NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF DIGITAL INDUSTRY TECHNOLOGIES

BSc THESIS

Implementation of an Autonomous Quadcopter Landing System on a Mobile Platform using Machine Vision Techniques

Dimitrios I. Fanourgakis
Spyridon Antonios K. Fronimos

Supervisor: **Michael G. Skarpetis, Associate Professor**

ATHENS
MAY 2026

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΤΕΧΝΟΛΟΓΙΩΝ ΨΗΦΙΑΚΗΣ ΒΙΟΜΗΧΑΝΙΑΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση Συστήματος Αυτόνομης Προσγείωσης Τετρακοπτέρου σε Κινητή Πλατφόρμα με Τεχνικές Μηχανικής Όρασης

Δημήτριος Ι. Φανουργάκης
Σπυρίδων Αντώνιος Κ. Φρόνιμος

Επιβλέπων: **Μιχαήλ Σκαρπέτης**, Αναπληρωτής Καθηγητής Συστημάτων Αυτομάτου Ελέγχου – Υδραυλικών και Πνευματικών Συστημάτων Αυτομάτου Ελέγχου

ΑΘΗΝΑ
ΜΑΪΟΣ 2026

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση Συστήματος Αυτόνομης Προσγείωσης Τετρακοπτέρου σε Κινητή Πλατφόρμα με τεχνικές μηχανικής όρασης

Δημήτριος Ι. Φανουργάκης

Α.Μ.: 1117202100219

Σπυρίδων Αντώνιος Κ. Φρόνιμος

Α.Μ.: 1117202000223

ΕΠΙΒΛΕΠΩΝ: **Μιχαήλ Σκαρπέτης,** Αναπληρωτής Καθηγητής Συστημάτων Αυτομάτου Ελέγχου – Υδραυλικών και Πνευματικών Συστημάτων Αυτομάτου Ελέγχου

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία αφορά τον σχεδιασμό, την κατασκευή και την πειραματική διερεύνηση ενός συστήματος αυτόνομης προσγείωσης τετρακοπτέρου σε κινητή πλατφόρμα, με χρήση τεχνικών μηχανικής όρασης. Στο πλαίσιο της εργασίας αναπτύχθηκε τετρακόπτερο γεωμετρίας τύπου F450, το οποίο κατασκευάστηκε μέσω τρισδιάστατης εκτύπωσης και εξοπλίστηκε με ελεγκτή πτήσης, σύστημα ασύρματης επικοινωνίας και κάμερα μηχανικής όρασης. Η επιλογή της γεωμετρίας F450 έγινε λόγω της μηχανικής απλότητας, της συμμετρικής διάταξης των κινητήρων και της δυνατότητας χρήσης συμβατών εξαρτημάτων, όπως ο πίνακας διανομής ισχύος.

Παράλληλα, σχεδιάστηκε και κατασκευάστηκε επίγειο τροχοφόρο όχημα διαφορικής οδήγησης, πάνω στο οποίο τοποθετήθηκε πλατφόρμα προσγείωσης με μηχανισμό αυτόματης εξισορρόπησης. Η πλατφόρμα υλοποιήθηκε με χρήση τρισδιάστατης σχεδίασης και εκτύπωσης, ενώ για τον έλεγχό της αξιοποιήθηκαν μικροελεγκτής, αισθητήρας αδρανειακής μέτρησης και σερβοκινητήρες. Στόχος του μηχανισμού αυτόματης εξισορρόπησης ήταν η διατήρηση της επιφάνειας προσγείωσης σε όσο το δυνατόν πιο οριζόντια θέση, ώστε να δημιουργηθεί ένα πιο σταθερό σημείο επαφής για το τετρακόπτερο.

Για την αναγνώριση της πλατφόρμας αναπτύχθηκαν αλγόριθμοι μηχανικής όρασης βασισμένοι σε δείκτες ArUco, ενώ πραγματοποιήθηκε βαθμονόμηση της κάμερας με χρήση ChArUco board. Η εικόνα από την κάμερα επεξεργάζεται με σκοπό την εκτίμηση της σχετικής θέσης του τετρακοπτέρου ως προς τον στόχο και την παραγωγή κατάλληλων εντολών καθοδήγησης. Επιπλέον, η λογική της αυτόνομης προσγείωσης εξετάστηκε μέσω προσομοίωσης, ώστε να αξιολογηθεί η συμπεριφορά του συστήματος σε διαφορετικές σχετικές θέσεις μεταξύ κάμερας και πλατφόρμας. Τα αποτελέσματα δείχνουν ότι ο συνδυασμός τρισδιάστατης κατασκευής, ενσωματωμένων συστημάτων, μηχανικής όρασης και ανοικτών πλατφορμών ελέγχου μπορεί να αποτελέσει λειτουργική βάση για την ανάπτυξη συστημάτων αυτόνομης προσγείωσης σε κινητές πλατφόρμες.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Αυτόματος Έλεγχος Βιομηχανικών Συστημάτων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Αυτόνομα ρομποτικά οχήματα, τετρακόπτερο, μηχανική όραση, ArUco markers, τρισδιάστατη εκτύπωση, κινητή πλατφόρμα.

ABSTRACT

This undergraduate thesis concerns the design, construction and experimental investigation of an autonomous quadcopter landing system on a mobile platform, using machine vision techniques. As part of the project, an F450-type quadcopter was developed and manufactured through 3D printing. The aerial vehicle was equipped with a flight controller, a wireless communication system and a machine vision camera. The F450 geometry was selected due to its mechanical simplicity, symmetrical motor arrangement and compatibility with suitable components, such as the power distribution board.

In parallel, a differential-drive ground vehicle was designed and constructed, carrying a landing platform with an automatic balancing mechanism. The platform was implemented through 3D design and printing, while its control system was based on a microcontroller, an inertial measurement sensor and servomotors. The objective of the automatic balancing mechanism was to maintain the landing surface in an approximately horizontal position, in order to provide a more stable contact point for the quadcopter.

For the recognition of the landing platform, machine vision algorithms based on ArUco markers were developed, while camera calibration was performed using a ChArUco board. The image captured by the camera is processed to estimate the relative position of the quadcopter with respect to the target and to generate appropriate guidance commands. In addition, the autonomous landing logic was examined through simulation, allowing the behaviour of the system to be evaluated under different relative positions between the camera and the platform. The results indicate that the combination of 3D manufacturing, embedded systems, machine vision and open-source control platforms can provide a functional basis for the development of autonomous landing systems on mobile platforms.

SUBJECT AREA: Automatic Control of Industrial Systems

KEYWORDS: Autonomous robotic vehicles, quadcopter, machine vision, ArUco markers, 3D printing, mobile platform.

ΕΥΧΑΡΙΣΤΙΕΣ

Για τη διεκπεραίωση της παρούσας Πτυχιακής Εργασίας, θα θέλαμε να ευχαριστήσουμε τον επιβλέποντα καθηγητή Μιχαήλ Σκαρπέτη, για τη συνεργασία και την πολύτιμη συμβολή του στην ολοκλήρωση της. Επιπλέον, θα θέλαμε να ευχαριστήσουμε ο ένας τον άλλον για την συνεργασία που είχαμε. Τέλος, ευχαριστούμε την συμφοιτήτρια μας, Αθανασία Λάμπρου, για την πολύτιμη βοήθεια και υποστήριξη καθ' όλη την έρευνα και συγγραφή.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΠΡΟΛΟΓΟΣ

Η παρούσα πτυχιακή εργασία εκπονήθηκε στο πλαίσιο του προγράμματος σπουδών του Τμήματος Τεχνολογιών Ψηφιακής Βιομηχανίας του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών. Αντικείμενό της αποτελεί η μελέτη και η υλοποίηση ενός συστήματος αυτόνομης προσγείωσης τετρακοπτέρου σε κινητή πλατφόρμα, με στόχο τη σύνδεση γνώσεων από διαφορετικά πεδία της σύγχρονης ψηφιακής βιομηχανίας.

Το θεωρητικό, σχεδιαστικό και προγραμματιστικό μέρος της εργασίας πραγματοποιήθηκε κυρίως σε ιδιωτικό χώρο εργασίας, όπου αναπτύχθηκαν τα μηχανικά σχέδια, το λογισμικό και η γενική αρχιτεκτονική του συστήματος. Το πειραματικό μέρος της εργασίας, το οποίο περιλάμβανε τη συναρμολόγηση, τις δοκιμές, τις ρυθμίσεις και την αξιολόγηση των επιμέρους υποσυστημάτων, πραγματοποιήθηκε στο εργαστήριο RCCL του Τμήματος.

Η επιλογή του συγκεκριμένου θέματος προέκυψε από το ενδιαφέρον μας για τα αυτόνομα ρομποτικά συστήματα και ειδικότερα για τη συνεργασία εναέριων και επίγειων οχημάτων. Η εργασία αποτέλεσε μία σύνθετη διαδικασία, καθώς απαιτούσε τον συνδυασμό θεωρητικής μελέτης, μηχανικού σχεδιασμού, κατασκευής, ηλεκτρολογικής διασύνδεσης, προγραμματισμού και πειραματικών δοκιμών.

Κατά τη διάρκεια της εκπόνησης της εργασίας αντιμετωπίστηκαν αρκετές πρακτικές δυσκολίες, οι οποίες ανέδειξαν τη σημασία της σωστής οργάνωσης, της επαναληπτικής δοκιμής και της προσαρμογής των αρχικών σχεδιαστικών επιλογών στις πραγματικές ανάγκες του πρωτοτύπου. Η διαδικασία αυτή συνέβαλε ουσιαστικά στην κατανόηση της μετάβασης από τη θεωρητική προσέγγιση στην πρακτική υλοποίηση ενός ολοκληρωμένου ρομποτικού συστήματος.

Σκοπός του παρόντος κειμένου είναι να παρουσιάσει με οργανωμένο τρόπο το θεωρητικό υπόβαθρο, τη μεθοδολογία, τη διαδικασία κατασκευής και προγραμματισμού, καθώς και την αξιολόγηση του συστήματος που αναπτύχθηκε. Η εργασία φιλοδοξεί να αποτελέσει βάση για μελλοντική βελτίωση και επέκταση αντίστοιχων συστημάτων αυτόνομης προσγείωσης σε κινητές πλατφόρμες.

1. ΒΙΒΛΙΟΓΡΑΦΙΚΗ ΑΝΑΣΚΟΠΗΣΗ

1.1 Θεωρητικό Πλαίσιο και Βασικές Έννοιες των Τετρακοπτέρων

Τα τετρακόπτερα (quadcopters) εντάσσονται στην ευρύτερη κατηγορία των Μη Επανδρωμένων Εναέριων Οχημάτων (Unmanned Aerial Vehicles - UAV) και αποτελούν ειδική μορφή πολυκοπτέρων (multirotors). Χαρακτηρίζονται από τη χρήση τεσσάρων ροτόρων, οι οποίοι παράγουν την απαιτούμενη άνωση και επιτρέπουν τον έλεγχο της κίνησης και του προσανατολισμού του οχήματος στον τρισδιάστατο χώρο [12], [16]. Η εξέλιξη της μικροηλεκτρονικής, των αισθητήρων, των συστημάτων αυτομάτου ελέγχου και των ενσωματωμένων υπολογιστικών μονάδων έχει συμβάλει σημαντικά στη διάδοση των τετρακοπτέρων τόσο στον ερευνητικό χώρο όσο και σε πρακτικές εφαρμογές.

Η ευρεία χρήση των τετρακοπτέρων οφείλεται σε συγκεκριμένα λειτουργικά χαρακτηριστικά που τα καθιστούν κατάλληλα για εφαρμογές όπου απαιτείται ευελιξία και ακρίβεια. Σε αντίθεση με τα συμβατικά αεροσκάφη σταθερών πτερύγων, τα τετρακόπτερα έχουν τη δυνατότητα κάθετης απογείωσης και προσγείωσης (Vertical Take-Off and Landing - VTOL), σταθερής αιώρησης (hovering), πτήσης σε χαμηλές ταχύτητες και εκτέλεσης ελιγμών σε περιορισμένο χώρο. Για τον λόγο αυτό χρησιμοποιούνται σε εφαρμογές όπως η εναέρια επιτήρηση, η χαρτογράφηση, η επιθεώρηση τεχνικών υποδομών, η γεωργία ακριβείας, η έρευνα και διάσωση, καθώς και η μεταφορά μικρών φορτίων [5], [12], [17], [57], [59].

Ως Μη Επανδρωμένο Εναέριο Όχημα ορίζεται κάθε ιπτάμενο σύστημα που λειτουργεί χωρίς την παρουσία ανθρώπινου χειριστή στο εσωτερικό του. Ο έλεγχος της πτήσης μπορεί να πραγματοποιείται είτε απομακρυσμένα, μέσω χειριστή και συστήματος τηλεχειρισμού, είτε αυτόνομα, μέσω ενσωματωμένων αλγορίθμων πλοήγησης, σταθεροποίησης και ελέγχου. Στην περίπτωση των τετρακοπτέρων, η δυνατότητα αυτόνομης λειτουργίας βασίζεται στη συνεχή επεξεργασία δεδομένων από αισθητήρες, όπως γυροσκόπια, επιταχυνσιόμετρα, βαρόμετρα, κάμερες ή άλλα συστήματα αντίληψης του περιβάλλοντος.

Από δυναμική άποψη, το τετρακόπτερο αποτελεί ένα εγγενώς ασταθές σύστημα, καθώς δεν μπορεί να διατηρήσει σταθερή πτήση χωρίς συνεχή έλεγχο των κινητήρων του. Η σταθεροποίηση επιτυγχάνεται μέσω της ηλεκτρονικής μονάδας ελέγχου πτήσης (flight controller), η οποία λαμβάνει δεδομένα από τους αισθητήρες, εκτιμά την κατάσταση του οχήματος και ρυθμίζει σε πραγματικό χρόνο την ταχύτητα περιστροφής των τεσσάρων κινητήρων. Με τον τρόπο αυτό ελέγχονται η κατακόρυφη κίνηση, η στάση και η συνολική συμπεριφορά του τετρακοπτέρου κατά την πτήση.

Στο πλαίσιο της παρούσας εργασίας, τα τετρακόπτερα παρουσιάζουν ιδιαίτερο ενδιαφέρον λόγω της ικανότητάς τους να αιωρούνται, να εκτελούν μικρές διορθωτικές κινήσεις και να προσεγγίζουν με ακρίβεια έναν στόχο προσγείωσης. Τα χαρακτηριστικά αυτά τα καθιστούν κατάλληλα για την ανάπτυξη συστημάτων αυτόνομης προσγείωσης σε κινητές πλατφόρμες, όπου απαιτείται συνδυασμός σταθερής πτήσης, εκτίμησης θέσης και συνεργασίας με συστήματα μηχανικής όρασης [1], [4], [7], [12], [15], [33], [57], [59].

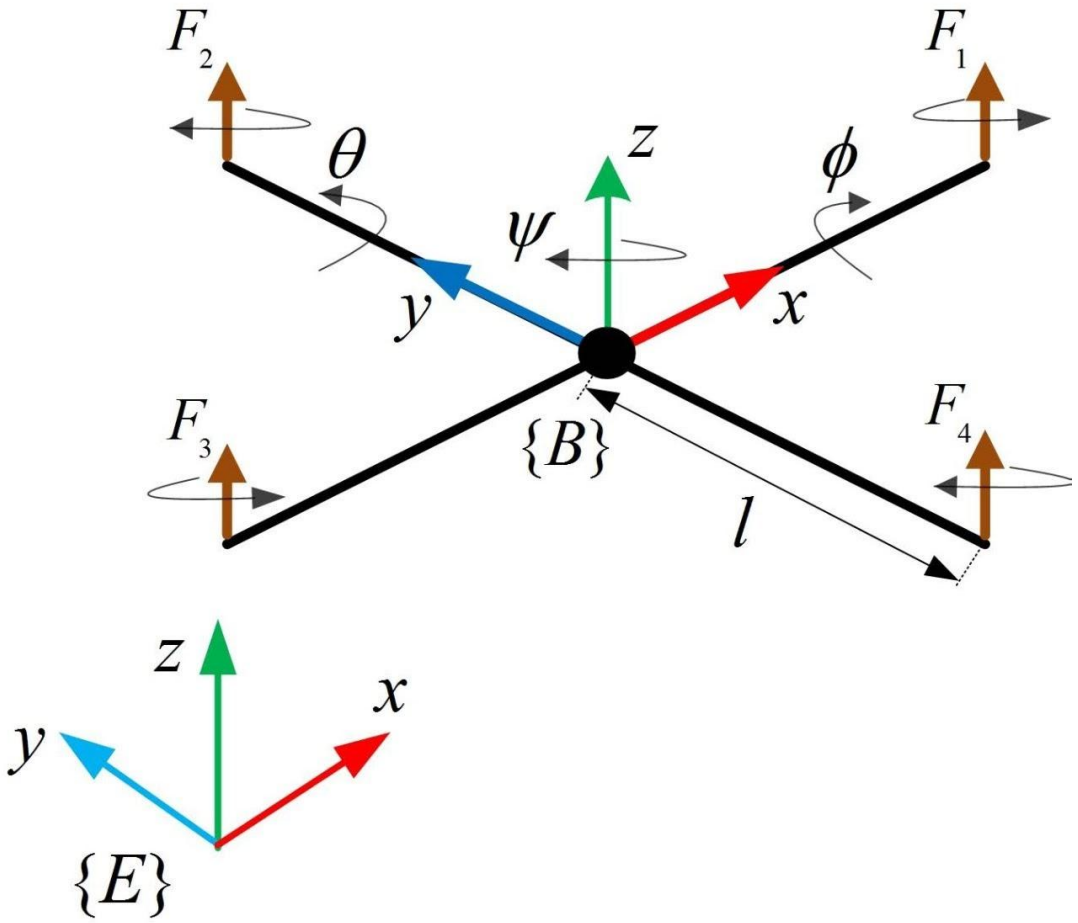
1.2 Δομή και Αρχή Λειτουργίας Τετρακοπτέρου

Η βασική δομή ενός τετρακοπτέρου στηρίζεται στη συμμετρική διάταξη τεσσάρων κινητήρων με έλικες, οι οποίοι τοποθετούνται στα άκρα ενός σταυροειδούς πλαισίου (Εικόνα 1.1). Η διάταξη αυτή επιτρέπει την παραγωγή της απαιτούμενης άνωσης, καθώς και τη δημιουργία των ροπών που απαιτούνται για τον έλεγχο της στάσης και της κίνησης του οχήματος. Στην παρούσα εργασία χρησιμοποιήθηκε γεωμετρία τύπου F450, η οποία προσφέρει επαρκή χώρο για την τοποθέτηση των ηλεκτρονικών εξαρτημάτων, συμμετρική κατανομή των κινητήρων και πρακτική ενσωμάτωση του συστήματος διανομής ισχύος [57], [68].

Για την εξισορρόπηση των ροπών αντίδρασης, οι τέσσερις κινητήρες δεν περιστρέφονται όλοι προς την ίδια κατεύθυνση. Δύο από αυτούς περιστρέφονται δεξιόστροφα (clockwise - CW), ενώ οι άλλοι δύο αριστερόστροφα (counter-clockwise - CCW). Οι κινητήρες που έχουν ίδια φορά περιστροφής τοποθετούνται διαμετρικά αντίθετα, ώστε οι ροπές που παράγονται από την περιστροφή των ελίκων να αλληλοαναιρούνται όταν το τετρακόπτερο βρίσκεται σε κατάσταση ισορροπίας. Με αυτόν τον τρόπο, όταν οι τέσσερις κινητήρες λειτουργούν με κατάλληλες και συμμετρικές ταχύτητες, το όχημα μπορεί να διατηρεί σταθερή αιώρηση (hovering), χωρίς ανεπιθύμητη περιστροφή γύρω από τον κατακόρυφο άξονά του.

Η κατακόρυφη κίνηση του τετρακοπτέρου επιτυγχάνεται μέσω της ταυτόχρονης μεταβολής της ταχύτητας περιστροφής όλων των κινητήρων. Όταν η συνολική παραγόμενη άνωση είναι μεγαλύτερη από το βάρος του οχήματος, το τετρακόπτερο ανέρχεται. Όταν η άνωση είναι μικρότερη από το βάρος, το όχημα κατέρχεται. Σε κατάσταση αιώρησης, η συνολική άνωση εξισορροπεί το βάρος, με αποτέλεσμα το τετρακόπτερο να διατηρεί περίπου σταθερό ύψος.

Ο έλεγχος του προσανατολισμού πραγματοποιείται μέσω διαφοροποιήσεων στις ταχύτητες περιστροφής των κινητήρων. Η περιστροφή γύρω από τον διαμήκη άξονα ονομάζεται roll και επιτυγχάνεται με μεταβολή της σχετικής ώσης μεταξύ των κινητήρων της αριστερής και της δεξιάς πλευράς. Η περιστροφή γύρω από τον εγκάρσιο άξονα ονομάζεται pitch και επιτυγχάνεται με διαφοροποίηση της ώσης μεταξύ των εμπρόσθιων και των οπίσθιων κινητήρων. Η περιστροφή γύρω από τον κατακόρυφο άξονα ονομάζεται yaw και προκαλείται από τη διαφορά ροπών μεταξύ των ζευγών κινητήρων αντίθετης φοράς περιστροφής. Οι τρεις αυτές κινήσεις περιγράφουν τη στάση του τετρακοπτέρου στον χώρο και αποτελούν βασικές μεταβλητές για τον έλεγχο της πτήσης.



Εικόνα 1.1 Γωνίες Euler και Δυνάμεις Ροτόρων [24]

Η λειτουργία του τετρακοπτέρου βασίζεται στη συνεχή συνεργασία της μονάδας ελέγχου πτήσης, των ηλεκτρονικών ρυθμιστών ταχύτητας και των αισθητήρων. Η μονάδα ελέγχου πτήσης (flight controller) λαμβάνει δεδομένα από αισθητήρες, όπως γυροσκόπια, επιταχυνσιόμετρα και βαρόμετρο, εκτιμά τη στάση και την κίνηση του οχήματος και αποστέλλει κατάλληλα σήματα στους ηλεκτρονικούς ρυθμιστές ταχύτητας (Electronic Speed Controllers - ESC) [13]. Οι ESC ρυθμίζουν την ταχύτητα περιστροφής των κινητήρων, επιτρέποντας στο τετρακόπτερο να σταθεροποιείται και να εκτελεί τις επιθυμητές κινήσεις.

Για την περιγραφή της κίνησης του τετρακοπτέρου χρησιμοποιούνται δύο βασικά συστήματα αναφοράς. Το πρώτο είναι το αδρανειακό σύστημα ή σύστημα γης, το οποίο θεωρείται σταθερό και χρησιμοποιείται για την περιγραφή της θέσης του οχήματος στον χώρο. Το δεύτερο είναι το σύστημα αναφοράς του σώματος, το οποίο είναι συνδεδεμένο με το τετρακόπτερο και μεταβάλλεται μαζί με αυτό. Η σχέση μεταξύ των δύο συστημάτων περιγράφεται μέσω των γωνιών Euler, δηλαδή των γωνιών roll, pitch και yaw, οι οποίες χρησιμοποιούνται για την αναπαράσταση του προσανατολισμού του οχήματος [2], [12]. Η αναλυτική μαθηματική περιγραφή αυτών των μεγεθών παρουσιάζεται στο επόμενο κεφάλαιο.

1.3 Βασικές Έννοιες και Κινηματικοί Περιορισμοί Ρομπότ Διαφορικής Οδήγησης

Τα ρομπότ διαφορικής οδήγησης αποτελούν μία από τις πιο διαδεδομένες κατηγορίες κινητών ρομποτικών συστημάτων, λόγω της απλής μηχανικής τους δομής και της σχετικά εύκολης υλοποίησης του ελέγχου κίνησης. Η λειτουργία τους βασίζεται στην ανεξάρτητη οδήγηση δύο κινητήριων τροχών, οι οποίοι είναι τοποθετημένοι στον ίδιο άξονα. Μεταβάλλοντας κατάλληλα τις ταχύτητες των δύο τροχών, το όχημα μπορεί να κινηθεί ευθύγραμμα, να εκτελέσει καμπύλη τροχιά ή να περιστραφεί γύρω από το κέντρο του.

Η ανάλυση της κίνησης ενός τέτοιου οχήματος βασίζεται κυρίως στην κινηματική (kinematics) [13], [14], [24], δηλαδή στη μελέτη της μεταβολής της θέσης και του προσανατολισμού του σώματος στον χρόνο, χωρίς να λαμβάνονται άμεσα υπόψη οι δυνάμεις που προκαλούν την κίνηση. Μέσω της κινηματικής ανάλυσης καθίσταται δυνατή η συσχέτιση των ταχυτήτων των δύο τροχών με τη συνολική μεταφορική και περιστροφική κίνηση του ρομπότ.

Βασικό χαρακτηριστικό των ρομπότ διαφορικής οδήγησης είναι ότι υπόκεινται σε μη ολόνομους περιορισμούς (non-holonomic constraints) [10], [13], [14]. Οι περιορισμοί αυτοί προκύπτουν από την παραδοχή ότι οι τροχοί κυλούν πάνω στην επιφάνεια χωρίς πλευρική ολίσθηση. Επομένως, το όχημα δεν μπορεί να μετακινηθεί στιγμιαία προς οποιαδήποτε κατεύθυνση στο επίπεδο, όπως θα μπορούσε ένα πανκατευθυντικό ρομπότ. Αντίθετα, η μετατόπισή του εξαρτάται από τον τρέχοντα προσανατολισμό του και από τη σχετική ταχύτητα των δύο τροχών.

Κατά την κίνηση του οχήματος, η σχέση μεταξύ των ταχυτήτων του αριστερού και του δεξιού τροχού καθορίζει την τροχιά που θα ακολουθήσει. Όταν οι δύο τροχοί περιστρέφονται με ίση ταχύτητα και ίδια φορά, το όχημα κινείται σε ευθεία γραμμή. Όταν οι δύο τροχοί περιστρέφονται με ίση ταχύτητα αλλά αντίθετη φορά, το όχημα περιστρέφεται επιτόπου. Στην περίπτωση όπου οι ταχύτητες των δύο τροχών είναι διαφορετικές, το όχημα ακολουθεί καμπύλη τροχιά.

Η καμπύλη αυτή κίνηση μπορεί να περιγραφεί μέσω της έννοιας του Στιγμιαίου Κέντρου Καμπυλότητας (Instantaneous Center of Curvature - ICC). Το σημείο αυτό αποτελεί το θεωρητικό κέντρο γύρω από το οποίο περιστρέφεται στιγμιαία το όχημα κατά την κίνησή του. Η θέση του ICC εξαρτάται από τη σχέση των ταχυτήτων των δύο τροχών. Όσο μεγαλύτερη είναι η διαφορά μεταξύ των ταχυτήτων τους, τόσο μικρότερη είναι η ακτίνα της καμπύλης τροχιάς που ακολουθεί το όχημα.

Στο πλαίσιο της παρούσας εργασίας, η διαφορική οδήγηση επιλέχθηκε για το επίγειο όχημα λόγω της απλότητας, της ευκολίας ελέγχου και της καταλληλότητάς της για την υποστήριξη της κινητής πλατφόρμας προσγείωσης. Η κατανόηση των κινηματικών περιορισμών του οχήματος είναι απαραίτητη, καθώς η κίνηση της επίγειας πλατφόρμας επηρεάζει άμεσα τη σχετική θέση μεταξύ του τετρακοπτέρου και του στόχου προσγείωσης.

1.4 Δομή και Αρχή Λειτουργίας Ρομπότ Διαφορικής Οδήγησης

Η μηχανική δομή ενός ρομπότ διαφορικής οδήγησης βασίζεται σε ένα κεντρικό πλαίσιο, πάνω στο οποίο τοποθετούνται τα βασικά μηχανικά και ηλεκτρονικά εξαρτήματα του συστήματος. Η κίνηση του οχήματος παράγεται από δύο ανεξάρτητα ελεγχόμενους

κινητήριους τροχούς, οι οποίοι βρίσκονται στον ίδιο άξονα και περιστρέφονται μέσω ξεχωριστών κινητήρων [10], [13], [24]. Η ανεξάρτητη οδήγηση των δύο τροχών επιτρέπει στο όχημα να μεταβάλλει τόσο τη γραμμική όσο και την περιστροφική του κίνηση, χωρίς την ανάγκη πρόσθετου μηχανισμού διεύθυνσης.

Για τη διατήρηση της σταθερότητας του πλαισίου χρησιμοποιείται συνήθως ένας ελεύθερος τροχός στήριξης, ο οποίος δεν συμμετέχει ενεργά στην παραγωγή της κίνησης, αλλά εξασφαλίζει την ισορροπία του οχήματος. Με αυτόν τον τρόπο, το όχημα μπορεί να κινηθεί με απλή μηχανική διάταξη, διατηρώντας παράλληλα επαρκή σταθερότητα για τη μεταφορά επιπλέον φορτίου. Στην παρούσα εργασία, το φορτίο αυτό είναι η πλατφόρμα προσγείωσης και ο μηχανισμός εξισορρόπησής της.

Η αρχή λειτουργίας του οχήματος στηρίζεται στη διαφοροποίηση των ταχυτήτων των δύο κινητήριων τροχών. Όταν οι δύο τροχοί περιστρέφονται με την ίδια ταχύτητα και προς την ίδια κατεύθυνση, το όχημα κινείται ευθύγραμμα. Όταν οι τροχοί περιστρέφονται με αντίθετες κατευθύνσεις, το όχημα μπορεί να περιστραφεί γύρω από το κέντρο του. Στην περίπτωση όπου οι δύο τροχοί έχουν διαφορετικές ταχύτητες, το όχημα ακολουθεί καμπύλη τροχιά. Με τον τρόπο αυτό, η συνολική κίνηση του ρομπότ καθορίζεται αποκλειστικά από τον κατάλληλο συνδυασμό των ταχυτήτων του αριστερού και του δεξιού τροχού.

Η επιλογή διαφορικής οδήγησης είναι ιδιαίτερα κατάλληλη για πειραματικές ρομποτικές εφαρμογές, καθώς προσφέρει απλότητα κατασκευής, μικρό αριθμό μηχανικών μερών και εύκολη ενσωμάτωση σε συστήματα μικροελεγκτών [13], [14]. Επιπλέον, η συγκεκριμένη διάταξη επιτρέπει ακριβή έλεγχο της κατεύθυνσης του οχήματος μέσω λογισμικού, γεγονός που την καθιστά κατάλληλη για την ανάπτυξη κινητής πλατφόρμας προσγείωσης.

Στο πλαίσιο της παρούσας εργασίας, το επίγειο όχημα λειτουργεί ως φορέας της πλατφόρμας προσγείωσης. Πάνω στο πλαίσιο τοποθετούνται ο μικροελεγκτής, ο οδηγός σερβοκινητήρων, ο αισθητήρας αδρανειακής μέτρησης, οι κινητήρες κίνησης και ο μηχανισμός εξισορρόπησης. Η πλατφόρμα προσγείωσης αποτελεί το ανώτερο τμήμα της κατασκευής και έχει ως στόχο να παρέχει στο τετρακόπτερο μία όσο το δυνατόν πιο σταθερή και οριζόντια επιφάνεια επαφής.

Η λειτουργία του οχήματος συνδέεται άμεσα με τον συνολικό στόχο της εργασίας, καθώς η κίνηση της επίγειας βάσης δημιουργεί ένα δυναμικό περιβάλλον προσγείωσης. Επομένως, το όχημα δεν αποτελεί απλώς μία μηχανική βάση, αλλά μέρος του συνολικού ρομποτικού συστήματος, το οποίο πρέπει να συνεργάζεται με το τετρακόπτερο και το σύστημα μηχανικής όρασης για την επίτευξη ασφαλούς και ελεγχόμενης προσέγγισης της πλατφόρμας.

2. ΜΑΘΗΜΑΤΙΚΗ ΑΝΑΛΥΣΗ ΚΑΙ ΜΟΝΤΕΛΟΠΟΙΗΣΗ

2.1 Μαθηματική Ανάλυση Τετρακοπτέρου

Η αναλυτική μαθηματική μοντελοποίηση ενός τετρακοπτέρου βασίζεται στις αρχές της μηχανικής στερεού σώματος και συγκεκριμένα στις εξισώσεις Newton-Euler, οι οποίες περιγράφουν τη συνδυασμένη μεταφορική και περιστροφική κίνηση του συστήματος [2] [5], [7], [12]. Η ανάπτυξη ενός τέτοιου μοντέλου είναι θεμελιώδους σημασίας, καθώς αποτελεί τη βάση για τον σχεδιασμό και την υλοποίηση προηγμένων αλγορίθμων ελέγχου, όπως ο γραμμικός τετραγωνικός ελεγκτής (LQR) και ο προγνωστικός έλεγχος μοντέλου (Model Predictive Control - MPC), οι οποίοι στοχεύουν στη σταθεροποίηση και τη βέλτιστη απόκριση του τετρακοπτέρου υπό διάφορες συνθήκες λειτουργίας.

2.1.1 Συστήματα Αναφοράς και Πίνακας Περιστροφής

Για την πλήρη περιγραφή της κίνησης ενός τετρακοπτέρου, ορίζονται δύο βασικά συστήματα συντεταγμένων. Το πρώτο είναι το αδρανειακό σύστημα αναφοράς $\{\mathrm{E}\}$, το οποίο θεωρείται σταθερό και χρησιμοποιείται για την περιγραφή της θέσης και της κίνησης στο χώρο. Το δεύτερο είναι το σύστημα αναφοράς του σώματος $\{\mathrm{B}\}$, το οποίο είναι προσδεδεμένο στο τετρακόπτερο και μεταβάλλεται μαζί με αυτό [2], [7], [12], [60], [61].

Ο προσανατολισμός του τετρακοπτέρου περιγράφεται μέσω των γωνιών Euler, οι οποίες ορίζονται ως

$$\Theta = [\varphi, \theta, \psi]^{\mathrm{T}}$$

Όπου φ είναι η γωνία κλίσης (roll), θ η γωνία πρόνευσης (pitch) και ψ η γωνία εκτροπής (yaw). Οι γωνίες αυτές επιτρέπουν την πλήρη αναπαράσταση της στάσης του σώματος ως προς το αδρανειακό σύστημα.

Η μετατροπή των μεγεθών από το σύστημα αναφοράς του σώματος στο αδρανειακό σύστημα πραγματοποιείται μέσω του πίνακα περιστροφής R_b. Ο πίνακας αυτός προκύπτει από τον διαδοχικό πολλαπλασιασμό των επιμέρους πινάκων περιστροφής γύρω από τους τρεις άξονες και δίνεται από την ακόλουθη μορφή:

$$R_b = \begin{bmatrix} cos\theta\ cos\psi & \sin\varphi\ \sin\theta\ \cos\psi - \cos\varphi\ \sin\psi & \cos\varphi\ \sin\theta\ \cos\psi + \sin\varphi\ \sin\psi \\ cos\theta\ sin\psi & \sin\varphi\ \sin\theta\ \sin\psi + \cos\varphi\ \cos\psi & \cos\varphi\ \sin\theta\ \sin\psi - \sin\varphi\ \cos\psi \\ -sin\theta & sin\varphi\ cos\theta & cos\varphi\ cos\theta \end{bmatrix}$$

2.1.2 Δυναμική της Μετατόπισης

Η γραμμική κίνηση του τετρακοπτέρου περιγράφεται μέσω της θέσης του στο χώρο

$$w = [x, y, z]^T$$

και της γραμμικής του ταχύτητας

$$u = [\dot{x}, \dot{y}, \dot{z}]^T$$

Η δυναμική της μετατόπισης διέπεται από τον δεύτερο νόμο του Νεύτωνα, σύμφωνα με τον οποίο το γινόμενο της μάζας m με την επιτάχυνση του σώματος ισούται με τη

συνισταμένη των δυνάμεων που ασκούνται σε αυτό. Στην περίπτωση του τετρακοπτέρου, η σχέση αυτή διατυπώνεται ως

$$m\dot{u} = R_b F_b - mge_3 + F_{drag}$$

Όπου $F_b = [0,0,\mathrm{T}]^{\mathrm{T}}$ εκφράζει τη συνολική δύναμη άνωσης που παράγεται από τους ρότορες στο σύστημα του σώματος, ενώ ο όρος F_{drag} αντιπροσωπεύει τις αεροδυναμικές δυνάμεις αντίστασης και τις εξωτερικές διαταραχές. Το διάνυσμα e_3 αντιστοιχεί στη μοναδιαία κατεύθυνση του κατακόρυφου άξονα στο αδρανειακό σύστημα, ενώ *g* είναι η επιτάχυνση της βαρύτητας.

Με τη χρήση του πίνακα περιστροφής R_b, η δύναμη άνωσης μετασχηματίζεται από το σύστημα του σώματος στο αδρανειακό σύστημα αναφοράς. Αναλύοντας την παραπάνω διανυσματική εξίσωση κατά μήκος των αξόνων x, y και z, προκύπτουν οι επιμέρους εξισώσεις που περιγράφουν τις γραμμικές επιταχύνσεις του τετρακόπτερου.

Συγκεκριμένα, η επιτάχυνση κατά τον άξονα x εξαρτάται από συνδυασμό των γωνιών προσανατολισμού και της συνολικής άνωσης, ενώ επηρεάζεται και από εξωτερικές διαταραχές. Αντίστοιχα, η επιτάχυνση κατά τον άξονα y προκύπτει από παρόμοιο συνδυασμό όρων, με διαφοροποίηση λόγω της γεωμετρίας της περιστροφής. Η επιτάχυνση κατά τον κατακόρυφο άξονα z καθορίζεται κυρίως από τη συνιστώσα της άνωσης στην κατακόρυφη διεύθυνση και την αντίθεση της βαρύτητας, καθώς και από τις επιδράσεις της αεροδυναμικής αντίστασης.

