντικά δεδομένα για την παρακολούθηση της μικροκλιματικής κατάστασης των καλλιεργειών, σε πραγματικό χρόνο, ενισχύοντας τη δυνατότητα λήψης αποφάσεων.

3.6.1 Σύνδεση αισθητήρα DHT22 με τον ESP32

Ο αισθητήρας DHT22 συνδέθηκε με τον ESP32 ως εξής:

- VCC (DHT22) στο 3,3V (ESP32)
- Data (DHT22) στο GPIO4 (ESP32)
- Ο τρίτος ακροδέκτης του DHT22 δεν χρησιμοποιείται
- GND (DHT22) στο GND (ESP32)

Ο ακροδέκτης Data παράγει ψηφιακό σήμα και θα μπορούσε να συνδεθεί σε ψηφιακή είσοδο του ESP32, παρόλα αυτά επιλέχθηκε το γενικής χρήσης pin GPIO4 λόγω διαθεσιμότητας και ευκολότερης σύνδεσης. Η τροφοδοσία του αισθητήρα παρέχεται από την έξοδο των 3,3V του ESP32, ενώ χρησιμοποιήθηκε εξωτερική pull-up αντίσταση 12kΩ, όπως προδιαγράφεται από τον κατασκευαστή.

3.6.2 Ανάπτυξη Κώδικα στον ESP32

Για την εύκολη ενσωμάτωση της λειτουργικότητας του αισθητήρα στον κώδικα του ESP32, χρησιμοποιήθηκε η DHT.h βιβλιοθήκη.

Παρακάτω παρατίθεται απόσπασμα του κώδικα στον ESP32 με την αρχικοποίηση του αισθητήρα, των μεταβλητών και την διαδικασία ανάγνωσης των τιμών του αισθητήρα:

```
#include <DHT.h>
float temp = 0.0;                    // Αρχικοποίηση μεταβλητής θερμοκρασίας
float hum = 0.0;                     // Αρχικοποίηση μεταβλητής υγρασίας

// Ρυθμίσεις για τον DHT αισθητήρα
#define DHTPIN 4                     // Ο ακροδέκτης στο οποίο είναι συνδεδεμένος ο αισθητήρας
#define DHTTYPE DHT22                // Ο τύπος του αισθητήρα
DHT dht(DHTPIN, DHTTYPE);            // Δήλωση αισθητήρα

void setup() {                       // Αρχή της setup
  Serial.begin(115200);              // Εκίννηση του serial monitor
    dht.begin();                     // Αρχικοποίηση DHT
  delay(1000);                       // Καθυστέρηση για αποφυγή λάθος τιμών κατά την εκίννηση
  temp = dht.readTemperature();      // Ανάγνωση τιμής θερμοκρασίας και αποθήκευση σε μεταβλητή
  if (temp == 0) {                   // Έλεγχος αισθητήρα
    Serial.println("DHT Sensor Init Failed!");

}
void loop() {                        // Αρχή της loop
  temp = dht.readTemperature();      // Ανάγνωση τιμής θερμοκρασίας και αποθήκευση σε μεταβλητή
  hum = dht.readHumidity();  // Ανάγνωση τιμής υγρασίας και αποθήκευση σε μεταβλητή
}
```

Η υλοποίηση περιλαμβάνει βασικό έλεγχο του αισθητήρα μέσω logging για αποσφαλμάτωση (debugging).

3.7 Ανάγνωση και Διαχείριση Δεδομένων Πτήσης μέσω MAVlink

Ο ESP32 λειτουργεί ως παθητικός ακροατής στο MAVlink stream που στέλνει ο ελεγκτής πτήσης, διαβάζοντας και αποκωδικοποιώντας συγκεκριμένους τύπους πακέτων.

3.7.1 Ανάπτυξη κώδικα στον ESP32

Για την διαχείριση των πακέτων MAVlink στον ESP32 χρησιμοποιήθηκε έτοιμη βιβλιοθήκη σε γλώσσα C (MAVlink.h). Η επιλογή των πακέτων έγινε μέσω απλού φιλτραρίσματος μηνυμάτων, βασισμένου στο message ID και το component ID του αποστολέα. Η υλοποίηση της διαχείρισης των MAVlink γίνεται μέσω της συνάρτησης handleMavlinkMessages(), η οποία στην συνέχεια καλείται μέσα στην loop().

Ο ESP32 προγραμματίστηκε να φιλτράρει πέντε είδη μηνυμάτων για ανάγνωση εφτά δεδομένων πτήσης που αποθηκεύονται σε εφτά αντίστοιχες μεταβλητές. Τα μηνύματα που φιλτράρονται είναι τα εξής:

- MAVLINK_MSG_ID_HEARTBEAT: Περιέχει δεδομένα κατάστασης drone (isAuto)
- GLOBAL_POSITION_INT: Περιέχει δεδομένα GPS (lat, lon, alt)
- GPS_RAW_INT: Περιέχει δεδομένα συνδεδεμένων δορυφόρων (sats)
- SYS_STATUS: Περιέχει δεδομένα κατάστασης μπαταρίας (vBat)
- VFR_HUD: Περιέχει δεδομένα ταχύτητας (speed)

Αρχικά, γίνεται έλεγχος νέου μηνύματος και όταν ανιχνευθεί, μέσω μίας switch ελέγχεται αν το μήνυμα είναι ένα από τα τέσσερα παραπάνω. Εφόσον ανιχνευθεί επιθυμητό μήνυμα, εκτελείται το αντίστοιχο case και αποθηκεύονται οι μεταβλητές που θέλουμε. Τα μηνύματα GLOBAL_POSITION_INT, GPS_RAW_INT, SYS_STATUS και VFR_HUD ανιχνεύονται για απλή αποθήκευση των δεδομένων σε μεταβλητές. Το μήνυμα MAVLINK_MSG_ID_HEARTBEAT χρησιμοποιείται κύριως για την ανίχνευση αυτόνομης αποστολής του drone και την αποθήκευση των δεδομένων στην κάρτα SD που θα εξηγηθεί στην επόμενη ενότητα.

Το Heartbeat μήνυμα έχει ID = 0 και μεταφέρει τον custom_mode, ο οποίος είναι ένας δυαδικός αριθμός από 32 bit (0=LSB, 31=MSB). Κάθε bit, που συνήθως αναφέρονται σαν flags, δηλώνει μία κατάσταση στην οποία βρίσκεται το drone. Το bit3 (0x00000008) αναφέρεται στην κατάσταση αυτόνομης πτήσης του drone και είναι αυτό που χρησιμοποιούμε για να ανιχνεύσουμε αν το drone εκτελεί αυτόνομη αποστολή.

Ο παρακάτω κώδικας δείχνει πώς γίνεται η αρχικοποίηση της UART θύρας, των μεταβλητών και το parsing/ανίχνευση MAVlink μηνυμάτων στον ESP32:

```
#include <MAVLink.h>
#include <HardwareSerial.h>


unsigned long lastSave = 0;                 // Χρονόμετρο για περιοδική αποθήκευση
bool isAuto = false;                        // Για έλεγχο αυτόνομης πτήσης
float temp = 0.0f, hum = 0.0f, lat = 0.0f,          // Αρχικοποίηση μεταβλητών δεδομένων
```

```
lon = 0.0f, alt = 0.0f, vBat = 0.0f, speed = 0.0f, climbRate = 0.0f;

// Αρχικοποίηση των UART pins
#define RX_PIN 16                    // RX2 του ESP32 (συνδέεται στο T1 του F405)
#define TX_PIN 17                    // TX2 του ESP32 (συνδέεται στο R1 του F405)
HardwareSerial mavSerial(1);                    // Χρησιμοποιούμε το Serial1 για MAVLink

void handleMavlinkMessages() {                  // Διαβάζει δεδομένα από MAVLink
 mavlink_message_t msg;
 mavlink_status_t status;

 while (mavSerial.available()) {
  uint8_t c = mavSerial.read();                 // Ανάγνωση εισερχόμενων μηνυμάτων
  if (mavlink_parse_char(MAVLINK_COMM_0, c, &msg, &status)) {       // Parsing χαρακτήρων μηνύματος
   Serial.print("Received MAVLink message: ");
   Serial.println(msg.msgid);                   // Εκτύπωση του ID του μηνύματος
   switch (msg.msgid) {                         // Έλεγχος τύπου μηνύματος
                                                // Έλεγχος αυτόνομης πτήσης
    case MAVLINK_MSG_ID_HEARTBEAT: {
     mavlink_heartbeat_t hb;
     mavlink_msg_heartbeat_decode(&msg, &hb);
     bool autoFlight = (hb.custom_mode == 3);          // Έλεγχος waypoint mission mode του drone

     if (autoFlight != isAuto) {
      isAuto = autoFlight;
      Serial.println(isAuto ? "Waypoints mission ENABLED" : "Waypoints mission DISABLED");
      if (isAuto) {
       lastSave = millis();             // Μηδενισμός χρονομέτρου κατά ενεργοποίηση
      }
     }
     break;
    }
    // Δεδομενα GPS
    case MAVLINK_MSG_ID_GLOBAL_POSITION_INT: {
     mavlink_global_position_int_t pos;
     mavlink_msg_global_position_int_decode(&msg, &pos);
     lat = pos.lat / 1.0e7;                     // Αποθήκευση γεωγραφικού πλάτους
     lon = pos.lon / 1.0e7;             // Αποθήκευση γεωγραφικού μήκους
     alt = pos.relative_alt / 1000.0;   // Αποθήκευση σχετικού υψόμετρου σε μέτρα
     break;
    }
    // Αριθμός Δορυφόρων
    case MAVLINK_MSG_ID_GPS_RAW_INT: {
```

```
        mavlink_gps_raw_int_t gps;
        mavlink_msg_gps_raw_int_decode(&msg, &gps);
        sats = gps.satellites_visible;
        break;
       }
       // Τάση μπαταρίας
       case MAVLINK_MSG_ID_SYS_STATUS: {
        mavlink_sys_status_t sys;
        mavlink_msg_sys_status_decode(&msg, &sys);
        vBat = sys.voltage_battery / 1000.0;          // Αποθήκευση τάσης μπαταρίας σε Volts
        break;
       }
       // Ταχύτητα
       case MAVLINK_MSG_ID_VFR_HUD: {
        mavlink_vfr_hud_t hud;
        mavlink_msg_vfr_hud_decode(&msg, &hud);
        speed = hud.groundspeed;                      // Αποθήκευση ταχύτητας εδάφους σε m/s
        break;
       }
     }
   }
 }
}
void setup() {                                        // Αρχή της setup
 Serial.begin(115200);                                // Εκίνηση του serial monitor
 mavSerial.begin(57600, SERIAL_8N1, RX_PIN, TX_PIN);          // Αρχικοποίηση MAVlink επικοινωνίας
}
void loop() {                                         // Αρχή της loop
 handleMavlinkMessages();              // Κλήση της συνάρτησης ανάγνωσης μηνυμάτων MAVlink
}
```

Το κομμάτι κώδικα για ανίχνευση αυτόνομης πτήσης χρησιμοποιείται για την περιοδική αποθήκευση των δεδομένων στην κάρτα SD όταν το drone πραγματοποιεί μία αυτόνομη αποστολή. Η τοπική αποθήκευση των δεδομένων καλύπτεται στην επόμενη ενότητα.

3.8 Τοπική Αποθήκευση Δεδομένων

Η δυνατότητα τοπικής αποθήκευσης των συλλεγόμμενων δεδομένων ενισχύει σημαντικά την αξιοπιστία και την πληρότητα της καταγραφής σε εφαρμογές γεωργίας ακριβείας, καθώς διασφαλίζει ότι συμαντικές πληροφορίες παραμένουν διαθέσιμες σε περίπτωση απώλειας σήματος ή κατά τη διάρκεια offline λειτουργίας. Στο πλαίσιο της παρούσας υλοποίησης, η αποθήκευση γίνεται σε αρχείο

κειμένου (.txt) στην κάρτα microSD και πραγματοποιείται περιοδικά κατά τη διάρκεια πτήσεων σε λειτουργία waypoint mission.

3.8.1 Ενσωμάτωση του MicroSD Module στον ESP32

Για την τοπική αποθήκευση χρησιμοποιήθηκε ένα συμβατό microSD module μικρών διαστάσεων, το οποίο συνδέθηκε στον ESP32 μέσω του διαύλου SPI (Serial Peripheral Interface). Οι γραμμές σύνδεσης που χρησιμοποιήθηκαν περιλαμβάνουν:

- MOSI (Master Out Slave In)
- MISO (Master in Slave Out)
- SCK (Serial Clock)
- CS (Chip Select)
- 5V
- GND

3.8.2 Λογική και Μορφή Αποθήκευσης

Η λογική αποθήκευσης ενεργοποιείται μόνο όταν το drone βρίσκεται σε λειτουργία αυτόνομης πτήσης (Auto mode) και εκτελεί προγραμματισμένη αποστολή με waypoints. Η ανίχνευση της λειτουργίας αυτής γίνεται μέσω της λήψης δεδομένων MAVlink από τον ελεγκτή πτήσης, όπου αναγνωρίζεται το αντίστοιχο flight mode.

Μόλις εντοπιστεί η είσοδος σε waypoint mission, το σύστημα ξεκινά την εγγραφή δεδομένων ανά διαστήματα 2 δευτερολέπτων. Οι μετρήσεις που αποθηκεύονται περιλαμβάνουν:

- Χρόνος από την εκίννηση (Timestamp)
- Θερμοκρασία (DHT22)
- Υγρασία (DHT22)
- Γεωγραφικές συντεταγμένες (GPS)
- Ύψος

Η αποθήκευση γίνεται σε αρχείο με κατάληξη .txt, ωστόσο η δομή των δεδομένων ακολουθεί την JSON μορφοποίηση, καθιστώντας τα δεδομένα ευανάγνωστα τόσο από άνθρωπο όσο και από λογισμικό.

Παρακάτω ακολουθεί ένα παράδειγμα καταγραφής όπως θα φαινόταν στο αρχείο datalog.txt:

{"timestamp":12,"temperature":33.10,"humidity":42.60,"latitude":40.664619,"longitude":22.947302,"altitude":5.25}
{"timestamp":14,"temperature":33.10,"humidity":42.60,"latitude":40.664623,"longitude":22.947296,"altitude":5.12}
{"timestamp":16,"temperature":33.40,"humidity":42.00,"latitude":40.664539,"longitude":22.947258,"altitude":4.94}

Τα πλεονεκτήματα της μεθόδου περιοδικής καταγραφής δεδομένων σε JSON κατά τη διάρκεια αυτόνομης αποστολής είναι:

- Ευκολία στην ανάλυση λόγω JSON μορφής δεδομένων
- Πληρότητα και συνέπεια κατά την καταγραφή
- Αποφυγή περιττών εγγραφών σε άλλες φάσης της πτήσης

3.8.3 Ανάπτυξη Κώδικα στον ESP32

Για την ανάπτυξη του κώδικα της τοπικής αποθήκευσης των δεδομένων χρησιμοποιήθηκαν οι βιβλιοθήκες SD.h, για τον έλεγχο των αρχείων της SD και η SPI.h, για την δυνατότητα χρήσης του διαύλου SPI. Δημιουργήθηκαν δύο κύριες συναρτήσεις, η initSDCard() για την αρχικοποίηση της κάρτας SD και η saveToSD() για την αποθήκευση των δεδομένων. Ωστόσο, διαχείριση των αρχείων χρησιμοποιείται και σε άλλα σημεία του κώδικα που πραγματεύονται την αποστολή των δεδομένων ή την διαγραφή τους, τα οποία θα καλυφθούν στην ενότητα 3.9.