Οι εξισώσεις αυτές λαμβάνουν την ακόλουθη μορφή:

$$\dot{x} = \frac{T}{m}(\cos\varphi \sin\theta \cos\psi + \sin\varphi \sin\psi) + d_x$$

$$\dot{y} = \frac{T}{m}(\cos\varphi \sin\theta \sin\psi - \sin\varphi \cos\psi) + d_y$$

$$\dot{z} = \frac{T}{m}(\cos\varphi \cos\theta) - g + d_z$$

όπου οι όροι d_x, d_y και d_z συγκεντρώνουν τις επιδράσεις των εξωτερικών διαταραχών και της αεροδυναμικής αντίστασης. Οι παραπάνω εξισώσεις αποτελούν θεμελιώδες μέρος του δυναμικού μοντέλου του τετρακοπτέρου, καθώς συνδέουν άμεσα την παραγόμενη άνωση και τον προσανατολισμό του σώματος με τη μεταφορική του κίνηση στον χώρο.

2.1.3 Δυναμική της Περιστροφής

Η περιστροφική κίνηση του τετρακοπτέρου περιγράφεται μέσω των εξισώσεων Euler, οι οποίες εκφράζουν τη σχέση μεταξύ της γωνιακής επιτάχυνσης και των ροπών που ασκούνται στο σώμα. Η γενική μορφή της εξίσωσης δίνεται από

$$J\dot{\omega} = -\omega(J\omega) + M_c + M_r + M_d$$

όπου το διάνυσμα $\omega = [p,q,r]^T$ αναπαριστά τη γωνιακή ταχύτητα του σώματος ως προς τους τρεις άξονες, ενώ ο πίνακας J (ή I) αποτελεί τον πίνακα ροπής αδράνειας του τετρακοπτέρου και, λόγω συμμετρίας, θεωρείται διαγώνιος με στοιχεία I_{xx}, I_{yy} και I_{zz}. Ο όρος M_c αντιστοιχεί στις ροπές ελέγχου που παράγονται από τους κινητήρες, ο M_r στις γυροσκοπικές ροπές που οφείλονται στην περιστροφή των ελίκων και ο M_d σε εξωτερικές διαταραχές.

Η ανάπτυξη της παραπάνω διανυσματικής εξίσωσης στους τρεις άξονες περιστροφής οδηγεί σε ένα σύστημα μη γραμμικών διαφορικών εξισώσεων, οι οποίες περιγράφουν τη δυναμική του προσανατολισμού του τετρακοπτέρου. Συγκεκριμένα, η γωνιακή επιτάχυνση γύρω από τον άξονα x (roll) εξαρτάται από τη σύζευξη των γωνιακών ταχυτήτων στους άλλους δύο άξονες, καθώς και από τη ροπή ελέγχου και τα γυροσκοπικά φαινόμενα. Αντίστοιχα, για τον άξονα y (pitch) και τον άξονα z (yaw), οι εξισώσεις λαμβάνουν παρόμοια μορφή, με διαφοροποίηση στους όρους σύζευξης και στις εφαρμοζόμενες ροπές.

Οι αναλυτικές μορφές των εξισώσεων είναι οι εξής:

$$\dot{p} = \frac{1}{I_{xx}}\left[\left(I_{yy} - I_{zz}\right)qr + U_2 + J_r\Omega q\right]$$

$$\dot{q} = \frac{1}{I_{yy}}\left[\left(I_{zz} - I_{xx}\right)pr + U_3 + J_r\Omega p\right]$$

$$\dot{r} = \frac{1}{I_{zz}}\left[\left(I_{xx} - I_{yy}\right)pq + U_4\right]$$

όπου οι όροι $U_2, U_3\ και\ U_4$ αντιστοιχούν στις ροπές ελέγχου γύρω από τους άξονες roll, pitch και yaw αντίστοιχα, ενώ ο όρος $J_r\Omega$ εκφράζει τη γυροσκοπική επίδραση που προκαλείται από τη συνολική περιστροφή των ελίκων. Οι εξισώσεις αυτές καταδεικνύουν τον έντονα μη γραμμικό και συζευγμένο χαρακτήρα της περιστροφικής δυναμικής, γεγονός που καθιστά αναγκαία τη χρήση κατάλληλων τεχνικών ελέγχου για τη σταθεροποίηση και τον ακριβή προσανατολισμό του τετρακόπτερου.

2.1.4 Είσοδοι Ελέγχου και Δυνάμεις Κινητήρων

Οι είσοδοι ελέγχου του τετρακοπτέρου συνδέονται άμεσα με τις δυνάμεις που παράγονται από τους τέσσερις κινητήρες $F_1, F_2, F_3\ και\ F_4$ με τρόπο που καθορίζει την άνωση και τις ροπές γύρω από τους άξονες του σώματος. Η συνολική άνωση U_1 προκύπτει από το άθροισμα των δυνάμεων των τεσσάρων ροτόρων:

$$U_1 = F_1 + F_2 + F_3 + F_4$$

και ελέγχει την κατακόρυφη μετατόπιση του τετρακοπτέρου.

Η ροπή γύρω από τον άξονα roll U_2 παράγεται από τη διαφορά δυνάμεων μεταξύ των δύο αριστερών και δεξιών κινητήρων:

$$U_2 = L(F_2 - F_4)$$

όπου L είναι το μήκος του βραχίονα από το κέντρο μάζας έως τον κινητήρα. Η ροπή γύρω από τον άξονα pitch U_3 καθορίζεται από τη διαφορά δυνάμεων μεταξύ των μπροστινών και πίσω κινητήρων:

$$U_3 = L(F_3 - F_1)$$

Τέλος, η ροπή γύρω από τον άξονα yaw U_4 προκύπτει από τη διαφορά των ροπών που προκαλούνται από τις δεξιόστροφες και αριστερόστροφες έλικες και εκφράζεται ως

$$U_4 = c_d(-F_1 + F_2 - F_3 + F_4)$$

Όπου c_d είναι ο συντελεστής που μετατρέπει τη διαφορά δυνάμεων σε καθαρή ροπή yaw.

2.1.5 Γραμμικοποίηση και Μοντέλο Χώρου Καταστάσεων

Κατά την πτήση αιώρησης (hovering) και κατά τις ομαλές κινήσεις προσγείωσης, οι γωνίες κλίσης (ϕ) και πρόνευσης (θ) προσεγγίζουν το μηδέν. Χρησιμοποιώντας την προσέγγιση μικρών γωνιών $(\sin x \approx x \, και \, \cos x \approx 1)$, οι σύνθετες τριγωνομετρικές συναρτήσεις απλοποιούνται, οδηγώντας σε αποσύνδεση της κίνησης κατά τους κύριους άξονες [37], [38], [60]. Οι εξισώσεις επιτάχυνσης κατά τους άξονες x, y και z απλοποιούνται ως εξής, υπό την υπόθεση $\psi \approx 0$ και αιώρησης $T = mg$:

$$\ddot{x} = g\theta - \frac{k_g}{m}\dot{x}$$

$$\ddot{y} = -g\varphi - \frac{k_g}{m}\dot{y}$$

$$\ddot{z} = \frac{T}{m} - g$$

Αυτή η γραμμικοποίηση επιτρέπει τη διαμόρφωση του πίνακα καταστάσεων A και του πίνακα εισόδων B για το διάνυσμα καταστάσεων

$$X(t) = \left[x, y, z, \varphi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\varphi}, \dot{\theta}, \dot{\psi}\right]^T$$

ορίζοντας το κλασικό γραμμικό σύστημα:

$$\dot{X}(t) = AX(t) + BU(t)$$

ή ισοδύναμα

$$\dot{X} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & g & 0 & -\frac{k_g}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -g & 0 & 0 & 0 & -\frac{k_g}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \varphi \\ \theta \\ \psi \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

Επιπλέον, χρησιμοποιείται εκτενώς σε αλγορίθμους ελέγχου αποσύζευξης εισόδων–εξόδων και μη αλληλοεπιδρώντος ελέγχου [45], [47], [51], [60], [66], [67], καθώς και σε ιεραρχικές και πολυμεταβλητές αρχιτεκτονικές ελέγχου πτήσης [53], [61]. Επίσης, σημαντικό πεδίο εφαρμογής αποτελούν οι ελεγκτές τριών όρων (PID) και οι εύρωστες παραλλαγές τους [38], [44], [64], ενώ τεχνικές Εύρωστου Ελέγχου έχουν αξιοποιηθεί τόσο για εύρωστη ακολούθηση εντολής και ελέγχου τροχιάς [37], [38], [43], [49], όσο και για εύρωστη αποκοπή διαταραχών και ριπών ανέμου [42], [62], [46]. Πολλές εφαρμογές

στον έλεγχο πτήσης αεροσκαφών, ελικοπτέρων, πυραύλων, τετρακοπτέρων και συστημάτων με αστοχίες ενεργοποιητών χρησιμοποιούν την αναπαράσταση στο χώρο κατάστασης, για παράδειγμα [39], [41], [43], [48], [60], [61], [66].

2.2 Μαθηματική Ανάλυση Οχήματος Διαφορικής Οδήγησης

Η μαθηματική ανάλυση ενός οχήματος διαφορικής οδήγησης βασίζεται κυρίως στην κινηματική περιγραφή της κίνησής του στο επίπεδο, καθώς στις περισσότερες εφαρμογές κινητής ρομποτικής το βασικό ζητούμενο είναι η συσχέτιση των ταχυτήτων των κινητήριων τροχών με τη θέση και τον προσανατολισμό του οχήματος [10], [13], [14], [24], [50], [67]. Σε αντίθεση με πιο σύνθετα συστήματα, όπως τα εναέρια οχήματα, στα ρομπότ διαφορικής οδήγησης η δυναμική ανάλυση συχνά παραλείπεται σε πρώτο στάδιο και δίνεται έμφαση στην κινηματική μοντελοποίηση, η οποία επαρκεί για τον σχεδιασμό αλγορίθμων πλοήγησης, οδομετρίας και ελέγχου κίνησης. Η ανάπτυξη ενός τέτοιου μοντέλου είναι ιδιαίτερα σημαντική, καθώς αποτελεί τη θεωρητική βάση για την υλοποίηση ελεγκτών παρακολούθησης τροχιάς, σταθεροποίησης προσανατολισμού και αυτόνομης πλοήγησης.

2.2.1 Συστήματα Αναφοράς και Περιγραφή Προσανατολισμού

Για την πλήρη περιγραφή της κίνησης ενός οχήματος διαφορικής οδήγησης ορίζονται δύο βασικά συστήματα αναφοράς. Το πρώτο είναι το αδρανειακό ή παγκόσμιο σύστημα αναφοράς $\{I\}$, το οποίο θεωρείται σταθερό και χρησιμοποιείται για τον προσδιορισμό της θέσης του οχήματος στο επίπεδο. Το δεύτερο είναι το σύστημα αναφοράς του σώματος $\{B\}$, το οποίο είναι στερεωμένο επάνω στο όχημα και μεταβάλλεται μαζί με αυτό κατά την κίνησή του. Η θέση και ο προσανατολισμός του ρομπότ στο επίπεδο περιγράφονται από το διάνυσμα κατάστασης

$$\xi_{\mathrm{UGV}}(t) = [x_{\mathrm{UGV}}(t), y_{\mathrm{UGV}}(t), \psi_{\mathrm{UGV}}(t)]^T$$

όπου $x_{\mathrm{UGV}}(t)$, $y_{\mathrm{UGV}}(t)$ είναι οι καρτεσιανές συντεταγμένες του κέντρου του οχήματος στο παγκόσμιο σύστημα και $\psi_{\mathrm{UGV}}(t)$ είναι η γωνία προσανατολισμού του ως προς τον άξονα z του αδρανειακού συστήματος. Η γωνία αυτή εκφράζει τη στροφή του τοπικού άξονα του οχήματος ως προς το σταθερό σύστημα και αποτελεί τη βασική μεταβλητή που συνδέει την κίνηση στο σύστημα του σώματος με την κίνηση στο παγκόσμιο επίπεδο [13], [14], [24].

Η μετάβαση από το σύστημα του σώματος στο αδρανειακό σύστημα πραγματοποιείται μέσω ενός επίπεδου πίνακα περιστροφής, ο οποίος για γωνία προσανατολισμού φ_{UGV} γράφεται ως

$$R\big(\psi_{\mathrm{UGV}}(t)\big) = \begin{bmatrix} \cos\psi_{\mathrm{UGV}}(t) & -\sin\psi_{\mathrm{UGV}}(t) \\ \sin\psi_{\mathrm{UGV}}(t) & \cos\psi_{\mathrm{UGV}}(t) \end{bmatrix}$$

Ο πίνακας αυτός χρησιμοποιείται για τον μετασχηματισμό διανυσμάτων ταχύτητας από το τοπικό σύστημα του οχήματος στο παγκόσμιο σύστημα αναφοράς. Επειδή το όχημα διαφορικής οδήγησης κινείται σε επίπεδο, η περιγραφή του προσανατολισμού απαιτεί μόνο μία γωνία περιστροφής, σε αντίθεση με τα τρισδιάστατα συστήματα όπου απαιτούνται περισσότερες γωνίες ή πίνακες πλήρους χωρικής περιστροφής.

2.2.2 Κινηματική της Μετατόπισης

Η μεταφορική κίνηση του οχήματος διαφορικής οδήγησης περιγράφεται μέσω της γραμμικής ταχύτητας $\upsilon_{\mathrm{UGV}}(t)$, η οποία αντιστοιχεί στην ταχύτητα του κέντρου του οχήματος κατά τη διαμήκη διεύθυνση του σώματός του. Υποθέτοντας ότι το όχημα κινείται χωρίς ολίσθηση, οι καρτεσιανές συνιστώσες της ταχύτητας στο αδρανειακό σύστημα δίνονται από τις σχέσεις

$$\frac{dx_{\mathrm{UGV}}(t)}{dt} = \upsilon_{\mathrm{UGV}}(t)\cos\psi_{\mathrm{UGV}}(t)$$

$$\frac{dy_{\mathrm{UGV}}(t)}{dt} = \upsilon_{\mathrm{UGV}}(t)\sin\psi_{\mathrm{UGV}}(t)$$

Οι εξισώσεις αυτές δείχνουν ότι η ταχύτητα του οχήματος στο παγκόσμιο σύστημα εξαρτάται τόσο από το μέτρο της γραμμικής ταχύτητας όσο και από τον τρέχοντα προσανατολισμό του. Συνεπώς, ακόμη και αν το όχημα κινείται με σταθερή γραμμική ταχύτητα, η τροχιά του στο επίπεδο μπορεί να είναι ευθύγραμμη ή καμπύλη, ανάλογα με τη μεταβολή της γωνίας ψ_{UGV} στον χρόνο. Το γενικό κινηματικό μοντέλο του οχήματος γράφεται στη μορφή

$$\frac{d}{dt}\begin{bmatrix} x_{\mathrm{UGV}}(t) \\ y_{\mathrm{UGV}}(t) \\ \psi_{\mathrm{UGV}}(t) \end{bmatrix} = \begin{bmatrix} \cos\psi_{\mathrm{UGV}}(t) & 0 \\ \sin\psi_{\mathrm{UGV}}(t) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \upsilon_{\mathrm{UGV}}(t) \\ \omega_{\mathrm{UGV}}(t) \end{bmatrix}$$

Ισοδύναμο με

$$\frac{d\xi_{\mathrm{UGV}}(t)}{dt} = S(\xi_{\mathrm{UGV}})\boldsymbol{\nu}_{\mathrm{UGV}}(t)$$

Όπου $\boldsymbol{\nu}_{\mathrm{UGV}}(t) = [\upsilon_{\mathrm{UGV}}(t) \quad \omega_{\mathrm{UGV}}(t)]^T$ είναι το διάνυσμα εισόδων ταχύτητας και $S(\xi_{\mathrm{UGV}})$ ο κινηματικός πίνακας μετασχηματισμού.

2.2.3 Κινηματικός Περιορισμός και Περιστροφική Κίνηση

Ένα από τα βασικά χαρακτηριστικά των οχημάτων διαφορικής οδήγησης είναι ότι υπόκεινται σε μη-ολόνομους περιορισμούς [10], [24]. Ο περιορισμός αυτός προκύπτει από την υπόθεση της ιδανικής κύλισης των τροχών, σύμφωνα με την οποία οι τροχοί κυλούν χωρίς πλευρική ολίσθηση. Φυσικά, αυτό σημαίνει ότι το όχημα δεν μπορεί να αποκτήσει στιγμιαία ταχύτητα κάθετη στον κύριο άξονά του. Ο κινηματικός αυτός περιορισμός εκφράζεται από τη σχέση

$$\frac{dx_{\mathrm{UGV}}(t)}{dt}\sin\psi_{\mathrm{UGV}}(t) - \frac{dy_{\mathrm{UGV}}(t)}{dt}\cos\psi_{\mathrm{UGV}}(t) = 0$$

ή ισοδύναμα από το γεγονός ότι η πλευρική ταχύτητα στο σύστημα του σώματος είναι μηδενική. Η ιδιότητα αυτή διαφοροποιεί τα οχήματα διαφορικής οδήγησης από άλλα οχήματα με πανκατευθυντική κίνηση, καθώς επιβάλλει ότι κάθε μεταβολή της θέσης στο επίπεδο πρέπει να συνοδεύεται από κατάλληλη μεταβολή του προσανατολισμού.

Η περιστροφική κίνηση του οχήματος περιγράφεται από τη γωνιακή ταχύτητα $\omega_{\mathrm{UGV}} = \frac{d\psi_{\mathrm{UGV}}(t)}{dt}$, η οποία εκφράζει τον ρυθμό μεταβολής του προσανατολισμού του. Η γωνιακή αυτή ταχύτητα δεν προκύπτει από ανεξάρτητο μηχανισμό περιστροφής, αλλά από τη διαφορά ταχυτήτων μεταξύ των δύο κινητήριων τροχών. Όταν οι ταχύτητες των

δύο τροχών είναι ίσες, η γωνιακή ταχύτητα μηδενίζεται και το όχημα κινείται σε ευθεία. Όταν οι ταχύτητες διαφέρουν, εμφανίζεται μη μηδενική γωνιακή ταχύτητα και το όχημα εκτελεί καμπύλη κίνηση ή επιτόπια περιστροφή.

2.2.4 Είσοδοι Ελέγχου και Συσχέτιση με τις Ταχύτητες των Τροχών

Στο όχημα διαφορικής οδήγησης οι είσοδοι ελέγχου συνδέονται άμεσα με τις γωνιακές ταχύτητες των δύο κινητήριων τροχών. Αν ω_{UGV_r} και ω_{UGV_l} είναι οι γωνιακές ταχύτητες του δεξιού και του αριστερού τροχού αντίστοιχα, r_{UGV_w} η ακτίνα των τροχών και d_{UGV_w} η απόσταση μεταξύ των δύο κινητήριων τροχών, τότε η γραμμική και η γωνιακή ταχύτητα του οχήματος δίνονται από τις σχέσεις

$$\upsilon_{\mathrm{UGV}}(t) = \frac{r_{\mathrm{UGV}_w}\left[\omega_{\mathrm{UGV}_r}(t) + \omega_{\mathrm{UGV}_l}(t)\right]}{2}$$

$$\omega_{\mathrm{UGV}}(t) = \frac{r_{\mathrm{UG}}\mathrm{V}_w\left[\omega_{\mathrm{UGV}_r}(t) - \omega_{\mathrm{UGV}_l}(t)\right]}{d_{\mathrm{UG}}\mathrm{V}_w}$$

Οι σχέσεις αυτές είναι θεμελιώδεις, διότι συνδέουν τις εντολές που εφαρμόζονται στους δύο κινητήρες με τη συνολική κινηματική συμπεριφορά του ρομπότ. Η γραμμική ταχύτητα εξαρτάται από το άθροισμα των γωνιακών ταχυτήτων των τροχών, ενώ η γωνιακή ταχύτητα εξαρτάται από τη διαφορά τους. Αυτό σημαίνει ότι η μεταφορική και η περιστροφική κίνηση του οχήματος μπορούν να ελεγχθούν μέσω κατάλληλου συνδυασμού των δύο κινητήριων τροχών [10], [24].

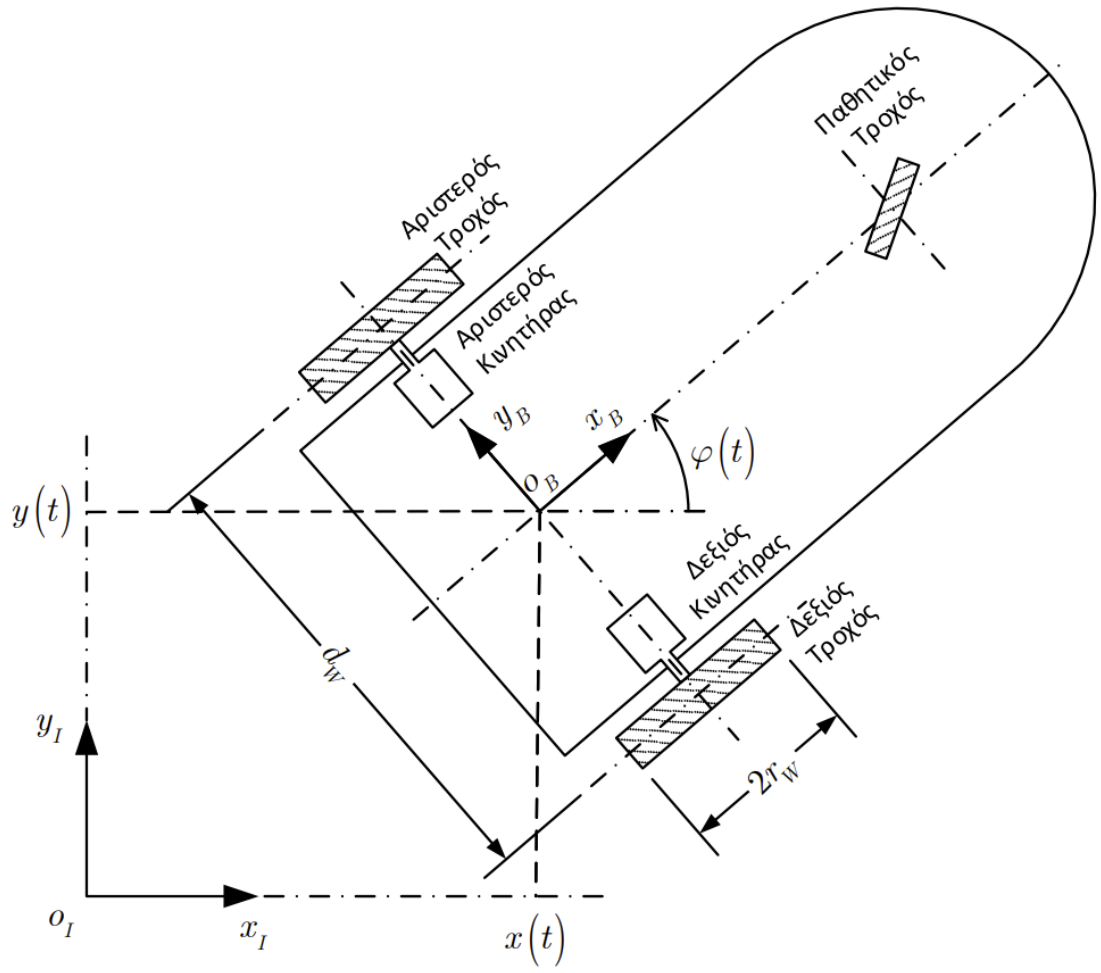
Αντικαθιστώντας τις παραπάνω σχέσεις στο γενικό κινηματικό μοντέλο, προκύπτει το ειδικό κινηματικό μοντέλο του οχήματος διαφορικής οδήγησης:

$$\frac{dx_{\mathrm{UGV}}(t)}{dt} = \frac{r_{\mathrm{UGV}_w}}{2}\cos\psi_{\mathrm{UGV}}(t)\left(\omega_{\mathrm{UGV}_r}(t) + \omega_{\mathrm{UGV}_l}(t)\right)$$

$$\frac{dy_{\mathrm{UGV}}(t)}{dt} = \frac{r_{\mathrm{UG}}\mathrm{V}_w}{2}\sin\psi_{\mathrm{UGV}}\,(t)\left(\omega_{\mathrm{UGV}_r}(t) + \omega_{\mathrm{UGV}_l}(t)\right)$$

$$\frac{d\psi_{\mathrm{UGV}}(t)}{dt} = \frac{r_{\mathrm{UGV}_w}}{d_{\mathrm{UGV}_w}}\left(\omega_{\mathrm{UGV}_r}(t) - \omega_{\mathrm{UGV}_l}(t)\right)$$

Το μοντέλο αυτό αποτελεί την κύρια αναλυτική σχέση μεταξύ των εισόδων ελέγχου και της κίνησης του οχήματος στο επίπεδο. Από αυτό προκύπτουν και οι βασικές μορφές κίνησης. Όταν ισχύει $\omega_{\mathrm{UGV}_r}(t) = \omega_{\mathrm{UGV}_l}(t)$, το όχημα κινείται ευθύγραμμα. Όταν ισχύει $\omega_{\mathrm{UGV}_r}(t) = -\omega_{\mathrm{UGV}_l}(t)$, η γραμμική ταχύτητα μηδενίζεται και το όχημα περιστρέφεται επιτόπου γύρω από το κέντρο του. Στη γενική περίπτωση όπου $\omega_{\mathrm{UGV}_r}(t) \neq \omega_{\mathrm{UGV}_l}(t)$, το όχημα διαγράφει καμπύλη τροχιά, περιστρεφόμενο στιγμιαία γύρω από ένα θεωρητικό σημείο που ονομάζεται Στιγμιαίο Κέντρο Καμπυλότητας.



Εικόνα 2.1 Ορισμός μεταβλητών και παραμέτρων ρομποτικού οχήματος διαφορικής οδήγησης [24]

2.2.5 Αντίστροφο Κινηματικό Πρόβλημα

Ιδιαίτερη σημασία στη μελέτη των οχημάτων διαφορικής οδήγησης έχει και το αντίστροφο κινηματικό πρόβλημα, δηλαδή ο προσδιορισμός των απαιτούμενων γωνιακών ταχυτήτων των τροχών όταν είναι γνωστές οι επιθυμητές τιμές της γραμμικής και της γωνιακής ταχύτητας του οχήματος. Η επίλυση του προβλήματος αυτού δίνει τις σχέσεις

$$\omega_{\mathrm{UGV}_r}(t) = \frac{2v_{\mathrm{UGV}}(t) + d_{\mathrm{UGV}_w}\omega_{\mathrm{UGV}}(t)}{2r_{\mathrm{UGV}_w}}$$

$$\omega_{\mathrm{UGV}_i}(t) = \frac{2v_{\mathrm{UGV}}(t) - d_{\mathrm{UGV}_w}\omega_{\mathrm{UGV}}(t)}{2r_{\mathrm{UGV}_w}}$$

Οι εξισώσεις αυτές χρησιμοποιούνται άμεσα σε αλγορίθμους ελέγχου κίνησης, καθώς επιτρέπουν τη μετατροπή των επιθυμητών εντολών πλοήγησης σε πραγματικές εντολές προς τους κινητήρες. Με τον τρόπο αυτό, ένας ελεγκτής τροχιάς μπορεί να υπολογίζει πρώτα τις επιθυμητές τιμές $v_{\mathrm{UGV}}(t)$ και $\omega_{\mathrm{UGV}}(t)$ και έπειτα, μέσω του αντίστροφου κινηματικού μοντέλου, να παράγει τις γωνιακές ταχύτητες των δύο τροχών που απαιτούνται για την υλοποίηση της επιθυμητής κίνησης.

2.2.6 Δυναμικές Εξισώσεις Πλατφόρμας

Η πλατφόρμα προσγείωσης που είναι τοποθετημένη επάνω στο όχημα εδάφους λειτουργεί ως ενεργά ελεγχόμενο μηχανικό σύστημα. Σκοπός της είναι να αντισταθμίζει την κλίση του οχήματος, ώστε η επιφάνεια προσγείωσης να παραμένει όσο το δυνατόν πιο κοντά στην οριζόντια θέση. Η ανάλυση βασίζεται στη δυναμική περιγραφή του μηχανισμού εξισορρόπησης ως σώματος με αδράνεια, απόσβεση, ροπή ενεργοποίησης και εξωτερικές διαταραχές.

Η κλίση του οχήματος ως προς τον ελεγχόμενο άξονα συμβολίζεται με $\theta_{UGV}\ (t)$, ενώ η αντίστοιχη γωνιακή ταχύτητα συμβολίζεται με $\widetilde{\omega}_{UGV}(t)$. Επομένως:

$$\widetilde{\omega}_{UGV}(t) = \dot{\theta}_{UGV}(t)$$

Η διορθωτική γωνία που εφαρμόζεται από τους σερβοκινητήρες της πλατφόρμας συμβολίζεται ως:

$$\theta_{UGV,c}(t)$$

Η τελική γωνία της επιφάνειας προσγείωσης ως προς το οριζόντιο επίπεδο προκύπτει από το άθροισμα της κλίσης του οχήματος και της διορθωτικής γωνίας της πλατφόρμας:

$$\theta_{pad}(t) = \theta_{UGV}(t) + \theta_{UGV,c}(t)$$

Για ιδανική εξισορρόπηση, η πλατφόρμα πρέπει να παραμένει οριζόντια, άρα:

$$\theta_{pad}(t) = 0$$

Επομένως, η επιθυμητή διορθωτική γωνία είναι:

$$\theta_{UGV,c,d}(t) = -\theta_{UGV}(t)$$

Η παραπάνω σχέση εκφράζει τον βασικό στόχο του ελέγχου: η πλατφόρμα πρέπει να κινείται αντίθετα από την κλίση του οχήματος, ώστε η τελική γωνία της επιφάνειας προσγείωσης να μηδενίζεται.

Η δυναμική της πλατφόρμας ως προς τον ελεγχόμενο άξονα μπορεί να περιγραφεί από εξίσωση δεύτερης τάξης:

$$J_{UGV,\theta}\ddot{\theta}_{UGV,c}(t) + B_{UGV,\theta}\dot{\theta}_{UGV,c}(t) = \tau_{UGV,\theta}(t) + d_{UGV,\theta}(t)$$

Όπου $J_{UGV,\theta}$ είναι η ισοδύναμη ροπή αδράνειας της πλατφόρμας ως προς τον άξονα κίνησης, $B_{UGV,\theta}$ είναι ο ισοδύναμος συντελεστής απόσβεσης, $\tau_{UGV,\theta}(t)$ είναι η ροπή που παράγεται από τους σερβοκινητήρες και $d_{UGV,\theta}(t)$ είναι ο όρος διαταραχών, ο οποίος περιλαμβάνει τις επιδράσεις από την κίνηση του οχήματος, τις ανωμαλίες του εδάφους, τις ταλαντώσεις και τις μηχανικές ανοχές της κατασκευής.

Επειδή η πλατφόρμα είναι τοποθετημένη επάνω στο κινούμενο όχημα, η γωνιακή επιτάχυνση του οχήματος επηρεάζει τη δυναμική του μηχανισμού. Για τον λόγο αυτό, η εξίσωση μπορεί να γραφεί πιο αναλυτικά ως:

$$J_{UGV,\theta}\ddot{\theta}_{UGV,c}(t) + B_{UGV,\theta}\dot{\theta}_{UGV,c}(t) = \tau_{UGV,\theta}(t) + d_{UGV,\theta}(t) - J_{UGV,\theta}\ddot{\theta}_{UGV}(t)$$

Ο όρος $-J_{UGV,\theta}\ddot{\theta}_{UGV}(t)$ εκφράζει την αδρανειακή επίδραση της μεταβολής της κλίσης του οχήματος επάνω στην πλατφόρμα. Δηλαδή, όσο πιο γρήγορα αλλάζει η γωνία του οχήματος, τόσο μεγαλύτερη είναι η δυναμική διαταραχή που πρέπει να αντισταθμίσει ο μηχανισμός.

Το σφάλμα εξισορρόπησης ορίζεται ως η απόκλιση της πλατφόρμας από την οριζόντια θέση:

$$e_{UGV}(t) = \theta_{pad}(t) - \theta_{ref}$$

Για οριζόντια πλατφόρμα ισχύει $\theta_{ref} = 0$ οπότε:

$$e_{UGV}(t) = \theta_{UGV}(t) + \theta_{UGV,c}(t)$$

Σκοπός του ελεγκτή είναι:

$$e_{UGV}(t) \to 0$$

Μία απλή μορφή εντολής διόρθωσης μπορεί να γραφεί ως:

$$\theta_{UGV,c,d}(t) = sat[-K_{UGV}\theta_{UGV}(t)]$$

όπου K_{UGV} είναι ο συντελεστής ενίσχυσης του ελεγκτή και $sat[\,]$ είναι η συνάρτηση κορεσμού, η οποία περιορίζει τη ζητούμενη γωνία στο επιτρεπτό εύρος κίνησης της πλατφόρμας.

Για τους δύο κατοπτρικά κινούμενους σερβοκινητήρες, οι εντολές μπορούν να εκφραστούν ως:

$$\theta_{s1}(t) = \theta_{s,0} + \theta_{UGV,c,d}(t)$$

$$\theta_{s2}(t) = \theta_{s,0} - \theta_{UGV,c,d}(t)$$

όπου $\theta_{s,0}$ είναι η ουδέτερη θέση των σερβοκινητήρων. Με αυτόν τον τρόπο, οι δύο σερβοκινητήρες κινούνται συμμετρικά και παράγουν τη διορθωτική περιστροφή της πλατφόρμας.

Συνοπτικά, η λειτουργία της αυτόματης εξισορρόπησης περιγράφεται από το ακόλουθο σύνολο σχέσεων:

$$\tilde{\omega}_{UGV}(t) = \dot{\theta}_{UGV}(t)$$

$$\theta_{pad}(t) = \theta_{UGV}(t) + \theta_{UGV,c}(t)$$

$$\theta_{UGV,c,d}(t) = -\theta_{UGV}(t)$$

$$J_{UGV,\theta}\ddot{\theta}_{UGV,c}(t) + B_{UGV,\theta}\dot{\theta}_{UGV,c}(t) = \tau_{UGV,\theta}(t) + d_{UGV,\theta}(t) - J_{UGV,\theta}\ddot{\theta}_{UGV}(t)$$

$$e_{UGV}(t) = \theta_{UGV}(t) + \theta_{UGV,c}(t)$$

$$\theta_{UGV,c,d}(t) = sat[-K_{UGV}\theta_{UGV}(t)]$$

Οι παραπάνω εξισώσεις συνδέουν την εκτιμώμενη γωνία του οχήματος με τη διορθωτική κίνηση της πλατφόρμας και δείχνουν ότι η αυτόματη εξισορρόπηση αποτελεί πρόβλημα δυναμικού ελέγχου. Η μέτρηση της γωνίας $\theta_{UGV}(t)$ και της γωνιακής ταχύτητας $\tilde{\omega}_{UGV}(t)$ χρησιμοποιείται για την παραγωγή της επιθυμητής διορθωτικής γωνίας $\theta_{UGV,c,d}(t)$, ενώ η πραγματική απόκριση της πλατφόρμας εξαρτάται από την αδράνεια, την απόσβεση, τη ροπή των σερβοκινητήρων και τις εξωτερικές διαταραχές.