Παρακάτω ακολουθεί ο κώδικας αρχικοποίησης και αποθήκευσης των δεδομένων στην κάρτα SD:

```
#include <SD.h>
#include <SPI.h>
// Global μεταβλητές
const unsigned long saveInterval = 2000;          // Για την αποθήκευση δεδομένων ανά 2 δευτερόλεπτα

#define SD_CS 2                                   // To Chip Select (CS) της SD κάρτας

bool initSDCard() {                               // Συνάρτηση αρχικοποίησης SD card
  int attempts = 5;                          // Μέγιστος αριθμός προσπαθειών
    while (attempts--) {
      if (SD.begin(SD_CS)) {
        Serial.println("SD card ready.");
        return true;                              // Επιτυχής αρχικοποίηση
      }
      Serial.println("SD initialization failed, retrying...");
      delay(1000);                                // Επανάληψη αρχικοποίησης
    }
  Serial.println("SD card initialization failed after multiple attempts.");
  return false;                                   // Αποτυχία αρχικοποίησης μετά από 5 προσπάθειες
}

// Συνάρτηση αποθήκευσης δεδομένων στην SD σε JSON μορφή
void saveToSD(float temp, float hum, float lat, float lon, float alt) {
  File txtFile = SD.open("/datalog.txt", FILE_APPEND);          // Άνοιγμα ή δημιουργία αρχείου
  if (!txtFile) {
```

```
  Serial.println("Error: Failed to open SD card file!");                    // Σφάλμα ανοίγματος αρχείου
  return;
 }
 txtFile.print("{\"timestamp\":");
 txtFile.print(millis() / 1000);              // Καταγραφή timestamp
 txtFile.print(",\"temperature\":");
 txtFile.print(temp, 2);                                  // Καταγραφή θερμοκρασίας
 txtFile.print(",\"humidity\":");
 txtFile.print(hum, 2);                                   // Καταγραφή υγρασίας
 txtFile.print(",\"latitude\":");
 txtFile.print(lat, 6);                                   // Καταγραφή γεωγραφικού πλάτους
 txtFile.print(",\"longitude\":");
 txtFile.print(lon, 6);                                   // Καταγραφή γεωγραφικού μήκους
 txtFile.print(",\"altitude\":");
 txtFile.print(alt, 2);                                   // Καταγραφή ύψους
 txtFile.println("}");
 txtFile.close();                                         // Κλείσιμο αρχείου
 Serial.println("Data logged successfully.");
}

void setup() {                                            // Αρχή της setup()
 Serial.begin(115200);                                    // Εκκίνηση του Serial monitor
 initSDCard();                                            // Αρχικοποίηση SD κάρτας
}

void loop() {                                             // Αρχή της loop()
 // Αποθήκευση κάθε 2 δευτερόλεπτα αν είναι σε auto το drone
 if (isAuto && (millis() - lastSave >= saveInterval)) {
  temp = dht.readTemperature();                                    // Ανανέωση τιμής θερμοκρασίας
  hum = dht.readHumidity();                               // Ανανέωση τιμής υγρασίας
  if (!isnan(temp) && !isnan(hum)) {                      // Έλεγχος αισθητήρα
   saveToSD(temp, hum, lat, lon, alt);                    // Κλήση συνάρτησης αποθήκευσης
   Serial.println("Data saved to SD (auto mode)");
  } else {
   Serial.println("Failed to read DHT sensor");
  }
  lastSave = millis();                                             // Μηδενισμός χρονομέτρου δευτερολέπτων
 }
}
```

Στον παραπάνω κώδικα παρουσιάζεται η αρχικοποίηση της SD κάρτας και η διαδικασία αποθήκευσης των δεδομένων σε μορφή JSON στο αρχείο datalog.txt. Η αρχικοποίηση του αισθητήρα DHT22 και της

επικοινωνίας MAVlink έχουν καλυφθεί στις προηγούμενες ενότητες και έχουν παραλειφθεί εδώ για καλύτερη ανάγνωση του κώδικα.

Αρχικά, γίνονται πέντε προσπάθειες εύρεσης της κάρτας SD για λόγους αποσφαλμάτωσης (debugging). Στη συνέχεια, μέσα στην loop ελέγχεται η κατάσταση αυτόνομης πτήσης (μεταβλητή isAuto) και συγκρίνεται με τον χρόνο από την προηγούμενη αποθήκευση (μεταβλητή lastSave), τιμές οι οποίες προκύπτουν μέσω της συνάρτησης handleMavlinkMessages() που καλύφθηκε στην ενότητα 3.6.1 . Αν το drone βρίσκεται σε κατάσταση Auto και έχει περάσει το διάστημα των δύο δευτερολέπτων (μεταβλητή saveInterval) από την προηγούμενη αποθήκευση, καλείται η συνάρτηση saveToSD() ώστε να προσθέσει τα νέα δεδομένα στην SD σε μορφή JSON.

3.9 Web Server και Αποστολή Δεδομένων

Η υλοποίηση ενός ενσωματωμένου web server στον NodeMCU ESP32 επιτρέπει την ασύρματη δίαθεση των δεδομένων που συλλέγονται από το σύστημα αισθητήρων, χωρίς την ανάγκη ενσύρματης μεταφοράς ή εξωτερικών υποδομών. Με αυτόν τον τρόπο, τα δεδομένα θερμοκρασίας, υγρασίας και πτήσης γίνονται άμεσα διαθέσιμα για ανάγνωση ή λήψη από την Android εφαρμογή που θα παρουσιαστεί στο επόμενο κεφάλαιο.

3.9.1 Αρχιτεκτονική Λειτουργίας

Η αρχιτεκτονική που ακολουθείται χωρίζεται σε τέσσερα βασικά μέρη:

1. Σύνδεση ESP32 σε δίκτυο Wi-Fi

   Το σύστημα ξεκινά σε λειτουργία πελάτη (STA mode) και συνδέεται σε προκαθορισμένο SSID με χρήση σταθερών στοιχείων πρόσβασης (SSID, password).

2. Δημιουργία Web server

   Ο ESP32 λειτουργεί ως HTTP server, ακούγοντας σε συγκεκριμένη θύρα (80).

3. Διάθεση δεδομένων σε μορφή JSON

   Ο web server παρέχει endpoints (π.χ. /sensor_data), τα οποία είτε επιστρέφουν σύνολα δεδομένων σε JSON, είτε εκτελούν κάποια διαδικασία (π.χ. διαγραφή δεδομένων SD).

4. Αίτηση και λήψη δεδομένων από την εφαρμογή

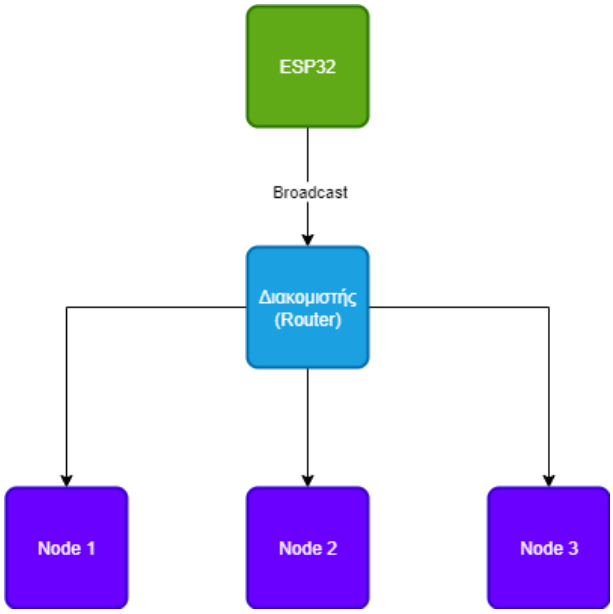
   Η εφαρμογή-πελάτης μπορεί να εκτελεί αιτήματα HTTP GET για να λαμβάνει ενημερωμένα δεδομένα, χωρίς να απαιτείται φυσική πρόσβαση στο drone ή την κάρτα SD.

Η χρήση web server ενσωματώνει το σύστημα σε τρίτες εφαρμογές ανοίγοντας τον δρόμο για την απεικόνιση και περαιτέρω ανάλυση των μετρήσεων σε αγροτικές καλλιέργειες. Η δυνατότητα παρακολούθησης σε πραγματικό χρόνο (real-time monitoring) και η άμεση πρόσβαση στα δεδομένα

χωρίς αφαίρεση του αποθηκευτικού μέσου επεκτείνουν την λειτουργικότητα και τις δυνατότητες του συνολικού συστήματος.

3.9.2 Διαμοιρασμός IP μέσω UDP Broadcast

Για να διευκολυνθεί η εύρεση του web server από την εφαρμογή-πελάτη, το σύστημα πραγματοποιεί περιοδικό διαμοιρασμό (broadcast) της τρέχουσας διεύθυνσης του ESP32. Στο σχήμα 3.2 φαίνεται ο τρόπος που επιτυγχάνεται ο διαμοιρασμός της IP του ESP32:



Σχήμα 3.2: Διάγραμμα διαμοιρασμού IP μέσω πρωτοκόλλου UDP

Με αυτήν την τεχνική, ο ESP32 στέλνει ένα σύντομο μήνυμα μέσω του πρωτοκόλλου UDP (User Diagram Protocol) προς την broadcast διεύθυνση του τοπικού δικτύου σε προκαθορισμένη θύρα. Όλες οι συσκευές που “ακούν” στην ίδια θύρα μπορούν να λάβουν το μήνυμα και να εντοπίσουν άμεσα τη διεύθυνση IP του ESP32.

Η λειτουργία είναι απλή:

- Ο ESP32 συνδέεται σε δίκτυο Wi-Fi και αποκτά δυναμική IP μέσω DHCP
- Δημιουργείται ένα UDP socket σε μία προκαθορισμένη θύρα (π.χ. 4210)
- Σε τακτά χρονικά διαστήματα ο ESP32 αποστέλλει την IP
- Η εφαρμογή-πελάτης λαμβάνει το broadcast και συνδέεται στον web server στην αντίστοιχη IP

3.9.3 Ανάπτυξη Κώδικα στον ESP32

Ο web server του ESP32 αξιοποιεί τη βιβλιοθήκη WiFi.h για τη σύνδεση σε τοπικό ασύρματο δίκτυο και την βιβλιοθήκη WiFiUdp.h για τον διαμοιρασμό της IP του μέσω UDP, ενώ η βιβλιοθήκη WebServer.h χρησιμοποιείται για τη δημιουργία των endpoints και την αποστολή των δεδομένων.

Παρακάτω ακολουθεί ο κώδικας υλοποίησης της ασύρματης σύνδεσης του ESP32 με προκαθορισμένο δίκτυο Wi-Fi:

```
#include <WiFi.h>

// Στοιχεία σύνδεσης Wi-Fi
const char* ssid = "SSID";                    // SSID του AP
const char* password = "Password";            // Κωδικός του AP

void setup() {                                //Αρχή της setup()
  Serial.begin(115200);                       // Εκκίνηση του Serial monitor
  WiFi.begin(ssid, password);                 // Αρχικοποίηση του Wi-Fi
  while (WiFi.status() != WL_CONNECTED) {
    Serial.println("Connecting to WiFi...");  // Αναμονή για σύνδεση με το δίκτυο
}
  Serial.println("Connected to WiFi");        // Επιτυχής σύνδεση
  Serial.print("ESP32 IP Address: ");
  Serial.println(WiFi.localIP());                     // Εμφάνιση διεύθυνσης IP στο Serial monitor

void loop() {                                 // Αρχή της loop()
}
```

Αρχικά, δηλώνονται τα στοιχεία του δικτύου (SSID, κωδικός). Μέσα στην setup() γίνεται η αρχικοποίηση του Wi-Fi και το πρόγραμμα "περιμένει" μέχρι να επιτευχθεί σύνδεση με το δίκτυο. Όταν ο ESP32 συνδεθεί στο προκαθορισμένο δίκτυο, εκτυπώνεται στο serial monitor η IP του.

Παρακάτω ακολουθεί ο κώδικας υλοποίησης του UDP socket και η έναρξη διαμοιρασμού της IP του ESP32:

```
#include <WiFiUdp.h>

WiFiUDP udp;                                  // Δημιουργία UDP socket
const unsigned int udpPort = 4210;            // Θύρα σύνδεσης
const char* udpAddress = "255.255.255.255";   // Broadcast address
```

```
unsigned long lastSend = 0;                    // Global μεταβλητές
const unsigned long interval = 5000;

void sendIP() {                                // Συνάρτηση για αποστολή της IP
  IPAddress ip = WiFi.localIP();

  udp.beginPacket(udpAddress, udpPort);        // Δημιουργία UDP πακέτου
  udp.print(ip);                               // Προσθήκη της IP στο πακέτο
  udp.endPacket();                             // Κλείσιμο πακέτου και αποστολή

  Serial.print("Sent IP via UDP: ");
  Serial.println(ip);
}

void setup() {                                 // Αρχή της setup()
  Serial.begin(115200);                        // Εκκίνηση του Serial monitor
  udp.begin(udpPort);                          // Αρχικοποίηση UDP
  delay(1000);                                 // Μικρή καθυστέρηση για ορθή αρχικοποίηση
  sendIP();                                    // Αποστολή της IP του ESP32

void loop() {                                  // Αρχή της loop()
  if (millis() - lastSend > interval) {
    sendIP();                                  // Αποστολή κάθε 5 δευτερόλεπτα
    lastSend = millis();
  }
}
```

Στην αρχή, καθορίζεται το UDP socket δηλώνοντας την θύρα και την broadcast address. Στη συνέχεια, δημιουργείται μία σύναρτηση η οποία αποστέλλει την IP του ESP32 μέσω UDP στο δίκτυο και καλείται κατά την αρχικοποίηση μέσα στην setup(). Τέλος, στην loop() καλούμε ξανά την sendIP() ανά διαστήματα πέντε δευτερολέπτων.

Παρακάτω ακολουθεί η υλοποίηση της ενσωμάτωσης του web server στον ESP32 που είναι υπεύθυνος για την αποστολή απαντήσεων στα HTTP GET αιτήματα μέσω endpoints:

```
#include <WebServer.h>                         // Βιβλιοθήκη για δημιουργία και διαχείριση HTTP server

WebServer server(80);                          // Ρύθμιση του Web Server στην θύρα 80 (HTTP)


// ---------------------- ΣΥΝΑΡΤΗΣΕΙΣ HANDLER ----------------------
```

```
// Επιστρέφει το περιεχόμενο του αρχείου datalog.txt από την SD σε μορφή JSON array
void handleGetSDData() {
  File file = SD.open("/datalog.txt");                  // Άνοιγμα αρχείου για ανάγνωση
  if (!file) {                                          // Αν αποτύχει το άνοιγμα
    server.send(500, "application/json", "{\"error\":\"Failed to open file\"}");
    return;
  }

  String data = "[";
  while (file.available()) {                            // Διαβάζει γραμμή-γραμμή
    data += file.readStringUntil('\n') + ",";           // Προσθήκη κόμματος μεταξύ εγγραφών
  }
  file.close();
                                                        // Αφαίρεση τελευταίου κόμματος αν υπάρχει
  if (data.length() > 1) {
    data.remove(data.length() - 1);
  }
  data += "]";                                          // Κλείσιμο JSON array
  server.send(200, "application/json", data);
}

// Διαγράφει το αρχείο datalog.txt από την SD
void clearSDData() {
  if (SD.exists("/datalog.txt")) {                      // Έλεγχος αν υπάρχει
    SD.remove("/datalog.txt");                          // Διαγραφή
    Serial.println("File deleted successfully");
    server.send(200, "application/json", "{\"deleted\":\"Data cleared successfully.\"}");
  } else {
    Serial.println("File not found");
    server.send(404, "application/json", "{\"error\":\"File not found.\"}");
  }
}

// Ελέγχει αν οι περιφερειακές συσκευές (DHT και SD) είναι διαθέσιμες
void checkHardwareStatus() {

    float checkTemp = dht.readTemperature();                    // Έλεγχος DHT αισθητήρα
    if (!isnan(checkTemp)) {
        dhtAvailable = true;
    } else {
        dhtAvailable = false;
    }
```

```
  // Έλεγχος SD Card (δημιουργία και διαγραφή test αρχείου)
  File testFile = SD.open("/test.txt", FILE_WRITE);
  if (testFile) {
    sdAvailable = true;
    testFile.close();
    SD.remove("/test.txt");
  } else {
    sdAvailable = false;
  }
}

void handleStatus() {                                   // Επιστρέφει σε JSON την κατάσταση Wi-Fi, DHT και SD
  checkHardwareStatus();                                // Κλήση της συνάρτησης ελέγχου

  String json = "{";
  json += "\"wifi_connected\":" + String(WiFi.status() == WL_CONNECTED ? "true" : "false") + ",";
  json += "\"dht_available\":" + String(dhtAvailable ? "true" : "false") + ",";
  json += "\"sd_available\":" + String(sdAvailable ? "true" : "false");
  json += "}";

  server.send(200, "application/json", json);
}

// Επιστρέφει δεδομένα αισθητήρων σε πραγματικό χρόνο σε JSON
void handleSensorData() {
  temp = dht.readTemperature();                 // Θερμοκρασία
  hum = dht.readHumidity();                     // Υγρασία

  // Έλεγχος αν τα δεδομένα είναι έγκυρα
  if (isnan(temp) || isnan(hum)) {
    server.send(500, "application/json", "{\"error\":\"Failed to read from DHT sensor\"}");
    return;
  }
  // Δημιουργία JSON string με όλα τα δεδομένα
  String json = "{";
  json += "\"temperature\":" + String(temp, 2) + ",";
  json += "\"humidity\":" + String(hum, 2) + ",";
  json += "\"latitude\":" + String(lat, 6) + ",";
  json += "\"longitude\":" + String(lon, 6) + ",";
  json += "\"altitude\":" + String(alt, 2) + ",";
  json += "\"sats\":" + String(sats) + ",";
  json += "\"vBat\":" + String(vBat, 2) + ",";
```

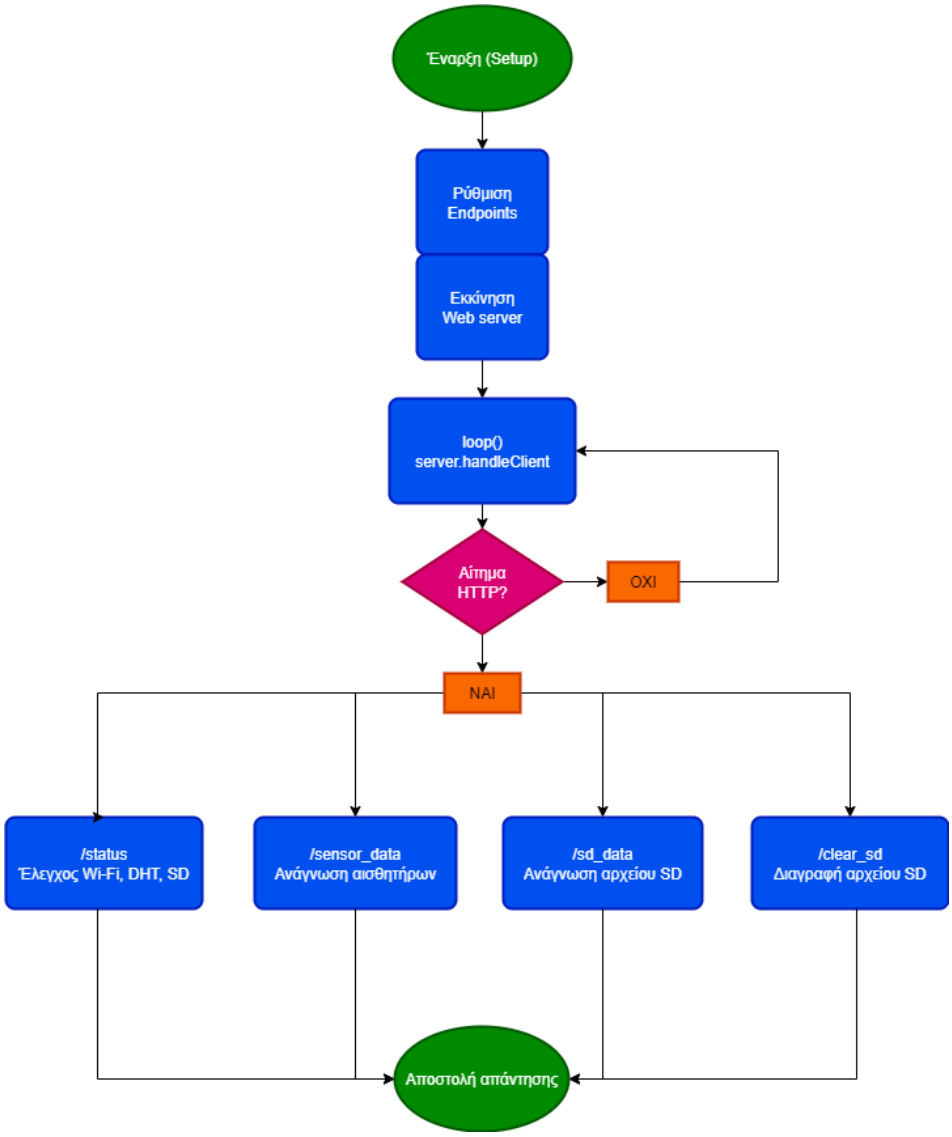
```
  json += "\"speed\":" + String(speed, 2);
  json += "}";
  server.send(200, "application/json", json);
}

// ---------------------- SETUP ----------------------
void setup() {
 Serial.begin(115200);                                    // Σειριακή επικοινωνία για debugging
 // Δήλωση των HTTP endpoints και αντιστοίχιση με handler συναρτήσεις
 server.on("/status", HTTP_GET, handleStatus);
 server.on("/sensor_data", HTTP_GET, handleSensorData);
 server.on("/sd_data", HTTP_GET, handleGetSDData);
 server.on("/clear_sd", HTTP_GET, clearSDData);
 server.begin();                                          // Εκκίνηση του Web Server
}

// ---------------------- LOOP ----------------------
void loop() {
 server.handleClient();                                   // Διαχείριση εισερχόμενων HTTP αιτημάτων
}
```

Κατά την εκκίνηση (setup), πραγματοποιείται αρχικοποίηση του serial monitor για debugging, δηλώνονται τα endpoints που θα εξυπηρετούνται από τον server και εκκινείται η λειτουργία του web server στη θύρα 80. Στη συνέχεια, στον κύριο βρόχο (loop), η συνάρτηση server.handleClient() εκτελείται συνεχώς, ώστε να ελέγχονται εισερχόμενα αιτήματα HTTP και να εκτελείται η αντίστοιχη handler συνάρτηση, με σκοπό την επιστροφή δεδομένων ή την εκτέλεση της ζητούμενης ενέργειας.

Τα endpoints και η λειτουργία τους είναι:

- /status : Μέσω της συνάρτησης handleStatus() πραγματοποιείται έλεγχος της κατάστασης Wi-Fi, του αισθητήρα DHT22 και της κάρτας SD επιστρέφοντας τα αποτελέσματα σε μορφή JSON.
- /sensor_data : Μέσω της συνάρτησης handleSensorData() επιστρέφονται σε πραγματικό χρόνο η θερμοκρασία, η υγρασία, οι συντεταγμένες GPS, το υψόμετρο, ο αριθμός δορυφόρων, η τάση της μπαταρίας και η ταχύτητα.
- /sd_data : Μέσω της συνάρτησης handleGetSDData διαβάζεται το αρχείο datalog.txt από την κάρτα SD και επιστρέφονται όλες οι εγγραφές σε μορφή JSON array.
- /clear_sd : Μέσω της συνάρτησης clearSDData διαγράφεται το αρχείο datalog.txt από την κάρτα SD και επιστρέφεται μήνυμα επιτυχίας ή αποτυχίας.

Στο σχήμα 3.3 παρουσιάζεται το διάγραμμα ροής του παραπάνω κώδικα για ευκολότερη κατανόηση της λειτουργίας του web server:



Σχήμα 3.3: Διάγραμμα ροής του κώδικα υλοποίησης web server στον ESP32

3.10 Επίλογος

Με την ολοκλήρωση της υλοποίησης του ενσωματωμένου συστήματος αισθητήρων και της ασύρματης διάθεσης των δεδομένων μέσω του web server, τίθεται η βάση για την ανάπτυξη της εφαρμογής απεικόνισης των δεδομένων. Στο επόμενο κεφάλαιο θα παρουσιαστεί η σχεδίαση και λειτουργία της

Android εφαρμογής που επιτρέπει την εύκολη πρόσβαση, οπτικοποίηση και διαχείριση των μετρήσεων, προσφέροντας μια ολοκληρωμένη λύση για την υποστήριξη της γεωργίας ακριβείας.

Κεφάλαιο 4ο: Ανάπτυξη Εφαρμογής Απεικόνισης και Διαχείρισης Δεδομένων

4.1 Εισαγωγή

Η ανάπτυξη μίας Android εφαρμογής απεικόνισης και διαχείρισης δεδομένων αποτελεί το τελικό στάδιο της ολοκληρωμένης λύσης που περιγράφεται στην παρούσα εργασία. Σκοπός της εφαρμογής είναι να επιτρέπει την άμεση και ευέλικτη πρόσβαση στα δεδομένα που συλλέγονται από το ενσωματωμένο σύστημα αισθητήρων του drone, παρέχοντας οπτικοποίηση κρίσιμων δεδομένων περιβάλλοντος και πτήσης, για την μεταγενέστερη εξαγωγή χρήσιμων συμπερασμάτων στο πλαίσιο της γεωργίας ακριβείας.

Η εφαρμογή λειτουργεί ως ενδιάμεσος κρίκος ανάμεσα στο ενσωματωμένο σύστημα και τον τελικό χρήστη, λαμβάνοντας δεδομένα από τον ESP32 μέσω του ενσωματωμένου web server, επεξεργάζοντάς τα και παρουσιάζοντάς τα με τρόπο κατανοητό και λειτουργικό. Η σχεδίαση της βασίζεται σε αρχές ευχρηστίας και επεκτασιμότητας, ώστε να μπορεί να προσαρμοστεί εύκολα σε μελλοντικές απαιτήσεις ή νέες πηγές δεδομένων.

4.2 Αρχιτεκτονική της Εφαρμογής

Η εφαρμογή FlyKonn έχει σχεδιαστεί με γνώμονα την αξιόπιστη λήψη, επεξεργασία και παρουσίαση των δεδομένων που αποστέλλονται από το ενσωματωμένο σύστημα αισθητήρων του drone. Η υλοποίησή της έγινε με τη γλώσσα προγραμματισμού Kotlin και για περιβάλλον ανάπτυξης χρησιμοποιήθηκε το Android Studio IDE.

Η αρχιτεκτονική της μπορεί να χωριστεί σε τρία κύρια μέρη:

1. Δικτυακή επικοινωνία με τον ESP32

   Περιλαμβάνει την ανακάλυψη της IP του ESP32 μέσω UDP broadcast και την ανταλλαγή δεδομένων μέσω REST API. Η επικοινωνία μέσω HTTP επιτυγχάνεται μέσω Retrofit.

2. Διαχείριση και επεξεργασία δεδομένων

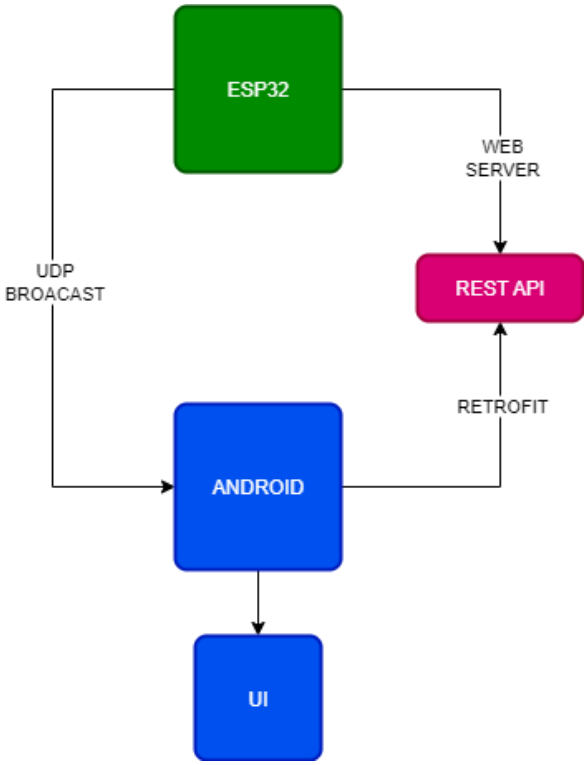
   Αναλαμβάνει το parsing των δεδομένων JSON που λαμβάνονται από τον ESP32 μετατρέποντάς τα σε κλάσεις δεδομένων Kotlin για εύκολη πρόσβαση και επεξεργασία.

3. Παρουσίαση και πλοήγηση

   Η προβολή των δεδομένων γίνεται μέσω ενός ViewPager με τρία fragments. Η πλοήγηση γίνεται μέσω ενός μενού στο κάτω μέρος της οθόνης.