2.2.7 Γραμμικοποίηση και Μοντέλο Χώρου Καταστάσεων

Για τον σχεδιασμό γραμμικών ελεγκτών, όπως ο γραμμικός τετραγωνικός ρυθμιστής (LQR), το μη γραμμικό κινηματικό μοντέλο του οχήματος διαφορικής οδήγησης μπορεί να γραμμικοποιηθεί γύρω από ένα σημείο λειτουργίας [14], [24]. Συνήθως, ως σημείο

λειτουργίας λαμβάνεται η κίνηση γύρω από έναν επιθυμητό προσανατολισμό ψ_{UGV0} και μία ονομαστική γραμμική ταχύτητα u_{UGV0}. Το διάνυσμα κατάστασης ορίζεται ως

$$X_{\mathrm{UGV}}(t) = \begin{bmatrix} x_{\mathrm{UGV}}(t) \\ y_{\mathrm{UGV}}(t) \\ \psi_{\mathrm{UGV}}(t) \end{bmatrix}$$

ενώ το διάνυσμα εισόδων ορίζεται ως

$$U_{UGV}(t) = \begin{bmatrix} u_{UGV}(t) \\ \widetilde{\omega}_{UGV}(t) \end{bmatrix}$$

οπότε το μη γραμμικό κινηματικό μοντέλο γράφεται στη μορφή

$$\dot{\mathrm{X}}_{UGV}(t) = f(X_{UGV}, U_{UGV})$$

με

$$\dot{\mathrm{x}}_{UGV}(t) = u_{UGV}(t)\cos\psi_{UGV}(t)$$
$$\dot{\mathrm{y}}_{UGV}(t) = u_{UGV}(t)\sin\psi_{UGV}(t)$$
$$\dot{\psi}_{UGV}(t) = \omega_{UGV}(t)$$

Για μικρές αποκλίσεις γύρω από το σημείο λειτουργίας, ορίζονται οι μεταβλητές διαταραχής

$$\delta X_{\mathrm{UGV}}(t) = X_{\mathrm{UGV}}(t) - X_{\mathrm{UGV0}}$$
$$\delta U_{\mathrm{UGV}}(t) = X_{\mathrm{UGV}}(t) - U_{\mathrm{UGV0}}$$

και το γραμμικοποιημένο μοντέλο πρώτης τάξης λαμβάνει τη μορφή

$$\delta\dot{\mathrm{X}}_{UGV}(t) = A_{\mathrm{UAV}}\delta X_{\mathrm{UAV}(t)} + B_{\mathrm{UGV}}\,\delta U_{\mathrm{UGV}}(t)$$

όπου οι πίνακες A_{UGV} και B_{UGV} προκύπτουν από τις Ιακωμβιανές μήτρες του πεδίου $f(X_{\mathrm{UGV}}, U_{UGV})$, δηλαδή

$$A_{\mathrm{UGV}} = \left.\frac{\partial f}{\partial X_{\mathrm{UGV}}}\right|_{(X_{\mathrm{UGV0}}.U_{\mathrm{UGV0}})}$$

$$B_{\mathrm{UGV}} = \left.\frac{\partial f}{\partial U_{\mathrm{UGV}}}\right|_{(X_{\mathrm{UGV0}}.U_{\mathrm{UGV0}})}$$

Για το παραπάνω κινηματικό μοντέλο, οι πίνακες αυτοί δίνονται από τις σχέσεις

$$A_{\mathrm{UGV}} = \begin{bmatrix} 0 & 0 & -u_{\mathrm{UGV0}}\sin\psi_{\mathrm{UGV0}} \\ 0 & 0 & u_{\mathrm{UGV0}}\,cos\,\psi_{\mathrm{UGV0}} \\ 0 & 0 & 0 \end{bmatrix}$$

$$B_{\mathrm{UGV}} = \begin{bmatrix} cos\,\psi_{\mathrm{UGV0}} & 0 \\ \sin\psi_{\mathrm{UGV0}_0} & 0 \\ 0 & 1 \end{bmatrix}$$

Επομένως, το γραμμικοποιημένο σύστημα χώρου καταστάσεων γράφεται ως

$$\delta\dot{\mathrm{X}}_{UGV}(t) = \begin{bmatrix} 0 & 0 & -u_{\mathrm{UGV0}}\sin\psi_{\mathrm{UGV0}} \\ 0 & 0 & u_{\mathrm{UGV0}}\,cos\,\psi_{\mathrm{UGV0}} \\ 0 & 0 & 0 \end{bmatrix}\delta X_{\mathrm{UGV}}(t) + \begin{bmatrix} cos\,\psi_{\mathrm{UGV0}} & 0 \\ \sin\psi_{\mathrm{UGV0}} & 0 \\ 0 & 1 \end{bmatrix}\delta U_{\mathrm{UGV}}(t)$$

Ιδιαίτερο ενδιαφέρον παρουσιάζει η περίπτωση ευθύγραμμης κίνησης, όπου $\psi_{\mathrm{UGV0}} = 0$. Τότε ισχύουν $\cos\psi_{\mathrm{UGV0}} = 1$ και $\sin\psi_{\mathrm{UGV0}} = 0$, οπότε οι πίνακες απλοποιούνται στις μορφές

$$A_{\mathrm{UGV}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & u_{\mathrm{UGV0}} \\ 0 & 0 & 0 \end{bmatrix}$$

$$B_{\mathrm{UGV}} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

και το γραμμικοποιημένο μοντέλο γίνεται

$$\delta\dot{x}_{UGV}(t) = \delta u_{\mathrm{UGV}}(t)$$

$$\delta\dot{y}_{UGV}(t) = u_{\mathrm{UGV0}}\,\delta\psi_{\mathrm{UGV}}(t)$$

$$\delta\dot{\psi}_{UGV}(t) = \delta\tilde{\omega}_{UGV}(t)$$

Σε διακριτό χρόνο, το μοντέλο μπορεί να γραφεί στη μορφή

$$X_{\mathrm{UGV}[k+1]} = A_{\mathrm{UGVd}} X_{\mathrm{UGV}[k]} + B_{\mathrm{UGVd}} U_{\mathrm{UGV}[k]}$$

όπου οι πίνακες A_{UGVd} και B_{UGVd} προκύπτουν από τη διακριτοποίηση του συνεχούς γραμμικοποιημένου μοντέλου και χρησιμοποιούνται για την υλοποίηση ψηφιακών ελεγκτών σε εφαρμογές πραγματικού χρόνου.

Ομοίως για το σύστημα εξισορρόπησης. Άρα το συνολικό σύστημα χώρου καταστάσεων γράφεται:

$$\dot{x}_b(t) = A_b x_b(t) + B_b u_b(t) + E_b w_b(t)$$

με:

$$x_b(t) = \begin{bmatrix} \theta_{UGV,c}(t) \\ \dot{\theta}_{UGV,c}(t) \end{bmatrix}$$

$u_b(t) =$ εντολή σερβοκινητήρων

και διάνυσμα διαταραχών:

$$w_b(t) = \begin{bmatrix} d_{UGV,\theta}(t) \\ \ddot{\theta}_{UGV}(t) \end{bmatrix}$$

Οι πίνακες του συστήματος είναι:

$$A_b = \begin{bmatrix} 0 & 1 \\ 0 & -\dfrac{B_{UGV,\theta}}{J_{UGV,\theta}} \end{bmatrix}$$

$$B_b = \begin{bmatrix} 0 \\ \dfrac{K_\tau}{J_{UGV,\theta}} \end{bmatrix}$$

$$E_b = \begin{bmatrix} 0 & 0 \\ \dfrac{1}{J_{UGV,\theta}} & -1 \end{bmatrix}$$

Η έξοδος του συστήματος είναι η τελική γωνία της πλατφόρμας:

$$y_b(t) = \theta_{pad}(t)$$

δηλαδή:

$$y_b(t) = \theta_{UGV}(t) + \theta_{UGV,c}(t)$$

ή σε μορφή εξόδου χώρου καταστάσεων:

$$y_b(t) = Cx(t) + \theta_{UGV}(t)$$

με:

$$C = [1 \quad 0]$$

Η αναπαράσταση στο χώρο κατάστασης και οι σύγχρονες τεχνικές ελέγχου έχουν ευρεία εφαρμογή σε συστήματα ρομποτικής και αυτόνομων οχημάτων. Συγκεκριμένα, έχουν αξιοποιηθεί σε προβλήματα ελέγχου και ακολούθησης εντολής αυτοεξισορροπούμενων ρομπότ με χρήση ασαφούς ελέγχου [52], σε συνεργατικό έλεγχο και οπτικοποίηση σχηματισμών κινητών ρομπότ [50], καθώς και σε προβλήματα πλευρικού ελέγχου οχημάτων και αυτόνομων συστημάτων καθοδήγησης [35], [40]. Επιπλέον, τεχνικές αποσύζευξης και εύρωστου ελέγχου έχουν εφαρμοστεί σε οχήματα τετραδιεύθυνσης (4WS), τόσο για μη αλληλοεπιδρώντα σχήματα ελέγχου [63], όσο και για εύρωστο έλεγχο οχημάτων με εμπρόσθια και οπίσθια διεύθυνση [65]. Αντίστοιχα, σύγχρονες προσεγγίσεις εύρωστου αποσυζευγμένου ελέγχου μεταβαλλόμενης ταχύτητας εφαρμόζονται και σε αυτόνομα οχήματα τετραδιεύθυνσης [36], επιβεβαιώνοντας τη σημασία των μεθόδων αυτών σε προηγμένα συστήματα ευφυών μεταφορών και ρομποτικής.

3. ΑΛΓΟΡΙΘΜΟΙ ΕΛΕΓΧΟΥ ΡΟΜΠΟΤΙΚΩΝ ΟΧΗΜΑΤΩΝ

3.1 Μαθηματική Ανάλυση Μοντέλου Εκτίμησης Γωνιακού Προσανατολισμού

Για την εκτίμηση του γωνιακού προσανατολισμού της πλατφόρμας αξιοποιήθηκαν οι μετρήσεις του γυροσκοπίου και του επιταχυνσιομέτρου της μονάδας αδρανειακής μέτρησης MPU6050. Το γυροσκόπιο παρέχει τη γωνιακή ταχύτητα ως προς κάθε άξονα, ενώ το επιταχυνσιόμετρο παρέχει πληροφορία σχετική με την κλίση του σώματος ως προς το πεδίο της βαρύτητας. Η χρήση αποκλειστικά των μετρήσεων του γυροσκοπίου οδηγεί σε συσσώρευση σφάλματος λόγω drift, ενώ η εκτίμηση που βασίζεται μόνο στο επιταχυνσιόμετρο επηρεάζεται από θόρυβο και δυναμικές επιταχύνσεις [52]. Για τον λόγο αυτό, η τελική εκτίμηση της γωνίας πραγματοποιείται με συνδυασμό των δύο αισθητηρίων μέσω συμπληρωματικού φίλτρου (complementary filter) [13].

Η μέτρηση του γυροσκοπίου ως προς κάθε άξονα μπορεί να περιγραφεί από τη σχέση:

$$\tilde{\omega}_{UGVm}(t) = \tilde{\omega}_{UGV}(t) + b_{UGV} + n_{UGVg}(t)$$

όπου $\tilde{\omega}_{UGVm}(t)$ είναι η μετρούμενη γωνιακή ταχύτητα, $\tilde{\omega}_{UGV}(t)$ η πραγματική γωνιακή ταχύτητα, b_{UGV} ο όρος μετατόπισης μηδενός (bias) του αισθητήρα και $n_{UGVg}(t)$ ο στοχαστικός θόρυβος της μέτρησης. Μετά την εκτίμηση του bias κατά τη διαδικασία βαθμονόμησης, η κινηματική εξίσωση της γωνίας διατυπώνεται ως:

$$\dot{\theta}_{UGV}(t) = \tilde{\omega}_{UGVm}(t) - b_{UGV}$$

Η ανωτέρω διαφορική εξίσωση διακριτοποιείται για χρονικό βήμα δειγματοληψίας Δt με τη μέθοδο Euler, οπότε η εκτίμηση της γωνίας από την ολοκλήρωση του γυροσκοπίου γράφεται ως:

$$\theta_{UGVg}(k) = \theta_{UGV}(k-1) + [\tilde{\omega}_{UGVm}(k) - b_{UGV}]\Delta t$$

όπου $\theta_{UGVg}(k)$ αποτελεί την πρόβλεψη της γωνίας στη διακριτή χρονική στιγμή k, με βάση τη μέτρηση του γυροσκοπίου και την αφαίρεση του bias.

Παράλληλα, η εκτίμηση των γωνιών κλίσης από το επιταχυνσιόμετρο προκύπτει από τη γεωμετρική προβολή του διανύσματος της βαρύτητας στους άξονες του αισθητήρα. Ειδικότερα, για τις γωνίες roll και pitch ισχύουν οι σχέσεις:

$$\varphi_{UGVacc}(k) = \operatorname{atan2}\,\alpha_{UGVy}(k), \alpha_{UGVz}(k))$$

$$\theta_{UGVacc}(k) = \operatorname{atan2}(-\alpha_{UGVx}(k), \sqrt{\alpha_{UGVy}{}^2(k) + \alpha_{UGVz}{}^2(k)})$$

όπου $\alpha_{UGVx}(k)$, $\alpha_{UGVy}(k)$ και $\alpha_{UGVz}(k)$ είναι οι μετρήσεις του επιταχυνσιομέτρου στους τρεις άξονες. Οι παραπάνω εξισώσεις παρέχουν μία ανεξάρτητη εκτίμηση της κλίσης, η οποία χαρακτηρίζεται από καλή μακροχρόνια σταθερότητα, αλλά χαμηλότερη ανθεκτικότητα σε δυναμικές μεταβολές.

Για τον συνδυασμό των δύο πηγών πληροφορίας χρησιμοποιήθηκε complementary filter, το οποίο για τη γωνία pitch γράφεται ως:

$$\theta_{UGV}(k) = \alpha_{UGV}\theta_{UGVg}(k) + (1 - \alpha_{UGV})\theta_{UGVacc}(k)$$

Αντικαθιστώντας την έκφραση της $\theta_{UGVg}(k)$, η παραπάνω σχέση λαμβάνει τη μορφή:

$$\theta_{UGV}(k) = \alpha_{UGV}\,[\theta_{UGV}(k-1) + (\tilde{\omega}_{UGVm}(k) - b_{UGV})\Delta t] + (1-\alpha_{UGV})\theta_{UGVacc}(k)$$

Αντίστοιχα, για τη γωνία roll η σχέση εκτίμησης διατυπώνεται ως:

$$\varphi_{UGV}(k) = \alpha_{UGV}\left[\varphi_{UGV}(k-1) + \left(\tilde{\omega}_{UGVx,m}(k) - b_{UGVx}\right)\Delta t\right] + (1-\alpha_{UGV})\varphi_{UGVacc}(k)$$

ενώ για τη γωνία pitch:

$$\theta_{UGV}(k) = \alpha_{UGV}\left[\theta_{UGV}(k-1) + \left(\tilde{\omega}_{UGVy,m}(k) - b_{UGVy}\right)\Delta t\right] + (1-\alpha_{UGV})\theta_{UGVacc}(k)$$

όπου $\alpha_{UGV} \in (0,1)$ είναι ο συντελεστής στάθμισης του complementary filter. Η τιμή του α_{UGV} επιλέγεται ώστε να δίνει μεγαλύτερη βαρύτητα στη βραχυχρόνια ακρίβεια του γυροσκοπίου και μικρότερη, αλλά σταθεροποιητική, βαρύτητα στην πληροφορία του επιταχυνσιομέτρου.

Επιπρόσθετα, για την έκφραση της απόκλισης της πλατφόρμας ως προς μία επιλεγμένη θέση αναφοράς, ορίζεται στιγμή μηδενισμού k_0, στην οποία καταγράφονται οι αρχικές γωνίες $\varphi_{UGV}(k_0)$ και $\theta_{UGV}(k_0)$. Οι σχετικές γωνίες ως προς τη θέση αυτή προκύπτουν από τις εξισώσεις:

$$\Delta\varphi_{UGV}(k) = \varphi_{UGV}(k) - \varphi_{UGV}(k_0)$$
$$\Delta\theta_{UGV}(k) = \theta_{UGV}(k) - \theta_{UGV}(k_0)$$

Με τον τρόπο αυτό, το σύστημα δεν εκτιμά μόνο τον απόλυτο προσανατολισμό, αλλά και τη μεταβολή της γωνίας ως προς τη θέση που ορίζεται ως σημείο αναφοράς από τον χρήστη.

Στην τελική υλοποίηση, η εκτιμώμενη γωνία πρόνευσης θ_{UGV} της πλατφόρμας δεν χρησιμοποιείται μόνο για την παρακολούθηση της κατάστασής της, αλλά και για την αυτόματη διόρθωση της θέσης της. Κατά τη διαδικασία μηδενισμού, το σύστημα καταγράφει τις τρέχουσες τιμές pitch και roll ως θέση αναφοράς. Η μηχανική διόρθωση της πλατφόρμας εφαρμόζεται στον άξονα που ελέγχεται από τους δύο σερβοκινητήρες. Αν η γωνία αυτού του άξονα συμβολιστεί ως θ_{UGV}, τότε η τιμή αναφοράς συμβολίζεται ως θ_{UGV0} και το σφάλμα κλίσης σε κάθε χρονική στιγμή υπολογίζεται από τη σχέση:

$$e_{UGV}(k) \;=\; \theta_{UGV}(k) \;-\; \theta_{UGV0}$$

όπου $\theta_{UGV}(k)$ είναι η τρέχουσα εκτιμώμενη γωνία που προκύπτει από το συμπληρωματικό φίλτρο και θ_{UGV0} είναι η γωνία αναφοράς. Αν η απόλυτη τιμή του σφάλματος είναι μικρότερη από ένα προκαθορισμένο όριο ε, δηλαδή αν ισχύει:

$$|e_{UGV}(k)| \;\leq\; \varepsilon$$

τότε δεν εφαρμόζεται νέα διόρθωση, ώστε να αποφεύγονται συνεχείς μικροκινήσεις λόγω θορύβου του αισθητήρα. Όταν το σφάλμα ξεπεράσει τη νεκρή ζώνη, δηλαδή όταν ισχύει:

$$e_{UGV}(k)| \;>\; \varepsilon$$

υπολογίζεται διορθωτική μεταβολή της γωνίας της πλατφόρμας:

$$\Delta\theta_{UGV}(k) \;=\; K \;\cdot\; e_{UGV}(k)$$

όπου K είναι ο συντελεστής ενίσχυσης του αυτόματου ελέγχου. Η μεταβολή αυτή περιορίζεται σε μέγιστο επιτρεπτό βήμα, ώστε η πλατφόρμα να μην κινείται απότομα.

Στη συνέχεια, η νέα επιθυμητή γωνία της πλατφόρμας προκύπτει από την προηγούμενη τιμή και τη διορθωτική μεταβολή:

$$\theta_{UGVc}(k) = \theta_{UGVc}(k-1) + \Delta\theta_{UGV}(k)$$

με περιορισμό στο επιτρεπτό εύρος λειτουργίας των σερβοκινητήρων. Στην παρούσα υλοποίηση το εύρος αυτό είναι:

$$0° \leq \theta_{UGVc}(k) \leq 35°$$

Οι δύο σερβοκινητήρες της πλατφόρμας λειτουργούν κατοπτρικά. Επομένως, αν η επιθυμητή γωνία του πρώτου σερβοκινητήρα είναι:

$$\theta_{UGV1}(k) = \theta_{UGVc}(k)$$

τότε η γωνία του δεύτερου σερβοκινητήρα δίνεται από τη σχέση:

$$\theta_{UGV2}(k) = 35° - \theta_{UGVc}(k)$$

Με αυτόν τον τρόπο υλοποιείται ένας απλός κλειστός βρόχος αυτόματης εξισορρόπησης, στον οποίο η μέτρηση του MPU6050 λειτουργεί ως ανάδραση και η πλατφόρμα διορθώνει σταδιακά τη θέση της, χωρίς απότομες κινήσεις ή συνεχείς ταλαντώσεις.

3.2 Μαθηματική Ανάλυση Ελεγκτή Πτήσης του ArduPilot Copter

Ο έλεγχος πτήσης του ArduPilot Copter βασίζεται σε ένα ιεραρχικό σύστημα ανάδρασης, το οποίο επιτρέπει τη σταθεροποίηση και τον έλεγχο του μη επανδρωμένου εναέριου οχήματος ως προς τον προσανατολισμό, τη γωνιακή ταχύτητα, την ταχύτητα μεταφοράς και τη θέση του στον χώρο. Η λειτουργία του συστήματος βασίζεται κυρίως σε διαδοχικούς ελεγκτές PID (Proportional–Integral–Derivative), οι οποίοι συνεργάζονται με αλγορίθμους εκτίμησης κατάστασης EKF (Extended Kalman Filter), ώστε να επιτυγχάνεται σταθερή και αξιόπιστη πτήση ακόμη και υπό παρουσία εξωτερικών διαταραχών [44], [61], [64].

Το τετρακόπτερο μοντελοποιείται ως μη γραμμικό δυναμικό σύστημα έξι βαθμών ελευθερίας, το οποίο περιγράφεται από μεταφορικές και περιστροφικές εξισώσεις κίνησης. Η μεταφορική δυναμική του συστήματος προκύπτει από τον δεύτερο νόμο του Νεύτωνα και γράφεται ως:

$$m\ddot{w} = RF_T + F_g + F_d$$

όπου m είναι η μάζα του οχήματος, $w = [x, y, z]^T$ το διάνυσμα θέσης, Ro πίνακας περιστροφής από το σύστημα σώματος στο αδρανειακό σύστημα αναφοράς, F_T η συνολική ώση των κινητήρων, $F_g = [0 \quad 0 \quad -mg]^T$ η βαρυτική δύναμη και F_d οι αεροδυναμικές διαταραχές που επιδρούν στο όχημα.

Παράλληλα, η περιστροφική δυναμική του συστήματος περιγράφεται μέσω των εξισώσεων Euler:

$$J\dot{\omega} + \omega(J\omega) = \tau$$

όπου J είναι ο πίνακας ροπής αδράνειας του σώματος, $\omega = [p, q, r]^T$ το διάνυσμα γωνιακών ταχυτήτων και τ το διάνυσμα ροπών ελέγχου που παράγονται από τους κινητήρες.

Για τον έλεγχο του προσανατολισμού του UAV, το ArduPilot χρησιμοποιεί ξεχωριστούς βρόχους ελέγχου για κάθε άξονα περιστροφής του οχήματος, δηλαδή για τους άξονες roll, pitch και yaw [36], [38], [60]. Ο εξωτερικός βρόχος ελέγχου υπολογίζει αρχικά την επιθυμητή γωνιακή ταχύτητα με βάση το σφάλμα μεταξύ της επιθυμητής και της πραγματικής γωνίας προσανατολισμού. Για τον άξονα roll, η επιθυμητή γωνιακή ταχύτητα δίνεται από τη σχέση:

$$p_{des}(k) = K_{p,\varphi}\left(\varphi_{ref}(k) - \varphi(k)\right)$$

Όπου $\varphi_{ref}(k)$ είναι η επιθυμητή γωνία roll, $\varphi(k)$ η πραγματική γωνία roll του οχήματος και $K_{p,\varphi}$ ο αναλογικός συντελεστής του ελεγκτή για τον συγκεκριμένο άξονα.

Αντίστοιχα, για τον άξονα pitch, η επιθυμητή γωνιακή ταχύτητα υπολογίζεται από:

$$q_{des}(k) = K_{p,\theta}\left(\theta_{ref}(k) - \theta(k)\right)$$

Όπου $q_{ref}(k)$ είναι η επιθυμητή γωνία pitch, $\theta(k)$ η πραγματική γωνία pitch και $K_{p,\theta}$ ο αναλογικός συντελεστής ελέγχου.

Τέλος για τον άξονα yaw, ο έλεγχος του προσανατολισμού πραγματοποιείται μέσω της σχέσης:

$$r_{des}(k) = K_{p,\psi}\left(\psi_{ref}(k) - \psi(k)\right)$$

Όπου $r_{ref}(k)$ είναι η επιθυμητή γωνία yaw, $\psi(k)$ η πραγματική γωνία yaw και $K_{p,\psi}$ ο αναλογικός συντελεστής.

Η επιθυμητή γωνιακή ταχύτητα τροφοδοτείται στον εσωτερικό βρόχο PID, ο οποίος ελέγχει τις πραγματικές γωνιακές ταχύτητες του συστήματος. Η μαθηματική μορφή του PID ελεγκτή διατυπώνεται ως:

$$u(k) = K_p e(k) + K_i \sum_{i=0}^{k} e(i)\Delta t + K_d \frac{e(k) - e(k-1)}{\Delta t}$$

όπου $u(k)$ είναι το σήμα ελέγχου, $e(k)$ το σφάλμα μεταξύ επιθυμητής και πραγματικής γωνιακής ταχύτητας, ενώ $K_p, K_i \; και \; K_d$ είναι οι αναλογικοί, ολοκληρωτικοί και διαφορικοί συντελεστές αντίστοιχα.

Το σφάλμα γωνιακής ταχύτητας για κάθε άξονα δίνεται από την σχέση:

$$e_\omega(k) = \omega_{des}(k) - \omega(k)$$

και επομένως η ροπή ελέγχου υπολογίζεται από τη σχέση:

$$\tau(k) = K_p e_\omega(k) + K_i \sum e_\omega(k)\Delta t \; + \; K_d \frac{de_\omega(k)}{dt}$$

Ο αναλογικός όρος αυξάνει την ταχύτητα απόκρισης του συστήματος, ο ολοκληρωτικός όρος εξαλείφει το μόνιμο σφάλμα, ενώ ο διαφορικός όρος βελτιώνει την απόσβεση και περιορίζει τις ταλαντώσεις.

Για την εκτίμηση της κατάστασης του UAV, το ArduPilot χρησιμοποιεί EKF, το οποίο συνδυάζει δεδομένα από γυροσκόπιο, επιταχυνσιόμετρο, GPS, μαγνητόμετρο και βαρόμετρο. Η πρόβλεψη της κατάστασης περιγράφεται από:

$$\hat{x}_k = f(\hat{x}_{k-1}, u_k, w_k)$$

ενώ η εξίσωση μέτρησης γράφεται ως:

$$z_k = h(\hat{x}_k, u_k)$$

όπου $\hat{x}_k$ είναι το εκτιμώμενο διάνυσμα κατάστασης, u_k οι είσοδοι ελέγχου, z_k οι μετρήσεις των αισθητήρων και w_k και u_k οι θόρυβοι συστήματος και μέτρησης αντίστοιχα.

Η τελική κατανομή της ώσης στους κινητήρες πραγματοποιείται μέσω πίνακα μίξης (motor mixing matrix). Για διάταξη τετρακόπτερου τύπου "X", οι δυνάμεις των κινητήρων υπολογίζονται από (όπου T είναι η συνολική ώση, τ_φ η ροπή roll, τ_θ η ροπή pitch και τ_ψ η ροπή yaw):

$$F_1 = T + \tau_\varphi + \tau_\theta + \tau_\psi$$

$$F_2 = T - \tau_\varphi + \tau_\theta - \tau_\psi$$

$$F_3 = T - \tau_\varphi - \tau_\theta + \tau_\psi$$

$$F_4 = T + \tau_\varphi - \tau_\theta - \tau_\psi$$

3.3 Μαθηματική Ανάλυση Συστήματος Landing Target του ArduPilot

Οι τεχνολογίες μηχανικής όρασης, τεχνητής νοημοσύνης και ενεργειακής υποστήριξης αποτελούν βασικούς άξονες ανάπτυξης σύγχρονων UAVs και αυτόνομων ρομποτικών συστημάτων. Ειδικότερα, έχουν μελετηθεί τεχνολογίες τροφοδοσίας για UAV και εφαρμογές machine vision, με στόχο τη συγκριτική αξιολόγηση ενεργειακών λύσεων και την ανάλυση μελλοντικών τάσεων [57], [58]. Παράλληλα, συστήματα ρομποτικής όρασης έχουν αξιοποιηθεί σε εφαρμογές τεχνητής νοημοσύνης και πολιτιστικής κληρονομιάς, όπως η ανάπτυξη τρισδιάστατα εκτυπωμένου ρομποτικού συστήματος όρασης για μεταγραφή και σχολιασμό χειρόγραφων [55]. Επιπλέον, σύγχρονες ερευνητικές προσεγγίσεις επικεντρώνονται στη χρήση UAV για καταγραφή ανθρώπινης κίνησης [34], σε τεχνικές localisation UAVs αποκλειστικά μέσω οπτικών δεδομένων [33], καθώς και σε προηγμένες μεθόδους computer vision και machine learning για αυτόνομη πλοήγηση UAV σε πραγματικές συνθήκες [56]. Αντίστοιχα, εφαρμογές αναγνώρισης προσώπου σε πλατφόρμες ArduPilot [54], συστήματα AIoT για έλεγχο UAVs [59] και αρχιτεκτονικές σχεδίασης purpose-built UAV βασισμένες σε Pixhawk και ArduPilot [68] αναδεικνύουν τη στενή σύνδεση της μηχανικής όρασης με τα σύγχρονα αυτόνομα εναέρια συστήματα.

Το σύστημα Landing Target του ArduPilot χρησιμοποιείται για την αυτόνομη και ακριβή προσγείωση ενός UAV πάνω σε προκαθορισμένο στόχο αναφοράς. Η λειτουργία του συστήματος βασίζεται στην επεξεργασία εικόνας και στον υπολογισμό της σχετικής θέσης του στόχου ως προς το UAV, ώστε να παράγονται οι κατάλληλες διορθωτικές εντολές πτήσης κατά τη διαδικασία καθόδου [33]. Η μέθοδος αυτή αξιοποιείται κυρίως σε εφαρμογές computer vision, όπου η κάμερα του UAV ανιχνεύει έναν οπτικό δείκτη, ArUco marker και υπολογίζει τη θέση του στόχου στο επίπεδο της εικόνας.

Αρχικά, θεωρείται ότι το κέντρο της εικόνας της κάμερας βρίσκεται στη θέση (u_0, v_0), ενώ η θέση του ανιχνευμένου στόχου αντιστοιχεί στις συντεταγμένες (u, v). Η απόκλιση του στόχου από το κέντρο της εικόνας εκφράζεται μέσω των σφαλμάτων εικόνας:

$$e_u = u - u_0$$

$$e_v = v - v_0$$

όπου e_u και e_v αποτελούν το οριζόντιο και κατακόρυφο σφάλμα αντίστοιχα. Τα μεγέθη αυτά εκφράζουν την απόσταση του στόχου από το κέντρο του οπτικού πεδίου και χρησιμοποιούνται για τον υπολογισμό της σχετικής θέσης του UAV ως προς τον στόχο προσγείωσης.

Η μετατροπή των σφαλμάτων εικόνας σε γωνιακές αποκλίσεις πραγματοποιείται μέσω του μοντέλου προβολής της κάμερας. Αν f_x και f_y είναι οι εστιακές αποστάσεις της κάμερας στους άξονες x και y, τότε οι αντίστοιχες γωνίες απόκλισης υπολογίζονται από τις σχέσεις:

$$a_x = \tan^{-1}\left(\frac{e_u}{f_x}\right)$$

$$a_y = \tan^{-1}\left(\frac{e_v}{f_y}\right)$$

Οι γωνίες a_x και a_y εκφράζουν τη σχετική κατεύθυνση του στόχου ως προς το UAV στο τρισδιάστατο περιβάλλον. Εφόσον είναι γνωστό το ύψος πτήσης h από τους αισθητήρες, η σχετική μετατόπιση του στόχου ως προς το όχημα προκύπτει μέσω γεωμετρικών σχέσεων:

$$x_t = h \tan a_x$$

$$y_t = h \tan a_y$$

όπου x_t και y_t αντιστοιχούν στις μετατοπίσεις του στόχου ως προς το UAV στους δύο οριζόντιους άξονες. Οι ποσότητες αυτές αποτελούν την είσοδο του συστήματος ελέγχου θέσης του ArduPilot.

Για τη διόρθωση της θέσης του UAV χρησιμοποιείται ελεγκτής PID, ο οποίος παράγει διορθωτικές ταχύτητες ή γωνίες πτήσης με σκοπό την ελαχιστοποίηση του σφάλματος στόχευσης. Η εξίσωση ελέγχου για τον άξονα x διατυπώνεται ως:

$$v_x(k) = K_p x_t(k) + K_i \sum_{i=0}^{k} x_t(i)\Delta t + K_d \frac{x_t(k) - x_t(k-1)}{\Delta t}$$

Αντίστοιχα, για τον άξονα y:

$$v_y(k) = K_p y_t(k) + K_i \sum_{i=0}^{k} y_t(i)\Delta t + K_d \frac{y_t(k) - y_t(k-1)}{\Delta t}$$

Όπου $v_x(k)$ και $v_y(k)$ είναι οι διορθωτικές ταχύτητες, K_p, K_i και K_d οι συντελεστές του PID ελεγκτή και Δt το χρονικό διάστημα δειγματοληψίας.

Κατά τη διαδικασία προσγείωσης, το UAV μειώνει σταδιακά το ύψος του, ενώ παράλληλα ο ελεγκτής προσαρμόζει συνεχώς τη θέση του ώστε να ελαχιστοποιηθεί η απόσταση από το κέντρο του στόχου. Η διαδικασία θεωρείται επιτυχής όταν οι αποκλίσεις στους δύο άξονες παραμένουν εντός προκαθορισμένων ορίων ανοχής:

$$|x_t| < \varepsilon_x$$

$$|y_t| < \varepsilon_y$$

όπου ε_x και ε_y είναι μικρές επιτρεπτές τιμές σφάλματος που καθορίζουν την ακρίβεια της προσγείωσης.

4. ΥΛΟΠΟΙΗΣΗ

4.1 Υλοποίηση Τετρακοπτέρου

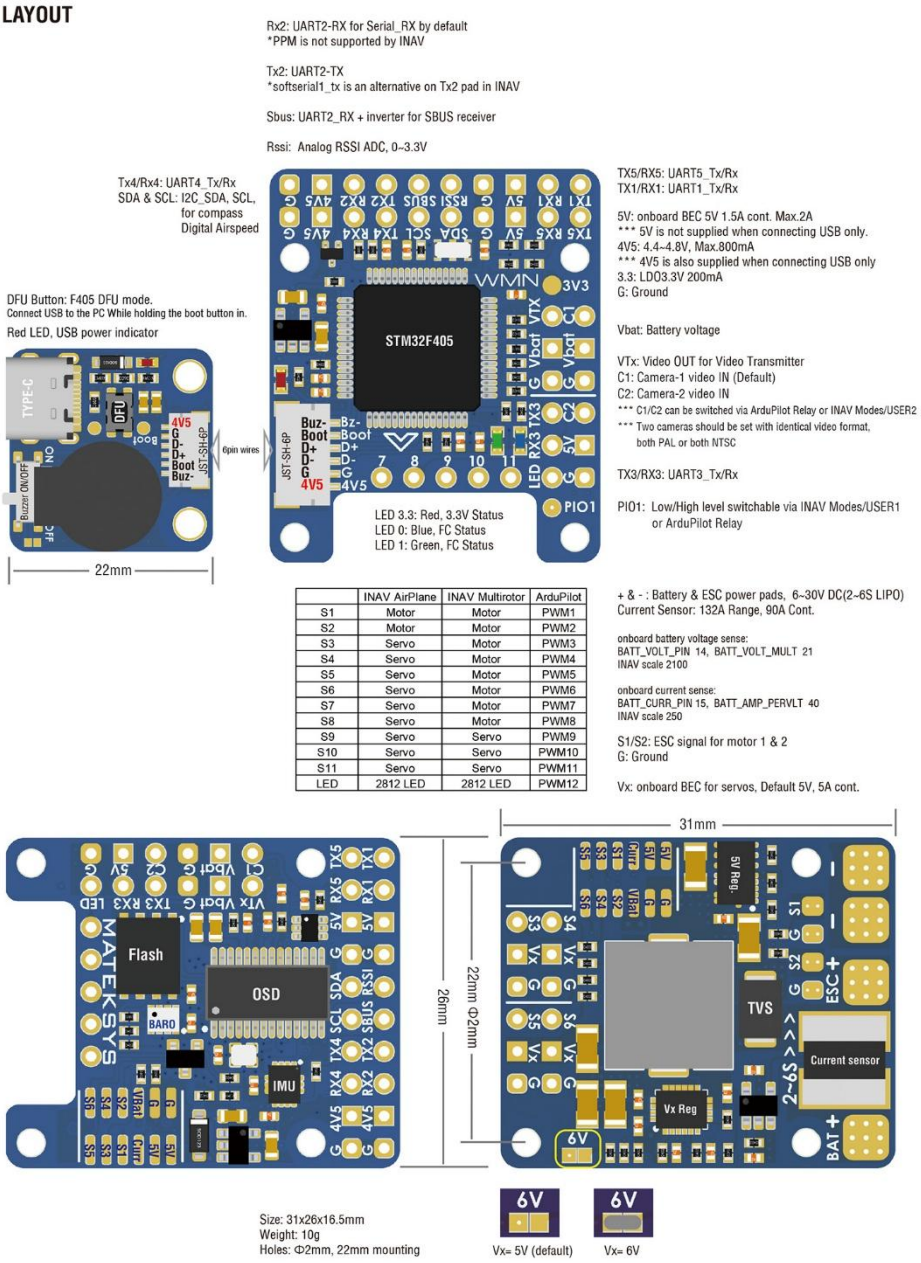
Η υλοποίηση του τετρακοπτέρου αποτέλεσε ένα από τα βασικά τμήματα της παρούσας εργασίας, καθώς το εναέριο όχημα έπρεπε να είναι ικανό να απογειώνεται, να διατηρεί σταθερή πτήση, να κινείται σε περιορισμένο χώρο και να εκτελεί διαδικασία προσγείωσης με τη βοήθεια μηχανικής όρασης. Για τον σκοπό αυτό, η σχεδίαση του UAV βασίστηκε σε γεωμετρία τύπου F450 (Εικόνα 4.3), η οποία προσφέρει συμμετρική διάταξη των κινητήρων, επαρκή χώρο για την τοποθέτηση των ηλεκτρονικών εξαρτημάτων και συμβατότητα με πίνακα διανομής ισχύος.

Η κατασκευή του τετρακοπτέρου αντιμετωπίστηκε ως συνδυασμός μηχανικού σχεδιασμού, επιλογής κατάλληλων εξαρτημάτων, ηλεκτρολογικής διασύνδεσης και παραμετροποίησης του συστήματος ελέγχου πτήσης. Ιδιαίτερη έμφαση δόθηκε στη δυνατότητα ενσωμάτωσης κάμερας μηχανικής όρασης και ασύρματης επικοινωνίας, καθώς τα δύο αυτά στοιχεία είναι απαραίτητα για τη λειτουργία του συστήματος αυτόνομης προσγείωσης.

4.1.1 Επιλογή Υλικών και Εξαρτημάτων

Η επιλογή των εξαρτημάτων του τετρακοπτέρου έγινε με βάση τις απαιτήσεις της εφαρμογής, δηλαδή τη σταθερή πτήση, τη δυνατότητα ελεγχόμενης προσγείωσης, την επαρκή ώση, την απλή συναρμολόγηση και τη συμβατότητα με λογισμικό ανοιχτού κώδικα. Βασικό στοιχείο του συστήματος αποτέλεσε ο ελεγκτής πτήσης (Flight Controller - FC), ο οποίος είναι υπεύθυνος για τη σταθεροποίηση του UAV και την εκτέλεση των εντολών πτήσης.