Η αρχιτεκτονική αυτή επιτρέπει τον εμπλουτισμό κάθε μέρους, διευκολύνοντας την ενσωμάτωση νέων λειτουργειών, όπως νέους αισθητήρες ή μεθόδους ανάλυσης δεδομένων.

Η συνολική ροή δεδομένων φαίνεται στο παρακάτω διάγραμμα:



Σχήμα 4.1: Διάγραμμα ροής δεδομένων μεταξύ ESP32 και εφαρμογής

4.3 Ανακάλυψη IP μέσω UDP Broadcast

Η εφαρμογή χρειάζεται να γνωρίζει την διεύθυνση IP του ESP32, καθώς αυτός λειτουργεί ως τοπικός web server. Ο ESP32 χρησιμοποιεί UDP broadcast και στέλνει περιοδικά την διεύθυνσή του, όπως υλοποιήθηκε στο προηγούμενο κεφάλαιο. Από την μεριά της εφαρμογής, υλοποιείται μέθοδος που λαμβάνει την διεύθυνση για την επιτυχή σύνδεση με το σύστημα αισθητήρων.

Η εφαρμογή χρησιμοποιεί την κλάση UdpListener για την αυτόματη ανίχνευση της διεύθυνσης IP του ESP32. Η επικοινωνία αυτή βασίζεται στα UDP broadcast μηνύματα που αποστέλλονται περιοδικά μέσω της θύρας 4210.

Παρακάτω ακολουθεί ο κώδικας υλοποίησης της κλάσης UdpListener:

```
Class UdpListener(
private val context: Context,
```

```
private val port: Int = 4210,
private val onIpReceived: (String) -> Unit
) {
    private var listeningJob: Job? = null
    fun startListening() {
        // Εκκίνηση coroutine στο IO dispatcher ώστε να μη μπλοκάρει το UI
        listeningJob = CoroutineScope(Dispatchers.IO).launch {

            // Απόκτηση MulticastLock για να επιτραπεί η λήψη broadcast/multicast πακέτων
            val wifiManager = context.applicationContext.getSystemService(Context.WIFI_SERVICE) as WifiManager
            val lock = wifiManager.createMulticastLock("udp_lock").apply { acquire() }
            // Δημιουργία UDP socket δεσμευμένο στη συγκεκριμένη θύρα και σε όλες τις διευθύνσεις
            val socket = DatagramSocket(port, InetAddress.getByName("0.0.0.0")).apply {
                broadcast = true            // Ενεργοποίηση λήψης broadcast πακέτων
            }

            val buffer = ByteArray(256)       // Buffer για αποθήκευση εισερχόμενων δεδομένων

            // Ατέρμονος βρόχος λήψης πακέτων όσο το coroutine είναι ενεργό
            while (isActive) {
                val packet = DatagramPacket(buffer, buffer.size)
                socket.receive(packet)       // Αναμονή λήψης UDP πακέτου
                // Μετατροπή των bytes σε String
                val message = String(packet.data, 0, packet.length)
                // Εκτέλεση του callback στο κύριο νήμα για ενημέρωση του UI
                withContext(Dispatchers.Main) {
                    onIpReceived(message)
                }
            }
            // Κλείσιμο socket και απελευθέρωση multicast lock όταν η ακρόαση σταματήσει
            socket.close()
            lock.release()
        }
    }
    fun stopListening() {
        // Διακοπή της coroutine που ακούει για UDP πακέτα
        listeningJob?.cancel()
    }
```

Η ακρόαση των μηνυμάτων γίνεται ασύγχρονα, μέσω των Kotlin Coroutines ώστε να τρέχει σε ξεχωριστό νήμα με τον Dispatchers.IO. Με αυτόν τον τρόπο, η διεπαφή χρήστη παραμένει ανεπηρέαστη, ενώ η εφαρμογή είναι σε θέση να δέχεται μηνύματα συνεχώς χωρίς μπλοκαρίσματα.

Για να επιτραπεί η λήψη broadcast πακέτων, η κλάση αποκτά ένα MulticastLock μέσω του WifiManager. Αυτό είναι απαραίτητο επειδή το Android, για λόγους εξοικονόμισης ενέργειας, περιορίζει τη ροή broadcast πακέτων και με το κλείδωμα αυτό διασφαλίζεται η λήψη. Στη συνέχεια, δημιουργείται ένα DatagramSocket στη θύρα 4210, με ρύθμιση για αποδοχή broadcast μηνυμάτων. Η κλάση εισέρχεται σε ατέρμονο βρόχο όπου περιμένει να ληφθεί πακέτο UDP, το οποίο διαβάζεται σε έναν προορισμένο buffer μεγέθους 256 bytes. Το περιεχόμενο του πακέτου (IP του ESP32), στην πορεία, μετατρέπεται σε κέιμενο.

Αφού ληφθεί το μήνυμα, η κλήση στην onIpReceived γίνεται μέσω του Dispachers.Main, ώστε η ενημέρωση της διεπαφής ή άλλες λειτουργίες που αφορούν το UI να γίνονται με ασφάλεια στο κύριο νήμα. Η κλάση διαθέτει και και μέθοδο stopListening() που ακυρώνει την coroutine και απελευθερώνει

τους πόρους. Με τον τρόπο αυτό η εφαρμογή διατηρεί συνεχώς ενημερωμένη την τρέχουσα διεύθυνση IP του ESP32, διευκολύνοντας την επόμενη φάση της επικοινωνίας μέσω HTTP REST κλήσεων.

4.4 Δικτυακή Επικοινωνία μέσω REST API

Η επικοινωνία της εφαρμογής FlyKonn με τον ESP32 βασίζεται σε ένα RESTful API που υλοποιείται στον ενσωματωμένο web server του ESP32. Μέσω HTTP κλήσεων, η εφαρμογή ζητάει συγκεκριμένα δεδομένα, όπως αισθητήρων ή κατάστασης συστήματος, και λαμβάνει απαντήσεις σε μορφή JSON.

Για την αποτελεσματική διαχείριση αυτής της επικοινωνίας, η εφαρμογή αξιοποιεί τη βιβλιοθήκη Retrofit, η οποία απλοποιεί την υλοποίηση HTTP client με σαφή τρόπο. Η Retrofit επιτρέπει τον ορισμό των διαφόρων endpoints μέσω Kotlin interfaces, όπου κάθε μέθοδος αντιστοιχεί σε συγκεκριμένη HTTP κλήση (GET, Post κ.λπ.).

Για την υλοποίηση της επικοινωνίας μέσω REST API κλήσεων δημιουργήθηκαν δύο αρχεία, η διεπαφή ApiService.kt και το αντικείμενο RetrofitClient.kt. Παρακάτω ακολουθεί ο κώδικας υλοποίησης της ApiService:

```
interface ApiService {
    @GET("status")
    fun getStatus(): Call<EspStatusResponse> // Επιστρέφει την κατάσταση των περιφερειακών
    @GET("sensor_data")
    fun getSensorData(): Call<SensorData> // Επιστρέφει τα δεδομένα αισθητήρων και πτήσης
    @GET("sd_data")
    fun getSdData(): Call<List<SdData>>  // Επιστρέφει τη λίστα των δεδομένων από την SD
    @GET("clear_sd")
    fun clearSDCard(): Call<ClearResponse> // Διαγράφει το αρχείο από την SD
}
```

Στο αρχείο ApiService.kt ορίζονται τα endpoints του API ως απλές Kotlin μέθοδοι με το κατάλληλο HTTP annotation. Συγκεκριμένα, υπάρχουν μέθοδοι για την ανάκτηση της κατάστασης του ESP32 (getStatus()), των δεδομένων αισθητήρων και πτήσης (getSensorData()), των αποθηκευμένων δεδομένων από την κάρτα SD (getSdData()), καθώς και για τον καθαρισμό της SD (clearSDCard()).

Παρακάτω παρατίθεται ο κώδικας υλοποίησης του αντικειμένου RetrofitClient, που διαχειρίζεται την σύνδεση με τον ESP32:

```
object RetrofitClient {
    private lateinit var retrofit: Retrofit       // Instance του Retrofit
    private lateinit var api: ApiService       // Αντικείμενο ApiService που περιέχει τα endpoints
    private var initialized = false       // Σημάδι για την αρχικοποίηση του RetrofitClient

    fun init(ip: String) {          // Αρχικοποίηση του RetrofitClient
        retrofit = Retrofit.Builder()
            .baseUrl("http://$ip/")      // Ορίζεται η βάση URL με το δυναμικό IP
            .addConverterFactory(GsonConverterFactory.create())
            .build()              // Μετατροπή JSON σε Kotlin
        api = retrofit.create(ApiService::class.java)
        initialized = true          // ApiService instance από το Retrofit
    }
    // Μέθοδος για απόκτηση του ApiService αντικειμένου
    fun getApi(): ApiService {
```

```
        if (!initialized) throw IllegalStateException("RetrofitClient not initialized. Call init(ip) first.")
        return api
    }
}
```

Το αντικείμενο RetrofitClient αρχικοποιείται με την δυναμική IP που λαμβάνει η εφαρμογή μέσω UDP broadcast. Στην init(ip: String) μέθοδο δημιουργείται ένα Retrofit instance με την βασική διεύθυνση URL, προστίθεται ο μετατροπέας Gson για αυτόματη μετατροπή των JSON δεδομένων σε Kotlin αντικείμενα και δημιουργείται το αντικείμενο API.

Οι υλοποιήσεις αυτές προσφέρουν σαφή και οργανωμένη πρόσβαση στα δεδομένα, ενώ η χρήση των Retrofit αντικειμένων επιτρέπει την ασύγχρονη εκτέλεση των κλήσεων μέσω callbacks. Έτσι, η εφαρμογή μπορεί να λαμβάνει και να επεξεργάζεται δεδομένα σε πραγματικό χρόνο χωρίς να επιρεάζεται η απόκριση του χρήστη.

4.5 Κλάσεις Δεδομένων

Για την αποτελεσματική διαχείριση και επεξεργασία των δεδομένων που λαμβάνονται από τον ESP32, χρησιμοποιήθηκαν συγκεκριμένες κλάσεις δεδομένων Kotlin, οι οποίες αντιπροσωπεύουν τα σύνολα δεδομένων που επιστρέφονται από το API.

Οι τέσσερεις δομές δεδομένων είναι:

- EspStatusResponse

  Περιλαμβάνει πληροφορίες για την κατάσταση του ESP32, όπως την κατάσταση σύνδεσης και των περιφερειακών του συστήματος

- SensorData

  Περιέχει τα δεδομένα αισθητήρων και πτήσης

- SdData

  Αποτελεί μία δομή που αναπαριστά τις εγγραφές που είναι αποθηκευμένες στην κάρτα SD για την ανάκτηση του ιστορικού πτήσεων.

- ClearResponse

  Αντιπροσωπεύει την απάντηση του API μετά από αίτημα για καθαρισμό της SD, επιστρέφοντας την επιτυχία ή την αποτυχία της ενέργειας.

Παρακάτω ακολουθούν οι τέσσερεις κλάσεις δεδομένων Kotlin:

```
// Η κλάση κατάστασης του ESP32
data class EspStatusResponse(
    val wifi_connected: Boolean,
    val dht_available: Boolean,
    val sd_available: Boolean
)
```

```
// Η κλάση δεδομένων αισθητήρων
data class SensorData(
    @SerializedName("temperature") val temperature: Float,
    @SerializedName("humidity") val humidity: Float,
    @SerializedName("latitude") val latitude: Float,
    @SerializedName("longitude") val longitude: Float,
    @SerializedName("altitude") val altitude: Float,
    @SerializedName("sats") val sats: Int,
    @SerializedName("vBat") val vBat: Float,
    @SerializedName("speed") val speed: Float
)

// Η κλάση δεδομένων της SD
data class SdData(
    val timestamp: Long,
    val temperature: Float,
    val humidity: Float,
    val latitude: Float,
    val longitude: Float,
    val altitude: Float
)

// Η κλάση απάντησης για την διαγραφή της SD
data class ClearResponse(
    @SerializedName("cleared") val message: String? = null,
    @SerializedName("error") val error: String? = null
)
```

Η χρήση κλάσεων δεδομένων επιτρέπει στην εφαρμογή να διαχειρίζεται με ευκολία τα δεδομένα, εξασφαλίζοντας την ασφάλεια τύπων και την αποφυγή σφαλμάτων κατά την χρήση των JSON δεδομένων.

4.6 Οργάνωση Λογικής και Πλοήγησης Εφαρμογής

Η σχεδίαση της εφαρμογής βασίζεται σε μία δομή που επιτρέπει τη διαχείριση και την παρουσίαση των δεδομένων μέσω τριών βασικών fragments, τα οποία προβάλλονται με την βοήθεια του στοιχείου ViewPager2. Η πλοήγηση μεταξύ των fragments και η επικοινωνία με το drone διαχειρίζονται μέσω της κύριας δραστηριότητας (MainActivity) που λειτουργεί ως κεντρικός συντονιστής.

4.6.1 Διαχείριση των Fragments

Η διαχείριση των fragments που συνθέτουν την εφαρμογή γίνεται μέσω ενός προσαρμογέα (adapter) που υλοποιείται μέσω της κλάσης ViewPagerAdapter. Η κλάση αυτή υλοποιεί τη λογική εμφάνισης του κατάλληλου fragment κατά την οριζόντια μετακίνηση (swipe), αντιστοιχίζοντας τις βασικές ενότητες της εφαρμογής με κάποιο αριθμό:

- Θεση 0 → DashboardFrag.kt
- Θέση 1 → WaypointsFrag.kt
- Θέση 2 → SettingsFrag.kt

Ο adapter εξασφαλίζει την ομαλή και αποδοτική διαχείριση των fragments κατά την πλοήγηση, αξιοποιώντας τις βέλτιστες πρακτικές της Android πλατφόρμας για ομαλή εμπειρία χρήστη και διατήρηση της κατάστασης μεταξύ των αλλαγών.

Η υλοποίηση του ViewPagerAdapter ακολουθεί την παρακάτω μορφή:

```
class ViewPagerAdapter(activity: FragmentActivity) : FragmentStateAdapter(activity) {
    override fun getItemCount() = 3 // Αριθμός των fragments που θα διαχειρίζεται ο adapter
    // Δημιουργεί και επιστρέφει το κατάλληλο fragment ανάλογα με τη θέση (position)
    override fun createFragment(position: Int): Fragment {
        return when (position) {
            0 -> DashboardFrag()      // Πρώτο fragment
            1 -> WaypointsFrag()      // Δεύτερο fragment
            2 -> SettingsFrag()   // Τρίτο fragment
            else -> DashboardFrag() // Προεπιλεγμένο fragment
        }
    }
}
```

Με αυτή την υλοποίηση επιτυγχάνεται η πλοήγηση μεταξύ των διαφορετικών ενοτήτων της εφαρμογής μέσω του συστήματος tabs του στοιχείουViewPager2.