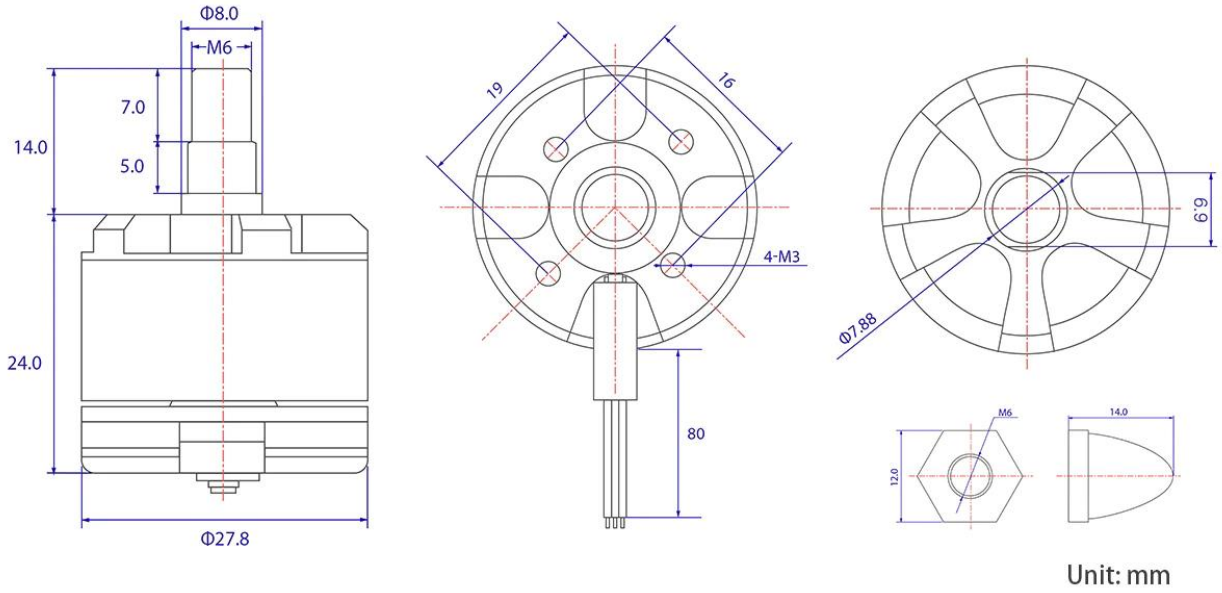
Για τον σκοπό αυτό επιλέχθηκε ο ελεγκτής πτήσης Matek F405-WMN (Εικόνα 4.1), ο οποίος βασίζεται στον μικροελεγκτή STM32F405RGT6 και διαθέτει ενσωματωμένη μονάδα αδρανειακής μέτρησης ICM42688-P και βαρόμετρο SPL06-001 [50], [67]. Η επιλογή του συγκεκριμένου ελεγκτή έγινε λόγω της συμβατότητάς του με το ArduPilot, καθώς και λόγω της δυνατότητας παραμετροποίησης του λογισμικού σύμφωνα με τις ανάγκες της εργασίας [23]. Στο πλαίσιο αυτό χρησιμοποιήθηκε κατάλληλη έκδοση του firmware, στην οποία διατηρήθηκαν οι λειτουργίες που ήταν απαραίτητες για την πειραματική διάταξη [19].



	INAV AirPlane	INAV Multirotor	ArduPilot
S1	Motor	Motor	PWM1
S2	Motor	Motor	PWM2
S3	Servo	Motor	PWM3
S4	Servo	Motor	PWM4
S5	Servo	Motor	PWM5
S6	Servo	Motor	PWM6
S7	Servo	Motor	PWM7
S8	Servo	Motor	PWM8
S9	Servo	Servo	PWM9
S10	Servo	Servo	PWM10
S11	Servo	Servo	PWM11
LED	2812 LED	2812 LED	PWM12

Εικόνα 4.1 Συνδεσιμότητα του MatekF405-WMN [31]

Για την πρόωση του τετρακοπτέρου επιλέχθηκαν τέσσερις κινητήρες A2212 920KV (Εικόνα 4.2) σε συνδυασμό με προπέλες 1045. Ο συνδυασμός αυτός κρίθηκε κατάλληλος για το εκτιμώμενο βάρος του UAV, παρέχοντας επαρκή ώση για απογείωση, αιώρηση και ελεγχόμενη κάθοδο. Οι κινητήρες συνδέθηκαν με τέσσερις ηλεκτρονικούς ρυθμιστές ταχύτητας (Electronic Speed Controllers - ESC) LANRC 45A, οι οποίοι υποστηρίζουν το λογισμικό BLHeli_S. Στη συνέχεια, το λογισμικό των ESC αντικαταστάθηκε με BlueJay, ώστε να επιτευχθεί καλύτερη συμβατότητα με τις απαιτήσεις ελέγχου του συστήματος.



Moto	KV (PRM/V)	Voltage (V)	Prop (Inch)	Amps (A)	Thrust (g)	Watts (W)	Efficiency (g/W)	Throttle (%)
A2212	920	11.1	9450	1.5	180	16.65	10.81	50%
				4.2	370	46.62	7.94	75%
				7.8	600	86.58	6.93	100%
			1045	1.8	210	19.98	10.51	50%
				5.2	420	57.72	7.28	75%
				10	680	111	6.13	100%

Εικόνα 4.2 Σχεδιάγραμμα και πληροφορίες για τα Motors [32]

Για την τροφοδοσία του συστήματος χρησιμοποιήθηκε μπαταρία Gens Ace Soaring 3S1P 2200 mAh, η οποία παρέχει ονομαστική τάση 11,1 V και είναι κατάλληλη για εφαρμογές UAV. Η διανομή ισχύος προς τα ESC και τα επιμέρους ηλεκτρονικά πραγματοποιήθηκε μέσω πίνακα διανομής ισχύος (Power Distribution Board - PDB), συμβατού με τη γεωμετρία τύπου F450.

Επιπλέον, χρησιμοποιήθηκε μονάδα Matek M10-5883, η οποία λειτουργεί ως σύστημα GNSS και μαγνητόμετρο για την υποστήριξη πιο αυτόματων λειτουργιών πτήσης. Η προσθήκη GNSS/compass κρίθηκε χρήσιμη, καθώς επιτρέπει στον ελεγκτή πτήσης να αξιοποιεί δεδομένα θέσης και μαγνητικής διεύθυνσης σε παγκόσμιο σύστημα αναφοράς, διευκολύνοντας τη χρήση αυτόνομων ή ημιαυτόνομων λειτουργιών του ArduPilot. Με αυτόν τον τρόπο, ο έλεγχος της πτήσης μπορεί να βασίζεται περισσότερο σε εντολές θέσης και ύψους, εκφρασμένες σε πραγματικές μονάδες μέτρησης, όπως μέτρα, αντί να απαιτείται αποκλειστικά άμεσος και συνεχής χειρισμός της ισχύος των κινητήρων μέσω του throttle. Η λειτουργία αυτή δεν αντικαθιστά το σύστημα μηχανικής όρασης για την τελική φάση προσγείωσης, αλλά παρέχει ένα πιο σταθερό και πρακτικό πλαίσιο για την αρχική πλοήγηση, την αιώρηση και την ελεγχόμενη προσέγγιση της πλατφόρμας.

Για τη μετάδοση εικόνας και την υποστήριξη της μηχανικής όρασης επιλέχθηκε ESP32-CAM με αισθητήρα OV2640. Αρχικά εξετάστηκε η δυνατότητα χρήσης της ίδιας μονάδας τόσο για λήψη εικόνας όσο και για επικοινωνία με τον υπολογιστή και τον ελεγκτή πτήσης. Ωστόσο, η ταυτόχρονη εκτέλεση αυτών των λειτουργιών οδήγησε σε αυξημένη καθυστέρηση και αστάθεια επικοινωνίας. Για τον λόγο αυτό, η τελική αρχιτεκτονική διαχώρισε τις λειτουργίες, χρησιμοποιώντας την ESP32-CAM για τη λήψη εικόνας και ξεχωριστή πλακέτα ESP32 DevKit V1 για την επικοινωνία και τη μεταφορά δεδομένων.

Τέλος, για τη μηχανική δομή του UAV επιλέχθηκε γεωμετρία τύπου F450. Η επιλογή αυτή έγινε λόγω της απλότητας της διάταξης, της συμμετρικής κατανομής των κινητήρων, της διαθεσιμότητας τρισδιάστατων μοντέλων και της δυνατότητας χρήσης έτοιμου πίνακα διανομής ισχύος σχεδιασμένου για τη συγκεκριμένη γεωμετρία. Με αυτόν τον τρόπο, το τετρακόπτερο μπορούσε να κατασκευαστεί μέσω τρισδιάστατης εκτύπωσης, διατηρώντας παράλληλα συμβατότητα με τα απαραίτητα ηλεκτρονικά και μηχανικά εξαρτήματα. Ως βασικό υλικό κατασκευής του σκελετού επιλέχθηκε PC, ενώ για τις μικρότερες βάσεις και τα βοηθητικά στηρίγματα χρησιμοποιήθηκε PLA. Με τον τρόπο αυτό επιτεύχθηκε συνδυασμός μηχανικής αντοχής στα κύρια δομικά μέρη και ευκολίας κατασκευής στα δευτερεύοντα εξαρτήματα.

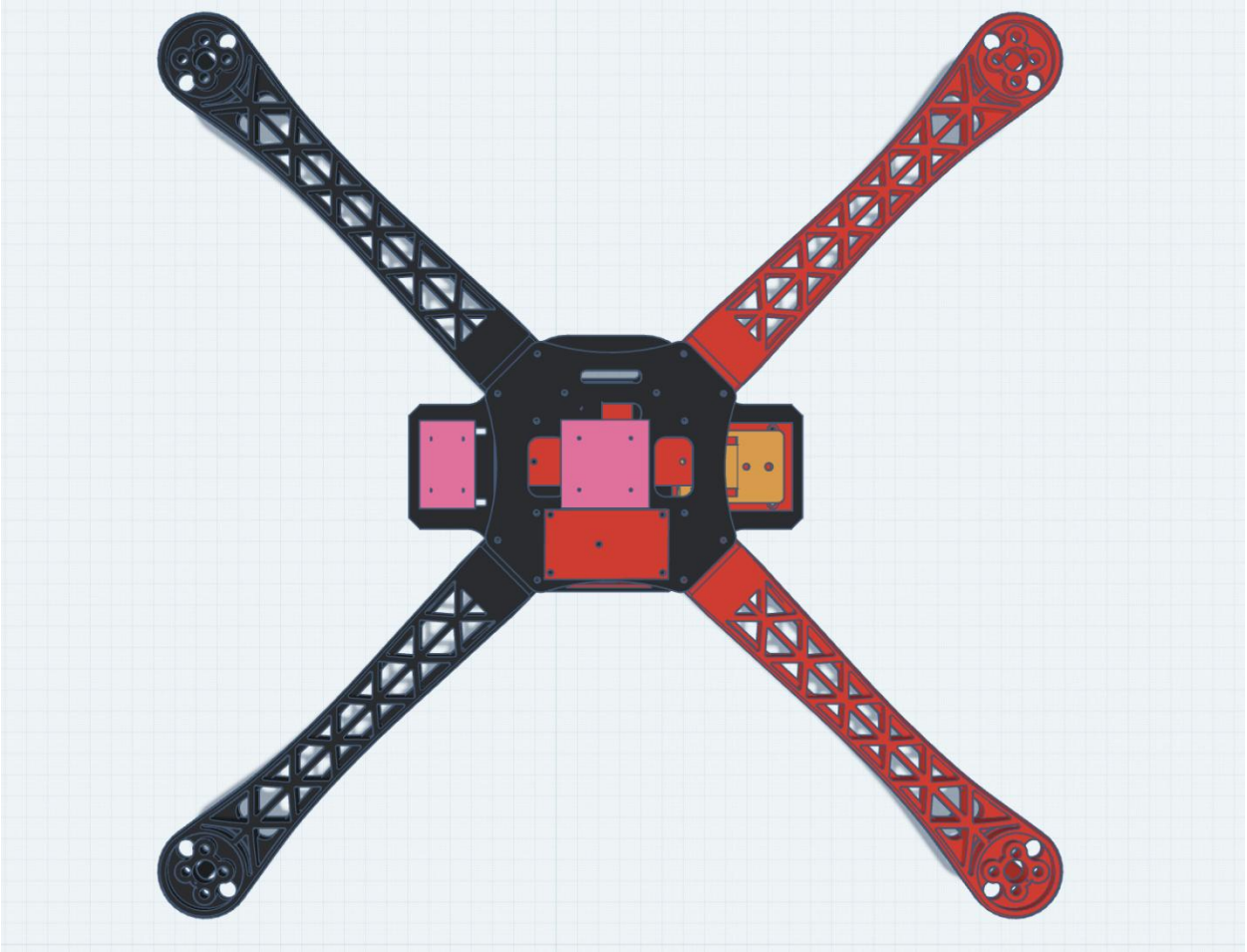
4.1.2 Σχεδίαση και Κατασκευή

Η σχεδίαση του τετρακοπτέρου βασίστηκε στη γεωμετρία τύπου F450, η οποία αποτελεί διαδεδομένη διάταξη σε τετρακόπτερα λόγω της σταθερότητας, της συμμετρίας και της απλής μηχανικής της μορφής. Η ύπαρξη διαθέσιμων τρισδιάστατων μοντέλων αποτέλεσε χρήσιμη βάση αναφοράς για τις βασικές διαστάσεις του σκελετού, ενώ τα επιμέρους μέρη προσαρμόστηκαν στις ανάγκες της παρούσας εργασίας.

Πέρα από τον βασικό σκελετό, σχεδιάστηκαν πρόσθετες βάσεις και προσαρμογές για την τοποθέτηση των ηλεκτρονικών εξαρτημάτων. Οι βάσεις αυτές αφορούσαν κυρίως την τοποθέτηση του ελεγκτή πτήσης, της κάμερας ESP32-CAM, της πλακέτας ESP32, της μπαταρίας και των υπόλοιπων βοηθητικών εξαρτημάτων. Η σχεδίαση πραγματοποιήθηκε με χρήση του TinkerCAD, έπειτα από μετρήσεις των πραγματικών διαστάσεων των εξαρτημάτων και επαναληπτικές διορθώσεις των μοντέλων.

Για την κατασκευή των βασικών μηχανικών μερών του τετρακοπτέρου επιλέχθηκε πολυανθρακικό υλικό (Polycarbonate - PC), καθώς προσφέρει αυξημένη μηχανική αντοχή και διατηρεί ικανοποιητική ακαμψία με μικρότερη ποσότητα υλικού σε σχέση με

το PLA. Η επιλογή αυτή ήταν σημαντική για τον σκελετό του UAV, όπου απαιτούνται σταθερότητα, αντοχή σε φορτία και όσο το δυνατόν χαμηλότερο βάρος. Αντίθετα, για μικρότερες βάσεις και βοηθητικά εξαρτήματα, όπως στηρίγματα ηλεκτρονικών μονάδων και προσαρμογές τοποθέτησης, χρησιμοποιήθηκε PLA (Polylactic Acid), καθώς είναι ευκολότερο στην εκτύπωση και επαρκές για εξαρτήματα που δεν καταπονούνται έντονα μηχανικά.



Εικόνα 4.3 Τρισδιάστατο Μοντέλο Τετρακοπτέρου.

Μετά την ολοκλήρωση της σχεδίασης (Εικόνα 4.3), τα απαραίτητα μηχανικά μέρη κατασκευάστηκαν μέσω τρισδιάστατης εκτύπωσης. Η διαδικασία αυτή επέτρεψε την προσαρμογή της κατασκευής στις ανάγκες του πρωτοτύπου, καθώς και τη γρήγορη τροποποίηση επιμέρους εξαρτημάτων όπου απαιτήθηκε. Η τελική συναρμολόγηση πραγματοποιήθηκε με στόχο τη σταθερή τοποθέτηση όλων των υποσυστημάτων, τη σωστή κατανομή του βάρους και τη δυνατότητα πρόσβασης στα κρίσιμα σημεία για έλεγχο, συντήρηση και επαναληπτικές δοκιμές (Εικόνα 4.4).

Εικόνα 4.4 Τελική μορφή του Τετρακοπτέρου.

4.1.3 Ηλεκτρολογική Διασύνδεση

Η ηλεκτρολογική διασύνδεση του τετρακοπτέρου αποτέλεσε κρίσιμο στάδιο της κατασκευής, καθώς από αυτήν εξαρτάται η ασφαλής τροφοδοσία των κινητήρων, η επικοινωνία των επιμέρους υποσυστημάτων και η σωστή λειτουργία του ελεγκτή πτήσης. Η διαδικασία περιλάμβανε τη σύνδεση των κινητήρων με τα ESC, τη σύνδεση των ESC με τον πίνακα διανομής ισχύος, την τροφοδοσία του ελεγκτή πτήσης και την καλωδίωση των μονάδων ESP32 και ESP32-CAM.

Κάθε κινητήρας συνδέθηκε με το αντίστοιχο ESC μέσω τριών αγωγών φάσης. Στη συνέχεια, τα καλώδια τροφοδοσίας των ESC συγκολλήθηκαν στον πίνακα διανομής ισχύος, μέσω του οποίου πραγματοποιείται η κατανομή της ενέργειας από την μπαταρία προς τους κινητήρες. Ο πίνακας διανομής ισχύος συνδέθηκε με τον ελεγκτή πτήσης, ενώ η κύρια τροφοδοσία του συστήματος καταλήγει σε σύνδεσμο τύπου XT60 για τη σύνδεση της μπαταρίας.

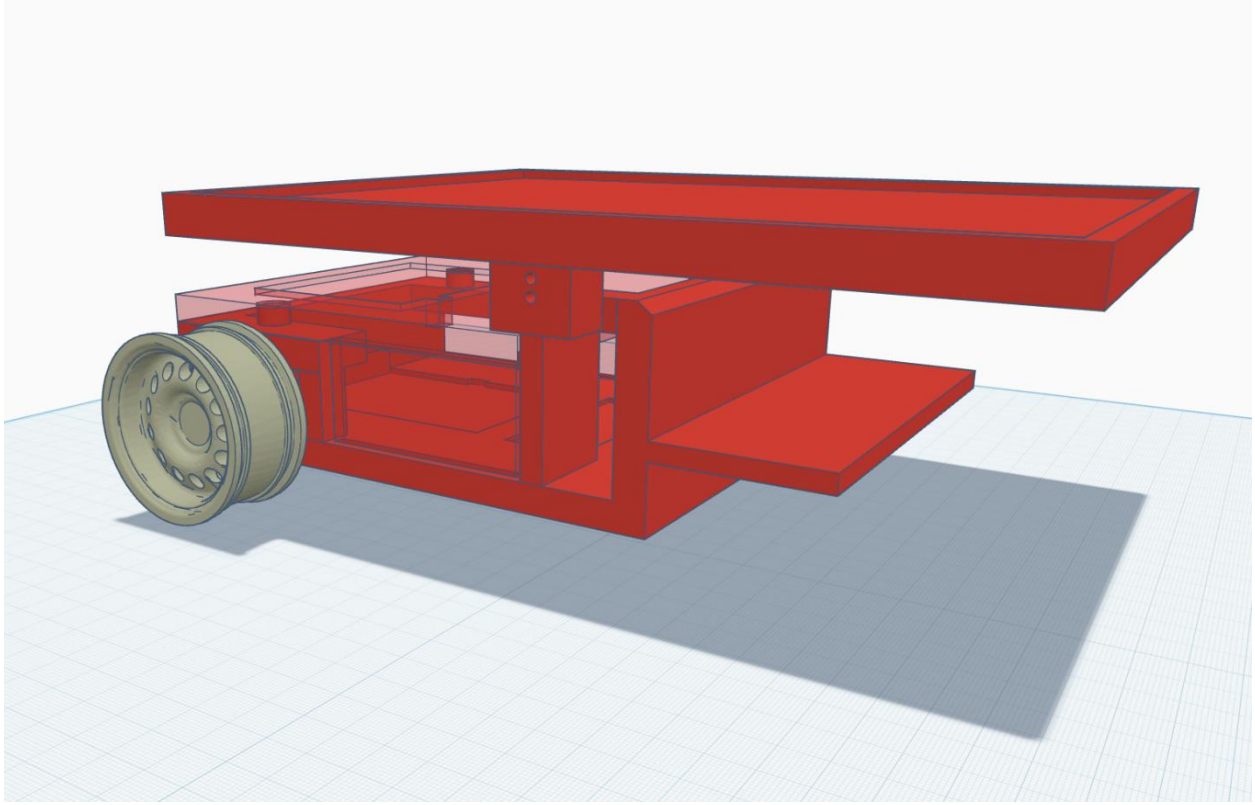
Ιδιαίτερη προσοχή δόθηκε στη σωστή φορά περιστροφής των κινητήρων, στην πολικότητα των συνδέσεων και στη μηχανική σταθερότητα των συγκολλήσεων. Για τις συνδέσεις σημάτων χρησιμοποιήθηκαν καλώδια Dupont, ώστε να είναι δυνατή η σύνδεση των ESC, της κάμερας και της πλακέτας ESP32 με τον ελεγκτή πτήσης. Η χρήση αποσπώμενων συνδέσεων διευκόλυνε τις δοκιμές, τις διορθώσεις και την αποσυναρμολόγηση των επιμέρους τμημάτων όπου αυτό κρίθηκε απαραίτητο.

Μετά την ολοκλήρωση της καλωδίωσης, πραγματοποιήθηκε έλεγχος των συνδέσεων πριν από την ενεργοποίηση του συστήματος. Ο έλεγχος αυτός ήταν απαραίτητος για την αποφυγή λανθασμένης πολικότητας, βραχυκυκλωμάτων ή αστοχίας στις γραμμές τροφοδοσίας και σημάτων. Με την ολοκλήρωση της ηλεκτρολογικής διασύνδεσης, το

UAV ήταν κατασκευαστικά έτοιμο για τις επόμενες φάσεις παραμετροποίησης, ελέγχου και δοκιμών.

4.2 Υλοποίηση Οχήματος Εδάφους

Η υλοποίηση του οχήματος εδάφους αποτέλεσε βασικό μέρος της συνολικής πειραματικής διάταξης, καθώς το όχημα λειτουργεί ως φορέας της πλατφόρμας προσγείωσης. Ο σχεδιασμός του έπρεπε να ικανοποιεί δύο βασικές απαιτήσεις: αφενός να επιτρέπει την ελεγχόμενη μετακίνηση της βάσης στο επίπεδο και αφετέρου να παρέχει επαρκή μηχανική σταθερότητα για την τοποθέτηση της κινούμενης πλατφόρμας και των ηλεκτρονικών υποσυστημάτων.



Εικόνα 4.5 Τρισδιάστατο μοντέλο οχήματος εδάφους.

Για την κίνηση του οχήματος επιλέχθηκε αρχιτεκτονική διαφορικής οδήγησης, καθώς προσφέρει απλή μηχανική κατασκευή, περιορισμένο αριθμό ενεργών κινητήριων μερών και δυνατότητα ελέγχου της κατεύθυνσης μέσω της σχετικής ταχύτητας των δύο πλευρικών τροχών [13], [24]. Η επιλογή αυτή ήταν κατάλληλη για την παρούσα εφαρμογή, αφού το όχημα είχε ως κύριο στόχο τη δημιουργία μιας κινητής βάσης πάνω στην οποία θα μπορούσε να τοποθετηθεί και να ελεγχθεί η πλατφόρμα προσγείωσης.

Εικόνα 4.6 Πρωτότυπα και αποτυχημένες δοκιμές.

Η κατασκευή του οχήματος οργανώθηκε σε διαδοχικά στάδια, τα οποία περιλάμβαναν την επιλογή των εξαρτημάτων, την τρισδιάστατη σχεδίαση του σασί και των βάσεων (Εικόνα 4.5), την κατασκευή των μηχανικών μερών μέσω τρισδιάστατης εκτύπωσης, τη συναρμολόγηση και την ηλεκτρολογική διασύνδεση των υποσυστημάτων. Με αυτόν τον τρόπο, η ανάπτυξη του οχήματος πραγματοποιήθηκε σταδιακά, επιτρέποντας διορθώσεις και προσαρμογές όπου αυτό κρίθηκε απαραίτητο (Εικόνα 4.6).

4.2.1 Επιλογή Υλικών και Εξαρτημάτων

Η επιλογή των υλικών και των εξαρτημάτων του οχήματος πραγματοποιήθηκε με βάση τις λειτουργικές απαιτήσεις της εφαρμογής. Το όχημα έπρεπε να είναι αρκετά σταθερό ώστε να υποστηρίζει την πλατφόρμα προσγείωσης, αρκετά απλό ώστε να μπορεί να συναρμολογηθεί και να ελεγχθεί αξιόπιστα, και αρκετά ευέλικτο ώστε να δέχεται τροποποιήσεις κατά τη διάρκεια των δοκιμών.

Ως κεντρική μονάδα ελέγχου επιλέχθηκε το Arduino Uno R4 WiFi (Εικόνα 4.7), το οποίο χρησιμοποιήθηκε για τον συντονισμό της λειτουργίας του οχήματος, την επεξεργασία των δεδομένων από τον αισθητήρα κλίσης και την παραγωγή των σημάτων ελέγχου προς τους ενεργοποιητές. Η ύπαρξη ενσωματωμένης ασύρματης επικοινωνίας

επέτρεψε επίσης την ανάπτυξη περιβάλλοντος χειρισμού και παρακολούθησης χωρίς την ανάγκη συνεχούς ενσύρματης σύνδεσης με υπολογιστή.

Για την οδήγηση των σερβοκινητήρων χρησιμοποιήθηκε ο οδηγός PCA9685 (Εικόνα 4.8), ο οποίος επιτρέπει τον έλεγχο πολλαπλών καναλιών PWM με σταθερό και επαναλήψιμο τρόπο. Η χρήση του κρίθηκε απαραίτητη, καθώς το σύστημα περιλαμβάνει τόσο σερβοκινητήρες κίνησης όσο και σερβοκινητήρες για την πλατφόρμα, οπότε απαιτήθηκε οργανωμένη και ανεξάρτητη παραγωγή σημάτων ελέγχου.

Για τη μέτρηση της κλίσης της πλατφόρμας χρησιμοποιήθηκε αισθητήρας MPU6050 (Εικόνα 4.9), ο οποίος συνδυάζει επιταχυνσιόμετρο και γυροσκόπιο. Τα δεδομένα του αξιοποιήθηκαν για την εκτίμηση του προσανατολισμού της πλατφόρμας και για την υλοποίηση λειτουργίας αυτόματης εξισορρόπησης. Μέσω των μετρήσεων του αισθητήρα, το Arduino μπορεί να υπολογίζει την απόκλιση της πλατφόρμας από τη θέση αναφοράς και να μεταβάλλει σταδιακά τη γωνία των σερβοκινητήρων, ώστε η επιφάνεια προσγείωσης να παραμένει όσο το δυνατόν πιο οριζόντια. Το Arduino Uno R4 WiFi, ο PCA9685 και ο MPU6050 συνδέθηκαν στο ίδιο δίαυλο επικοινωνίας I2C, μέσω των γραμμών SDA και SCL. Στην πράξη, η σύνδεση του MPU6050 πραγματοποιήθηκε μέσω των διαθέσιμων ακροδεκτών διέλευσης του PCA9685, καθώς το πρωτόκολλο I2C επιτρέπει τη σύνδεση περισσότερων από μία συσκευών στον ίδιο δίαυλο.

Για την κίνηση του οχήματος χρησιμοποιήθηκαν δύο σερβοκινητήρες συνεχούς περιστροφής Tower Pro MG996R, τοποθετημένοι συμμετρικά στην αριστερή και τη δεξιά πλευρά του σασί. Η διάταξη αυτή υλοποιεί τη λογική της διαφορικής οδήγησης, καθώς η κίνηση και η στροφή του οχήματος προκύπτουν από τη σχετική λειτουργία των δύο κινητήρων. Για την κίνηση της επάνω πλατφόρμας χρησιμοποιήθηκαν σερβοκινητήρες θέσης FeeTech FT5320M, οι οποίοι επιτρέπουν ελεγχόμενη μεταβολή της γωνίας κλίσης της βάσης.

Τα μηχανικά μέρη του οχήματος και της πλατφόρμας κατασκευάστηκαν κυρίως από PLA μέσω τρισδιάστατης εκτύπωσης. Το PLA επιλέχθηκε λόγω της ευκολίας εκτύπωσης, της επαρκούς ακαμψίας για τις ανάγκες του πρωτοτύπου και της δυνατότητας γρήγορης παραγωγής και αντικατάστασης εξαρτημάτων. Επιπλέον, χρησιμοποιήθηκαν τροχοί κατάλληλων διαστάσεων, στοιχεία στήριξης, καλωδίωση και βοηθητικά εξαρτήματα για τη μηχανική και ηλεκτρολογική ολοκλήρωση του συστήματος.

4.2.2 Τρισδιάστατη Σχεδίαση και Διαστασιολόγηση

Η τρισδιάστατη σχεδίαση του οχήματος προηγήθηκε της κατασκευής και βασίστηκε στη μέτρηση των πραγματικών διαστάσεων των εξαρτημάτων που θα τοποθετούνταν στο σασί. Η διαδικασία αυτή ήταν απαραίτητη, καθώς το όχημα έπρεπε να υποστηρίζει μηχανικά τους κινητήρες, τους τροχούς, την πλατφόρμα, τους σερβοκινητήρες, το Arduino, τον οδηγό PCA9685, τον αισθητήρα MPU6050 και την απαραίτητη καλωδίωση.

Κατά τη σχεδίαση δόθηκε ιδιαίτερη έμφαση στην κατανομή του διαθέσιμου χώρου. Τα ηλεκτρονικά εξαρτήματα έπρεπε να τοποθετηθούν σε σημεία που να παρέχουν προστασία και σταθερότητα, αλλά ταυτόχρονα να παραμένουν προσβάσιμα για προγραμματισμό, έλεγχο και συντήρηση. Παράλληλα, οι σερβοκινητήρες της πλατφόρμας έπρεπε να τοποθετηθούν με τρόπο που να επιτρέπει την ομαλή μεταφορά

της κίνησης στην επάνω βάση, χωρίς να δημιουργούνται μηχανικές παρεμβολές με το υπόλοιπο σασί.

Η διαστασιολόγηση του οχήματος επηρεάστηκε επίσης από την ανάγκη διατήρησης χαμηλού κέντρου βάρους. Επειδή η πλατφόρμα προσγείωσης τοποθετείται στο επάνω μέρος της κατασκευής, η βάση του οχήματος έπρεπε να παρέχει αρκετή επιφάνεια στήριξης και επαρκή πλευρική σταθερότητα. Για τον λόγο αυτό, η θέση των κινητήριων τροχών, η απόσταση μεταξύ τους και η τοποθέτηση των βαρύτερων εξαρτημάτων εξετάστηκαν με στόχο τη μείωση ανεπιθύμητων ταλαντώσεων κατά την κίνηση.

Η σχεδίαση πραγματοποιήθηκε με επαναληπτική διαδικασία. Μετά την αρχική δημιουργία του μοντέλου, πραγματοποιήθηκαν έλεγχοι συμβατότητας με τα πραγματικά εξαρτήματα και ακολούθησαν διορθώσεις στις διαστάσεις, στις θέσεις στήριξης και στα ανοίγματα διέλευσης καλωδίων. Η ανάγκη αυτών των διορθώσεων ήταν αναμενόμενη, καθώς η τρισδιάστατη σχεδίαση ενός λειτουργικού πρωτοτύπου δεν περιορίζεται στη γεωμετρική αναπαράσταση των μερών, αλλά απαιτεί συνεχή έλεγχο της πρακτικής τους συνεργασίας ως ενιαίο σύστημα.

4.2.3 Κατασκευή Μηχανικών Μερών με Τρισδιάστατη Εκτύπωση

Μετά την ολοκλήρωση της σχεδίασης, τα μηχανικά μέρη του οχήματος κατασκευάστηκαν με τη μέθοδο της τρισδιάστατης εκτύπωσης. Η επιλογή της συγκεκριμένης τεχνολογίας ήταν κατάλληλη για την ανάπτυξη του πρωτοτύπου, καθώς επέτρεψε την ταχεία παραγωγή εξαρτημάτων, την εύκολη τροποποίηση των σχεδίων και την προσαρμογή της κατασκευής στις πραγματικές ανάγκες της εφαρμογής.

Το PLA χρησιμοποιήθηκε ως βασικό υλικό εκτύπωσης για το σασί, τις βάσεις στήριξης και τα επιμέρους μηχανικά τμήματα της πλατφόρμας. Η επιλογή του βασίστηκε στην ικανοποιητική του ακαμψία, στην ευκολία εκτύπωσης και στη δυνατότητα γρήγορης επανάληψης της κατασκευής σε περίπτωση αστοχίας ή ανάγκης τροποποίησης. Για ένα πειραματικό πρωτότυπο, τα χαρακτηριστικά αυτά ήταν ιδιαίτερα σημαντικά, καθώς επέτρεψαν τη σταδιακή βελτίωση των εξαρτημάτων χωρίς μεγάλο κόστος ή καθυστέρηση.

Η κατασκευή δεν περιορίστηκε στην εκτύπωση ενός απλού πλαισίου. Περιλάμβανε επιμέρους βάσεις για τους κινητήρες, σημεία στήριξης για την πλατφόρμα, υποδοχές για ηλεκτρονικές μονάδες και βοηθητικά σημεία για τη διέλευση ή τη σταθεροποίηση καλωδίων. Μετά την εκτύπωση κάθε τμήματος πραγματοποιήθηκε δοκιμαστική τοποθέτηση, ώστε να επιβεβαιωθεί ότι οι διαστάσεις ήταν συμβατές με τα πραγματικά εξαρτήματα και ότι δεν υπήρχαν εμπόδια στην κίνηση των μηχανικών μερών.

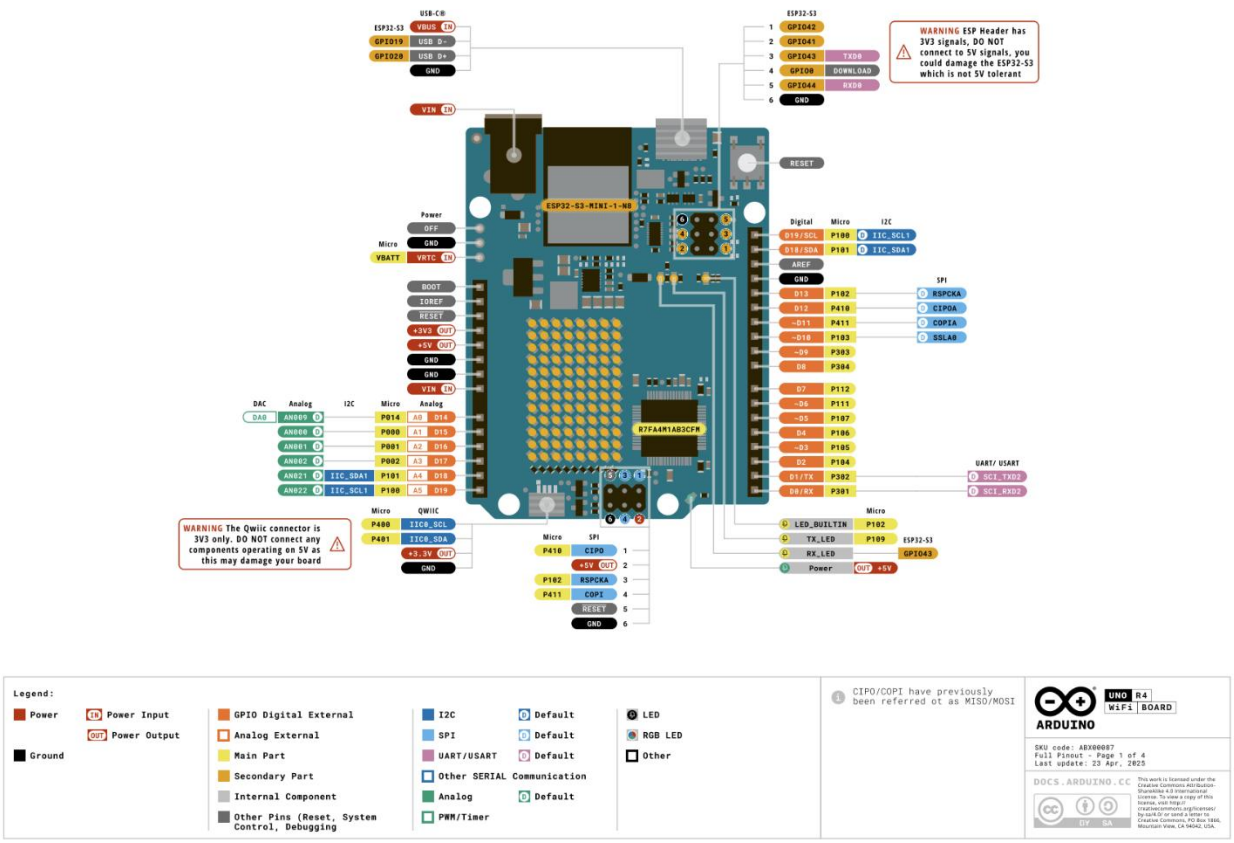
Η τρισδιάστατη εκτύπωση διευκόλυνε σημαντικά την επαναληπτική ανάπτυξη του πρωτοτύπου. Σε περιπτώσεις όπου κάποιο εξάρτημα δεν παρείχε την απαιτούμενη μηχανική σταθερότητα ή δεν επέτρεπε σωστή τοποθέτηση των υποσυστημάτων, το μοντέλο τροποποιήθηκε και εκτυπώθηκε εκ νέου. Με αυτόν τον τρόπο, η τελική κατασκευή προέκυψε μέσα από διαδοχικές βελτιώσεις, οι οποίες βασίστηκαν τόσο στις σχεδιαστικές απαιτήσεις όσο και στις πρακτικές παρατηρήσεις κατά τη συναρμολόγηση.

4.2.4 Συναρμολόγηση και Ηλεκτρολογική Διασύνδεση

Η συναρμολόγηση του οχήματος ξεκίνησε με την τοποθέτηση των βασικών μηχανικών τμημάτων του σασί και των κινητήριων σερβοκινητήρων. Οι δύο σερβοκινητήρες συνεχούς περιστροφής τοποθετήθηκαν συμμετρικά, ώστε να σχηματίζουν τη διάταξη διαφορικής οδήγησης. Στη συνέχεια τοποθετήθηκαν οι τροχοί και ελέγχθηκε η μηχανική ευθυγράμμιση, καθώς τυχόν αποκλίσεις στη θέση των τροχών μπορούν να επηρεάσουν την ομαλότητα της κίνησης.

Ακολούθησε η εγκατάσταση των σερβοκινητήρων που χρησιμοποιούνται για τη μεταβολή της κλίσης της πλατφόρμας. Η τοποθέτησή τους έγινε με τρόπο ώστε η κίνηση να μεταφέρεται στην επάνω βάση χωρίς εμπλοκές ή υπερβολικές μηχανικές καταπονήσεις. Η κινούμενη πλατφόρμα τοποθετήθηκε στο ανώτερο τμήμα του οχήματος, ώστε να λειτουργεί ως επιφάνεια προσγείωσης και ταυτόχρονα ως μηχανικό τμήμα που μπορεί να μεταβάλλει τη γωνία του.

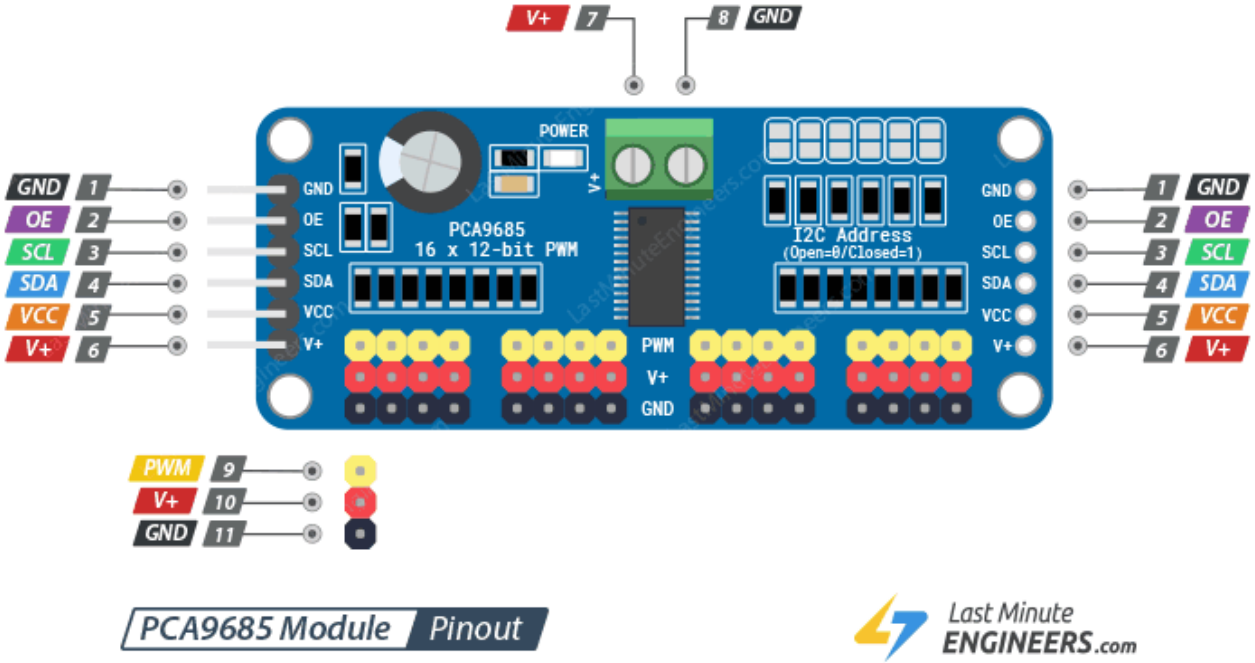
Μετά την ολοκλήρωση της μηχανικής συναρμολόγησης, τοποθετήθηκαν οι ηλεκτρονικές μονάδες. Το Arduino Uno R4 WiFi τοποθετήθηκε ως κεντρική μονάδα διαχείρισης, ο PCA9685 ως οδηγός των σερβοκινητήρων και ο MPU6050 ως αισθητήρας μέτρησης της κλίσης και της γωνιακής συμπεριφοράς της πλατφόρμας. Η διάταξη των ηλεκτρονικών έγινε με κριτήριο τη σταθερή στερέωση, την προστασία των συνδέσεων και την εύκολη πρόσβαση για προγραμματισμό και έλεγχο.



Εικόνα 4.7 Συνδεσιμότητα του Arduino Uno R4 Wi-Fi [28]

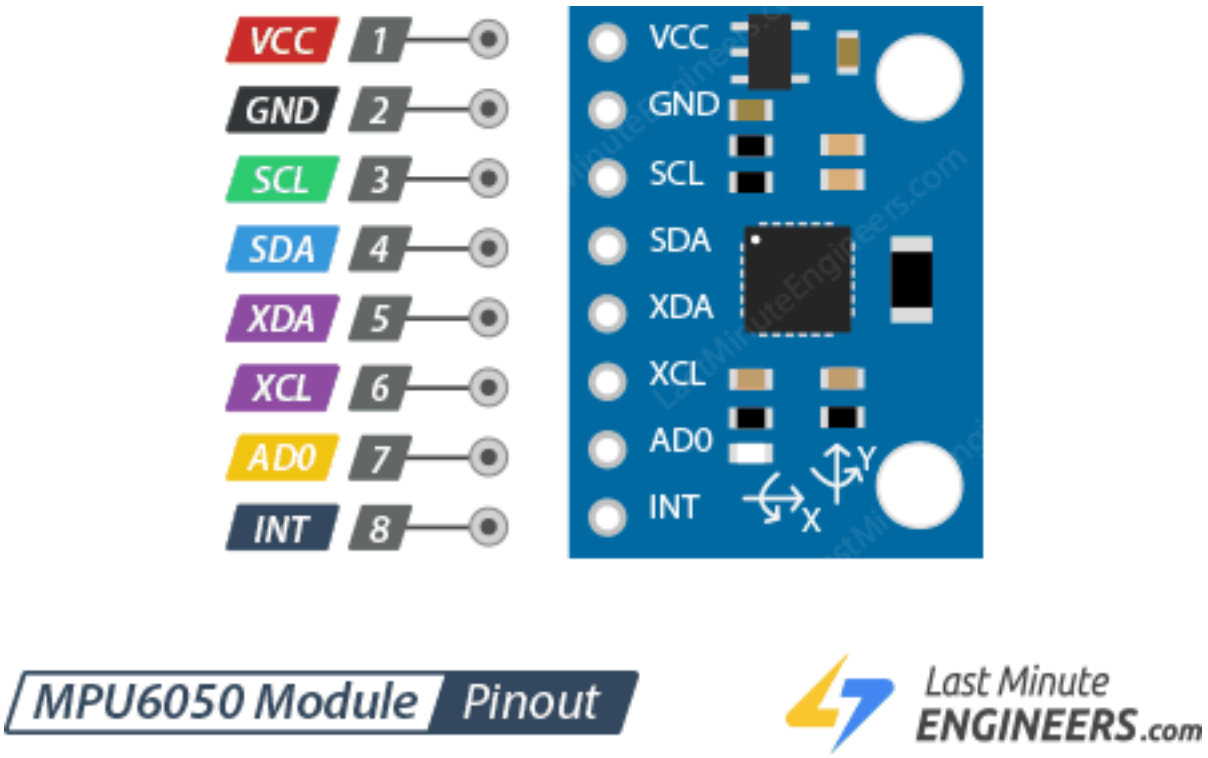
Η ηλεκτρολογική διασύνδεση βασίστηκε στον διαχωρισμό σημάτων ελέγχου και τροφοδοσίας. Το Arduino επικοινωνεί με τον PCA9685 και τον MPU6050 μέσω I2C, ενώ ο PCA9685 παράγει τα PWM σήματα για τους σερβοκινητήρες. Ιδιαίτερη προσοχή

δόθηκε στην κοινή γείωση των υποσυστημάτων, καθώς η σωστή αναφορά γείωσης είναι απαραίτητη για την αξιόπιστη λειτουργία των σημάτων ελέγχου. Παράλληλα, η καλωδίωση οργανώθηκε ώστε να μην παρεμποδίζει την κίνηση της πλατφόρμας ή των τροχών.



Εικόνα 4.8 Συνδεσιμότητα του PCA9685 [29]

Μετά τη σύνδεση των υποσυστημάτων πραγματοποιήθηκαν βασικοί έλεγχοι λειτουργίας. Ελέγχθηκε η απόκριση των κινητήριων σερβοκινητήρων στις εντολές κίνησης, η λειτουργία των σερβοκινητήρων της πλατφόρμας, η επικοινωνία μέσω I2C και η λήψη δεδομένων από τον MPU6050. Οι έλεγχοι αυτοί ήταν απαραίτητοι πριν από τη συνολική δοκιμή του οχήματος, καθώς επέτρεψαν την απομόνωση πιθανών σφαλμάτων στη σύνδεση, στην τροφοδοσία ή στη μηχανική συναρμολόγηση.



Εικόνα 4.9 Συνδεσιμότητα του MPU6050 [30]

4.2.5 Τελική Μορφή του Πρωτοτύπου

Η τελική μορφή του πρωτοτύπου αποτελείται από ένα τροχοφόρο όχημα διαφορικής οδήγησης, πάνω στο οποίο έχει ενσωματωθεί κινούμενη πλατφόρμα προσγείωσης (Εικόνα 4.10). Το κάτω τμήμα της κατασκευής περιλαμβάνει το σασί, τους κινητήριους σερβοκινητήρες, τους τροχούς και τις ηλεκτρονικές μονάδες ελέγχου. Το επάνω τμήμα περιλαμβάνει την πλατφόρμα και τους σερβοκινητήρες θέσης που επιτρέπουν τη μεταβολή της κλίσης της.

Εικόνα 4.10 Τελική μορφή οχήματος εδάφους.

Η διάταξη των εξαρτημάτων επιλέχθηκε ώστε να εξασφαλίζεται ισορροπία μεταξύ μηχανικής σταθερότητας και λειτουργικής προσβασιμότητας. Τα βασικά ηλεκτρονικά μέρη παραμένουν προσβάσιμα για επαναπρογραμματισμό, έλεγχο καλωδιώσεων και συντήρηση, ενώ η πλατφόρμα έχει τοποθετηθεί στο άνω μέρος ώστε να λειτουργεί ως εμφανής και αξιοποιήσιμη επιφάνεια για τη διαδικασία προσγείωσης.

Το τελικό πρωτότυπο επιτρέπει τον έλεγχο της κίνησης του οχήματος, τον χειροκίνητο έλεγχο της γωνίας της πλατφόρμας, καθώς και τη λειτουργία αυτόματης εξισορρόπησης με χρήση των μετρήσεων του MPU6050. Ο χρήστης μπορεί να ορίσει τη θέση αναφοράς της πλατφόρμας μέσω της διαδικασίας μηδενισμού και στη συνέχεια να ενεργοποιήσει την αυτόματη λειτουργία, κατά την οποία το Arduino διορθώνει σταδιακά τη θέση των σερβοκινητήρων.

Συνολικά, η υλοποίηση του οχήματος εδάφους ανέδειξε τη σημασία της συνδυασμένης μηχανικής, ηλεκτρονικής και προγραμματιστικής σχεδίασης. Η χρήση τρισδιάστατης εκτύπωσης επέτρεψε την προσαρμογή της κατασκευής στις ανάγκες της εφαρμογής, ενώ η χρήση μικροελεγκτή, αισθητήρα αδρανειακής μέτρησης και σερβοκινητήρων επέτρεψε τη δημιουργία μιας λειτουργικής κινητής πλατφόρμας προσγείωσης.

5. Προγραμματισμός

5.1 Λειτουργίες Τετρακοπτέρου

Το λογισμικό του τετρακοπτέρου αποτελείται από τρία βασικά επίπεδα λειτουργίας: το firmware του ελεγκτή πτήσης, το firmware των ηλεκτρονικών ρυθμιστών ταχύτητας και το λογισμικό επικοινωνίας και λήψης εικόνας που εκτελείται στις πλακέτες ESP32 και ESP32-CAM. Ο συνδυασμός αυτών των επιπέδων επιτρέπει τη σταθεροποίηση του UAV, την επικοινωνία με τον υπολογιστή και την υποστήριξη της διαδικασίας αναγνώρισης της πλατφόρμας προσγείωσης.

Στον ελεγκτή πτήσης εγκαταστάθηκε προσαρμοσμένη έκδοση του ArduPilot, με ενεργοποιημένες τις λειτουργίες που ήταν απαραίτητες για την παρούσα εργασία [19] [23]. Ιδιαίτερη σημασία είχε η υποστήριξη του Precision Landing, καθώς η λειτουργία αυτή επιτρέπει στον ελεγκτή πτήσης να αξιοποιεί δεδομένα θέσης στόχου που προέρχονται από εξωτερικό σύστημα μηχανικής όρασης. Με τον τρόπο αυτό, το τετρακόπτερο μπορεί να λαμβάνει πληροφορίες σχετικά με τη θέση της πλατφόρμας και να τις χρησιμοποιεί κατά τη διαδικασία προσέγγισης και προσγείωσης.

Η μονάδα GNSS/compass αξιοποιείται για την υποστήριξη λειτουργιών πτήσης που απαιτούν εκτίμηση θέσης και μαγνητικής διεύθυνσης στο παγκόσμιο σύστημα αναφοράς, όπως η σταθερή αιώρηση και η πιο ελεγχόμενη μετακίνηση του UAV. Η τελική ευθυγράμμιση με την πλατφόρμα, ωστόσο, εξακολουθεί να βασίζεται στο σύστημα μηχανικής όρασης και στα δεδομένα των ArUco markers.

Παράλληλα, χρησιμοποιήθηκε πλακέτα ESP32 DevKit V1 για την ασύρματη επικοινωνία μεταξύ του ελεγκτή πτήσης και του υπολογιστή. Στην πλακέτα αυτή εγκαταστάθηκε DroneBridge for ESP32, το οποίο ρυθμίστηκε σε λειτουργία σημείου πρόσβασης (Access Point - AP). Μέσω αυτού του δικτύου συνδέονται ο υπολογιστής και η ESP32-CAM, επιτρέποντας την ανταλλαγή δεδομένων και την απομακρυσμένη παρακολούθηση της λειτουργίας του συστήματος. Η ESP32-CAM χρησιμοποιείται αποκλειστικά για τη λήψη εικόνας από την πλατφόρμα προσγείωσης, ώστε να μειώνεται το υπολογιστικό φορτίο και να διαχωρίζεται η λειτουργία της εικόνας από τη λειτουργία τηλεμετρίας και επικοινωνίας.

Η συνολική αρχιτεκτονική του τετρακοπτέρου βασίζεται επομένως στον διαχωρισμό των επιμέρους λειτουργιών. Ο ελεγκτής πτήσης είναι υπεύθυνος για τη σταθεροποίηση και τον έλεγχο πτήσης, η πλακέτα ESP32 λειτουργεί ως γέφυρα επικοινωνίας μέσω Wi-Fi και η ESP32-CAM παρέχει τις εικόνες που απαιτούνται για την επεξεργασία από τον αλγόριθμο μηχανικής όρασης. Η λεπτομερής λειτουργία του αλγορίθμου αναγνώρισης ArUco και της αποστολής δεδομένων προσγείωσης αναλύεται στην ενότητα 5.3.

5.1.1 Προγραμματισμός Τετρακοπτέρου

Η προετοιμασία του τετρακοπτέρου ξεκίνησε από τη ρύθμιση των ηλεκτρονικών ρυθμιστών ταχύτητας. Στα ESC εγκαταστάθηκε το firmware BlueJay μέσω του ESC Configurator, ώστε να επιτευχθεί σταθερή λειτουργία και συμβατότητα με το πρωτόκολλο ελέγχου που χρησιμοποιήθηκε στο σύστημα [25], [26]. Η επικοινωνία μεταξύ του ελεγκτή πτήσης και των ESC πραγματοποιήθηκε με χρήση του

πρωτοκόλλου DShot600, το οποίο επιτρέπει ψηφιακή μετάδοση εντολών προς τους κινητήρες.

Στη συνέχεια πραγματοποιήθηκε η παραμετροποίηση του ελεγκτή πτήσης μέσω του ArduPilot Methodic Configurator [18]. Το εργαλείο αυτό χρησιμοποιήθηκε για τη σταδιακή εισαγωγή των βασικών χαρακτηριστικών του UAV και για τη δημιουργία ενός αρχικού συνόλου παραμέτρων λειτουργίας [22], [23]. Στη διαδικασία αυτή δηλώθηκαν η κατηγορία πλαισίου Quad, η διάταξη X, τα χαρακτηριστικά της μπαταρίας, το συνολικό βάρος του UAV, οι έξοδοι σύνδεσης των ESC, το πρωτόκολλο DShot600, η σειριακή θύρα επικοινωνίας με την ESP32, το μέγεθος των προπελών, η συνδεσιμότητα της μονάδας GNSS/compass και τα βασικά στοιχεία των κινητήρων. Συγκεκριμένα, χρησιμοποιήθηκαν οι έξοδοι S3, S4, S5 και S6 για τα ESC, η επικοινωνία με την ESP32 πραγματοποιήθηκε μέσω RX5/TX5, η μπαταρία δηλώθηκε ως 3S 2200 mAh στα 11,1 V, οι προπέλες ως 1045 και οι κινητήρες με 14 μαγνητικούς πόλους.

Μέσω της ίδιας διαδικασίας ρυθμίστηκαν επίσης παράμετροι που επηρεάζουν άμεσα τη συμπεριφορά πτήσης, όπως η εκτίμηση της ώσης αιώρησης και το φίλτρο Harmonic Notch. Η παράμετρος ώσης αιώρησης είναι σημαντική, επειδή σχετίζεται με το ποσοστό ισχύος που απαιτείται ώστε το UAV να διατηρεί σταθερό ύψος. Αντίστοιχα, το φίλτρο Harmonic Notch χρησιμοποιείται για τη μείωση των κραδασμών που προέρχονται κυρίως από τους κινητήρες και τις προπέλες, βελτιώνοντας την ποιότητα των μετρήσεων της αδρανειακής μονάδας του ελεγκτή πτήσης.

Μετά την αρχική παραμετροποίηση πραγματοποιήθηκαν ελεγχόμενες δοκιμές λειτουργίας, με στόχο την επιβεβαίωση της ορθής φοράς περιστροφής των κινητήρων, της σωστής αντιστοίχισης των εξόδων του ελεγκτή πτήσης και της συνολικής απόκρισης του UAV. Οι πρώτες δοκιμές έγιναν σε περιορισμένο και ελεγχόμενο περιβάλλον, ώστε να μειωθεί ο κίνδυνος αστοχίας κατά την αρχική ενεργοποίηση των κινητήρων. Μετά τις απαραίτητες διορθώσεις και την επιβεβαίωση των βασικών ρυθμίσεων, το UAV ήταν έτοιμο για τις πρώτες πτητικές δοκιμές.

Κατά τις πρώτες πτήσεις συλλέχθηκαν δεδομένα λειτουργίας, τα οποία χρησιμοποιήθηκαν για περαιτέρω ρύθμιση του συστήματος. Η διαδικασία αυτή ήταν απαραίτητη, καθώς η θεωρητική παραμετροποίηση δεν επαρκεί από μόνη της για τη βέλτιστη συμπεριφορά ενός πραγματικού τετρακοπτέρου. Παράγοντες όπως το πραγματικό βάρος, οι κραδασμοί, η απόκριση των κινητήρων, η κατανομή μάζας και η ποιότητα των μετρήσεων των αισθητήρων επηρεάζουν τη σταθερότητα της πτήσης και απαιτούν πρακτική επαλήθευση.

Εφόσον ο στόχος της παρούσας εργασίας δεν ήταν η πλήρης ανάπτυξη ενός νέου ελεγκτή πτήσης, αλλά η δημιουργία πλατφόρμας ικανής να συνεργαστεί με σύστημα μηχανικής όρασης για διαδικασία προσγείωσης, η παραμετροποίηση περιορίστηκε στις λειτουργίες που ήταν αναγκαίες για τις δοκιμές. Μετά την ολοκλήρωση των βασικών ρυθμίσεων, το UAV μπορούσε να χρησιμοποιηθεί ως φυσικό πρωτότυπο για τη μελέτη της επικοινωνίας, της λήψης εικόνας και της λογικής αυτόνομης προσγείωσης.

5.1.2 Προγραμματισμός ESP32 και ESP32-CAM

Το υποσύστημα επικοινωνίας και λήψης εικόνας υλοποιήθηκε με δύο ξεχωριστές μονάδες: μία πλακέτα ESP32 DevKit V1 και μία ESP32-CAM με αισθητήρα εικόνας OV2640. Ο διαχωρισμός αυτός επιλέχθηκε ώστε να μειωθεί το φορτίο της ESP32-CAM και να επιτευχθεί πιο σταθερή λειτουργία. Η ESP32 DevKit V1 χρησιμοποιήθηκε για την

ασύρματη επικοινωνία και τη μεταφορά δεδομένων μεταξύ του υπολογιστή και του ελεγκτή πτήσης, ενώ η ESP32-CAM χρησιμοποιήθηκε για τη λήψη εικόνων της πλατφόρμας προσγείωσης [59], [61].

Στην ESP32 DevKit V1 εγκαταστάθηκε το DroneBridge for ESP32 και η συσκευή ρυθμίστηκε σε λειτουργία Access Point [27]. Με αυτόν τον τρόπο δημιουργήθηκε ασύρματο δίκτυο στο οποίο μπορούσαν να συνδεθούν ο υπολογιστής και η ESP32-CAM. Η σειριακή επικοινωνία με τον ελεγκτή πτήσης πραγματοποιήθηκε μέσω των ακροδεκτών RX2 και TX2, οι οποίοι χρησιμοποιήθηκαν για τη λειτουργία serial-to-Wi-Fi. Η σύνδεση αυτή επέτρεψε τη μεταφορά δεδομένων MAVLink μεταξύ του ελεγκτή πτήσης και του υπολογιστή [23].

Στην ESP32-CAM εγκαταστάθηκε πρόγραμμα σε C++, το οποίο συνδέει την κάμερα στο ασύρματο δίκτυο που δημιουργεί η ESP32 DevKit V1 και ενεργοποιεί τις απαραίτητες HTTP λειτουργίες. Συγκεκριμένα, υλοποιήθηκαν δύο βασικά endpoints: το /health, το οποίο χρησιμοποιείται για τον έλεγχο της κατάστασης της κάμερας, και το /capture.jpg, το οποίο χρησιμοποιείται για τη λήψη νέας εικόνας. Τα δύο αυτά endpoints αξιοποιούνται από το Python script που εκτελείται στον υπολογιστή, ώστε να ελέγχει αν η κάμερα είναι διαθέσιμη και να ζητά εικόνες για επεξεργασία.

Η επιλογή λήψης εικόνας μέσω αιτήματος από τον υπολογιστή, αντί για συνεχή μετάδοση βίντεο, έγινε ώστε να απλοποιηθεί η επεξεργασία και να περιοριστεί η χρήση του ασύρματου δικτύου. Με αυτήν την προσέγγιση, ο αλγόριθμος μηχανικής όρασης ζητά νέα εικόνα μόνο όταν είναι έτοιμος να την επεξεργαστεί, μειώνοντας την πιθανότητα καθυστερήσεων ή απώλειας δεδομένων. Η αρχιτεκτονική αυτή ταιριάζει στη λογική της παρούσας εργασίας, όπου η κύρια απαίτηση είναι η περιοδική εκτίμηση της σχετικής θέσης του στόχου προσγείωσης και όχι η συνεχής μετάδοση υψηλής ανάλυσης βίντεο.

Συνολικά, ο προγραμματισμός των μονάδων ESP32 και ESP32-CAM επέτρεψε τη δημιουργία ενός απλού αλλά λειτουργικού συστήματος επικοινωνίας και λήψης εικόνας. Η ESP32 λειτουργεί ως γέφυρα μεταξύ του υπολογιστή και του ελεγκτή πτήσης, ενώ η ESP32-CAM παρέχει τις εικόνες που απαιτούνται για την ανίχνευση των ArUco markers. Με αυτόν τον τρόπο, το τετρακόπτερο αποκτά τη δυνατότητα συνεργασίας με εξωτερικό υπολογιστικό σύστημα, το οποίο αναλαμβάνει την επεξεργασία εικόνας και την παραγωγή των κατάλληλων εντολών καθοδήγησης.

5.2 Προγραμματισμός Οχήματος Εδάφους

Ο προγραμματισμός του οχήματος εδάφους πραγματοποιήθηκε σε γλώσσα C++ στο περιβάλλον του Arduino IDE. Ο κώδικας εγκαταστάθηκε στο Arduino Uno R4 WiFi, το οποίο λειτουργεί ως κεντρική μονάδα ελέγχου του επίγειου συστήματος. Η λειτουργία του προγράμματος περιλαμβάνει την ασύρματη σύνδεση του οχήματος σε δίκτυο Wi-Fi, τη δημιουργία τοπικής ιστοσελίδας χειρισμού, την παραγωγή σημάτων PWM για τους σερβοκινητήρες και την επεξεργασία των μετρήσεων του αισθητήρα MPU6050.

Η επικοινωνία με τον χρήστη πραγματοποιείται μέσω ενσωματωμένου web interface, το οποίο φιλοξενείται απευθείας από το Arduino. Μέσω της ιστοσελίδας ο χρήστης μπορεί να ελέγχει την κίνηση του οχήματος, δίνοντας εντολές για εμπρόσθια κίνηση, οπίσθια κίνηση, στροφή αριστερά, στροφή δεξιά και ακινητοποίηση. Οι εντολές αυτές αντιστοιχούν σε HTTP διαδρομές, όπως /f, /b, /l, /r και /s, και μετατρέπονται σε κατάλληλους παλμούς PWM προς τους δύο σερβοκινητήρες συνεχούς περιστροφής.

Για λόγους ασφαλείας, κατά την αρχική φόρτωση της ιστοσελίδας αποστέλλεται εντολή παύσης, ώστε το όχημα να ξεκινά από κατάσταση ακινησίας.

Η παραγωγή των σημάτων ελέγχου προς τους σερβοκινητήρες πραγματοποιείται μέσω του οδηγού PCA9685. Στον κώδικα ορίζονται ξεχωριστά κανάλια για τον αριστερό και τον δεξιό τροχό, καθώς και για τους δύο σερβοκινητήρες της πλατφόρμας. Συγκεκριμένα, οι κινητήριοι σερβοκινητήρες συνδέονται στα κανάλια 0 και 15, ενώ οι σερβοκινητήρες της πλατφόρμας στα κανάλια 4 και 11. Για τους κινητήριους σερβοκινητήρες χρησιμοποιείται ουδέτερη τιμή παλμού 1500 μs, η οποία αντιστοιχεί σε στάση, ενώ μεγαλύτερες ή μικρότερες τιμές προκαλούν περιστροφή προς αντίθετες κατευθύνσεις. Με αυτόν τον τρόπο υλοποιούνται οι βασικές κινήσεις του οχήματος μέσω κατάλληλου συνδυασμού παλμών στον αριστερό και τον δεξιό τροχό.

Το Arduino επικοινωνεί με τον PCA9685 και τον MPU6050 μέσω του διαύλου I2C. Ο PCA9685 χρησιμοποιείται για την παραγωγή των PWM σημάτων, ενώ ο MPU6050 παρέχει μετρήσεις από το επιταχυνσιόμετρο και το γυροσκόπιο. Στον κώδικα, ο αισθητήρας αρχικοποιείται με εύρος επιταχυνσιομέτρου ±8g και εύρος γυροσκοπίου 500 deg/s. Τα δεδομένα του αισθητήρα χρησιμοποιούνται για την εκτίμηση της κλίσης της πλατφόρμας, η οποία εμφανίζεται στην ιστοσελίδα ελέγχου. Με αυτόν τον τρόπο ο χρήστης μπορεί να παρακολουθεί την κατάσταση της πλατφόρμας κατά τη διάρκεια των δοκιμών.

Η εκτίμηση της κλίσης βασίζεται στον συνδυασμό των μετρήσεων του γυροσκοπίου και του επιταχυνσιομέτρου. Ο κώδικας ολοκληρώνει τις γωνιακές ταχύτητες του γυροσκοπίου για την εκτίμηση των γωνιών pitch και roll, ενώ παράλληλα υπολογίζει τις αντίστοιχες γωνίες από το επιταχυνσιόμετρο. Στη συνέχεια εφαρμόζεται συμπληρωματικό φίλτρο, ώστε η γρήγορη απόκριση του γυροσκοπίου να συνδυάζεται με τη μακροπρόθεσμη σταθερότητα του επιταχυνσιομέτρου. Το αποτέλεσμα εξομαλύνεται επιπλέον πριν εμφανιστεί στο web interface, ώστε η ένδειξη της κλίσης να είναι πιο σταθερή και ευανάγνωστη.

Για τον έλεγχο της πλατφόρμας χρησιμοποιούνται δύο σερβοκινητήρες θέσης, οι οποίοι λειτουργούν κατοπτρικά μέσα σε προκαθορισμένο εύρος λειτουργίας. Η επιθυμητή γωνία περιορίζεται από τον κώδικα στο διάστημα 0-35 μοιρών, ενώ ο δεύτερος σερβοκινητήρας λαμβάνει συμπληρωματική τιμή ως προς το ίδιο εύρος, δηλαδή $35 - \theta$. Η εντολή γωνίας αποστέλλεται μέσω της διαδρομής /Platform?deg=..., είτε από αριθμητικό πεδίο είτε από slider της ιστοσελίδας. Όταν ο χρήστης δώσει χειροκίνητη εντολή γωνίας, η αυτόματη εξισορρόπηση απενεργοποιείται, ώστε ο χειροκίνητος έλεγχος να έχει προτεραιότητα έναντι του αυτόματου ελέγχου.

Στην τελική έκδοση του κώδικα προστέθηκε λειτουργία αυτόματης εξισορρόπησης της πλατφόρμας. Η λειτουργία αυτή βασίζεται στις μετρήσεις του MPU6050 και ενεργοποιείται από τον χρήστη μέσω του web interface. Αρχικά, ο χρήστης εκτελεί τη διαδικασία ZERO NOW, με την οποία η τρέχουσα γωνία της πλατφόρμας αποθηκεύεται ως θέση αναφοράς. Στη συνέχεια, με την επιλογή ENABLE AUTO LEVEL, ο κώδικας συγκρίνει περιοδικά την τρέχουσα κλίση με τη θέση αναφοράς και εφαρμόζει μικρές διορθώσεις στη γωνία της πλατφόρμας. Η διόρθωση γίνεται αθροιστικά, με συντελεστή ενίσχυσης και μέγιστο επιτρεπτό βήμα ανά κύκλο, ώστε η κίνηση να είναι ομαλή και να αποφεύγονται απότομες μεταβολές.

Ιδιαίτερη σημασία δόθηκε στην ομαλή κίνηση της πλατφόρμας. Αντί η επιθυμητή γωνία να εφαρμόζεται απότομα, ο κώδικας μεταβάλλει σταδιακά τη θέση των σερβοκινητήρων μέχρι να φτάσει στην τελική τιμή. Η μετάβαση γίνεται με βήμα μίας μοίρας και μικρή

χρονική καθυστέρηση μεταξύ των βημάτων, ώστε η κίνηση να είναι πιο ομαλή. Η προσέγγιση αυτή μειώνει τις απότομες μηχανικές φορτίσεις, προστατεύει τους σερβοκινητήρες και κάνει τη συμπεριφορά της πλατφόρμας πιο σταθερή.

Για την αποφυγή συνεχών μικροκινήσεων λόγω θορύβου του αισθητήρα, εφαρμόζεται νεκρή ζώνη σφάλματος. Όταν η απόκλιση της πλατφόρμας από τη θέση αναφοράς βρίσκεται εντός του επιτρεπτού ορίου, ο κώδικας διατηρεί την τρέχουσα θέση των σερβοκινητήρων χωρίς νέα διόρθωση. Όταν η απόκλιση υπερβεί το όριο αυτό, η επιθυμητή γωνία μεταβάλλεται σταδιακά και περιορίζεται στο επιτρεπτό εύρος 0-15 μοιρών. Η συνάρτηση αυτόματης εξισορρόπησης εκτελείται συνεχώς μέσα στο βασικό loop() του Arduino, μαζί με την ανάγνωση του αισθητήρα και την ομαλή ενημέρωση της θέσης της πλατφόρμας.

Ο κώδικας περιλαμβάνει επίσης διαδικασίες βαθμονόμησης και μηδενισμού του αισθητήρα κλίσης. Η βαθμονόμηση χρησιμοποιείται για τον υπολογισμό των offsets του γυροσκοπίου, ώστε να μειωθεί η σταδιακή απόκλιση που εμφανίζεται κατά την ολοκλήρωση των γωνιακών ταχυτήτων. Ο μηδενισμός επιτρέπει στον χρήστη να ορίσει την τρέχουσα θέση της πλατφόρμας ως θέση αναφοράς για την αυτόματη εξισορρόπηση. Στην παρούσα έκδοση, οι τιμές αυτές χρησιμοποιούνται κατά τη διάρκεια λειτουργίας του συστήματος και επαναϋπολογίζονται όταν ο χρήστης εκτελέσει εκ νέου τις αντίστοιχες διαδικασίες.

Κατά την εκκίνηση του προγράμματος πραγματοποιείται αρχικοποίηση των βασικών υποσυστημάτων. Ενεργοποιείται η σειριακή επικοινωνία, αρχικοποιείται ο δίαυλος I2C, ρυθμίζεται ο PCA9685 στη συχνότητα λειτουργίας των σερβοκινητήρων, ακινητοποιούνται οι τροχοί και τοποθετείται η πλατφόρμα στην αρχική της θέση. Στη συνέχεια αρχικοποιείται ο MPU6050, μηδενίζεται η εσωτερική κατάσταση του φίλτρου εκτίμησης και λαμβάνονται αρχικές μετρήσεις ώστε να σταθεροποιηθεί η ένδειξη της κλίσης. Οι τιμές βαθμονόμησης και μηδενισμού δεν φορτώνονται από μόνιμη μνήμη, αλλά ορίζονται κατά τη λειτουργία του συστήματος μέσω των επιλογών CALIBRATE και ZERO NOW. Μετά την ολοκλήρωση της αρχικοποίησης ξεκινά η λειτουργία του web server. Κατά τη διάρκεια της εκκίνησης χρησιμοποιείται η ενσωματωμένη LED matrix του Arduino Uno R4 WiFi ως ένδειξη ότι το σύστημα βρίσκεται σε διαδικασία προετοιμασίας.

Η ιστοσελίδα του οχήματος περιλαμβάνει τα βασικά στοιχεία χειρισμού του συστήματος: κουμπιά κίνησης του οχήματος, χειροκίνητο έλεγχο της γωνίας της πλατφόρμας μέσω αριθμητικού πεδίου και slider, καθώς και ξεχωριστή περιοχή για τη λειτουργία αυτόματης εξισορρόπησης. Μέσω των επιλογών CALIBRATE και ZERO NOW ο χρήστης μπορεί να βαθμονομήσει το γυροσκόπιο και να ορίσει τη θέση αναφοράς της πλατφόρμας. Στη συνέχεια, με τα κουμπιά ENABLE AUTO LEVEL και DISABLE AUTO LEVEL μπορεί να ενεργοποιεί ή να απενεργοποιεί την αυτόματη διόρθωση. Οι αποκρίσεις των αντίστοιχων HTTP διαδρομών επιστρέφονται ως απλό κείμενο κατάστασης, ώστε ο χρήστης να γνωρίζει αν η εντολή εκτελέστηκε επιτυχώς. Έτσι, το web interface λειτουργεί τόσο ως χειριστήριο του οχήματος όσο και ως πίνακας ενεργοποίησης και ρύθμισης της πλατφόρμας.

Συνολικά, ο προγραμματισμός του οχήματος εδάφους επέτρεψε τη δημιουργία ενός λειτουργικού επίγειου υποσυστήματος, το οποίο συνδυάζει κίνηση διαφορικής οδήγησης, χειροκίνητο έλεγχο της πλατφόρμας, αυτόματη εξισορρόπηση με ανάδραση από τον MPU6050 και ασύρματη διεπαφή χειρισμού. Η υλοποίηση αυτή αποτέλεσε απαραίτητο στάδιο για τη συνεργασία του οχήματος με το τετρακόπτερο, καθώς παρέχει

μία κινητή και ενεργά ρυθμιζόμενη επιφάνεια πάνω στην οποία βασίζεται το σενάριο αυτόνομης προσγείωσης.

5.3 Ανίχνευση ArUco και Προσγείωση Τετρακοπτέρου

Η διαδικασία αυτόνομης προσέγγισης και προσγείωσης του τετρακοπτέρου βασίστηκε στη χρήση δεικτών ArUco markers, οι οποίοι τοποθετήθηκαν πάνω στην πλατφόρμα προσγείωσης [20]. Οι δείκτες αυτοί επιλέχθηκαν επειδή μπορούν να ανιχνευθούν αξιόπιστα από αλγορίθμους μηχανικής όρασης και επειδή, όταν είναι γνωστές οι πραγματικές τους διαστάσεις, μπορούν να χρησιμοποιηθούν για την εκτίμηση της σχετικής θέσης και απόστασης μεταξύ κάμερας και στόχου.

Η βασική λογική του συστήματος είναι ότι η κάμερα του τετρακοπτέρου λαμβάνει εικόνες της πλατφόρμας, οι εικόνες επεξεργάζονται σε υπολογιστή με χρήση Python και OpenCV, και στη συνέχεια η εκτιμώμενη θέση του στόχου αποστέλλεται στον ελεγκτή πτήσης μέσω MAVLink. Με αυτόν τον τρόπο, το σύστημα μηχανικής όρασης λειτουργεί ως εξωτερικός αισθητήρας θέσης, παρέχοντας στο ArduPilot τις πληροφορίες που απαιτούνται για τη λειτουργία Precision Landing.

Η χρήση GNSS/compass αφορά κυρίως τη γενική πλοήγηση και τη σταθερότητα των αυτόνομων λειτουργιών πτήσης. Η εκτίμηση της σχετικής θέσης κατά την τελική προσέγγιση της πλατφόρμας πραγματοποιείται από το σύστημα μηχανικής όρασης, μέσω των ArUco markers και των μηνυμάτων Precision Landing.