4.6.2 DashboardFrag – Ζωντανή Απεικόνιση Δεδομένων και Χάρτης

Η κλάση DashboardFrag.kt αποτελεί το βασικό περιβάλλον χρήστη για την παρουσίαση των real-time δεδομένων αισθητήρων και πτήσης, καθώς και της απεικόνισης της θέσης του drone σε έναν ενσωματωμένο χάρτη. Το fragment αυτό φορτώνει το σχετικό layout (frag_dashboard.xml) και περιλαμβάνει στοιχεία κειμένου TextView για την εμφάνιση δεδομένων όπως θερμοκρασία, υγρασία, συντεταγμένες, υψόμετρο, ταχύτητα, τάση μπαταρίας και αριθμό δορυφόρων.

Παρακάτω ακολουθεί η υλοποίηση της κλάσης DashboardFrag.kt:

```
class DashboardFrag : Fragment(R.layout.frag_dashboard) {

    // Μεταβλητές που αντιπροσωπεύουν τα TextView του layout για την απεικόνιση δεδομένων αισθητήρων
    private lateinit var tempRtSensor: TextView   // Θερμοκρασία
    private lateinit var humRtSensor: TextView    // Υγρασία
    private lateinit var mapWebView: WebView      // WebView για την απεικόνιση χάρτη
    private lateinit var altRtSensor: TextView    // Υψόμετρο
    private lateinit var spdRtSensor: TextView    // Ταχύτητα
    private lateinit var vbatRtSensor: TextView   // Τάση μπαταρίας
    private lateinit var satsRtSensor: TextView   // Δορυφόροι GPS

    /**
     * Μέθοδος που καλείται μετά τη δημιουργία του view του Fragment.
     * Συνδέει τα στοιχεία του UI με τις μεταβλητές και ρυθμίζει το WebView.
     */
    @SuppressLint("ClickableViewAccessibility")
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // Σύνδεση των στοιχείων του layout με τις αντίστοιχες μεταβλητές
        tempRtSensor = view.findViewById(R.id.tempRtSensor)
        humRtSensor = view.findViewById(R.id.humRtSensor)
```

```
        mapWebView = view.findViewById(R.id.mapWebView)
        altRtSensor = view.findViewById(R.id.altRtSensor)
        spdRtSensor = view.findViewById(R.id.spdRtSensor)
        vbatRtSensor = view.findViewById(R.id.vbatRtSensor)
        satsRtSensor = view.findViewById(R.id.satsRtSensor)

        // Πρόσβαση στο ViewPager2 για έλεγχο της πλοήγησης μεταξύ οθονών
        val viewPager2: ViewPager2 = requireActivity().findViewById(R.id.viewPager)

        // Ενεργοποίηση υποστήριξης JavaScript στο WebView
        mapWebView.settings.javaScriptEnabled = true

        // Φόρτωση τοπικού HTML αρχείου που περιέχει τον χάρτη
        mapWebView.loadUrl("file:///android_asset/map.html")

        // Απενεργοποίηση λειτουργίας swipe του ViewPager2 όταν ο χρήστης αλληλεπιδρά με το χάρτη
        mapWebView.setOnTouchListener { _, event ->
            when (event.action) {
                MotionEvent.ACTION_DOWN, MotionEvent.ACTION_MOVE -> {
                    viewPager2.isUserInputEnabled = false  // Απενεργοποίηση swipe
                }
                MotionEvent.ACTION_UP, MotionEvent.ACTION_CANCEL -> {
                    viewPager2.isUserInputEnabled = true  // Επανενεργοποίηση swipe
                }
            }
            false     // Επιστρέφουμε false για να συνεχίσει το WebView να δέχεται τα events
        }
    }

    // Μέθοδος για ενημέρωση των δεδομένων των αισθητήρων και ανανέωση της θέσης στο χάρτη.
    fun updateSensorData(
        temperature: String,
        humidity: String,
        latitude: String,
        longitude: String,
        altitude: String,
        sats: String,
        vBat: String,
        speed: String
    ) {
        // Ενημέρωση των TextView με τις τιμές των αισθητήρων
        tempRtSensor.text = temperature
        humRtSensor.text = humidity
        altRtSensor.text = altitude
        spdRtSensor.text = speed
        vbatRtSensor.text = vBat
        satsRtSensor.text = sats

        // Μετατροπή των συντεταγμένων σε αριθμητικές τιμές (Double)
        val latitudeDouble = latitude.toDoubleOrNull() ?: 0.0
        val longitudeDouble = longitude.toDoubleOrNull() ?: 0.0

        // Εκτέλεση JavaScript κώδικα για ενημέρωση της θέσης του drone στον χάρτη
        val jsCode = "javascript:updateDronePosition($latitudeDouble, $longitudeDouble)"
        mapWebView.post {
            mapWebView.evaluateJavascript(jsCode, null)
        }
    }
}
```

Στην αρχή της υλοποίησης γίνεται δήλωση των μεταβλητών διεπαφής χρήστη, οι οποίες περιλαμβάνουν TextView για την απεικόνιση των δεδομένων σε πραγματικό χρόνο, καθώς και ένα στοιχείο WebView για την προβολή του χάρτη πλοήγησης.

Η αρχικοποίηση αυτών των στοιχείων γίνεται στη μέθοδο onViewCreated, η οποία εκτελείται μετά τη φόρτωση του layout. Σε αυτό το στάδιο γίνεται αντιστοίχιση των μεταβλητών με τα αντίστοιχα στοιχεία του γραφικού περιβάλλοντος, ενεργοποίηση της υποστήριξης Javascript στο WebView και ρύθμιση της αλληλεπίδρασης με τον χάρτη έτσι ώστε να αποτρέπεται η ακούσια εναλλαγή σελίδων του ViewPager2 κατά την πλοήγηση.

Στη συνέχεια, η μέθοδος updateSensorData είναι υπεύθυνη για την ενημέρωση των δεδομένων στην οθόνη. Δέχεται ως παραμέτρους τις τιμές των αισθητήρων – πτήσης και ενημερώνει δυναμικά τα αντίστοιχα στοιχεία διεπαφής. Επιπλέον, μετατρέπει τις συντεταγμένες σε αριθμούς και καλεί μέσω Javascript τη συνάρτηση updateDronePosition που βρίσκεται στο αρχείο map.html, προκειμένου να ανανεωθεί η απεικόνιση της θέσης του drone στο χάρτη.

Η υλοποίηση του DashboardFrag συνδυάζει την απεικόνιση κρίσιμων δεδομένων πτήσης με την οπτική παρακολούθηση της πορείας του drone, παρέχοντας στον χρήστη ένα ολοκληρωμένο εργαλείο επιτήρησης και ελέγχου.

4.6.3 WaypointsFrag – Διαχείριση Δεδομένων από SD Κάρτα

Η κλάση WaypointsFrag αποτελεί το fragment της εφαρμογής που είναι υπεύθυνο για την ανάκτηση και εμφάνιση δεδομένων που έχουν συλλεχθεί κατά την αυτόνομη πτήση του drone μέσω της κάρτας SD, καθώς και για την δυνατότητα εκκαθάρισης αυτών των δεδομένων.

Η υλοποίηση της WaypointsFrag ακολουθεί την παρακάτω μορφή:

```
class WaypointsFrag : Fragment() {

    private lateinit var textView: TextView // Πεδίο εμφάνισης δεδομένων

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Σύνδεση layout με το fragment
        val rootView = inflater.inflate(R.layout.frag_waypoints, container, false)

        // Σύνδεση του TextView για την εμφάνιση δεδομένων
        textView = rootView.findViewById(R.id.sdDataTextView)

        // Ορισμός λειτουργίας κουμπιού εκκαθάρισης SD κάρτας
        rootView.findViewById<Button>(R.id.btnClearSD).setOnClickListener { clearSDCard() }

        // Ανάκτηση δεδομένων από τον server
        fetchDataFromServer()

        return rootView
    }

    /**
     * Ανάκτηση των δεδομένων από την SD κάρτα μέσω Retrofit API.
     * Τα δεδομένα φορτώνονται και εμφανίζονται στο textView.
     */
    private fun fetchDataFromServer() {
        RetrofitClient.getApi().getSdData().enqueue(object : Callback<List<SdData>> {
```

```
        override fun onResponse(call: Call<List<SdData>>, response: Response<List<SdData>>) {
            if (response.isSuccessful) {
                displayData(response.body())
            } else {
                textView.text = "Failed to load data"
            }
        }

        override fun onFailure(call: Call<List<SdData>>, t: Throwable) {
            textView.text = "Failed to load data"
        }
    })
}

/**
 * Εμφανίζει τα δεδομένα της SD κάρτας με μορφοποίηση.
 */
private fun displayData(dataList: List<SdData>?) {
    if (!dataList.isNullOrEmpty()) {
        val sb = StringBuilder()
        for (data in dataList) {
            sb.append("Timestamp: ${data.timestamp}\n")
            sb.append("Temperature: ${data.temperature}°C\n")
            sb.append("Humidity: ${data.humidity}%\n")
            sb.append("Latitude: ${data.latitude}\n")
            sb.append("Longitude: ${data.longitude}\n")
            sb.append("Altitude: ${data.altitude}m\n\n")
        }
        textView.text = sb.toString()
    } else {
        textView.text = "No data available"
    }
}

/**
 * Εκκαθαρίζει τα δεδομένα της SD κάρτας καλώντας το API.
 * Εμφανίζει μήνυμα επιτυχίας ή αποτυχίας στον χρήστη.
 */
private fun clearSDCard() {
    RetrofitClient.getApi().clearSDCard().enqueue(object : Callback<ClearResponse> {
        override fun onResponse(call: Call<ClearResponse>, response: Response<ClearResponse>) {
            val message = response.body()?.message
                ?: response.body()?.error
                ?: "SD data cleared successfully"
            Toast.makeText(requireContext(), message, Toast.LENGTH_LONG).show()
        }

        override fun onFailure(call: Call<ClearResponse>, t: Throwable) {
            Toast.makeText(requireContext(), "Connection problem detected!", Toast.LENGTH_LONG).show()
        }
    })
}
}
```

Κατά την δημιουργία της προβολής μέσω της μεθόδου onCreateView, φορτώνεται το layout του fragment (frag_waypoints.xml) και συνδέονται τα στοιχεία διεπαφής χρήστη, όπως το κουμπί εκκαθάρισης και το TextView στο οποίο εμφανίζονται τα δεδομένα. Η λειτουργία του κουμπιού ορίζεται μέσω listener (onClickListener), το οποίο καλεί τη μέθοδο clearSDCard() για την αποστολή αιτήματος διαγραφής στον ESP32.

Η μέθοδος fetchDataFromServer() χρησιμοποιεί τη Retrofit για να στείλει αίτημα προς τον ESP32 και να ανακτήσει τα αποθηκευμένα δεδομένα της κάρτας SD. Κατά την επιτυχή απόκριση, η μέθοδος displayData() δημιουργεί μία λίστα αντικειμένων τύπου SdData που επιστρέφεται από τον server (Ενότητα 4.5). Το αποτέλεσμα προβάλλεται στο TextView, ώστε ο χρήσης να έχει πλήρη εικόνα των δεδομένων που έχουν καταγραφεί στην κάρτα SD.

Η μέθοδος clearSDCard() αποστέλλει το αίτημα διαγραφής των δεδομένων μέσω της Retrofit. Αν η ενέργεια είναι επιτυχής στέλνεται μήνυμα επιβεβαίωσης, ενώ σε άλλη περίπτωση εμφανίζονται κατάλληλα μηνύματα προς τον χρήστη.

Έτσι, το συγκεκριμένο fragment προσφέρει έναν ολοκληρωμένο τρόπο προβολής των δεδομένων της SD, ενώ η δυνατότητα διαγραφής των δεδομένων εξαλείφει την ανάγκη χρήσης πρόσθετου μέσου για την διαχείριση τους.

4.6.4 SettingsFrag – Σελίδα Ρυθμίσεων Εφαρμογής

Η υλοποίηση του fragment ρυθμίσεων της εφαρμογής διατηρείται προς το παρόν σε μία απλή και λειτουργική μορφή, με βασικό στοιχείο έναν διακόπτη που επιτρέπει την αλληλεπίδραση με το χρήστη για αλλαγή της κατάστασής του. Η ύπαρξη ξεχωριστού fragment για τις ρυθμίσεις δημιουργεί ένα ολοκληρωμένο περιβάλλον, το οποίο μπορεί να υποστηρίξει μελλοντικές βελτιώσεις, όπως επιλογές παραμετροποίησης, διαχείρισης ειδοποιήσεων ή δημιουργίας προφίλ χρήστη.

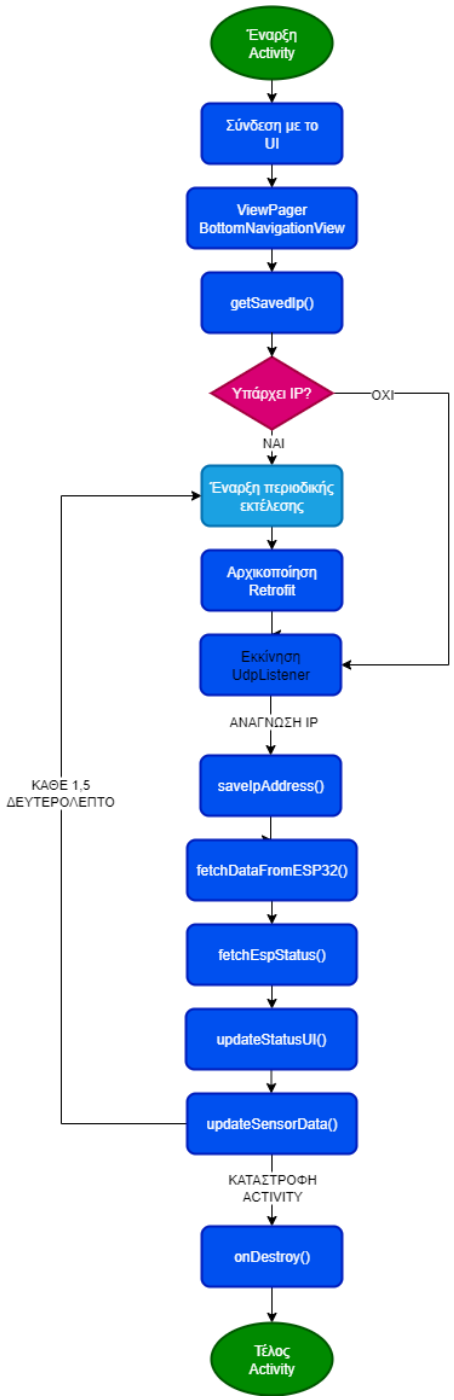
Ο κώδικας υλοποίησης του SettingsFrag παρουσιάζεται παρακάτω:

```
class SettingsFrag : Fragment() {

    private lateinit var themeSwitch: SwitchCompat
    private lateinit var rootView: View
    private lateinit var themeTextView: TextView

    // Δημιουργία της προβολής
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {                                   // Σύνδεση με το layout
        rootView = inflater.inflate(R.layout.frag_settings, container, false)

        themeSwitch = rootView.findViewById(R.id.themeSwitch)
        themeTextView = rootView.findViewById(R.id.themeText)

        // Αρχικό κείμενο
        themeTextView.text = "Dark Mode"

        // Listener που αλλάζει μόνο το κείμενο ανάλογα με το switch
        themeSwitch.setOnCheckedChangeListener { _, isChecked ->
            if (isChecked) {
                themeTextView.text = "Light Mode"
            } else {
                themeTextView.text = "Dark Mode"
            }
        }

        return rootView
```

```
    }
}
```