Η αρχιτεκτονική που υλοποιήθηκε αποτελείται από τέσσερα βασικά στάδια. Αρχικά, η ESP32-CAM λαμβάνει εικόνα της πλατφόρμας μέσω του endpoints /capture.jpg. Στη συνέχεια, το Python script αναζητά στην εικόνα τους γνωστούς ArUco markers και υπολογίζει τη θέση τους στο επίπεδο της εικόνας. Έπειτα, με χρήση των πραγματικών διαστάσεων και θέσεων των markers πάνω στην πλατφόρμα, υπολογίζεται η τρισδιάστατη θέση της πλατφόρμας ως προς την κάμερα. Τέλος, η θέση αυτή μετατρέπεται στο σύστημα συντεταγμένων του UAV και αποστέλλεται στον ελεγκτή πτήσης με μήνυμα LANDING_TARGET [2], [12], [15].

5.3.1 ChArUco και Calibration

Πριν από την ανίχνευση των ArUco markers για την προσγείωση, ήταν απαραίτητη η βαθμονόμηση της κάμερας. Η διαδικασία αυτή επιτρέπει τον υπολογισμό των εσωτερικών παραμέτρων της κάμερας, όπως το μητρώο κάμερας, καθώς και των συντελεστών παραμόρφωσης του φακού [20]. Οι παράμετροι αυτές είναι απαραίτητες για την ακριβέστερη εκτίμηση της θέσης του στόχου στον τρισδιάστατο χώρο, καθώς η εικόνα που λαμβάνεται από την κάμερα δεν αντιστοιχεί ιδανικά στο πραγματικό γεωμετρικό μοντέλο λόγω παραμορφώσεων του φακού.

Για τη βαθμονόμηση χρησιμοποιήθηκε πίνακας ChArUco, δηλαδή συνδυασμός σκακιέρας και ArUco markers [33]. Η χρήση ChArUco board επιλέχθηκε επειδή επιτρέπει ακριβέστερο εντοπισμό γωνιών σε σχέση με τη χρήση απλών ArUco markers, ενώ ταυτόχρονα διατηρεί τη δυνατότητα αναγνώρισης του πίνακα ακόμη και όταν δεν είναι πλήρως ορατός στην εικόνα [20]. Για τη δημιουργία του πίνακα χρησιμοποιήθηκε Python script, το οποίο παράγει εικόνα ChArUco board με διαστάσεις 7 x 5 τετραγώνων, λεξικό DICT_4X4_50, ονομαστικό μήκος τετραγώνου 0,040 m και

ονομαστικό μήκος marker 0,030 m. Το παραγόμενο αρχείο αποθηκεύεται ως charuco_board.png και χρησιμοποιείται για εκτύπωση.

Μετά την εκτύπωση του ChArUco board, οι πραγματικές διαστάσεις μετρήθηκαν εκ νέου, ώστε να ληφθούν υπόψη πιθανές αποκλίσεις που προκύπτουν από την εκτύπωση. Για τον λόγο αυτό, στα scripts λήψης και βαθμονόμησης χρησιμοποιήθηκαν οι διορθωμένες τιμές των φυσικών διαστάσεων, δηλαδή μήκος τετραγώνου 0,036 m και μήκος marker 0,027 m. Η διόρθωση αυτή είναι σημαντική, επειδή η εκτίμηση της απόστασης εξαρτάται άμεσα από τις πραγματικές διαστάσεις του μοτίβου αναφοράς.

Η λήψη των εικόνων βαθμονόμησης πραγματοποιήθηκε με ξεχωριστό Python script, το οποίο επικοινωνεί με την ESP32-CAM μέσω HTTP και ζητά εικόνες από τη διαδρομή /capture.jpg. Κάθε εικόνα μετατρέπεται σε grayscale και στη συνέχεια γίνεται ανίχνευση των ArUco markers και των ChArUco corners. Η εικόνα αποθηκεύεται μόνο όταν ανιχνευθεί επαρκής αριθμός γωνιών, συγκεκριμένα τουλάχιστον 8 ChArUco corners. Με αυτόν τον τρόπο αποφεύγεται η χρήση εικόνων που δεν παρέχουν αρκετή γεωμετρική πληροφορία για αξιόπιστη βαθμονόμηση.

Στη συνέχεια, οι αποθηκευμένες εικόνες επεξεργάζονται από το script βαθμονόμησης. Το πρόγραμμα διαβάζει όλες τις εικόνες από τον φάκελο calib_images, εντοπίζει τα ChArUco corners και κρατά μόνο τις εικόνες που διαθέτουν επαρκή αριθμό έγκυρων σημείων. Αν ο αριθμός των έγκυρων εικόνων είναι μικρότερος από 10, η διαδικασία διακόπτεται και απαιτείται η λήψη περισσότερων εικόνων από διαφορετικές γωνίες και αποστάσεις. Μετά την ολοκλήρωση της βαθμονόμησης, αποθηκεύονται το μητρώο κάμερας, οι συντελεστές παραμόρφωσης, το reprojection error και το μέγεθος της εικόνας στο αρχείο cam_calib_esp32cam.npz.

Το αρχείο αυτό χρησιμοποιείται στη συνέχεια από το κύριο πρόγραμμα ανίχνευσης ArUco, ώστε οι υπολογισμοί θέσης να βασίζονται στις πραγματικές παραμέτρους της ESP32-CAM. Η ύπαρξη σωστού calibration είναι ιδιαίτερα σημαντική, διότι μικρά σφάλματα στην εκτίμηση της κάμερας μπορούν να οδηγήσουν σε σφάλματα στην εκτίμηση της σχετικής θέσης της πλατφόρμας, ιδιαίτερα όταν το UAV βρίσκεται κοντά στον στόχο προσγείωσης.

5.3.2 ArUco Detection και Precision Landing

Μετά τη βαθμονόμηση της κάμερας, αναπτύχθηκε το κύριο Python script για την ανίχνευση της πλατφόρμας και την αποστολή δεδομένων Precision Landing στον ελεγκτή πτήσης. Το πρόγραμμα φορτώνει αρχικά το αρχείο cam_calib_esp32cam.npz, από το οποίο λαμβάνει το μητρώο κάμερας και τους συντελεστές παραμόρφωσης [20]. Στη συνέχεια ελέγχει τη διαθεσιμότητα της ESP32-CAM μέσω του endpoint /health και ανοίγει σύνδεση MAVLink προς το σύστημα DroneBridge μέσω UDP. Στην υλοποίηση που χρησιμοποιήθηκε, η σύνδεση MAVLink γίνεται με τη μορφή udpout:192.168.2.1:14550.

Η πλατφόρμα προσγείωσης περιγράφεται στον κώδικα μέσω προκαθορισμένης γεωμετρίας markers. Συγκεκριμένα, ορίζεται ένας μεγάλος marker με ID 7 και πλευρά 0,115 m, καθώς και τρεις μικρότεροι markers με IDs 20, 21 και 22 και πλευρά 0,040 m. Οι μικροί markers τοποθετούνται σε γνωστές θέσεις ως προς το επιθυμητό σημείο επαφής της κάμερας πάνω στην πλατφόρμα, ενώ ο μεγάλος marker τοποθετείται πιο πίσω και χρησιμοποιείται για ευκολότερη αναγνώριση από μεγαλύτερη απόσταση. Η χρήση πολλαπλών markers επιτρέπει στο σύστημα να εκμεταλλεύεται όσα markers

είναι ορατά κάθε στιγμή, αντί να εξαρτάται αποκλειστικά από έναν μόνο δείκτη [33], [54], [59].

Σε κάθε κύκλο λειτουργίας, το πρόγραμμα ζητά νέα εικόνα από την ESP32-CAM και τη μετατρέπει σε grayscale. Στη συνέχεια εκτελείται ανίχνευση ArUco markers με χρήση του λεξικού DICT_4X4_50. Για τη βελτίωση της ακρίβειας ανίχνευσης χρησιμοποιείται υπο-εικονοστοιχειακή βελτίωση των γωνιών των markers, ενώ οι παράμετροι adaptive thresholding ρυθμίζονται ώστε να διευκολύνεται η ανίχνευση σε διαφορετικές συνθήκες φωτισμού.

Αφού εντοπιστούν οι markers, το πρόγραμμα συγκρίνει τα IDs που ανιχνεύθηκαν με τα IDs που έχουν οριστεί στη γεωμετρία της πλατφόρμας. Για κάθε γνωστό marker δημιουργούνται αντιστοιχίες μεταξύ των πραγματικών τρισδιάστατων σημείων του marker πάνω στην πλατφόρμα και των αντίστοιχων δισδιάστατων σημείων στην εικόνα. Τα πραγματικά σημεία υπολογίζονται από το γνωστό μέγεθος του marker, τη θέση του κέντρου του ως προς το σημείο προσγείωσης και την πιθανή περιστροφή του πάνω στην πλατφόρμα. Με αυτόν τον τρόπο, το σύστημα δημιουργεί ένα ενιαίο σύνολο αντιστοιχιών από όλους τους ορατούς και γνωστούς markers.

Η εκτίμηση της θέσης της πλατφόρμας πραγματοποιείται με χρήση της συνάρτησης solvePnP, η οποία υπολογίζει τη σχετική θέση και περιστροφή της πλατφόρμας ως προς την κάμερα. Το αποτέλεσμα της διαδικασίας είναι το διάνυσμα περιστροφής rvec και το διάνυσμα μετατόπισης $tvec$. Για τη διαδικασία προσγείωσης, ιδιαίτερο ενδιαφέρον έχει το $tvec$, καθώς εκφράζει τη θέση του στόχου σε σχέση με την κάμερα. Η εκτίμηση αυτή βασίζεται στις πραγματικές διαστάσεις των markers και στις παραμέτρους βαθμονόμησης της κάμερας.

Επειδή το OpenCV και το ArduPilot χρησιμοποιούν διαφορετικά συστήματα συντεταγμένων, απαιτείται μετατροπή των αξόνων πριν από την αποστολή των δεδομένων στον ελεγκτή πτήσης. Στον κώδικα θεωρείται ότι η κάμερα είναι στραμμένη προς τα κάτω και ότι το επάνω μέρος της εικόνας αντιστοιχεί στο εμπρόσθιο μέρος του UAV. Στο σύστημα της κάμερας του OpenCV, ο άξονας x αντιστοιχεί προς τα δεξιά της εικόνας, ο άξονας y προς τα κάτω της εικόνας και ο άξονας z κατά μήκος του οπτικού άξονα. Για τη μετατροπή στο σύστημα BODY_FRD του ArduPilot, όπου x είναι εμπρός, y δεξιά και z κάτω, χρησιμοποιείται η αντιστοίχιση $x\_body = -y\_cam$, $y\_body = x\_cam$ και $z\_body = z\_cam$.

Από τη θέση του στόχου στο σύστημα BODY_FRD υπολογίζονται τα μεγέθη που απαιτούνται για το μήνυμα LANDING_TARGET [23]. Η τρισδιάστατη απόσταση υπολογίζεται ως το μέτρο του διανύσματος θέσης, ενώ οι γωνιακές αποκλίσεις υπολογίζονται από τις σχέσεις $angle\_x = atan2(y, z)$ και $angle\_y = atan2(x, z)$. Τα μεγέθη αυτά, μαζί με τις συντεταγμένες x, y, z, αποστέλλονται στον ελεγκτή πτήσης μέσω MAVLink. Το πεδίο position_valid τίθεται ίσο με 1, δηλώνοντας ότι η αποστελλόμενη θέση είναι πλήρης και έγκυρη τρισδιάστατη εκτίμηση.

Το μήνυμα LANDING_TARGET αποστέλλεται περιοδικά με χρονική περίοδο 0,08 s, δηλαδή περίπου 12,5 φορές ανά δευτερόλεπτο, εφόσον υπάρχει έγκυρη εκτίμηση της θέσης του στόχου. Παράλληλα, το πρόγραμμα στέλνει MAVLink heartbeat ως onboard controller, ώστε να δηλώνει την παρουσία του εξωτερικού υπολογιστικού συστήματος. Η διαδικασία αυτή επιτρέπει στο ArduPilot να λαμβάνει συνεχώς ενημερωμένη εκτίμηση της σχετικής θέσης της πλατφόρμας κατά τη φάση προσέγγισης και προσγείωσης.

Για λόγους ελέγχου και αποσφαλμάτωσης, το πρόγραμμα εμφανίζει παράθυρο παρακολούθησης στο οποίο σχεδιάζονται οι ανιχνευμένοι markers, τα ID που

χρησιμοποιούνται, η εκτιμώμενη θέση στο σύστημα BODY_FRD και η συνολική απόσταση από τον στόχο. Επιπλέον, όταν εντοπίζονται μόνο οι μεγάλοι markers, μόνο οι μικροί markers ή συνδυασμός αυτών, το πρόγραμμα εμφανίζει αντίστοιχη ένδειξη λειτουργίας [12], [15], [16]. Η δυνατότητα αυτή ήταν χρήσιμη κατά τη διάρκεια των δοκιμών, καθώς επέτρεψε την άμεση παρακολούθηση της συμπεριφοράς του αλγορίθμου και την επιβεβαίωση ότι η ανίχνευση και η εκτίμηση θέσης λειτουργούσαν σωστά.

Συνολικά, η διαδικασία ArUco Detection και Precision Landing επιτρέπει τη σύνδεση του συστήματος μηχανικής όρασης με τον ελεγκτή πτήσης. Η ESP32-CAM παρέχει τις εικόνες, ο υπολογιστής εκτελεί την επεξεργασία και την εκτίμηση θέσης, ενώ το ArduPilot λαμβάνει τα δεδομένα στόχου μέσω MAVLink. Με αυτόν τον τρόπο, η πλατφόρμα προσγείωσης μετατρέπεται από απλό οπτικό αντικείμενο σε μετρήσιμο στόχο στον χώρο, ο οποίος μπορεί να χρησιμοποιηθεί από το UAV για ακριβέστερη προσέγγιση και προσγείωση.

6. Προσομοίωση και Πείραμα

6.1 Όχημα Εδάφους

Η αξιολόγηση του οχήματος εδάφους πραγματοποιήθηκε μέσω φυσικών δοκιμών, καθώς το συγκεκριμένο υποσύστημα είχε σχετικά απλή μηχανική και κινηματική δομή και μπορούσε να ελεγχθεί άμεσα σε πραγματικό περιβάλλον. Σε αντίθεση με το τετρακόπτερο, όπου η προσομοίωση είναι απαραίτητη λόγω των αυξημένων κινδύνων κατά την πτήση και της πολυπλοκότητας της δυναμικής συμπεριφοράς, το όχημα εδάφους μπορούσε να δοκιμαστεί με ασφάλεια σε ελεγχόμενο χώρο [10], [24].

Η θεωρητική ανάλυση της διαφορικής οδήγησης, καθώς και η προηγούμενη περιγραφή της μηχανικής και ηλεκτρολογικής υλοποίησης, παρείχαν την απαραίτητη βάση για τη μετάβαση στο πειραματικό στάδιο. Σκοπός των δοκιμών ήταν να επιβεβαιωθεί ότι το όχημα μπορεί να κινηθεί σύμφωνα με τις εντολές του χρήστη, ότι η πλατφόρμα μπορεί να μεταβάλει την κλίση της εντός των επιτρεπτών ορίων και ότι οι μετρήσεις του αισθητήρα MPU6050 μπορούν να χρησιμοποιηθούν για την παρακολούθηση της κατάστασης της πλατφόρμας.

Στο πλαίσιο της παρούσας εργασίας, η κίνηση του οχήματος πραγματοποιήθηκε χειροκίνητα μέσω της ιστοσελίδας που δημιουργεί το Arduino Uno R4 WiFi κατά την εκκίνηση του συστήματος. Η πλατφόρμα προσγείωσης, ωστόσο, υποστηρίζει τόσο χειροκίνητο έλεγχο γωνίας όσο και λειτουργία αυτόματης εξισορρόπησης. Η επιλογή αυτή επέτρεψε την ασφαλή δοκιμή του οχήματος από τον χρήστη, ενώ παράλληλα έδωσε τη δυνατότητα αξιολόγησης της απόκρισης της πλατφόρμας όταν ο έλεγχος της γωνίας πραγματοποιείται αυτόματα με βάση τις μετρήσεις του MPU6050.

6.1.1 Πείραμα

Μετά την ολοκλήρωση της σχεδίασης, της τρισδιάστατης εκτύπωσης, της συναρμολόγησης και της ηλεκτρολογικής διασύνδεσης, πραγματοποιήθηκαν δοκιμές για την επιβεβαίωση της σωστής λειτουργίας του οχήματος εδάφους. Το πρώτο στάδιο των δοκιμών έγινε χωρίς την πλατφόρμα προσγείωσης, με στόχο τον έλεγχο της βασικής κινητικής συμπεριφοράς του οχήματος και της απόκρισης των κινητήριων σερβοκινητήρων στις εντολές του web interface.

Στο στάδιο αυτό ελέγχθηκαν οι βασικές εντολές κίνησης: εμπρόσθια κίνηση, οπίσθια κίνηση, στροφή αριστερά, στροφή δεξιά και ακινητοποίηση. Οι εντολές αποστέλλονταν μέσω της ιστοσελίδας ελέγχου και μετατρέπονταν από το Arduino σε κατάλληλους παλμούς PWM προς τους δύο σερβοκινητήρες συνεχούς περιστροφής. Η λειτουργία της ακινητοποίησης επιβεβαιώθηκε ως βασική συνθήκη ασφαλείας, καθώς το όχημα έπρεπε να ξεκινά και να επανέρχεται σε κατάσταση στάσης χωρίς ανεπιθύμητη κίνηση.

Κατά τις αρχικές δοκιμές παρατηρήθηκε μικρή απόκλιση στην ταχύτητα περιστροφής των δύο τροχών. Το φαινόμενο αυτό είναι αναμενόμενο σε σερβοκινητήρες συνεχούς περιστροφής, καθώς ακόμη και μικρές κατασκευαστικές αποκλίσεις ή μικρές διαφορές στο φορτίο μπορούν να οδηγήσουν σε διαφορετική πραγματική ταχύτητα για την ίδια τιμή παλμού. Για την αντιμετώπιση του προβλήματος πραγματοποιήθηκε πρακτική διόρθωση των τιμών ελέγχου, ώστε η κίνηση του οχήματος να γίνει πιο ομαλή και

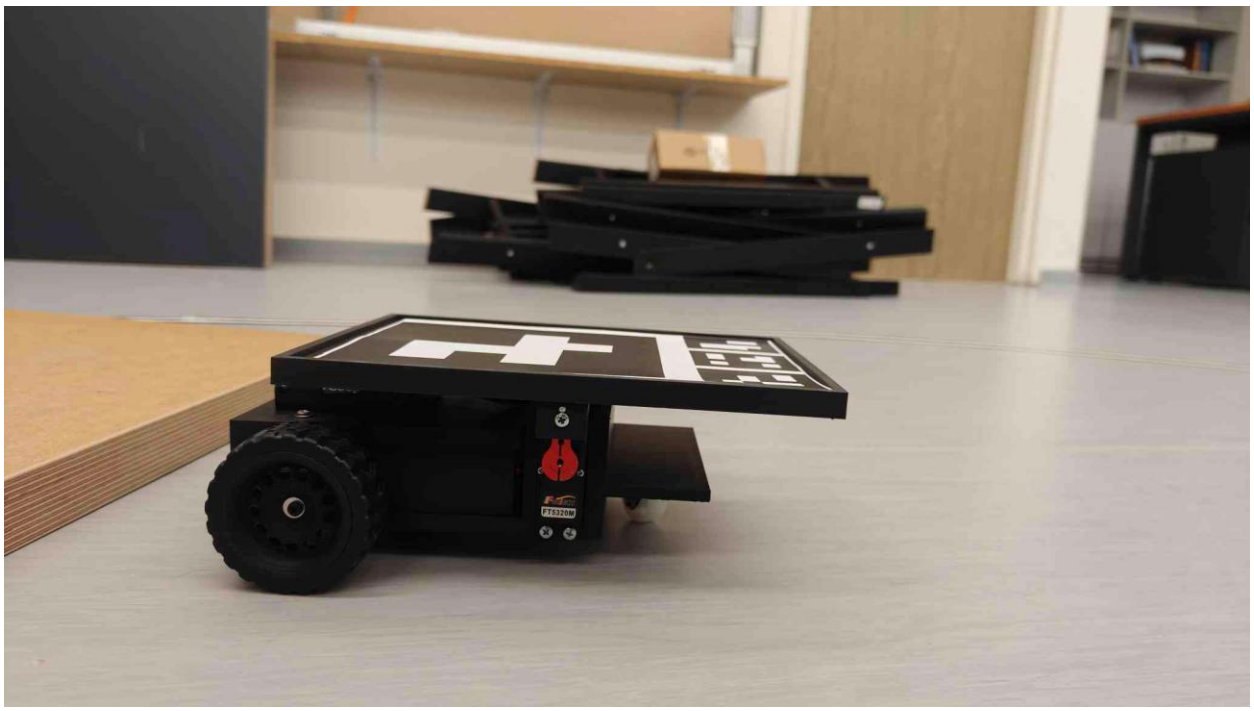
προβλέψιμη. Με αυτόν τον τρόπο βελτιώθηκε η ευθύγραμμη κίνηση και μειώθηκε η ανεπιθύμητη πλευρική απόκλιση.

Στη συνέχεια δοκιμάστηκαν διαφορετικοί τρόποι στροφής του οχήματος. Αρχικά εξετάστηκε η περίπτωση όπου οι δύο τροχοί περιστρέφονται ταυτόχρονα σε αντίθετες κατευθύνσεις, ώστε το όχημα να περιστρέφεται σχεδόν γύρω από το κέντρο του. Ωστόσο, για την τελική λειτουργία επιλέχθηκε απλούστερη προσέγγιση, κατά την οποία ο ένας τροχός παραμένει σταθερός και περιστρέφεται ο τροχός της αντίθετης πλευράς από την επιθυμητή κατεύθυνση στροφής. Δηλαδή, για αριστερή στροφή κινείται κυρίως ο δεξιός τροχός, ενώ για δεξιά στροφή κινείται κυρίως ο αριστερός. Η επιλογή αυτή μείωσε την πολυπλοκότητα του ελέγχου και περιόρισε τις απαιτήσεις ισχύος κατά τις στροφές.

Μετά την επιβεβαίωση της βασικής κίνησης, εγκαταστάθηκε η πλατφόρμα προσγείωσης πάνω στο όχημα μαζί με τον αισθητήρα MPU6050. Στο δεύτερο αυτό στάδιο οι δοκιμές έγιναν πιο σύνθετες, καθώς έπρεπε να ελεγχθεί όχι μόνο η κίνηση του οχήματος, αλλά και η λειτουργία της πλατφόρμας και η αξιοπιστία των μετρήσεων κλίσης. Ιδιαίτερη προσοχή δόθηκε στον θόρυβο των μετρήσεων, στη σταδιακή απόκλιση του γυροσκοπίου και στην καθυστέρηση που μπορούσε να προκύψει από τη συνεχή ανάγνωση και αποστολή δεδομένων προς την ιστοσελίδα ελέγχου.

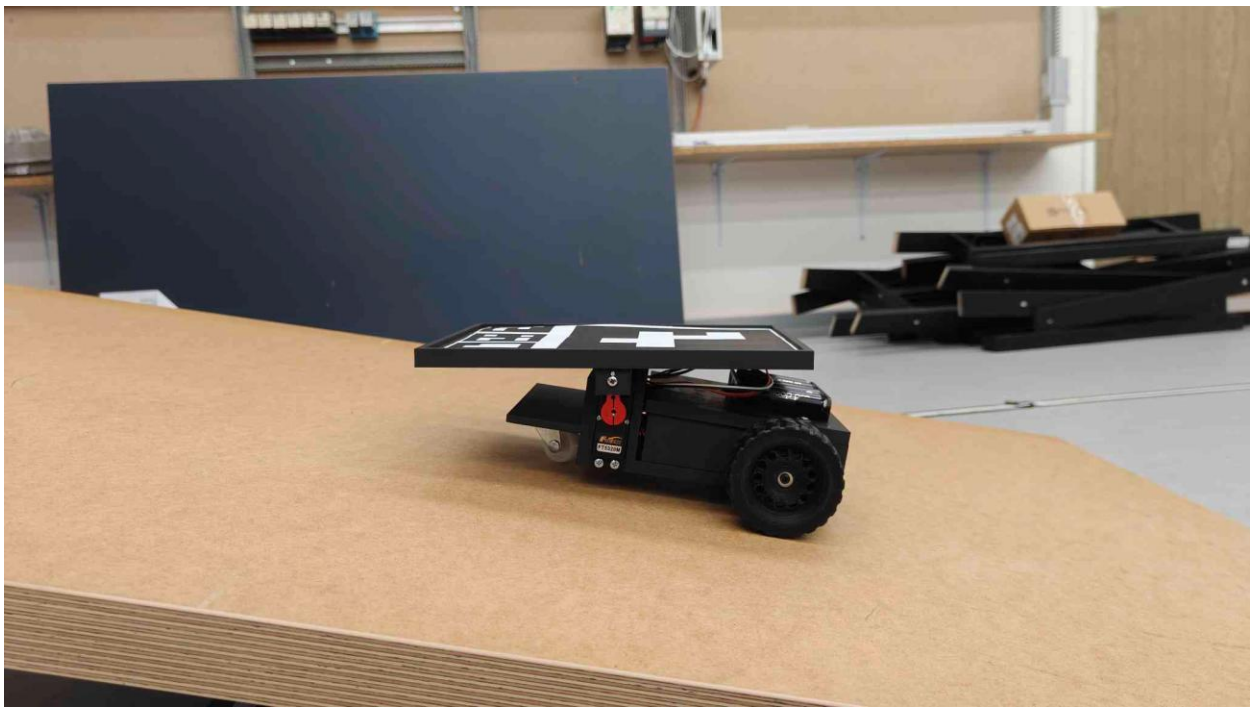
Για την παρακολούθηση της πλατφόρμας, το web interface εμφάνιζε τις μετρήσεις κλίσης και επέτρεπε στον χρήστη να μεταβάλλει τη γωνία των σερβοκινητήρων της πλατφόρμας. Η πλατφόρμα δοκιμάστηκε αρχικά σε χειροκίνητη λειτουργία, ώστε να επιβεβαιωθεί η σωστή απόκριση των σερβοκινητήρων και το επιτρεπτό εύρος κίνησης. Στη συνέχεια ενεργοποιήθηκε η λειτουργία αυτόματης εξισορρόπησης. Αφού ορίστηκε η τρέχουσα θέση ως θέση αναφοράς μέσω της διαδικασίας μηδενισμού, το σύστημα χρησιμοποιούσε τις μετρήσεις του MPU6050 για να υπολογίζει την απόκλιση της πλατφόρμας και να διορθώνει σταδιακά τη γωνία των σερβοκινητήρων. Η δοκιμή αυτή επέτρεψε την αξιολόγηση της σταθερότητας του αισθητήρα, της ομαλότητας των διορθώσεων και της ικανότητας της πλατφόρμας να επανέρχεται κοντά στην οριζόντια θέση χωρίς συνεχή χειροκίνητη παρέμβαση.

Από τις δοκιμές προέκυψε ότι το όχημα εδάφους μπορεί να λειτουργήσει ως κινητή βάση της πλατφόρμας προσγείωσης και να ελεγχθεί αξιόπιστα μέσω της ιστοσελίδας του Arduino. Παράλληλα, επιβεβαιώθηκε ότι οι μετρήσεις του MPU6050 μπορούν να αξιοποιηθούν όχι μόνο για παρακολούθηση της κλίσης, αλλά και για την αυτόματη διόρθωση της γωνίας της πλατφόρμας. Η λειτουργία αυτόματης εξισορρόπησης βελτιώνει τη χρησιμότητα της πλατφόρμας ως επιφάνειας προσγείωσης, καθώς μειώνει την ανάγκη συνεχούς χειροκίνητης ρύθμισης και επιτρέπει στο σύστημα να διατηρεί πιο σταθερή θέση αναφοράς.



Εικόνα 6.1 Όχημα σε σημείο ισορροπίας.

Εικόνα 6.2 Όχημα σε κλίση, χωρίς αυτόματη εξισορρόπηση.



Εικόνα 6.3 Όχημα σε κλίση, με αυτόματη εξισορρόπηση.

6.2 Τετρακόπτερο

Η προσομοίωση αποτέλεσε απαραίτητο στάδιο της παρούσας εργασίας, καθώς επέτρεψε την αξιολόγηση της λογικής αυτόνομης προσγείωσης πριν από την πλήρη εφαρμογή της στο φυσικό τετρακόπτερο. Σε αντίθεση με το όχημα εδάφους, το UAV αποτελεί δυναμικά ασταθές σύστημα και κάθε λανθασμένη εντολή μπορεί να οδηγήσει σε απώλεια ελέγχου ή μηχανική αστοχία [2], [4], [5], [15]. Για τον λόγο αυτό, η λειτουργία του αλγορίθμου μηχανικής όρασης, η διαδικασία ευθυγράμμισης και η αποστολή δεδομένων προς το σύστημα ελέγχου πτήσης εξετάστηκαν αρχικά σε περιβάλλον προσομοίωσης.

Μέσω της προσομοίωσης μελετήθηκε η συμπεριφορά του συστήματος σε διαφορετικές σχετικές θέσεις μεταξύ κάμερας και πλατφόρμας προσγείωσης. Ο βασικός στόχος ήταν να επιβεβαιωθεί ότι ο αλγόριθμος μπορεί να αναγνωρίζει τους ArUco markers, να εκτιμά τη σχετική θέση του στόχου, να διακρίνει την αρχική από την τελική φάση προσέγγισης και να παράγει κατάλληλα δεδομένα καθοδήγησης. Η διαδικασία αυτή αποτέλεσε ενδιάμεσο στάδιο μεταξύ της θεωρητικής ανάπτυξης του αλγορίθμου και της πρακτικής δοκιμής του στο φυσικό UAV.

Η αξιολόγηση του τετρακοπτέρου οργανώθηκε σε τρία επίπεδα. Αρχικά εξετάστηκε η λειτουργία του αλγορίθμου ArUco σε διαφορετικά σενάρια σχετικής μετατόπισης. Στη συνέχεια πραγματοποιήθηκε πιο ολοκληρωμένη προσομοίωση με χρήση Gazebo Sim και ArduPilot SITL, ώστε να ελεγχθεί η συνεργασία μεταξύ του προσομοιωμένου UAV, της κάμερας, των ArUco markers και του συστήματος ελέγχου πτήσης. Τέλος, πραγματοποιήθηκε φυσικό πείραμα με το πραγματικό τετρακόπτερο, αφού είχαν ολοκληρωθεί οι βασικές ρυθμίσεις και οι έλεγχοι ασφαλείας.

6.2.1 Προσομοίωση ArUco

Η ανάγκη για προσομοίωση του αλγορίθμου ArUco προέκυψε από το γεγονός ότι η αυτόνομη προσγείωση αποτελεί διαδικασία υψηλής ακρίβειας και απαιτεί διαδοχική λήψη αποφάσεων [2], [7], [15]. Το UAV πρέπει αρχικά να εντοπίζει την πλατφόρμα, να εκτιμά τη σχετική του θέση ως προς αυτήν, να διορθώνει τυχόν οριζόντια απόκλιση και, όταν βρίσκεται εντός αποδεκτών ορίων, να εισέρχεται στην τελική φάση καθόδου. Η διαδικασία αυτή συνδέεται με τη λειτουργία Precision Landing του ArduPilot, στην οποία αποστέλλονται δεδομένα τύπου LANDING_TARGET μέσω MAVLink [59], [60].

Για την αρχική αξιολόγηση του αλγορίθμου χρησιμοποιήθηκε προσομοιωμένο περιβάλλον, στο οποίο η κάμερα του UAV τοποθετήθηκε σε διαφορετικές θέσεις ως προς την πλατφόρμα προσγείωσης. Σε κάθε σενάριο λαμβανόταν εικόνα, γινόταν ανίχνευση του αντίστοιχου ArUco marker και υπολογίζονταν οι μεταβλητές θέσης, απόστασης και γωνιακής απόκλισης [20]. Με βάση τις τιμές αυτές, ο αλγόριθμος επέλεγε την αντίστοιχη κατάσταση λειτουργίας και την κατάλληλη λογική ενέργεια διόρθωσης.

Πρέπει να τονιστεί ότι οι εντολές όπως MOVE_RIGHT, MOVE_LEFT, MOVE_FORWARD, MOVE_BACKWARD, LOWER_SLOW και LAND χρησιμοποιούνται στο στάδιο αυτό ως λογικές έξοδοι αξιολόγησης του αλγορίθμου. Δηλαδή δεν αποτελούν απαραίτητα άμεσες χειροκίνητες εντολές προς τους κινητήρες, αλλά δείχνουν ποια απόφαση θα έπρεπε να λάβει το σύστημα με βάση τη θέση του marker στην εικόνα. Στο πλήρες σύστημα, η σχετική θέση μετατρέπεται σε δεδομένα καθοδήγησης ή σε μηνύματα LANDING_TARGET προς το ArduPilot [23].

Για την αξιολόγηση της συμπεριφοράς πραγματοποιήθηκαν επτά βασικά σενάρια. Τα σενάρια αυτά επιλέχθηκαν ώστε να καλύπτουν τις βασικές περιπτώσεις που αναμένονται κατά την προσέγγιση της πλατφόρμας: κεντραρισμένη κάμερα, πλευρικές αποκλίσεις, εμπρόσθια και οπίσθια απόκλιση, μεγάλο ύψος και τελικό όριο προσγείωσης. Με τον τρόπο αυτό ελέγχθηκε όχι μόνο η ικανότητα ανίχνευσης του marker, αλλά και η ορθότητα της λογικής απόφασης του αλγορίθμου [12], [16].

Στο πρώτο σενάριο, η κάμερα ήταν σχεδόν κεντραρισμένη σε χαμηλό ύψος. Το σύστημα αναγνώρισε ότι η οριζόντια απόκλιση ήταν μικρή και επέλεξε κατάσταση τελικής καθόδου, παράγοντας την εντολή LOWER_SLOW. Στο δεύτερο και τρίτο σενάριο εφαρμόστηκε πλευρική απόκλιση προς αντίθετες κατευθύνσεις. Το σύστημα αντέδρασε με τις εντολές MOVE_RIGHT και MOVE_LEFT, επιβεβαιώνοντας ότι το πρόσημο της πλευρικής μετατόπισης ερμηνεύεται σωστά. Στο τέταρτο και πέμπτο σενάριο εξετάστηκε η εμπρόσθια και οπίσθια απόκλιση, με αποτέλεσμα την παραγωγή των εντολών MOVE_FORWARD και MOVE_BACKWARD.

Το έκτο σενάριο αφορούσε μεγαλύτερο ύψος, με μέση απόσταση περίπου 1,155 m από την πλατφόρμα. Σε αυτή την περίπτωση, το σύστημα δεν επέλεξε τελική κάθοδο, αλλά κατάσταση αρχικής ευθυγράμμισης με το μεγάλο marker, δηλαδή ALIGN_BIG_MARKER. Η επιλογή αυτή είναι σημαντική, καθώς δείχνει ότι ο αλγόριθμος διαφοροποιεί τη συμπεριφορά του ανάλογα με την απόσταση από τον στόχο. Τέλος, στο έβδομο σενάριο η μέση απόσταση ήταν περίπου 0,149 m, τιμή που αντιστοιχεί στο τελικό όριο προσγείωσης. Το σύστημα αναγνώρισε ότι το UAV βρίσκεται αρκετά κοντά στην πλατφόρμα και επέλεξε την κατάσταση LAND.

Τα αποτελέσματα των δοκιμών δείχνουν ότι ο αλγόριθμος ανταποκρίνεται σωστά στις βασικές περιπτώσεις σχετικής μετατόπισης μεταξύ UAV και πλατφόρμας. Σε όλα τα σενάρια, η παρατηρούμενη έξοδος συμφώνησε με την αναμενόμενη συμπεριφορά. Στα

σενάρια πλευρικής μετατόπισης, οι μέσες γωνιακές αποκλίσεις ήταν περίπου +6,05 μοίρες και -6,23 μοίρες, οδηγώντας αντίστοιχα στις εντολές MOVE_RIGHT και MOVE_LEFT. Αντίστοιχα, στις εμπρόσθιες και οπίσθιες αποκλίσεις, η μεταβολή της γωνιακής απόκλισης στον δεύτερο άξονα οδήγησε στις αντίστοιχες εντολές διόρθωσης.

Συνολικά, η αρχική προσομοίωση ArUco επιβεβαίωσε ότι η λογική εκτίμησης θέσης και επιλογής κατάστασης λειτουργεί με συνεπή τρόπο. Το αποτέλεσμα αυτό ήταν απαραίτητο πριν από τη μετάβαση σε πιο σύνθετη προσομοίωση, όπου εκτός από την ανίχνευση του marker έπρεπε να εξεταστεί και η αλληλεπίδραση με το σύστημα πτήσης του UAV.

6.2.2 Προσομοίωση Ολοκληρωμένου Συστήματος

Πέρα από την παραπάνω προσομοίωση για τον έλεγχο της γενικής λειτουργίας του αλγόριθμου ανίχνευσης, χρειάστηκε να κάνουμε μια δεύτερη, πιο ενδελεχή προσομοίωση για δοκιμή του ολοκληρωμένου συστήματος πτήσης, αναγνώρισης και προσγείωσης [59], [60], [61].

Για την υλοποίηση της προσομοίωσης χρησιμοποιήθηκε το Gazebo Sim σε συνδυασμό με το ArduPilot SITL (Εικόνα 6.1) [21]. Το Gazebo χρησιμοποιήθηκε για την τρισδιάστατη αναπαράσταση του περιβάλλοντος και του τετρακοπτέρου, ενώ το ArduPilot SITL χρησιμοποιήθηκε για την προσομοίωση της πτητικής συμπεριφοράς και των λειτουργιών του αυτόματου πιλότου [21], [23]. Η επικοινωνία μεταξύ του προσομοιωμένου UAV και του υπολογιστή πραγματοποιήθηκε μέσω MAVLink, ενώ το Mission Planner χρησιμοποιήθηκε για την παρακολούθηση της τηλεμετρίας και των καταστάσεων πτήσης [22]. Το μοντέλο του UAV προσαρμόστηκε ώστε να αντιστοιχεί στο φυσικό πρωτότυπο της εργασίας. Συγκεκριμένα, το οπτικό μοντέλο βασίστηκε σε πλαίσιο τύπου DJI F450 και περιλάμβανε αναπαραστάσεις των βασικών εξαρτημάτων: flight controller Matek F405-WMN, μπαταρία 3S 2200 mAh στα 11,1 V, τέσσερις κινητήρες A2212 920KV, ESC LANRC 45A, προπέλες 1045, ESP32-CAM και ξεχωριστή πλακέτα ESP32 για επικοινωνία [20]. Η κάμερα τοποθετήθηκε στην κάτω πλευρά του UAV, με προσανατολισμό προς την πλατφόρμα. Η συνολική μάζα του φυσικού UAV εκτιμήθηκε σε περίπου 0,890 kg. Η τιμή αυτή προέκυψε από τη μάζα του UAV χωρίς προπέλες, περίπου 0,840 kg, και τη συνολική μάζα των τεσσάρων προπελών, περίπου 0,050 kg. Το δυναμικό μοντέλο της προσομοίωσης ρυθμίστηκε ώστε να προσεγγίζει αυτή τη μάζα και να επιτυγχάνει σταθερή απογείωση και αιώρηση.

Η πρώτη βασική μαθηματική παράμετρος της προσομοίωσης είναι η απαιτούμενη ώση για αιώρηση. Για μάζα:

$$m = 0{,}890 \text{ kg}$$

και επιτάχυνση βαρύτητας:

$$g = 9{,}81 \text{ m/s}^2$$

το συνολικό βάρος του UAV είναι:

$$W = m \cdot g$$
$$W = 0{,}890 \cdot 9{,}81 = 8{,}73 \text{ N}$$

Επειδή το UAV είναι τετρακόπτερο, η απαιτούμενη ώση ανά κινητήρα κατά την αιώρηση είναι:

$$T_{hover} = \frac{W}{4}$$
$$T_{hover} = \frac{8{,}73}{4} = 2{,}18 \text{ N}$$

Η τιμή αυτή αντιστοιχεί περίπου σε:

$$T_{hover} \approx 0{,}222 \text{ kgf}$$

δηλαδή περίπου 222 g ώσης ανά κινητήρα.

Σύμφωνα με τα δεδομένα του κινητήρα A2212 920KV με προπέλα 1045 και τροφοδοσία 11,1 V, η ώση ανά κινητήρα είναι περίπου 210 g στο 50% throttle, 420 g στο 75% throttle και 680 g στο 100% throttle. Επομένως, το θεωρητικό σημείο αιώρησης βρίσκεται λίγο πάνω από το 50% throttle. Με γραμμική παρεμβολή μεταξύ 50% και 75%:

$$T_{required} = 222{,}5 \text{ g}$$
$$T_{50\%} = 210 \text{ g}$$
$$T_{75\%} = 420 \text{ g}$$

Η επιπλέον ώση που απαιτείται πάνω από το 50% είναι:

$$222{,}5 - 210 = 12{,}5 \text{ g}$$

Η αύξηση ώσης μεταξύ 50% και 75% είναι:

$$420 - 210 = 210 \text{ g}$$

Άρα το ποσοστό του διαστήματος που απαιτείται είναι:

$$\frac{12{,}5}{210} = 0{,}0595$$

και η αντίστοιχη αύξηση throttle είναι:

$$0{,}0595 \cdot 25\% = 1{,}49\%$$

Συνεπώς, το αναμενόμενο throttle αιώρησης είναι:

$$50\% + 1{,}49\% = 51{,}49\%$$

δηλαδή:

$$MOT\_THST\_HOVER \approx 0{,}515$$

Κατά τη βαθμονόμηση της προσομοίωσης, η αρχική δυναμική συμπεριφορά δεν αντιστοιχούσε πλήρως στη μάζα και στην ώση του φυσικού UAV. Για τον λόγο αυτό έγιναν επαναληπτικές ρυθμίσεις στο μοντέλο ώσης των προπελών. Η τελική τιμή που υπολογίστηκε από το ArduPilot για την παράμετρο αιώρησης ήταν:

$$MOT\_THST\_HOVER \approx 0{,}476$$

Η τιμή αυτή βρίσκεται κοντά στη θεωρητική τιμή 0,515 και θεωρήθηκε ικανοποιητική για τη συνέχεια των πειραμάτων, καθώς το UAV μπορούσε να απογειωθεί, να αιωρηθεί και να ανταποκριθεί σε εντολές καθοδήγησης χωρίς να εμφανίζει ανεξέλεγκτη συμπεριφορά.

Η κάμερα της προσομοίωσης τοποθετήθηκε στην κάτω πλευρά του UAV, ώστε να αναπαριστά τη λειτουργία της ESP32-CAM στο φυσικό σύστημα. Η ανάλυση της κάμερας ορίστηκε σε:

$$640 \times 480 \text{ pixels}$$

με οριζόντιο πεδίο θέασης:

$$HFOV = 1{,}047 \text{ rad}$$

Η κάμερα προσεγγίστηκε με το μοντέλο pinhole camera. Η εστιακή απόσταση σε pixels υπολογίζεται από τη σχέση:

$$f_x = \frac{w}{2\tan\left(\frac{HFOV}{2}\right)}$$

όπου $w = 640$ pixels. Επομένως:

$$f_x = \frac{640}{2\tan\left(\frac{1{,}047}{2}\right)}$$
$$f_x \approx 554 \text{ pixels}$$

Θεωρώντας τετραγωνικά pixels:

$$f_y = f_x$$

και οπτικό κέντρο:

$$c_x = 320, c_y = 240$$

ο πίνακας εσωτερικών παραμέτρων της κάμερας είναι:

$$K = \begin{bmatrix} 554 & 0 & 320 \\ 0 & 554 & 240 \\ 0 & 0 & 1 \end{bmatrix}$$

Ο πίνακας αυτός χρησιμοποιήθηκε από τον αλγόριθμο εκτίμησης στάσης ArUco. Για κάθε marker είναι γνωστή η πραγματική του διάσταση. Στην προσομοίωση χρησιμοποιήθηκαν δύο markers:

$$ID = 23, s = 0{,}110 \text{ m}$$
$$ID = 7, s = 0{,}045 \text{ m}$$

Για κάθε marker, οι τέσσερις γωνίες του στο τοπικό σύστημα συντεταγμένων ορίστηκαν ως:

$$(-s/2, +s/2, 0)$$
$$(+s/2, +s/2, 0)$$
$$(+s/2, -s/2, 0)$$
$$(-s/2, -s/2, 0)$$

Με βάση τα σημεία αυτά και τις αντίστοιχες γωνίες που ανιχνεύονται στην εικόνα, επιλύεται το πρόβλημα Perspective-n-Point. Το αποτέλεσμα είναι ένα διάνυσμα μεταφοράς:

$$t = \begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{bmatrix}$$

το οποίο εκφράζει τη θέση του marker ως προς το σύστημα συντεταγμένων της κάμερας.

Η απευθείας χρήση του διανύσματος της κάμερας δεν είναι επαρκής, επειδή η κάμερα δεν βρίσκεται ακριβώς στο κέντρο του UAV. Στο μοντέλο της προσομοίωσης, η κάμερα θεωρήθηκε ότι βρίσκεται περίπου 0,045 m μπροστά από το κέντρο του UAV και περίπου 0,050 m κάτω από αυτό. Επομένως, η θέση της κάμερας ως προς το σώμα του UAV είναι:

$$X_{cam\_offset} = 0{,}045 \text{ m}$$
$$Y_{cam\_offset} = 0$$
$$Z_{cam\_offset} = 0{,}050 \text{ m}$$

Το OpenCV χρησιμοποιεί σύστημα κάμερας όπου ο άξονας x_{cam} αντιστοιχεί στη δεξιά κατεύθυνση της εικόνας, ο άξονας y_{cam} στην κάτω κατεύθυνση της εικόνας και ο άξονας z_{cam} στην κατεύθυνση θέασης της κάμερας. Για κάμερα στραμμένη προς τα κάτω, το διάνυσμα μετασχηματίζεται στο σύστημα σώματος του UAV τύπου FRD, όπου X είναι εμπρός, Y δεξιά και Z κάτω.

Ο μετασχηματισμός που χρησιμοποιήθηκε είναι:

$$X_{body} = -y_{cam}$$
$$Y_{body} = x_{cam}$$
$$Z_{body} = z_{cam}$$

Λαμβάνοντας υπόψη και τη μετατόπιση της κάμερας από το κέντρο του UAV:

$$X_{vehicle} = 0{,}045 - y_{cam}$$
$$Y_{vehicle} = x_{cam}$$
$$Z_{vehicle} = 0{,}050 + z_{cam}$$

Με αυτόν τον τρόπο, το σφάλμα θέσης του marker εκφράζεται ως προς το κέντρο του UAV και μπορεί να χρησιμοποιηθεί είτε για εξωτερικό έλεγχο καθοδήγησης είτε για αποστολή μηνυμάτων LANDING_TARGET προς το ArduPilot.

Στην προσομοίωση χρησιμοποιήθηκαν δύο ArUco markers, με διαφορετικό ρόλο. Το μεγάλο marker με ID 23 χρησιμοποιήθηκε για αρχική ανίχνευση και χονδρική ευθυγράμμιση, ενώ το μικρό marker με ID 7 χρησιμοποιήθηκε για την τελική φάση προσγείωσης. Η χρήση δύο markers επιτρέπει καλύτερη λειτουργία σε διαφορετικά ύψη: το μεγάλο marker είναι πιο εύκολα ανιχνεύσιμο από μεγαλύτερη απόσταση, ενώ το μικρό marker δίνει πιο ακριβές σημείο αναφοράς κοντά στην πλατφόρμα. Στην πλατφόρμα, τα δύο markers δεν τοποθετούνται στο ίδιο φυσικό σημείο. Το μεγάλο marker τοποθετήθηκε σε απόσταση περίπου:

$$y_{big} = -0{,}035 \text{ m}$$

από το κέντρο της πλατφόρμας, ενώ το μικρό marker τοποθετήθηκε σε απόσταση:

$$y_{small} = +0{,}060 \text{ m}$$

Η απόσταση μεταξύ των δύο κέντρων είναι:

$$\Delta x = 0{,}060 - (-0{,}035)$$
$$\Delta x = 0{,}095 \text{ m}$$

Η παρατήρηση αυτή ήταν σημαντική κατά τη δοκιμή της προσομοίωσης. Όταν ο αλγόριθμος επέτρεπε ελεύθερη εναλλαγή μεταξύ των δύο markers κατά την τελική προσγείωση, το σημείο στόχου μετατοπιζόταν απότομα κατά περίπου 9,5 cm. Αυτό προκαλούσε απότομες διορθώσεις, υπερδιόρθωση και ασταθή επαφή με την πλατφόρμα. Για τον λόγο αυτό η τελική λογική του συστήματος διαμορφώθηκε έτσι ώστε το μεγάλο marker να χρησιμοποιείται μόνο για αρχική ευθυγράμμιση και το μικρό marker να "κλειδώνει" ως ο μοναδικός στόχος κατά την τελική προσγείωση.

Η τελική λογική της προσομοίωσης οργανώθηκε ως μηχανή καταστάσεων. Η μηχανή καταστάσεων αντιστοιχεί στη ρεαλιστική διαδικασία λειτουργίας του συστήματος, όπου η πλατφόρμα δεν χρειάζεται να τοποθετηθεί με απόλυτη ακρίβεια κάτω από το UAV, αλλά απλώς να βρεθεί εντός του οπτικού πεδίου της κάμερας [2], [12], [15], [16].

Οι βασικές καταστάσεις είναι οι εξής:

ALIGN_BIG: Το UAV χρησιμοποιεί το μεγάλο marker 23 για την αρχική ευθυγράμμιση. Η κατάσταση αυτή επιτρέπει στο σύστημα να λειτουργεί ακόμη και όταν η πλατφόρμα βρίσκεται σχετικά μακριά από το κέντρο της εικόνας.

SEARCH_SMALL_DESCEND: Όταν το μεγάλο marker έχει κεντραριστεί για συγκεκριμένο αριθμό διαδοχικών πλαισίων, το UAV αρχίζει να κατεβαίνει αργά, ενώ συνεχίζει να διατηρεί την ευθυγράμμιση ως προς το marker 23. Στόχος αυτής της φάσης είναι να πλησιάσει αρκετά ώστε να γίνει ορατό το μικρό marker.

ALIGN_SMALL: Μόλις το μικρό marker 7 εντοπιστεί σταθερά, το σύστημα αλλάζει στόχο και ευθυγραμμίζεται με αυτό. Η μετάβαση αυτή γίνεται πριν την τελική προσγείωση, ώστε να αποφευχθεί οποιαδήποτε αλλαγή στόχου κατά την κάθοδο.

WAIT_FOR_CONFIRMATION: Όταν το μικρό marker είναι κεντραρισμένο και σταθερό, το σύστημα περιμένει εντολή από τον χρήστη. Στην προσομοίωση η εντολή αυτή δόθηκε με πάτημα ENTER, ενώ στο φυσικό σύστημα μπορεί να αντιστοιχεί σε κουμπί ή εντολή από τη διεπαφή ελέγχου.

PRECISION_LAND_SMALL_LOCKED: Μετά την επιβεβαίωση, ενεργοποιείται LAND mode στο ArduPilot και αποστέλλονται μόνο δεδομένα LANDING_TARGET για το marker 7. Δεν επιτρέπεται επιστροφή στο μεγάλο marker, ώστε να αποφευχθεί απότομη μετατόπιση του στόχου.

Η μηχανή καταστάσεων επέτρεψε την ασφαλή μετάβαση από την αρχική αναγνώριση στην τελική προσγείωση. Επίσης, κατέδειξε ότι η πλατφόρμα δεν χρειάζεται να τοποθετηθεί με ακρίβεια στο τελικό σημείο προσγείωσης. Η ευθυγράμμιση ολοκληρώνεται από το UAV με βάση την οπτική αναγνώριση.

Η πειραματική διαδικασία περιλάμβανε πέντε βασικές δοκιμές.

Αρχικά πραγματοποιήθηκε βαθμονόμηση της πτητικής συμπεριφοράς. Το UAV ρυθμίστηκε σε συνολική μάζα περίπου 0,890 kg και το μοντέλο ώσης προσαρμόστηκε μέχρι να επιτευχθεί σταθερή απογείωση και αιώρηση. Η τελική τιμή MOT_THST_HOVER ήταν περίπου 0,476.

Στη συνέχεια δοκιμάστηκε η διαδικασία προσέγγισης της πλατφόρμας. Η πλατφόρμα ξεκίνησε από θέση εκτός του οπτικού πεδίου της κάμερας και μετακινήθηκε σταδιακά προς το UAV. Κατά τα πρώτα δευτερόλεπτα το σύστημα βρισκόταν στην κατάσταση SEARCH_MARKER. Όταν το μεγάλο marker εισήλθε στο οπτικό πεδίο, η κατάσταση άλλαξε σε MARKER_ACQUIRED και μετά από σταθερή ανίχνευση διαδοχικών πλαισίων το σύστημα δήλωσε ότι ήταν έτοιμο να ξεκινήσει η διαδικασία ευθυγράμμισης.

Η τρίτη δοκιμή αφορούσε την ευθυγράμμιση με το μεγάλο marker. Το UAV χρησιμοποίησε τον εξωτερικό ελεγκτή GUIDED mode και μετακινήθηκε αργά προς το κέντρο του marker 23. Μετά την επίτευξη σταθερής ευθυγράμμισης, η κατάσταση άλλαξε σε SEARCH_SMALL_DESCEND.

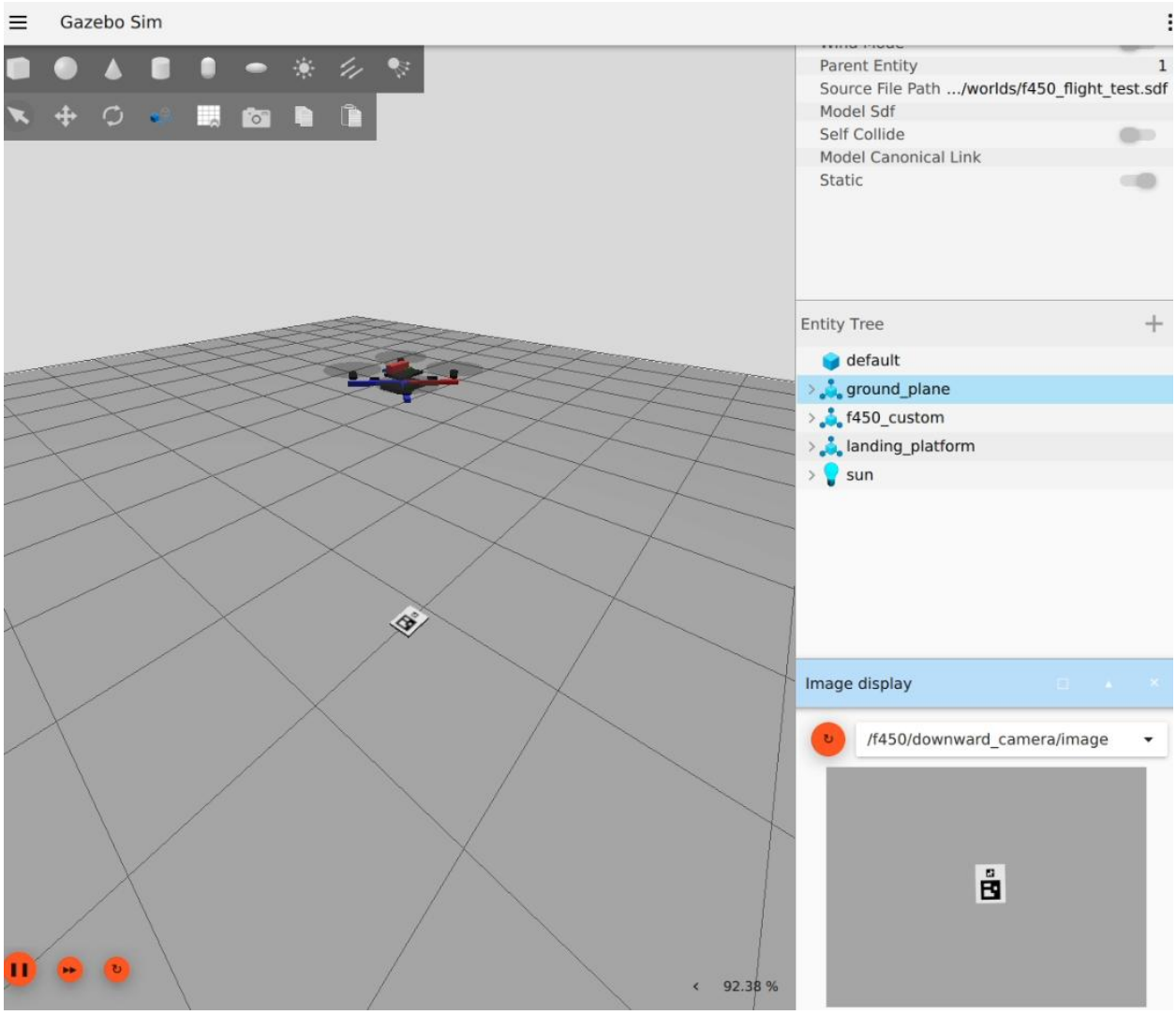
Στην τέταρτη δοκιμή, το UAV κατέβαινε αργά ενώ συνέχιζε να διορθώνει τη θέση του με βάση το μεγάλο marker. Όταν το μικρό marker 7 έγινε ορατό, το σύστημα άλλαξε κατάσταση σε ALIGN_SMALL. Από εκεί και μετά η τελική ευθυγράμμιση βασίστηκε μόνο στο μικρό marker.

Στην πέμπτη δοκιμή, μετά την ευθυγράμμιση στο μικρό marker, δόθηκε επιβεβαίωση από τον χρήστη και ενεργοποιήθηκε LAND mode. Κατά την τελική φάση αποστέλλονταν στο ArduPilot μόνο μηνύματα LANDING_TARGET για το marker 7. Η διαδικασία οδήγησε το UAV στην πλατφόρμα και το σύστημα δήλωσε αφόπλιση μετά την επαφή με την επιφάνεια.

Η προσομοίωση επιβεβαίωσε τη λειτουργική αλληλουχία του προτεινόμενου συστήματος αυτόνομης προσγείωσης. Το UAV μπόρεσε να απογειωθεί, να αιωρηθεί, να ανιχνεύσει την πλατφόρμα όταν αυτή εισήλθε στο οπτικό πεδίο της κάμερας, να ευθυγραμμιστεί αρχικά με το μεγάλο ArUco marker, να κατέβει μέχρι την ανίχνευση του μικρού marker, να ευθυγραμμιστεί με αυτό και να ενεργοποιήσει την τελική διαδικασία προσγείωσης μετά από επιβεβαίωση του χρήστη.

Το σημαντικότερο συμπέρασμα είναι ότι η πλατφόρμα δεν χρειάζεται να τοποθετηθεί με μεγάλη ακρίβεια στο τελικό σημείο προσγείωσης. Η λειτουργία της είναι να φέρει το οπτικό σήμα εντός του πεδίου ανίχνευσης του UAV. Η τελική ακρίβεια επιτυγχάνεται από το σύστημα όρασης και τον ελεγκτή ευθυγράμμισης του UAV [5], [6], [9], [11], [17].

Η προσομοίωση ανέδειξε επίσης κρίσιμα τεχνικά ζητήματα που θα πρέπει να ληφθούν υπόψη και στην πραγματική υλοποίηση, όπως η ανάγκη σταθερής επιλογής marker κατά την τελική φάση, η αποφυγή απότομων διορθώσεων, η χρήση φίλτρων εξομάλυνσης και η ύπαρξη χειροκίνητης επιβεβαίωσης πριν την προσγείωση. Συνεπώς, η προσομοίωση αποτέλεσε ουσιαστικό εργαλείο επαλήθευσης, καθώς επέτρεψε τη δοκιμή της λογικής ελέγχου πριν από την εφαρμογή της στο φυσικό UAV.



Εικόνα 6.4 Αναπαράσταση του οχήματος στο περιβάλλον Gazebo

6.2.3 Φυσικό Πείραμα

Το φυσικό πείραμα του UAV πραγματοποιήθηκε μετά την ολοκλήρωση της κατασκευής, της ηλεκτρολογικής διασύνδεσης, της παραμετροποίησης του ελεγκτή πτήσης και των βασικών δοκιμών ασφαλείας. Το στάδιο αυτό ήταν απαραίτητο, καθώς η προσομοίωση μπορεί να επιβεβαιώσει τη λογική του συστήματος, αλλά δεν μπορεί να αναπαραστήσει πλήρως όλους τους παράγοντες που εμφανίζονται στην πραγματική πτήση, όπως οι κραδασμοί, οι μικρές μηχανικές αποκλίσεις, οι διακυμάνσεις της τροφοδοσίας, οι καθυστερήσεις επικοινωνίας και οι πραγματικές συνθήκες φωτισμού [2], [4], [8], [12].

Κατά τα πρώτα στάδια των δοκιμών παρουσιάστηκαν τεχνικές δυσκολίες, οι οποίες οδήγησαν σε επαναξιολόγηση των αρχικών επιλογών. Πραγματοποιήθηκαν αλλαγές σε

επιμέρους εξαρτήματα, όπως ESC, προπέλες, κινητήρες, μπαταρία και αρχιτεκτονική επικοινωνίας μεταξύ UAV, υπολογιστή και αλγορίθμου αναγνώρισης. Η διαδικασία αυτή, αν και χρονοβόρα, ήταν καθοριστική για τη διαμόρφωση της τελικής μορφής του τετρακοπτέρου και ανέδειξε τη σημασία της σωστής προσομοίωσης και της σταδιακής επαλήθευσης πριν από τις πτητικές δοκιμές.

Η τελική παραμετροποίηση του UAV πραγματοποιήθηκε με χρήση του ArduPilot Methodic Configurator. Μέσω της διαδικασίας αυτής εισήχθησαν οι βασικές παράμετροι του οχήματος, όπως το είδος πλαισίου, το βάρος, η μπαταρία, οι κινητήρες, οι προπέλες, το πρωτόκολλο επικοινωνίας με τα ESC και οι σειριακές θύρες επικοινωνίας. Οι αρχικές δοκιμές πραγματοποιήθηκαν σε ελεγχόμενο περιβάλλον με περιορισμό της κίνησης του UAV, ώστε να ελεγχθούν η ορθή φορά περιστροφής των κινητήρων, η απόκριση του συστήματος και η βασική σταθερότητα χωρίς κίνδυνο ανεξέλεγκτης πτήσης [33], [59], [61].

Μετά την ολοκλήρωση των ελέγχων ασφαλείας πραγματοποιήθηκε πτήση σε εξωτερικό χώρο του εργαστηρίου. Η πρώτη ελεύθερη πτήση χρησιμοποιήθηκε κυρίως για τη συλλογή δεδομένων και την αυτόματη προσαρμογή συγκεκριμένων παραμέτρων από το ArduPilot. Παρότι η αρχική απόκριση δεν ήταν πλήρως ομαλή, το UAV κατάφερε να απογειωθεί και να διατηρήσει ελεγχόμενη πτήση. Τα δεδομένα της πτήσης χρησιμοποιήθηκαν στη συνέχεια για τα επόμενα βήματα ρύθμισης του Methodic Configurator.

Ακολούθησε δεύτερη πτήση, κατά την οποία εκτελέστηκε το απαιτούμενο Lua script στον ελεγκτή πτήσης για την ολοκλήρωση των επόμενων βημάτων παραμετροποίησης. Η δεύτερη πτήση παρουσίασε βελτιωμένη συμπεριφορά σε σχέση με την πρώτη, γεγονός που έδειξε ότι οι διορθώσεις των παραμέτρων και η βαθμονόμηση είχαν θετική επίδραση στη σταθερότητα του UAV. Μετά την ολοκλήρωση των βασικών ρυθμίσεων, το τετρακόπτερο θεωρήθηκε έτοιμο για τη δοκιμή συνεργασίας με τον αλγόριθμο μηχανικής όρασης.

Στην ολοκληρωμένη δοκιμή, το UAV απογειώθηκε και σταθεροποιήθηκε σε ύψος περίπου 1,5 m. Στη συνέχεια, το όχημα εδάφους μετακινήθηκε κάτω από το UAV μέχρι η πλατφόρμα προσγείωσης και ο μεγάλος ArUco marker να εισέλθουν στο οπτικό πεδίο της κάμερας. Με την ανίχνευση του marker, το σύστημα αναγνώρισε τη σχετική θέση της πλατφόρμας και το UAV εισήλθε σε φάση ευθυγράμμισης [4], [8], [12], [15]. Η λειτουργία αυτή αντιστοιχεί στη φάση Precision Loiter, κατά την οποία το UAV χρησιμοποιεί τις οπτικές μετρήσεις για να παραμείνει ευθυγραμμισμένο με τον στόχο.

Όταν η ευθυγράμμιση με το μεγάλο marker σταθεροποιήθηκε, το σύστημα ενημέρωσε τον χρήστη ότι ήταν έτοιμο για την έναρξη της διαδικασίας προσγείωσης. Η τελική κάθοδος ξεκίνησε μετά από επιβεβαίωση του χρήστη, ώστε να υπάρχει πρόσθετο επίπεδο ασφάλειας πριν από την προσγείωση. Στη συνέχεια, το UAV εισήλθε στη φάση Precision Landing και άρχισε να κατέρχεται, ενώ ο υπολογιστής λάμβανε συνεχώς εικόνες από την κάμερα, υπολόγιζε τη σχετική θέση του marker και έστελνε τις αντίστοιχες πληροφορίες στο ArduPilot.

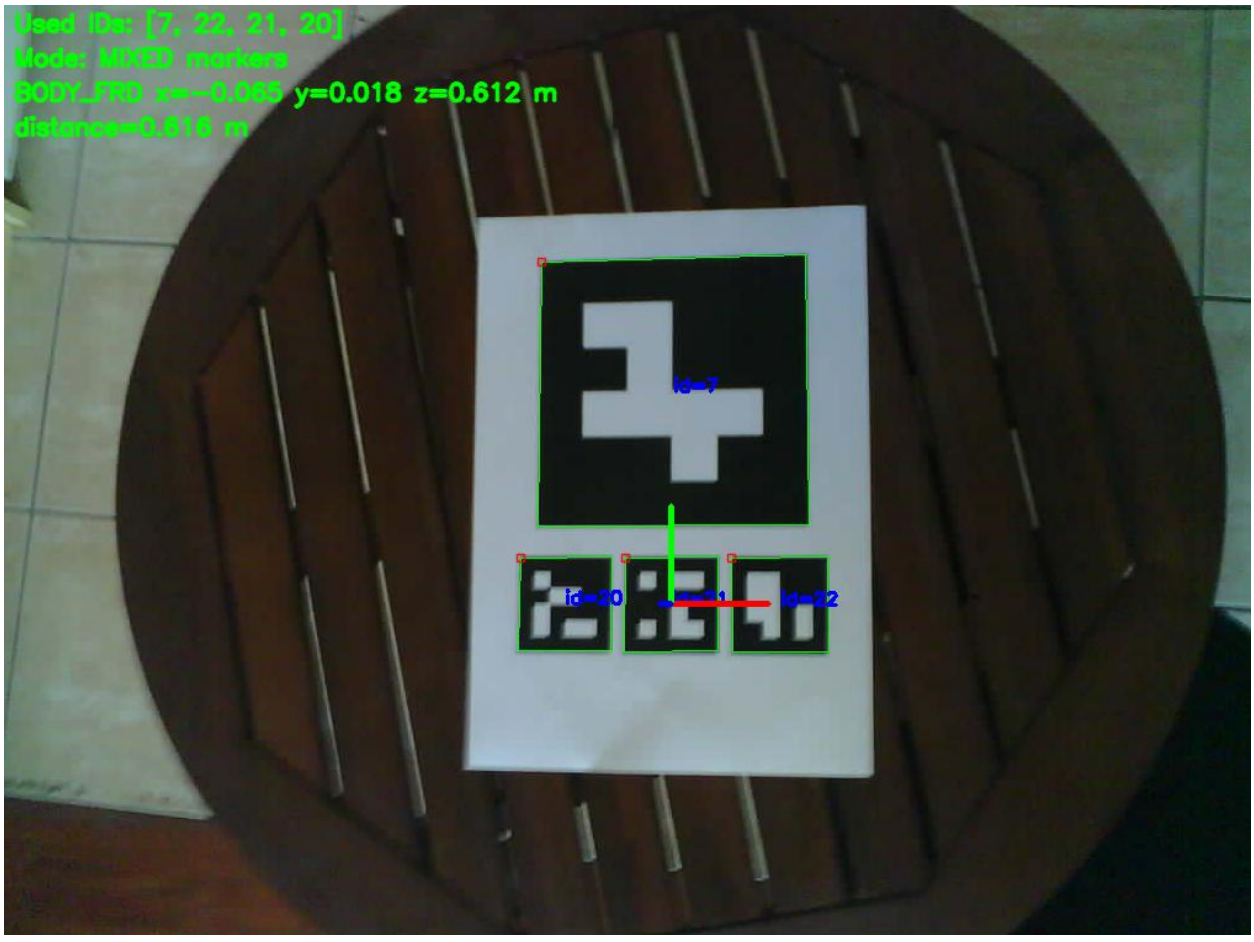
Κατά τη μείωση του ύψους, το μικρό marker έγινε ορατό από την κάμερα. Από το σημείο αυτό και μετά, οι υπολογισμοί της τελικής ευθυγράμμισης βασίστηκαν στο μικρό marker, το οποίο αντιστοιχούσε στο ακριβέστερο σημείο αναφοράς για την προσγείωση. Η αλλαγή αυτή είναι σημαντική, επειδή το μεγάλο marker διευκολύνει την αρχική ανίχνευση από μεγαλύτερο ύψος, ενώ το μικρό marker επιτρέπει καλύτερη ακρίβεια στην τελική φάση. Όταν το UAV έφτασε περίπου στα 20 cm πάνω από την πλατφόρμα,

η διαδικασία συνέχισε ως απλή προσγείωση, χωρίς περαιτέρω εξάρτηση από τον marker.

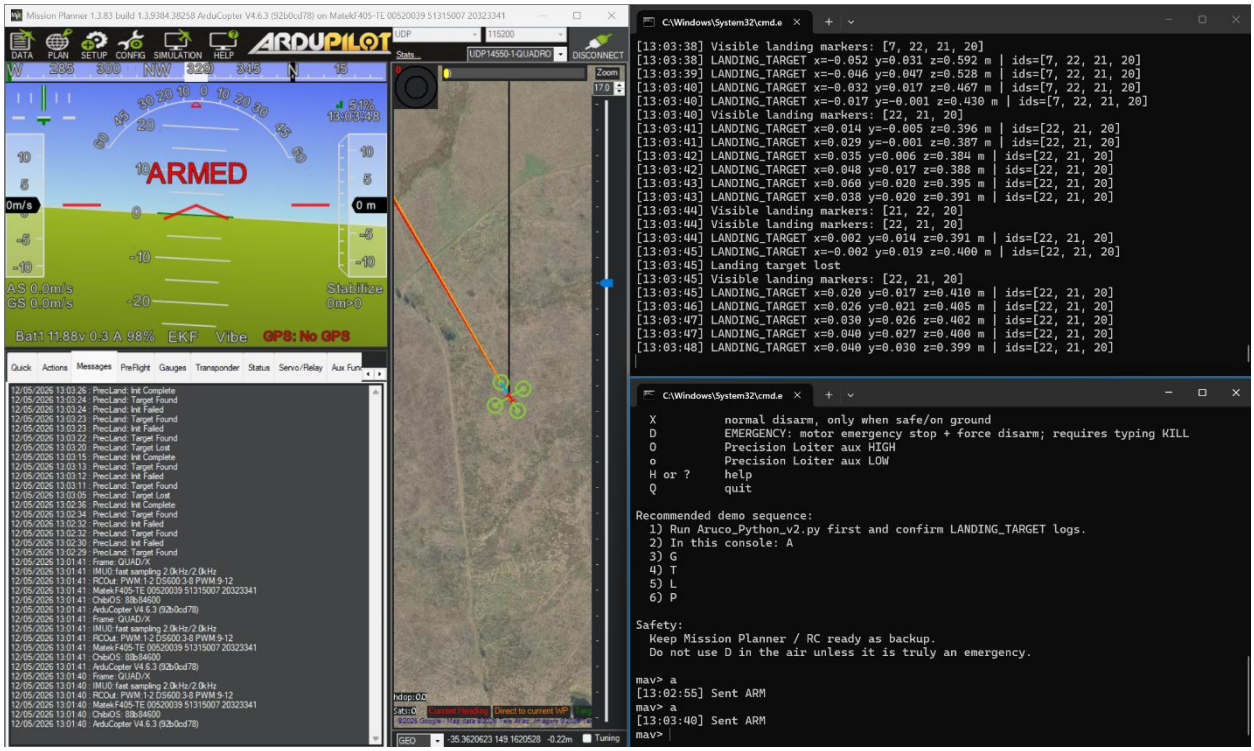
Η δοκιμή ολοκληρώθηκε ομαλά και επαναλήφθηκε ώστε να επιβεβαιωθεί η σταθερότητα της διαδικασίας. Τα αποτελέσματα έδειξαν ότι το σύστημα μπορεί να συνδυάσει το φυσικό UAV, την ESP32-CAM, την επεξεργασία εικόνας στον υπολογιστή, την αποστολή δεδομένων προς το ArduPilot και την κινητή πλατφόρμα προσγείωσης σε μία ενιαία πειραματική διάταξη. Παράλληλα, το φυσικό πείραμα επιβεβαίωσε τα βασικά συμπεράσματα της προσομοίωσης, δηλαδή ότι το μεγάλο marker είναι κατάλληλο για αρχική αναγνώριση και ευθυγράμμιση, ενώ το μικρό marker είναι καταλληλότερο για την τελική φάση προσέγγισης.

Συνολικά, το φυσικό πείραμα επιβεβαίωσε τη λειτουργικότητα της αρχιτεκτονικής που αναπτύχθηκε. Παρότι η διαδικασία απαιτεί προσεκτική προετοιμασία, σωστή βαθμονόμηση, αξιόπιστη επικοινωνία και σταθερή πτητική συμπεριφορά, η τελική δοκιμή έδειξε ότι η συνεργασία UAV, μηχανικής όρασης και κινητής πλατφόρμας μπορεί να υλοποιηθεί πρακτικά σε πειραματικό επίπεδο.