4.6.5 Συντονισμός Εφαρμογής – MainActivity

Η κύρια Activity της εφαρμογής έχει ως βασικό ρόλο τη διαχείριση της διεπαφής χρήστη και το συντονισμό της επικοινωνίας με το ESP32. Για την καλύτερη κατανόηση της λειτουργίας και της δομής της, ο κώδικας έχει χωριστεί και αναλύεται κάθε κομμάτι του ξεχωριστά. Το διάγραμμα ροής της MainActivity φαίνεται στο σχήμα 4.2:



Σχήμα 4.2: Διάγραμμα ροής της MainActivity

1. Δήλωση μεταβλητών και αρχικοποίηση UI

Η κύρια δραστηριότητα της εφαρμογής αναλαμβάνει τον συντονισμό και τη διαχείριση των δεδομένων που προέρχονται από το drone μέσω του ESP32. Για αυτόν τον σκοπό, δηλώνονται αρχικά σημαντικές μεταβλητές και στοιχεία διεπαφής χρήστη (UI), ώστε να μπορούν να χρησιμοποιηθούν σε όλη τη διάρκεια της εκτέλεσης της εφαρμογής. Ο κώδικας ακολουθεί παρακάτω:

```
// Listener UDP που λαμβάνει την IP διεύθυνση από το ESP32
private lateinit var udpListener: UdpListener

// Αποθηκεύει την τρέχουσα IP του ESP32
private var espIp: String? = null

// Συστατικά του UI
private lateinit var viewPager: ViewPager2
private lateinit var bottomNavigationView: BottomNavigationView
private lateinit var fragmentAdapter: ViewPagerAdapter
private lateinit var statusTextView: TextView
private lateinit var ipTextView: TextView
```

Αυτές οι μεταβλητές αρχικοποιούνται και συνδέονται με τα στοιχεία του αντίστοιχου layout μέσα στη μέθοδο onCreate(), η οποία αποτελεί το βασικό σημείο εκκίνησης της δραστηριότητας.

2. Ρύθμιση UI και έναρξη επικοινωνίας στο onCreate()

Η μέθοδος onCreate() είναι το σημείο όπου εκκινείται η εφαρμογή μόλις ξεκινήσει η MainActivity. Εδώ ορίζουμε το layout, αρχικοποιούμε τους adapters, ρυθμίζουμε τη πλοήγηση και ξεκινάμε τη διαδικασία λήψης IP από το ESP32:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    // Ορισμός του layout της κύριας οθόνης
    setContentView(R.layout.activity_main)

    // Σύνδεση των UI στοιχείων με τα IDs του layout
    viewPager = findViewById(R.id.viewPager)
    bottomNavigationView = findViewById(R.id.bottom_navigation)
    ipTextView = findViewById(R.id.ipTextView)
    statusTextView = findViewById(R.id.statusTextView)

    // Αρχικοποίηση του adapter για το ViewPager
    fragmentAdapter = ViewPagerAdapter(this)
    viewPager.adapter = fragmentAdapter
```

Στη συνέχεια, γίνεται συγχρονισμός του ViewPager με το BottomNavigationView (μενού πλοήγησης), ώστε όταν αλλάζει σελίδα ο χρήστης μέσω swipe, να ενημερώνεται το BottomNavigationView με τη σωστή επιλογή:

```
viewPager.registerOnPageChangeCallback(object : ViewPager2.OnPageChangeCallback() {
    override fun onPageSelected(position: Int) {
        super.onPageSelected(position)
        val menuItemId = when (position) {
            0 -> R.id.nav_home
            1 -> R.id.nav_sensors
            2 -> R.id.nav_settings
            else -> R.id.nav_home
        }
        bottomNavigationView.selectedItemId = menuItemId
```

```
    }
})
```

Αντίστροφα, όταν ο χρήστης πατήσει ένα tab στο μενού, αλλάζουμε την τρέχουσα σελίδα στο ViewPager:

```
bottomNavigationView.setOnItemSelectedListener { item ->
    val page = when (item.itemId) {
        R.id.nav_home -> 0
        R.id.nav_sensors -> 1
        R.id.nav_settings -> 2
        else -> 0
    }
    viewPager.setCurrentItem(page, true)
    true
}
```

Αν έχουμε ήδη συνδεθεί σε προηγούμενη εκτέλεση της εφαρμογής, φορτώνουμε την IP από τα SharedPreferences για να μην περιμένουμε νέο UDP μήνυμα:

```
val savedIp = getSavedIp()
if (savedIp != null) {
    espIp = savedIp
    ipTextView.text = "IP: $savedIp"
    RetrofitClient.init(savedIp) // Αρχικοποίηση Retrofit
    startRepeatingTask()        // Ξεκινάμε αμέσως τη λήψη δεδομένων
}
```

Αν δεν υπάρχει αποθηκευμένη IP ή θέλουμε να ενημερωθεί, ξεκινάμε έναν UdpListener που ακούει broadcast από το ESP32:

```
udpListener = UdpListener(this) { ip ->
    espIp = ip
    runOnUiThread {
        ipTextView.text = "IP: $ip"
    }
    saveIp(ip)
    RetrofitClient.init(ip)
    if (!isRepeatingTaskStarted) {
        startRepeatingTask()
    }
}
udpListener.startListening()
```

3. Συναρτήσεις επικοινωνίας με Retrofit

Η εφαρμογή χρησιμοποιεί το Retrofit για να επικοινωνεί με τον server του ESP32, προκειμένου να λαμβάνει δεδομένα αισθητήρων και κατάσταση συσκευής. Οι βασικές κλήσεις είναι:

- fetchEspStatus()

Αυτή η μέθοδος κάνει ασύγχρονη κλήση στο endpoint που επιστρέφει την κατάσταση λειτουργίας του ESP32:

```
private fun fetchEspStatus() {
    RetrofitClient.getApi().getStatus().enqueue(object : Callback<EspStatusResponse> {
        override fun onResponse(call: Call<EspStatusResponse>, response: Response<EspStatusResponse>) {
            if (response.isSuccessful && response.body() != null) {
                updateStatusUI(response.body()!!)
            } else {
                showDisconnected()
            }
        }

        override fun onFailure(call: Call<EspStatusResponse>, t: Throwable) {
            showDisconnected()
        }
    })
}
```

Αν η κλήση πετύχει και επιστραφούν δεδομένα, περνάμε την απάντηση στη updateStatusUI() για να ενημερωθεί το περιβάλλον χρήστη. Αν αποτύχει η κλήση ή δεν είναι επιτυχής, εμφανίζεται μήνυμα αποσύνδεσης μέσω showDisconnected().

- FetchDataFromESP32()

  Αυτή η μέθοδος καλεί το API που επιστρέφει τα δεδομένα αισθητήρων και πτήσης**:**

```
private fun fetchDataFromESP32() {
    RetrofitClient.getApi().getSensorData().enqueue(object : Callback<SensorData> {
        override fun onResponse(call: Call<SensorData>, response: Response<SensorData>) {
            if (response.isSuccessful) {
                val data = response.body()
                data?.let {
                    updateSensorData(
                        it.temperature,
                        it.humidity,
                        it.latitude,
                        it.longitude,
                        it.altitude,
                        it.sats,
                        it.vBat,
                        it.speed
                    )
                }
            }
        }

        override fun onFailure(call: Call<SensorData>, t: Throwable) {
            // Σε περίπτωση αποτυχίας, ενημερώνουμε με προεπιλεγμένες αρνητικές τιμές
            updateSensorData(-1f, -1f, -1f, -1f, -1f, 2, -1f, -1f)
        }
    })
}
```

Σε περίπτωση επιτυχίας, περνάμε τα ληφθέντα δεδομένα στη μέθοδο updateSensorData() που ενημερώνει το Dashboard. Σε αποτυχία, στέλνουμε προεπιλεγμένες τιμές που δείχνουν αποσύνδεση ή απουσία δεδομένων.

- updateSensorData()

  Η μέθοδος περνάει τις νέες μετρήσεις στο DashboardFrag ώστε να εμφανίζονται στον χρήστη:

```
private fun updateSensorData(
    temperature: Float,
    humidity: Float,
    latitude: Float,
    longitude: Float,
    altitude: Float,
    sats: Int,
    vBat: Float,
    speed: Float
) {
    val fragment = supportFragmentManager.findFragmentByTag("f" + viewPager.currentItem) as? DashboardFrag
    fragment?.updateSensorData(
        "${temperature}°C",
        "${humidity}%",
        "$latitude",
        "$longitude",
        "${altitude}m",
        "$sats",
        "${vBat}V",
        "${speed}m/s"
    )
}
```

- updateStatusUI()

  Η μέθοδος ενημερώνει το UI με το αν ο ESP32 είναι συνδεδεμένος και αν λειτουργεί σωστά:

```
private fun updateStatusUI(status: EspStatusResponse) {
    when {
        status.wifi_connected && status.dht_available && status.sd_available -> {
            statusTextView.text = "Connected"
            statusTextView.setTextColor(Color.GREEN)
        }
        status.wifi_connected && (!status.dht_available || !status.sd_available) -> {
            statusTextView.text = "Check sensor connection"
            statusTextView.setTextColor(Color.YELLOW)
        }
        else -> {
            showDisconnected()
        }
    }
}

// Εμφανίζει κατάσταση αποσύνδεσης
private fun showDisconnected() {
    statusTextView.text = "Disconnected"
    statusTextView.setTextColor(Color.RED)
}
```

4. Διαχείριση περιοδικών εργασιών

Η εφαρμογή χρειάζεται να ανανεώνει συνεχώς τα δεδομένα από το ESP32 σε τακτά χρονικά διαστήματα, ώστε να ενημερώνεται το περιβάλλον χρήστη σε πραγματικό χρόνο. Για αυτόν τον σκοπό, χρησιμοποιείται ένας Handler μαζί με ένα Runnable. Μέσα στο run() του Runnable, καλούνται οι δύο βασικές μέθοδοι fetchDataFromESP32() και fetchEspStatus() για να πάρουν δεδομένα από τη συσκευή.

Στο τέλος, το ίδιο το Runnable προγραμματίζεται να ξανατρέξει μετά από 1500 ms (1.5 δευτερόλεπτα), δημιουργώντας έτσι έναν ατελή βρόχο. Η μέθοδος startRepeatingTask() ξεκινά την πρώτη εκτέλεση του updateRunnable. Παρακάτω ακολουθεί ο κώδικας:

```
private val updateRunnable = object : Runnable {
    override fun run() {
        fetchDataFromESP32()   // Λήψη δεδομένων αισθητήρων
        fetchEspStatus()       // Λήψη κατάστασης ESP32
        handler.postDelayed(this, 1500) // Επανεκτέλεση μετά 1.5 δευτερόλεπτα
    }
}
private fun startRepeatingTask() {      // Έναρξη περιοδικής εκτέλεσης
    if (!isRepeatingTaskStarted) {
        handler.post(updateRunnable)
        isRepeatingTaskStarted = true
    }
```

5. Αποθήκευση και ανάκτηση διεύθυνσης IP

Η εφαρμογή χρειάζεται να γνωρίζει τη διεύθυνση IP του ESP32, ώστε να μπορεί να επικοινωνεί μαζί του μέσω των HTTP κλήσεων του Retrofit. Για να μη ζητείται από τον χρήστη κάθε φορά, η IP αποθηκεύεται τοπικά στη συσκευή χρησιμοποιώντας SharedPreferences, όπως παρακάτω:

```
private fun saveIpAddress(ip: String) {
    val sharedPref = getSharedPreferences("AppSettings", Context.MODE_PRIVATE)
    with(sharedPref.edit()) {
        putString("ESP32_IP", ip) // Αποθήκευση IP με το κλειδί "ESP32_IP"
        apply()                   // Εφαρμογή αλλαγών ασύγχρονα
    }
}
```

Για την ανάγνωση της αποθηκευμένης IP χρησιμοποιείται η getIpAddress που διαβάζει την IP με το ίδιο κλειδί, ενώ αν η IP δεν έχει αποθηκευτεί προηγουμένως, επιστρέφεται null τιμή:

```
private fun getIpAddress(): String? {
    val sharedPref = getSharedPreferences("AppSettings", Context.MODE_PRIVATE)
    return sharedPref.getString("ESP32_IP", null) // Επιστρέφει null αν δεν υπάρχει
}
```

6. Κλείσιμο και καθαρισμός πόρων

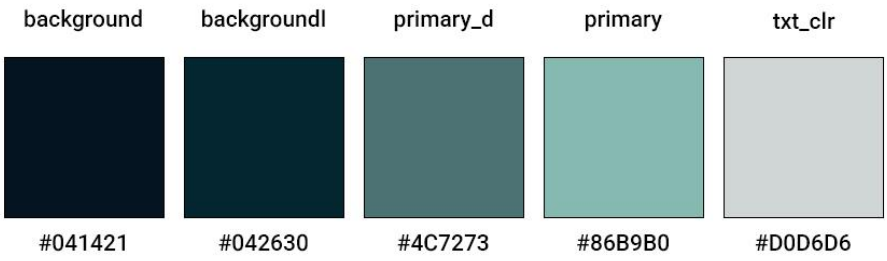
Όταν η MainActivity καταστρέφεται (π.χ. ο χρήστης κλείσει την εφαρμογή ή το σύστημα ανακτήσει μνήμη), πρέπει να εξασφαλίσουμε ότι θα σταματήσουν όλες οι διαδικασίες που τρέχουν στο παρασκήνιο, ώστε να αποφύγουμε σπατάλη πόρων ή σφάλματα.Ο κώδικας που το επιτυγχάνει είναι:

```
override fun onDestroy() {
    super.onDestroy()
    handler.removeCallbacks(updateRunnable) // Σταματά την περιοδική εκτέλεση
    udpListener.stopListening() // Κλείνει τον UDP listener
}
```

Με την ολοκλήρωση της ανάλυσης της MainActivity, καλύψαμε τον βασικό μηχανισμό επικοινωνίας, ελέγχου και ενημέρωσης της εφαρμογής.

4.7 Διεπαφή Χρήστη (UI)

Η διεπαφή χρήστη ή αλλιώς UI της εφαρμογής σχεδιάστηκε με γνώμονα την απλότητα και την άμμεση οπτική αναγνώριση των κρίσιμων δεδομένων του συστήματος. Η παλέτα χρωμάτων που χρησιμοποιήθηκε τονίζει τα σημαντικά σημεία που αφορούν τον χρήστη, διατηρώντας παράλληλα την ομοιομορφία σε όλα τα στοιχεία της εφαρμογής.



Εικόνα 4.1: Η παλέτα χρωμάτων της εφαρμογής

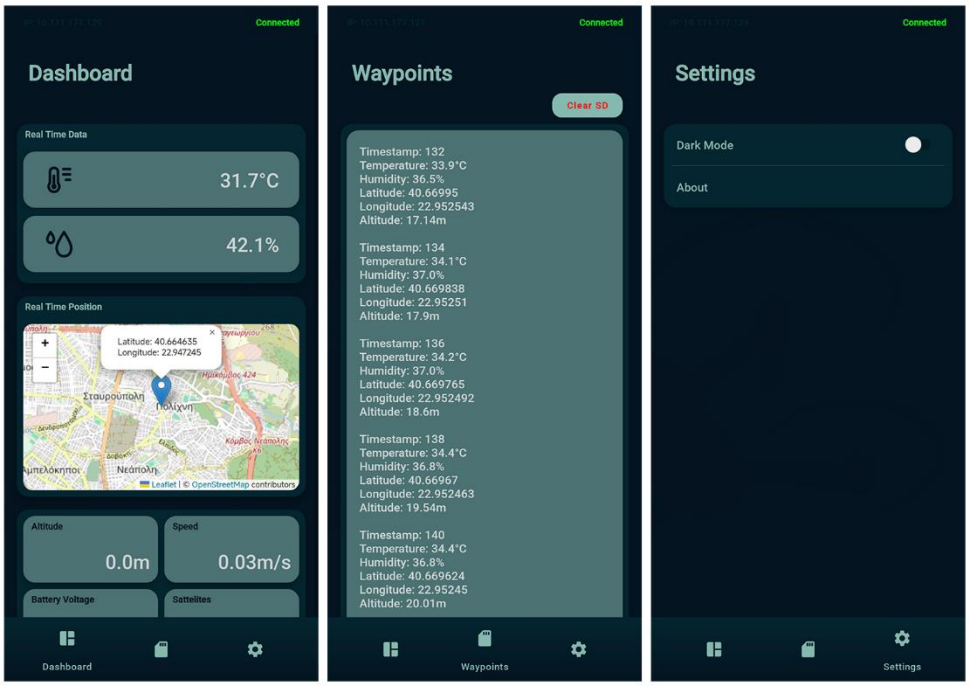
Για γραμματοσειρά του κειμένου της εφαρμογής επιλέχθηκε η Roboto Medium, μία γραμματοσειρά σχεδιασμένη για Android, η οποία διακρίνεται για την εύκολη ανάγνωση των χαρακτήρων και για την σύγχρονη αισθητική της.

Εικόνα 4.2: Η γραμματοσειρά Roboto Medium
[https://d29fhpw069ctt2.cloudfront.net/font/60123/preview/7d0d7c2659fed15e3296b61b9bc05051.jpg]

Η σχεδίαση των τριών καρτελών (fragments) ακολουθεί το ίδιο μοτίβο, με τον τίτλο της καρτέλας στο πάνω μέρος και από κάτω το περιεχόμενο. Πάνω από τον τίτλο κάθε καρτέλας, ο χρήστης μπορεί να δει στα αριστερά την IP του ESP32 και στα δεξιά την κατάσταση του συστήματος αισθητήρων (Connected, Check Sensor Connection, Disconnected).

Η καρτέλα Dashboard είναι η κεντρική οθόνη της εφαρμογής και περιέχει τα δεδομένα των αισθητήρων και πτήσης του drone σε πραγματικό χρόνο. Η καρτέλα Waypoints περιλαμβάνει τα δεδομένα που έχει αποθηκεύσει ο ESP32 στην κάρτα SD κατά την διάρκεια αυτόνομων πτήσεων και ένα κουμπί για την εκκαθάριση των δεδομένων. Η καρτέλα Settings δεν προσφέρει κάτι ουσιαστικό ακόμα αλλά κάνει την εφαρμογή να φαίνεται πιο ολοκληρωμένη.

Παρακάτω παρουσιάζονται οι τρεις καρτέλες (fragments) της εφαρμογής στην τελική τους μορφή:



Εικόνα 4.3: Το UI της εφαρμογής

4.8 Επίλογος

Με την ολοκλήρωση του 4ου κεφαλαίου ολοκληρώνεται η ανάλυση όλων των υποσυστημάτων του συστήματος, από την δημιουργία του κύριου συστήματος πτήσης έως την απεικόνιση των δεδομένων στον τελικό χρήστη.

Κεφάλαιο 5ο: Συμπεράσματα και Μελλοντικές Βελτιώσεις

5.1 Συνολική Αξιολόγηση Έργου

Η παρούσα εργασία παρουσίασε την ανάπτυξη και υλοποίηση ενός ολοκληρωμένου αυτόνομου μη επανδρωμένου drone που συλλέγει, επεξεργάζεται και αποστέλλει δεδομένα ασύρματα για απεικόνιση προς τον τελικό χρήστη μέσω μίας Android εφαρμογής. Το βασικό σύστημα πτήσης του drone συνεργάζεται με ένα ενσωματωμένο σύστημα αισθητήρων για το συνδυασμό χρήσιμων δεδομένων αισθητήρων – πτήσης, για τον έλεγχο και παρατήρηση γεωργικών εκτάσεων, καθώς και για την μετέπειτα ανάλυσή τους. Η ικανότητα του drone να εκτελεί σχεδιασμένες αποστολές αυτόνομα και με ασφάλεια μέσω μηχανισμών – δικλείδων ασφαλείας, αναβαθμίζει το σύστημα ως προς την αξιοπιστία του. Η δυνατότητα του συστήματος να συλλέγει και να αποθηκεύει τοπικά κρίσιμα δεδομένα αγροτικών καλλιεργειών, χωρίς να επηρεάζεται από εξωτερικούς παράγοντες, όπως η μη κάλυψη δικτύου ή διακοπές στην επικοινωνία, εξασφαλίζει την επιτυχία και την χρηστικότητά του για περιοχές που περιβάλλονται από έντονη βλάστηση. Τέλος, ο χρήστης μπορεί να έχει μία πλήρη εικόνα της κατάστασης και της θέσης του συστήματος μέσω μίας εφαρμογής που πλησιάζει τα σύγχρονα δεδομένα και δίνει πληροφορίες άμεσα και με μεγάλη αξιοπιστία.

Ιδιαίτερη σημασία έχει η επιτυχημένη λειτουργία της επικοινωνίας μεταξύ του συστήματος αισθητήρων και του ελεγκτή πτήσης μέσω του πρωτοκόλλου MAVlink, παρότι η υποστήριξη του INAV για το πρωτόκολλο είναι ακόμη περιορισμένη. Η εφαρμογή απέδειξε ότι είναι εφικτή η σταθερή και αξιόπιστη μετάδοση δεδομένων, διασφαλίζοντας ότι οι πληροφορίες των αισθητήρων φτάνουν σωστά στην εφαρμογή και αποθηκεύονται για περαιτέρω επεξεργασία.

Η εφαρμογή αποδείχθηκε εύχρηστη και αξιόπιστη, καθώς επιτρέπει τον έλεγχο της ορθής επικοινωνίας όλων των υποσυστημάτων και της επικοινωνίας με τον ESP32 χωρίς την ανάγκη άμεσης φυσικής πρόσβασης στο σύστημα αισθητήρων. Παράλληλα, παρέχει εύκολη διαχείριση και παρακολούθηση των δεδομένων που συλλέγονται, προσφέροντας μια πλήρη εικόνα της κατάστασης του συστήματος σε πραγματικό χρόνο και διευκολύνοντας την ανάλυση και αξιοποίηση των μετρήσεων.

Η Π.Ε. συνολικά επιβεβαίωσε την λειτουργικότητα ενός ολοκληρωμένου συστήματος συλλογής δεδομένων, ενώ ταυτόχρονα ανέδειξε τις δυνατότητες επέκτασης, τη σημασία της σωστής διαχείρισης δικτύου και της φιλικής προς το χρήστη διεπαφής. Παράλληλα, η υλοποίηση του συστήματος επιτεύχθηκε με χαμηλό κόστος, καθιστώντας το σύστημα προσβάσιμο για πειραματικούς σκοπούς ή εφαρμογές γεωργίας ακριβείας μικρής κλίμακας χωρίς να θυσιάζεται η αξιοπιστία και η αποτελεσματικότητά του.

5.2 Τελικά Χαρακτηριστικά Συστήματος

Το τελικό σύστημα κατάφερε να πετύχει τους αρχικούς στόχους του, όσον αφορά τα χαρακτηριστικά που ήταν επιθυμητό να έχει κατά την ολοκλήρωσή του για την εφαρμογή του στην γεωργία ακριβείας.

Μέσα από συνεχείς δοκιμαστικές αυτόνομες πτήσεις του συστήματος, μετρήθηκαν δεδομένα που αφορούν την αυτονομία, την κατανάλωση ενέργειας, την ορθή συλλογή δεδομένων και την ακρίβεια της πλοήγησης του συστήματος κατά την διάρκεια των πτήσεων. Ο πίνακας 5.1 παρουσιάζει τα τεχνικά χαρακτηριστικά του τελικού συστήματος όπως αυτά μετρήθηκαν:

Πίνακας 5.1: Τεχνικά χαρακτηριστικά συστήματος

Τεχνικά Χαρακτηριστικά Συστήματος				
Χαρακτηριστικό	**Υποσύστημα**	**Ελάχιστη**	**Τυπική**	**Μέγιστη**
Τάση λειτουργίας (V)	-	18V	22,2V	25,2V
Κατανάλωση ρεύματος (A)	Ελεγκτής πτήσης (FC)	200mA	-	300mA
	Κινητήρες	4A	30A	44A
	GPS	-	10mA	-
	Δέκτης	-	22,5mA	-
	ESP32	150mA	-	260mA
	DHT22	0,5mA	1mA	1,5mA
	microSD module	5mA	-	100mA
Κατανάλωση ισχύος (W)	-	97,4W	680,3W	992,2W
Χωρητικότητα μπαταρίας (mAh)	-	-	4.500mAh	-
Διάρκεια αυτόνομης πτήσης	-	20mins	-	25mins
Βάρος	-			1,130kg
Ακρίβεια Αισθητήρων				
Αισθητήρας	**Παράμετρος**	**Εύρος μέτρησης**		**Ακρίβεια**
DHT22	Θερμοκρασία	-40°C έως 80°C		±0,5°C
	Υγρασία	0 - 100% RH		±2-3% RH
GPS	Γεωγραφικό πλάτος	-		±1,5m
	Γεωγραφικό μήκος	-		±1,5m
	Υψόμετρο	-		±1m
	Ταχύτητα	-		±0,05m/s
Εμβέλεια επικοινωνίας				
Είδος επικοινωνίας	**Ισχύς εκπομπής**	**Ελάχιστη**	**Μέγιστη**	**Σχόλια**
ESP32 WiFi	100mW	~100m	~200m	Επιρρεάζεται από εμπόδια
Πομπός - Δέκτης Ραδιοχειρισμού	100mW	1km	1,5km	Επιρρεάζεται από εμπόδια
Πλοήγηση	Μόνο λήψη	∞	∞	Περιορίζεται μόνο από την αυτονομία του συστήματος

Με βάση τον πίνακα μπορούμε να επιβεβαιώσουμε πως το σύστημα καθίσταται κατάλληλο για εφαρμογή στη γεωργία ακριβείας, όπου η αυτονομία, η χαμηλή κατανάλωση ενέργειας και η ακρίβεια των αισθητήρων είναι κρίσιμα στοιχεία. Η μέγιστη κατανάλωση ρεύματος στους κινητήρες φτάνει τα 44A (λόγω μέγιστης συνεχούς τάσης αποφόρτισης της μπαταρίας στα ~45A), με τα υπόλοιπα στοιχεία να καταναλώνουν σημαντικά λιγότερη ενέργεια, γεγονός που υποστηρίζει τη χαμηλή κατανάλωση και βέλτιστη απόδοση της μπαταρίας. Η μπαταρία του συστήματος, χωρητικότητας 4.500mAh, είναι ικανή να υποστηρίξει αυτόνομες πτήσεις έως 25 λεπτά, επιτρέποντας την κάλυψη μεγάλων γεωργικών εκτάσεων με μία μόνο φόρτιση. Στην μεγάλη αυτονομία συμβάλει αρκετά το ιδανικό συνολικό βάρος των 1,130 kg του συστήματος, το οποίο δεν επιβαρύνει αρκετά τους τέσσερεις κινητήρες και κάνει το drone πιο ευέλικτο σε γρήγορες αλλαγές κατεύθυνσης.

Η ακρίβεια των αισθητήρων του DHT22 και του GPS, αποτελούν μεγάλο πλεονέκτημα του συστήματος ως προς την ασφαλή και αποδοτική εκτέλεση αυτόνομων πτήσεων. Ο DHT22 μετρά θερμοκρασία και υγρασία με ανοχή ±0,5°C και 2-3% RH αντίστοιχα, το οποίο είναι αρκετό για ανάλυση μικροκλίματος σε γεωργικές καλλιέργειες. Το γεωγραφικό πλάτος και μήκος έχουν ακρίβεια ±1,5m, ενώ το υψόμετρο ποτέ δεν ξεπέρασε το 1 μέτρο απόκλιση από το ιδανικό. Η απόκλιση της γεωργαφικής θέσης του drone, αν και παρουσιάζεται αυξημένη, στην πράξη σπάνια ξεπέρασε τα ±0,5m παρά μόνο σε περιπτώσεις αδύναμίας σύνδεσης του GPS με αρκετούς δορυφόρους λόγω εμποδίων ή σε συνθήκες ισχυρών ανέμων.

Η εμβέλεια του WiFi του συστήματος αισθητήρων μέσω του ESP32 αγγίζει τα 200 μέτρα περίπου σε ανοιχτό πεδίο, για αποστολή των δεδομένων πτήσης σε πραγματικό χρόνο μέσω δικτύου. Αυτό αποτελεί έναν περιορισμό του drone, όπως αναφέρεται και στην ενότητα 5.4, ο οποίος έχει ληφθεί υπόψη για επίλυση σε μελλοντική αναβάθμιση του συστήματος. Ο χειροκίνητος ελεγχος είναι εφικτός σε απόσταση έως και 1,5km μακριά από το σύστημα, εφόσον η σύνδεση είναι LOS (Line-of-Sight) και μπορεί να χρησιμεύσει σε περιπτώσεις που ο χειριστής θέλει να αποφύγει κάποιο εμπόδιο ή αν επιθυμεί απλή παρακολούθηση κάποιας γεωργικής έκτασης.

Η συνδυασμένοι παράγοντες χαμηλής κατανάλωσης, αξιόπιστων αισθητήρων και επαρκούς αυτονομίας επιτρέπει στο μη επανδρωμένο σύστημα να εκτελεί αυτόνομες αποστολές συλλογής δεδομένων με μεγάλη ακρίβεια και αποτελεσματικότητα, υποστηρίζοντας την εφαρμογή του στη γεωργία ακριβείας.

5.3 Προκλήσεις και Αντιμετώπιση

Κατά την υλοποίηση του συστήματος προέκυψαν αρκετές προκλήσεις που είτε προσπεράστηκαν, είτε άλλαξαν τον τρόπο ή την δομή της υλοποίησης.

Στην παραμετροποίηση του συστήματος του drone μέσω του λογισμικού πτήσης INAV Configurator, υπήρξαν δυσκολίες κατά το calibration του μαγνητόμετρου του GPS module. Το GPS για να δουλέψει σωστά και χωρίς παρεμβολές δείχνοντας την σωστή κατεύθυνση που “κοιτάει” το drone, έπρεπε να είναι σωστά τοποθετημένο στο πλαίσιο και μακριά από καλώδια υψηλής ισχύος. Για τον σκοπό αυτό, προστέθηκε στο πλαίσιο ένα πλαστικό εξάρτημα στήριξης του GPS, τοποθετώντας το μακρύτερα από τα άλλα εξαρτήματα του drone και κυρίως των καλωδίων της μπαταρίας. Παρόλα αυτά, μετά από αυτή την μετατροπή, το μαγνητόμετρο συνέχισε να έχει απόκλιση περίπου 90° μοιρών από την σωστή κατεύθυνση. Η λύση του προβλήματος ήταν η σωστή τοποθέτηση του GPS μέσα στο γραφικό περιβάλλον του INAV (καρτέλα Alignment Tool). Έγινε οριζόντια στροφή 90° μοιρών του στοιχείου

GPS και ρυθμίστηκε η κλίση του στις 25° μοίρες, με αποτέλεσμα το μαγνητόμετρο μετά το calibration να δείχνει στην σωστή κατεύθυνση.

Κατά την υλοποίηση του συστήματος αισθητήρων και συγκεκριμένα στο κομμάτι ανίχνευσης αυτόνομης αποστολής μέσω του πρωτοκόλλου MAVlink, δεν ήταν ξεκάθαρο ποια μηνύματα στέλνει ο ελεγκτής πτήσης, ούτε τι πληροφορίες περιέχουν τα μηνύματα, λόγω μη ύπαρξης ξεκάθαρου οδηγού χρήσης της υλοποίησης του πρωτοκόλλου από το INAV (πιθανότατα λόγω περιορισμένης υποστήριξης από το λογισμικό). Για την αναζήτηση του κατάλληλου μηνύματος και του bit που χρειαζόταν για την ανίχνευση της αυτόνομης αποστολής, χρησιμοποιήθηκε η SD κάρτα που ενσωματώθηκε στον ESP32 και αρκετές δοκιμαστικές πτήσεις σε αυτόνομη λειτουργία. Σε κάθε πτήση ο ESP32 κατέγραφε στην κάρτα όλα τα MAVlink μηνύματα που έστελνε το drone, μέχρι που βρέθηκε το κατάλληλο bit που χρειαζόταν (μήνυμα HEARTBEAT, ID = 0, custom_mode = 3).

Η επικοινωνία του ESP32 με την εφαρμογή, σε πρώτο στάδιο, υλοποιήθηκε μέσω στατικής IP. Ο κώδικας του ESP32, διαμόρφωνε την IP διεύθυνση με Host ID = 200 (192.168.1.200), και παράλληλα η εφαρμογή για να λάβει τα δεδομένα έστελνε HTTP αιτήματα σε αυτήν την IP. Παρατηρήθηκε όμως, ότι υπήρχαν προβλήματα στην επικοινωνία σε περιπτώσεις όπου η συγκεκριμένη IP είχε καταληφθεί από άλλη συσκευή (IP conflict) ή στο ενδεχόμενο όπου ένας δρομολογητής ή διακομιστής έδινε στον ESP32 μία διεύθυνση με διαφορετικό Network ID (π.χ. 192.168.10.200). Για παράδειγμα, η λειτουργία Hotspot των smartphones, για λόγους ασφαλείας είναι ρυθμισμένη να δίνει στις συσκευές διευθύνσεις με διαφορετικό Network ID κάθε φορά που συνδέονται. Έτσι, για την επιτυχή σύνδεση ESP32-Εφαρμογής κάθε φορά που ενεργοποιείται το σύστημα, υλοποιήθηκε κώδικας στον ESP32 που αποστέλλει την διεύθυνση IP του σε όλες τις συσκευές του δικτύου μέσω UDP broadcast. Από την μεριά της εφαρμογής, μέσω της Retrofit, ο κώδικας διαβάζει την IP του ESP32, την αποθηκεύει σε μία μεταβλητή και την χρησιμοποιεί στο baseUrl για να στείλει τα HTTP GET αιτήματα.

5.4 Επεκτασιμότητα και Μελλοντικές Βελτιώσεις

Η αρχιτεκτονική του συστήματος διευκολύνει την επεκτασιμότητα, επιτρέποντας την εύκολη ενσωμάτωση νέων λειτουργιών και τεχνολογιών.

Η προσθήκη επιπλέον αισθητήρων μελλοντικά θα μπορούσε να ενισχύσει σημαντικά τις δυνατότητες παρακολούθησης και καταγραφής δεδομένων. Ενδεικτικά, η χρήση αισθητήρων ποιότητας αέρα (π.χ. CO_2, PM2.5/PM10), υγρασίας εδάφους, βαρομετρικής πίεσης, φωτεινότητας και ηλιακής ακτινοβολίας θα προσέφερε πιο ολοκληρωμένη εικόνα για την ποιότητα των καλλιεργειών και θα άνοιγε τον δρόμο για περεταίρω ανάλυση. Επιπλέον, η προσθήκη μίας RGB κάμερας για παρακολούθηση του αγρού απομακρυσμένα θα μπορουσε να θεωρηθεί αναγκαία για μεγάλες εκτάσεις καλλιεργιών όπου είναι δύσκολη η επιτήρηση με φυσικό τρόπο. Η προσθήκη πολυφασματικών ή θερμικών καμερών θα άνοιγε τον δρόμο για ανάλυση δεδομένων και πιο αποτελεσματική διαχείριση των δεδομένων, παρόλα αυτά η

χρήση τόσο εξειδικευμένων αισθητήρων ανεβάζει ραγδαία το κόστος κατασκευής και υλοποίησης του συστήματος.

Η επικοινωνία μεταξύ του συστήματος αισθητήρων και της εφαρμογής βασίζεται σε τοπικό δίκτυο Wi-Fi, που αποτελεί καλή λύση αλλά περιορίζει την λειτουργικότητα του συστήματος λόγω μικρής εμβέλειας. Μία αποτελεσματική λύση θα ήταν η ενσωμάτωση ενός GSM/LTE module στο σύστημα του ESP32 για καλυψη σε απομακρυσμένες περιοχές χωρίς την ανάγκη τοπικής σύνδεσης. Ο ESP32 θα μπορούσε να στέλνει τα δέδομένα σε έναν server (broker), μέσω του πρωτοκόλλου MQTT και η εφαρμογή με την σειρά της να ανακτά τα δεδομένα αυτά για απεικόνιση. Το MQTT αποτελεί ένα ελαφρύ πρωτόκολλο ανταλλαγής μηνυμάτων, επιτρέποντας χαμηλή κατανάλωση πόρων και άμεση ενημέρωση σε πραγματικό χρόνο.

5.5 Κοστολόγιο Υλικών Κατασκευής

Το κόστος των εξαρτημάτων και πρόσθετων υλικών που χρησιμοποιήθηκαν για την επιτυχή ολοκλήρωση του έργου ακολουθεί στον παρακάτω πίνακα:

Πίνακας 5.2: Κοστολόγιο υλικών κατασκευής

Κατασκευή του drone		
Πλαίσιο	GEPRC Mark4 7”	**45,96 €**
Κινητήρες	YSIDO 2807 1.300kV	**41,73 €**
FC – ESC Stack	SpeedyBee F405 v.4 BLS 60A	**60,95 €**
Μπαταρία	4x Molicel P45B	**48,88 €**
GPS	GEPRC M10Q	**25,98 €**
Προπέλες	HQProp 7040 7x4x3mm	**2,94 €**
Δέκτης	RadioMaster RP1	**19,93 €**
Πομπός	RadioMaster Pocket (LBT)	**89,58 €**
Πρόσθετα υλικά	Καλώδια, βίδες, βύσματα κ.α.	**20,00 €**
Ενσωματωμένο σύστημα αισθητήρων		
Μικροελεγκτής	NodeMCU ESP-32	**10,00 €**
Μονάδα αποθήκευσης	Micro SD TF card module	**1,50 €**
Αισθητήρας θερμοκρασίας-υγρασίας	DHT22 AM2302	**9,00 €**
BEC	Diatone Mamba micro 2A 5V/9V	**3,69 €**
Micro SD κάρτα	Intenso microSD HC 32GB	**6,00 €**
Διάτρητες πλακέτες	2x 4x6mm	**2,00 €**
Κουτί ηλεκτρονικών	4x6mm	**0,80 €**
Πρόσθετα υλικά	Καλώδια, pin headers-connectors	**5,00 €**
	Σύνολο:	**393,94 €**

ΒΙΒΛΙΟΓΡΑΦΙΑ

Research Articles

[1] D. C. Tsouros, S. Bibi and P. G. Sarigiannidis, "A Review on UAV-Based Applications for Precision Agriculture," Information, vol. 10, no. 11, p. 349, Nov. 2019, doi: 10.3390/info10110349.

[2] H. Ben Salah, A. Ouled Belgacem, S. Rejeb and K. Rejeb, "Use of UAV-Based Multispectral Imagery for Monitoring Olive Trees in Arid Regions: A Case Study in Tunisia," Sustainability, vol. 15, no. 2, p. 1157, Jan. 2023, doi: 10.3390/su15021157.

[3] Peksa, J.; Mamchur, D. A Review on the State of the Art in Copter Drones and Flight Control Systems. Sensors 2024, 24, 3349. https://doi.org/10.3390/s24113349

[4] Aliane, N. A Survey of Open-Source UAV Autopilots.Electronics 2024, 13, 4785. https://

doi.org/10.3390/electronics13234785.

[5] Siva Sivamani GK, Gudipalli A. Design and implementation of DATA logging and stabilization system for a UAV. Heliyon. 2024 Feb 14;10(4):e26394. doi: 10.1016/j.heliyon.2024.e26394. PMID: 38390123; PMCID: PMC10881421.

[6] "User interface for monitoring the flight of a drone," *Embention*. [Online]. Available: https://www.embention.com/news/user-interface-for-monitoring-drone/. [Accessed: Aug. 13, 2025].

[7] P. Radoglou-Grammatikis, P. Sarigiannidis, T. Lagkas, and I. Moscholios, "A compilation of UAV applications for precision agriculture," Comput. Netw., vol. 172, p. 107148, 2020, doi: 10.1016/j.comnet.2020.107148.

[8] L. Roth and B. Streit, "Predicting cover crop biomass by lightweight UAS-based RGB and NIR photography: an applied photogrammetric approach," Precision Agriculture, vol. 19, pp. 93–114, 2018, doi: 10.1007/s11119-017-9501-1.

[9] A. Agrawal and M. Y. Arafat, "Transforming Farming: A Review of AI-Powered UAV Technologies in Precision Agriculture," Drones, vol. 8, no. 11, art. 664, Nov. 2024, doi: 10.3390/drones8110664.

[10] S. Bonny, K. Johansen, Y. M. Malbéteau, and M. F. McCabe, "Detecting Plant Stress Using Thermal and Optical Imagery From an Unoccupied Aerial Vehicle," Frontiers in Plant Science, vol. 12, 2021, doi:10.3389/fpls.2021.734944.

[11] S. Lu, T. Zhang, and F. Tian, "Evaluation of Crop Water Status and Vegetation Dynamics for Alternate Partial Root-Zone Drip Irrigation of Alfalfa: Observation with an UAV Thermal Infrared Imagery," Frontiers in Environmental Science, vol. 10, 2022, doi: 10.3389/fenvs.2022.791982.

[12] M. Li, R. R. Shamshiri, C. Weltzien, and M. Schirrmann, "Crop Monitoring Using Sentinel-2 and UAV Multispectral Imagery: A Comparison Case Study in Northeastern Germany," Remote Sensing, vol. 14, no. 17, art. 4426, 2022, doi: 10.3390/rs14174426. [Online]. Available: https://doi.org/10.3390/rs14174426

Papers

[13]H. Ruff, G. Calhoun, E. Frost, K. Behymer, and J. Bartik, “Comparison of Adaptive, Adaptable, and Hybrid Automation for Surveillance Task Completion in a Multi-Task Environment,” Proc. Hum. Factors Ergonom. Soc. Annu. Meeting, vol. 62, no. 1, pp. 155–159, 2018, doi: 10.1177/1541931218621036.

[14] J.-P. Huttner and M. Friedrich, “Current Challenges in Mission Planning Systems for UAVs: A Systematic Review,” in *2023 Integrated Communication, Navigation and Surveillance Conference (ICNS 2023)*, Washington D.C., USA, Apr. 18–20, 2023. doi: 10.1109/ICNS58246.2023.10124299.

[15]F. Z. Bassine, T. E. Epule, A. Kechchour, and A. Chehbouni, “Recent applications of machine learning, remote sensing, and IoT approaches in yield prediction: A critical review,” arXiv, Jun. 7, 2023. [Online]. Available: https://arxiv.org/abs/2306.04566

Internet Sites

[16] “ESP32,” *Wikipedia*, Available: https://en.wikipedia.org/wiki/ESP32. Accessed: Aug. 13, 2025.

[17] Wikipedia contributors, "Real-time kinematic positioning," Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/Real-time_kinematic_positioning. [Accessed: 13-Aug-2025].

[18] DroneDeploy, "RTK vs PPK: Choosing the right GPS correction method for drone mapping," DroneDeploy, 25-Apr-2025. [Online]. Available: https://www.dronedeploy.com/blog/what-is-the-difference-between-rtk-ppk-and-gcp-and-why-does-it-matter. [Accessed: 13-Aug-2025].

[19] DJI, "DJI Enterprise – Drone Solutions for Your Business," DJI Enterprise. [Online]. Available: https://enterprise.dji.com. [Accessed: 13-Aug-2025].

[20] DJI, "DJI Agras T50," DJI Agricultural Drones. [Online]. Available: https://ag.dji.com/t50. [Accessed: 13-Aug-2025].

Repositories

[21] S. Cocchi, “Esp32-cam-telemetry: DIY drone telemetry system using an esp32-cam,” GitHub. [Online]. Available: https://github.com/sebastiano123-c/Esp32-cam-telemetry. [Accessed: Aug. 13, 2025].