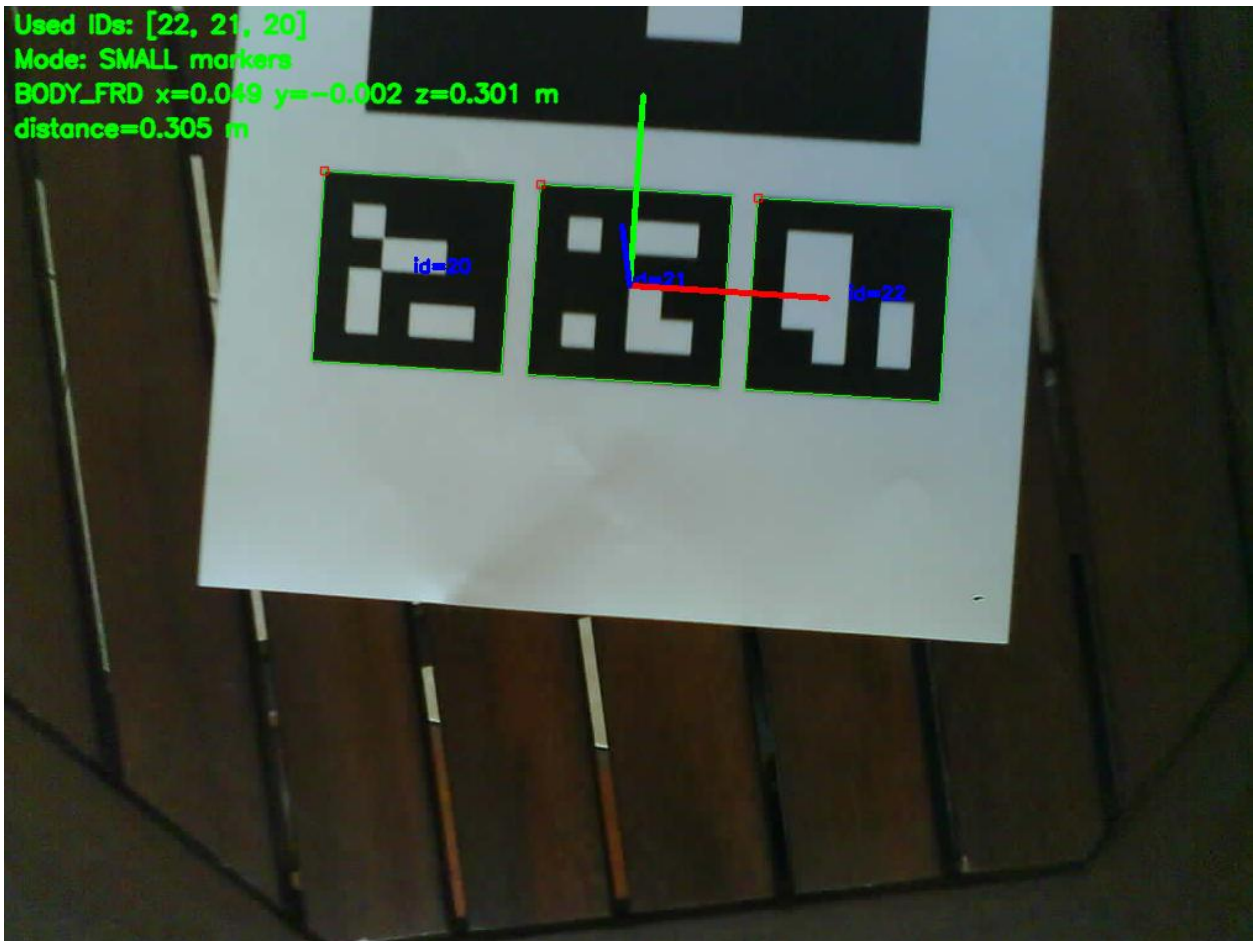
Ακολουθούν φωτογραφίες από την κάμερα του UAV καθώς και τις μετρήσεις του υπολογιστή:



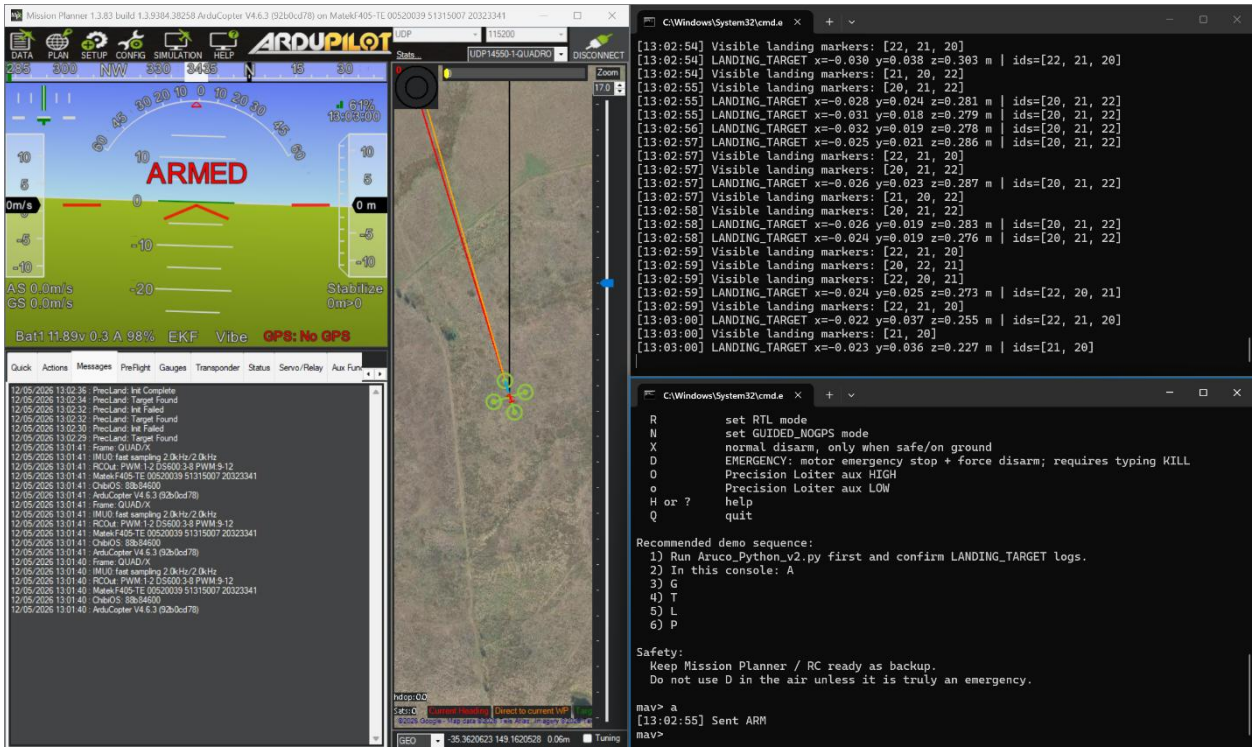
Εικόνα 6.5 Το UAV σε ύψος 0,6 μέτρα κατά τη διάρκεια της προσγείωσης.



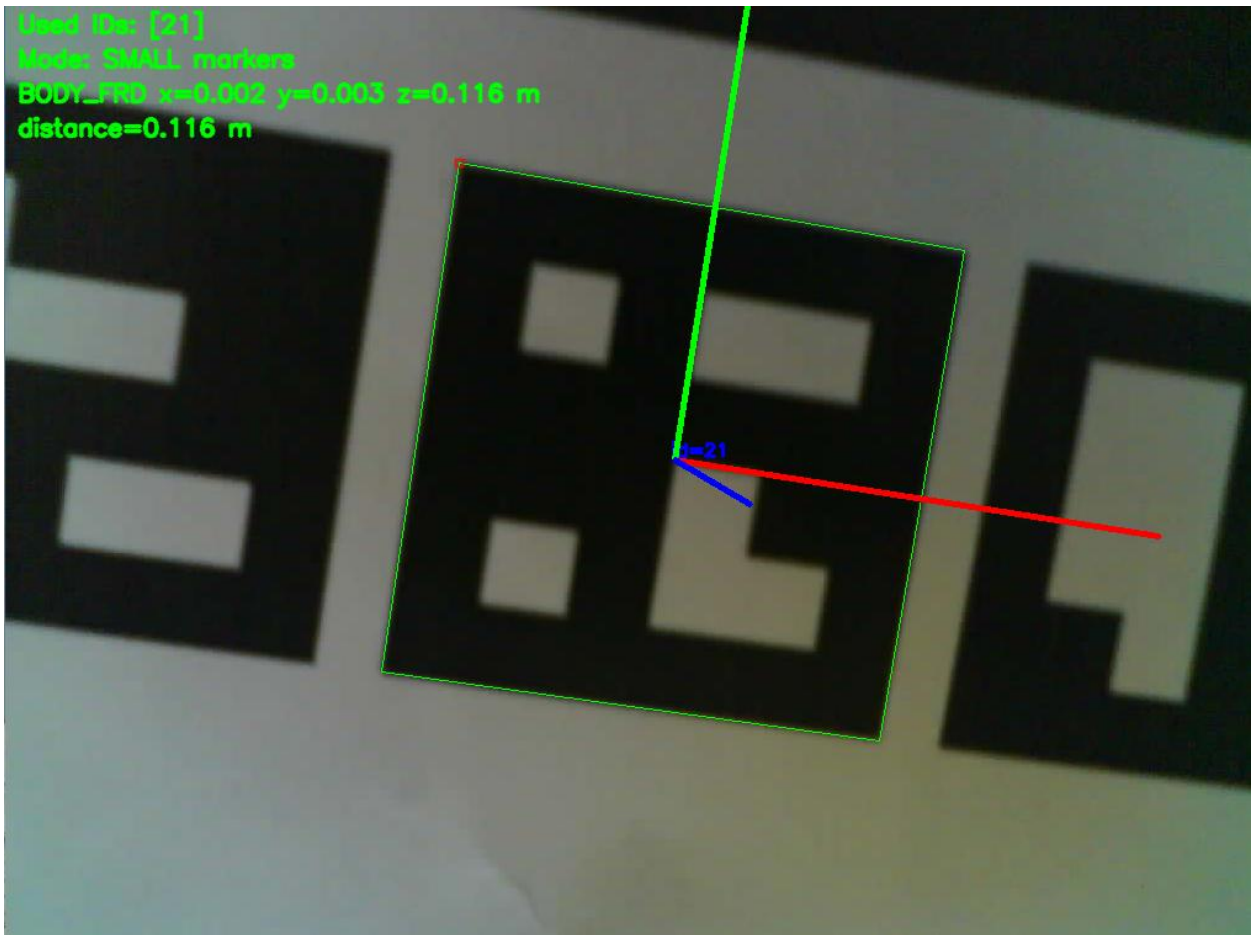
Εικόνα 6.6 Οι μετρήσεις του υπολογιστή σε ύψος 0,4 μέτρα.



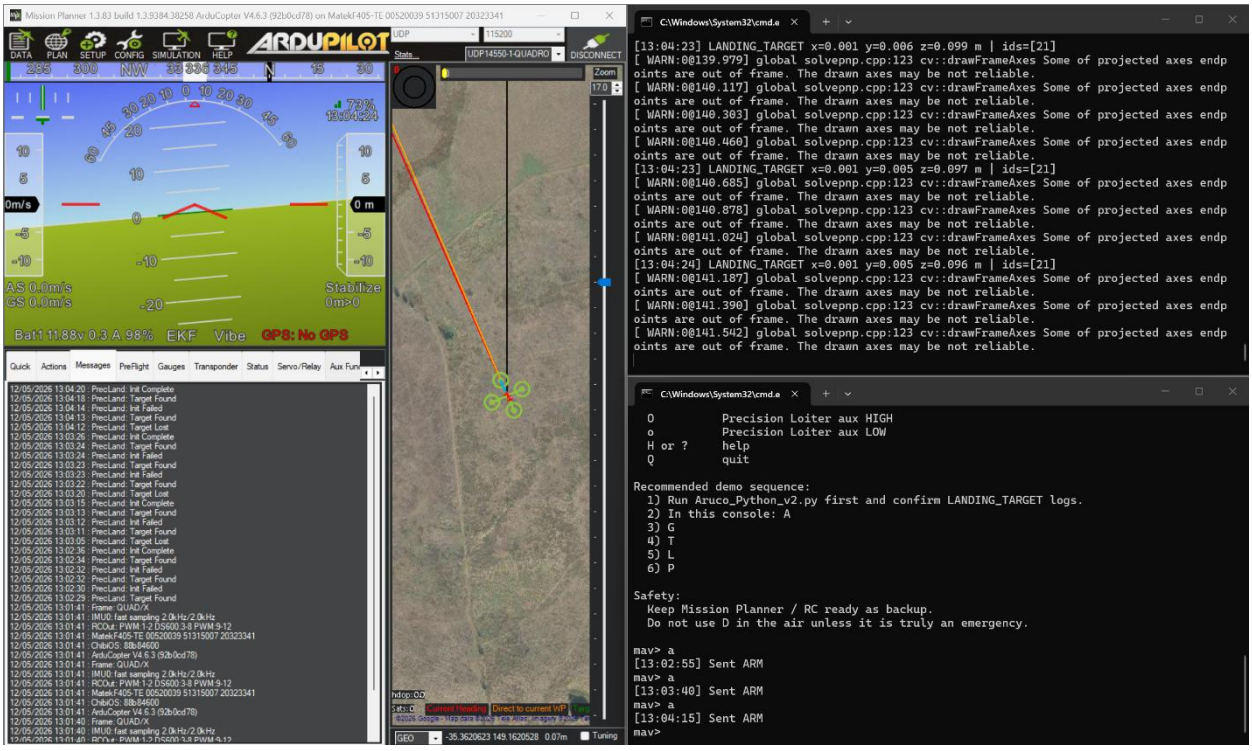
Εικόνα 6.7 Το UAV σε ύψος 0,3 μέτρα κατά τη διάρκεια της προσγείωσης.



Εικόνα 6.8 Οι μετρήσεις του υπολογιστή σε ύψος 0,2 μέτρα.



Εικόνα 6.9 Το UAV σε ύψος 0,1 μέτρα κατά τη διάρκεια της προσγείωσης.



Εικόνα 6.10 Οι μετρήσεις του υπολογιστή σε ύψος 0,09 μέτρα.

7. Συμπέρασμα

Η παρούσα πτυχιακή εργασία είχε ως αντικείμενο τον σχεδιασμό, την κατασκευή και την πειραματική διερεύνηση ενός ολοκληρωμένου συστήματος αυτόνομης προσγείωσης τετρακοπτέρου σε κινητή και εξισορροπούμενη πλατφόρμα, αξιοποιώντας τεχνικές μηχανικής όρασης. Μέσα από την εργασία συνδυάστηκαν επιμέρους γνωστικά αντικείμενα, όπως η ρομποτική, τα μη επανδρωμένα εναέρια οχήματα, τα συστήματα αυτόματου ελέγχου, η επεξεργασία εικόνας, η τρισδιάστατη σχεδίαση και εκτύπωση, καθώς και ο προγραμματισμός ενσωματωμένων συστημάτων [33], [57], [59].

Αρχικά, πραγματοποιήθηκε θεωρητική και μαθηματική ανάλυση των βασικών υποσυστημάτων. Αναλύθηκε η δυναμική και κινηματική συμπεριφορά του τετρακοπτέρου, με έμφαση στις γωνίες Euler, στις δυνάμεις των ροτόρων και στις εισόδους ελέγχου που καθορίζουν την κίνηση και τη σταθεροποίησή του. Παράλληλα, μελετήθηκε το όχημα διαφορικής οδήγησης, καθώς και οι κινηματικοί περιορισμοί που διέπουν την κίνησή του στο επίπεδο. Η θεωρητική αυτή προσέγγιση αποτέλεσε τη βάση για την κατανόηση της λειτουργίας του συνολικού συστήματος και για τη μετέπειτα πρακτική υλοποίηση.

Στο κατασκευαστικό μέρος, αναπτύχθηκε ένα πρωτότυπο σύστημα αποτελούμενο από δύο βασικές μονάδες: το τετρακόπτερο και το επίγειο όχημα με την κινούμενη πλατφόρμα προσγείωσης. Η πλατφόρμα σχεδιάστηκε ώστε να μπορεί να μεταβάλλει την κλίση της και να διατηρείται όσο το δυνατόν πιο οριζόντια, αξιοποιώντας δεδομένα από αισθητήρα αδρανειακής μέτρησης και κατάλληλη οδήγηση σερβοκινητήρων. Η χρήση του Arduino Uno R4 WiFi, του οδηγού PCA9685 και του αισθητήρα MPU6050 επέτρεψε την υλοποίηση ενός λειτουργικού συστήματος ελέγχου της πλατφόρμας, ενώ η τρισδιάστατη εκτύπωση συνέβαλε στη δημιουργία προσαρμοσμένων μηχανικών μερών με βάση τις ανάγκες της εφαρμογής.

Ιδιαίτερη σημασία είχε η ανάπτυξη του συστήματος μηχανικής όρασης. Η χρήση ArUco markers επέτρεψε την αναγνώριση της πλατφόρμας και την εκτίμηση της σχετικής θέσης του UAV ως προς αυτήν. Μέσω διαδικασίας βαθμονόμησης με ChArUco board και κατάλληλων Python scripts, κατέστη δυνατή η σύνδεση των μετρήσεων εικόνας με φυσικές αποστάσεις, γεγονός απαραίτητο για την εφαρμογή ακριβέστερης προσέγγισης και προσγείωσης. Το σύστημα λήψης εικόνας μέσω ESP32-CAM και η επεξεργασία των δεδομένων στον υπολογιστή αποτέλεσαν μία πρακτική και χαμηλού κόστους λύση για την υλοποίηση της αναγνώρισης και παρακολούθησης του στόχου.

Από την ανάπτυξη και τις δοκιμές του συστήματος προέκυψε ότι η αυτόνομη προσγείωση σε κινητή πλατφόρμα αποτελεί ένα σύνθετο πρόβλημα, καθώς απαιτεί τον συντονισμό πολλών διαφορετικών υποσυστημάτων. Η σταθερότητα του τετρακοπτέρου, η ακρίβεια της κάμερας, η ποιότητα της βαθμονόμησης, η καθυστέρηση στη μετάδοση εικόνων, η αξιοπιστία της ανίχνευσης του marker και η μηχανική σταθερότητα της πλατφόρμας επηρεάζουν άμεσα το τελικό αποτέλεσμα. Παρά τις τεχνικές δυσκολίες που παρουσιάστηκαν, η εργασία απέδειξε ότι είναι εφικτή η δημιουργία μιας πειραματικής διάταξης η οποία συνδυάζει UAV, κινητή ρομποτική βάση και μηχανική όραση για την προσέγγιση του προβλήματος της αυτόνομης προσγείωσης.

Σημαντικό συμπέρασμα της εργασίας είναι ότι τα συστήματα χαμηλού κόστους, όπως οι μικροελεγκτές, οι απλές κάμερες και οι ανοικτές πλατφόρμες λογισμικού, μπορούν να χρησιμοποιηθούν αποτελεσματικά για την ανάπτυξη σύνθετων ρομποτικών εφαρμογών. Ωστόσο, η επίτευξη υψηλής ακρίβειας και αξιοπιστίας απαιτεί προσεκτική ρύθμιση, επαναλαμβανόμενες δοκιμές και σωστή συνεργασία μεταξύ υλικού και

λογισμικού. Η μετάβαση από τη θεωρητική ανάλυση στην πραγματική υλοποίηση ανέδειξε πρακτικά ζητήματα τα οποία δεν είναι πάντα εμφανή στο στάδιο του σχεδιασμού, όπως οι μηχανικές ανοχές, οι καθυστερήσεις επικοινωνίας, οι περιορισμοί ισχύος και η ευαισθησία των αισθητήρων.

Συνολικά, η εργασία πέτυχε τον βασικό της στόχο, δηλαδή την ανάπτυξη και τεκμηρίωση ενός ολοκληρωμένου πρωτοτύπου για αυτόνομη προσγείωση τετρακοπτέρου σε εξισορροπούμενη κινητή πλατφόρμα με χρήση μηχανικής όρασης. Το αποτέλεσμα αποτελεί μία λειτουργική βάση για περαιτέρω έρευνα και βελτίωση, καθώς μπορεί να επεκταθεί με πιο εξελιγμένους αλγορίθμους ελέγχου, ταχύτερη επεξεργασία εικόνας, καλύτερη εκτίμηση θέσης, χρήση επιπλέον αισθητήρων και πληρέστερη ενσωμάτωση των εντολών προσγείωσης στον ελεγκτή πτήσης. Μελλοντικά, το σύστημα θα μπορούσε να εξελιχθεί ώστε να λειτουργεί με μεγαλύτερη αυτονομία, υψηλότερη ακρίβεια και αυξημένη αξιοπιστία σε πραγματικές συνθήκες, αποτελώντας μία ολοκληρωμένη ρομποτική εφαρμογή συνεργασίας εναέριου και επίγειου οχήματος.

8. Κώδικες

8.1 Κώδικας ESP32-CAM – C++

```
#include <Arduino.h>
#include <WiFi.h>
#include <WebServer.h>
#include "esp_camera.h"

// USER CONFIG

// Wi-Fi network the ESP32-CAM will join.
const char* WIFI_SSID = "DroneBridge for ESP32";
const char* WIFI_PASS = "dronebridge";
IPAddress local_IP(192, 168, 2, 10);
IPAddress gateway(192, 168, 2, 1);
IPAddress subnet(255, 255, 255, 0);
IPAddress dns1(192, 168, 2, 1);
IPAddress dns2(8, 8, 8, 8);

// Camera capture settings
framesize_t CAMERA_FRAME_SIZE = FRAMESIZE_XGA;  // 1024x768
int CAMERA_JPEG_QUALITY = 12;                   // lower = better quality

// AI-THINKER ESP32-CAM PINOUT
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27
#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22

// GLOBALS
WebServer server(80);

// HELPERS
void sendCorsHeaders() {
  server.sendHeader("Access-Control-Allow-Origin", "*");
  server.sendHeader("Cache-Control", "no-store, no-cache, must-revalidate");
```

```
}

void sendBinaryResponse(int code, const char* contentType, const uint8_t* data,
size_t len) {
  sendCorsHeaders();
  server.setContentLength(len);
  server.send(code, contentType, "");
  WiFiClient client = server.client();
  if (data && len > 0) {
    client.write(data, len);
  }
}

bool initCamera() {
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;

  if (psramFound()) {
    config.frame_size = CAMERA_FRAME_SIZE;
    config.jpeg_quality = CAMERA_JPEG_QUALITY;
    config.fb_count = 2;
    config.grab_mode = CAMERA_GRAB_LATEST;
    config.fb_location = CAMERA_FB_IN_PSRAM;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 14;
    config.fb_count = 1;
    config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
    config.fb_location = CAMERA_FB_IN_DRAM;
```

```
  }

  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("[CAM] init failed: 0x%x\n", err);
    return false;
  }

  sensor_t* s = esp_camera_sensor_get();
  if (s) {
    s->set_brightness(s, 1);
    s->set_contrast(s, 0);
    s->set_saturation(s, 0);
    s->set_whitebal(s, 1);
    s->set_awb_gain(s, 1);
    s->set_exposure_ctrl(s, 1);    // auto exposure ON
    s->set_gain_ctrl(s, 1);        // auto gain ON
    s->set_aec2(s, 1);             // improved exposure control
    s->set_ae_level(s, 1);         // range is effectively around -3..3
    s->set_framesize(s, psramFound() ? CAMERA_FRAME_SIZE : FRAMESIZE_SVGA);
  }

  Serial.println("[CAM] init ok");
  return true;
}

void connectWiFi() {
  WiFi.mode(WIFI_STA);
  WiFi.setSleep(false);

  if (!WiFi.config(local_IP, gateway, subnet, dns1, dns2)) {
    Serial.println("[WIFI] Static IP config failed");
  }

  WiFi.begin(WIFI_SSID, WIFI_PASS);

  Serial.printf("[WIFI] connecting to %s", WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("[WIFI] connected, IP = ");
  Serial.println(WiFi.localIP());
}

// HTTP HANDLERS
void handleHealth() {
```

```
  String json = "{";
  json += "\"ok\":true,";
  json += "\"ip\":\"" + WiFi.localIP().toString() + "\"";
  json += "}";
  sendCorsHeaders();
  server.send(200, "application/json", json);
}

void handleCaptureJpg() {
  camera_fb_t* fb = esp_camera_fb_get();
  if (!fb) {
    sendCorsHeaders();
    server.send(500, "text/plain", "capture failed");
    return;
  }

  sendCorsHeaders();
  server.setContentLength(fb->len);
  server.send(200, "image/jpeg", "");

  WiFiClient client = server.client();
  client.write(fb->buf, fb->len);

  esp_camera_fb_return(fb);
}

void handleRoot() {
  String msg;
  msg += "ESP32-CAM Camera\n";
  msg += "GET /health\n";
  msg += "GET /capture.jpg\n";
  msg += "\n";
  msg += "ESP32 IP: " + WiFi.localIP().toString() + "\n";
  sendCorsHeaders();
  server.send(200, "text/plain", msg);
}

void setupHttp() {
  server.on("/", HTTP_GET, handleRoot);
  server.on("/health", HTTP_GET, handleHealth);
  server.on("/capture.jpg", HTTP_GET, handleCaptureJpg);

  server.onNotFound([]() {
    sendCorsHeaders();
    server.send(404, "text/plain", "not found");
  });

  server.begin();
```

```
  Serial.println("[HTTP] server started");
}

// SETUP / LOOP
void setup() {
  delay(1000);

  Serial.begin(115200);
  Serial.println();
  Serial.println("ESP32-CAM Camera starting...");

  connectWiFi();

  if (!initCamera()) {
    Serial.println("Camera init failed; restarting in 5 seconds...");
    delay(5000);
    ESP.restart();
  }

  setupHttp();

  Serial.println("Ready.");
  Serial.print("Health: http://");
  Serial.print(WiFi.localIP());
  Serial.println("/health");

  Serial.print("Capture: http://");
  Serial.print(WiFi.localIP());
  Serial.println("/capture.jpg");
}

void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("[WIFI] Disconnected, reconnecting...");
    WiFi.disconnect();
    WiFi.begin(WIFI_SSID, WIFI_PASS);

    unsigned long start = millis();
    while (WiFi.status() != WL_CONNECTED && millis() - start < 10000) {
      delay(500);
      Serial.print(".");
    }
    Serial.println();

    if (WiFi.status() == WL_CONNECTED) {
      Serial.print("[WIFI] Reconnected, IP = ");
      Serial.println(WiFi.localIP());
    } else {
```

```
      Serial.println("[WIFI] Reconnect failed");
    }
  }

  server.handleClient();
  delay(1);
}
```

8.2 Κώδικας Οχήματος Εδάφους – C++

```
#include <Wire.h>                       //Arduino Wire Library - Handles I2C
Communication - SDA and SCL Pins10
#include <Adafruit_PWMServoDriver.h>    //PCA9685 (Servo Driver) Library
#include <WiFiS3.h>
#include <math.h>
#include <Arduino_LED_Matrix.h>

// WiFi Connection Settings
const char* ssid = "dfansumu";
const char* pass = "dfansumu";
WiFiServer server(80);  //Web Server Object "server" on Port 80, for the webpage

// LED Settings
ArduinoLEDMatrix matrix;

const uint32_t ALL_ON_FRAME[3] = {
  0xFFFFFFFF,
  0xFFFFFFFF,
  0xFFFFFFFF
};

// PCA9685 (Servo Driver)
Adafruit_PWMServoDriver pwm(0x40); //"Names" the PCA9685 as "pwm" for future use,
sets its I2C address as 0x40

// Servo Cannels - Names the I2C channels according to the Servo that is attached
to them
#define WHEEL_LEFT       0
#define WHEEL_RIGHT     15
#define PLATFORM_LEFT    4
#define PLATFORM_RIGHT  11

// Continuous servo pulse values - in microseconds. 1500 is stop, >1500 is
Counter Clockwise rotation, <1500 is Clockwise
#define STOP_WHEELS 1500
#define FWD_LEFT    1750 //Forward
#define FWD_RIGHT   1300
#define REV_LEFT    1300 //Reverse
```

```
#define REV_RIGHT   1790

// PLATFORM SERVO CALIBRATION - Effectively sets minimum and maximum "positions"
for the platform servos 600 for 0 degrees, 2400 for 180
static int PLATFORM_MIN_LEFT      =   600;
static int PLATFORM_MAX_LEFT      =  1000;
static int PLATFORM_MIN_RIGHT     =   750;
static int PLATFORM_MAX_RIGHT     =  1150;
static int PlatformLastDeg  =     0;
static int PLATFORM_MAX_DEG =    35;

// Platform SMOOTH MOTION
int PlatformCurrentDeg                    =  0; //Stores current Degrees, starts
at 0
int PlatformTargetDeg                     =  0; //Stores target Degrees
unsigned long PlatformLastStepMs          =  0; //Stores when last movement
occured
const unsigned long Platform_STEP_DELAY_MS =  10; //Sets delay between movements
const int Platform_STEP_DEG               =  1; //Sets number of degrees per
movement
unsigned long PlatformLastRefreshMs       =  0;
const unsigned long Platform_REFRESH_MS    = 500;

struct GyroOffsets {
  long gyro_x_cal;
  long gyro_y_cal;
  long gyro_z_cal;
};
struct GyroVectors {
  int16_t gyro_x;
  int16_t gyro_y;
  int16_t gyro_z;
};
GyroVectors gV;
GyroOffsets gO = {0, 0, 0};

long acc_x = 0, acc_y = 0, acc_z = 0;
bool imuOk = false;
bool set_gyro_angles = false;

float angle_pitch_acc = 0.0f;
float angle_roll_acc  = 0.0f;
float angle_pitch     = 0.0f;
float angle_roll      = 0.0f;
float angle_pitch_output = 0.0f;
float angle_roll_output  = 0.0f;
unsigned long imuLoopTimerUs = 0;
```

```
// AUTO-LEVELLING STATE
static bool autoLevelEnabled = false;          // Auto-level stays OFF until Zero
is set and user enables it

static const float AUTO_LEVEL_GAIN = 0.12f;     // Servo degrees per measured
vehicle pitch degree
static const float AUTO_LEVEL_DEADBAND_DEG = 0.15f; // Smaller dead zone, so
auto-level starts earlier
static const int AUTO_LEVEL_SIGN = 1;          // Change to -1 if the platform
moves the wrong way
static const float AUTO_LEVEL_MAX_STEP_DEG = 0.5f;
static const float AUTO_LEVEL_MIN_STEP_DEG = 0.25f; // Minimum useful correction
after leaving deadband

static const unsigned long AUTO_LEVEL_UPDATE_MS = 50;
static unsigned long autoLevelLastMs = 0;

static float autoLevelTargetDeg = 0.0f;

static float zeroX       =  0.0f; //Sets current X as 0 when levelling
static float zeroY       =  0.0f; //Sets current Y as 0 when levelling
static bool zeroCaptured = false; //Stores if levelling has occured, angle logic
doesn't use above values until "true"

// HELPERS
int clampInt(int v, int lo, int hi) { //Function to keep integer "v" within "lo"
and "hi" range.
  if (v < lo) return lo;
  if (v > hi) return hi;
  return v;
}

float clampFloat(float v, float lo, float hi) { //Same but for float numbers
  if (v < lo) return lo;
  if (v > hi) return hi;
  return v;
}

// SERVO UTILITIES
void setServoUs(uint8_t ch, int us) { //Function to limit Servo us in given range
and convert us to Driver Ticks for Wheels and send the ticks to the Wheel Servos
  us = clampInt(us, 900, 2100);
  int ticks = us * 4096L / 20000L;
  pwm.setPWM(ch, 0, ticks);
}

uint16_t usToTicks50Hz(int us) { //Function to limit Servo us in given range and
convert us to Driver Ticks for Platform
```

```
  us = clampInt(us, 500, 2500);
  long ticks = (long)us * 4096L / 20000L;
  return (uint16_t)clampInt((int)ticks, 0, 4095);
}

int degToUsPlatformLeft(int degL) { // Converts a physical servo angle to
microseconds
  degL = clampInt(degL, 0, PLATFORM_MAX_DEG);
  long usL = PLATFORM_MIN_LEFT + (long)(PLATFORM_MAX_LEFT - PLATFORM_MIN_LEFT) *
degL / PLATFORM_MAX_DEG;
  return (int)usL;
}

int degToUsPlatformRight(int degR) {
  degR = clampInt(degR, 0, PLATFORM_MAX_DEG);
  long usR = PLATFORM_MIN_RIGHT + (long)(PLATFORM_MAX_RIGHT - PLATFORM_MIN_RIGHT)
* degR / PLATFORM_MAX_DEG;
  return (int)usR;
}

void setPlatformLeftDeg(int degL) {
  int usL = degToUsPlatformLeft(degL);
  pwm.setPWM(PLATFORM_LEFT, 0, usToTicks50Hz(usL));
}

void setPlatformRightDeg(int degR) {
  int usR = degToUsPlatformRight(degR);
  pwm.setPWM(PLATFORM_RIGHT, 0, usToTicks50Hz(usR));
}

void applyPlatformMirrorDeg(int degL) {
  degL = clampInt(degL, 0, PLATFORM_MAX_DEG);
  int degR = PLATFORM_MAX_DEG - degL;

  setPlatformLeftDeg(degL);
  setPlatformRightDeg(degR);

  PlatformLastDeg = degL;
}

void updatePlatformSmooth() {
  unsigned long now = millis();

  // When the platform is already at the target, keep re-sending the last
position
  if (PlatformCurrentDeg == PlatformTargetDeg) {
    if (now - PlatformLastRefreshMs >= Platform_REFRESH_MS) {
      applyPlatformMirrorDeg(PlatformCurrentDeg);
```

```
      PlatformLastRefreshMs = now;
    }
    return;
  }

  // Smooth movement delay
  if (now - PlatformLastStepMs < Platform_STEP_DELAY_MS) return;

  if (PlatformCurrentDeg < PlatformTargetDeg) {
    PlatformCurrentDeg += Platform_STEP_DEG;
  } else {
    PlatformCurrentDeg -= Platform_STEP_DEG;
  }

  PlatformCurrentDeg = clampInt(PlatformCurrentDeg, 0, PLATFORM_MAX_DEG);

  applyPlatformMirrorDeg(PlatformCurrentDeg);

  PlatformLastStepMs = now;
  PlatformLastRefreshMs = now;
}

// ROVER MOVEMENT
void stopRover() {
  setServoUs(WHEEL_LEFT, STOP_WHEELS);
  setServoUs(WHEEL_RIGHT, STOP_WHEELS);
}

void moveForward() {
  setServoUs(WHEEL_LEFT, FWD_LEFT);
  setServoUs(WHEEL_RIGHT, FWD_RIGHT);
}

void moveBackward() {
  setServoUs(WHEEL_LEFT, REV_LEFT);
  setServoUs(WHEEL_RIGHT, REV_RIGHT);
}

void turnLeft() {
  setServoUs(WHEEL_LEFT, STOP_WHEELS);
  setServoUs(WHEEL_RIGHT, FWD_RIGHT);
}

void turnRight() {
  setServoUs(WHEEL_LEFT, FWD_LEFT);
  setServoUs(WHEEL_RIGHT, STOP_WHEELS);
}
```

```
// REQUEST PARSING
int getQueryInt(const String& req, const String& key, int defVal) { //Checks and
read given "Degrees", falls back to PlatformLastDeg
  int q = req.indexOf('?');
  if (q < 0) return defVal;
  int k = req.indexOf(key + "=", q);
  if (k < 0) return defVal;
  k += key.length() + 1;
  int end = req.indexOf('&', k);
  if (end < 0) end = req.indexOf(' ', k);
  if (end < 0) end = req.length();
  String val = req.substring(k, end);
  val.trim();
  if (val.length() == 0) return defVal;
  return val.toInt();
}

String getPathFromReqLine(const String& reqLine) { //Extra fallback in case HTTP
query cannot be located otherwise
  int a = reqLine.indexOf(' ');
  if (a < 0) return "/";
  int b = reqLine.indexOf(' ', a + 1);
  if (b < 0) return "/";
  return reqLine.substring(a + 1, b);
}

// LEVELLING
void configureMPU6050() {
  // Wake up
  Wire.beginTransmission(0x68);
  Wire.write(0x6B);
  Wire.write(0x00);
  Wire.endTransmission();

  // Accelerometer
  Wire.beginTransmission(0x68);
  Wire.write(0x1C);
  Wire.write(0x10);
  Wire.endTransmission();

  // Gyro
  Wire.beginTransmission(0x68);
  Wire.write(0x1B);
  Wire.write(0x08);
  Wire.endTransmission();
}

bool readMPU6050Raw() {
```

```
  Wire.beginTransmission(0x68);
  Wire.write(0x3B);
  if (Wire.endTransmission(false) != 0) return false;

  int n = Wire.requestFrom(0x68, 14);
  if (n != 14) return false;

  acc_x = (Wire.read() << 8) | Wire.read();
  acc_y = (Wire.read() << 8) | Wire.read();
  acc_z = (Wire.read() << 8) | Wire.read();
  Wire.read(); Wire.read(); // temperature bytes not used
  gV.gyro_x = (Wire.read() << 8) | Wire.read();
  gV.gyro_y = (Wire.read() << 8) | Wire.read();
  gV.gyro_z = (Wire.read() << 8) | Wire.read();

  return true;
}

void resetImuFilterState() {
  set_gyro_angles = false;
  angle_pitch_acc = 0.0f;
  angle_roll_acc  = 0.0f;
  angle_pitch     = 0.0f;
  angle_roll      = 0.0f;
  angle_pitch_output = 0.0f;
  angle_roll_output  = 0.0f;
}

void calibrateGyroOffsets() {
  long sx = 0, sy = 0, sz = 0;
  int good = 0;

  resetImuFilterState();

  for (int i = 0; i < 2000; i++) {
    if (readMPU6050Raw()) {
      sx += gV.gyro_x;
      sy += gV.gyro_y;
      sz += gV.gyro_z;
      good++;
    }
    delay(3);
  }

  if (good > 0) {
    gO.gyro_x_cal = sx / good;
    gO.gyro_y_cal = sy / good;
    gO.gyro_z_cal = sz / good;
```

```
  } else {
    g0.gyro_x_cal = 0;
    g0.gyro_y_cal = 0;
    g0.gyro_z_cal = 0;
  }

  resetImuFilterState();
  imuLoopTimerUs = micros();
}

void imuTick() {
  if (!imuOk) return;

  unsigned long now = micros();
  if ((long)(now - imuLoopTimerUs) < 4000) return; // ~250Hz
  float dt = (now - imuLoopTimerUs) * 1.0e-6f;
  imuLoopTimerUs = now;

  if (!readMPU6050Raw()) return;

  float gx = ((float)gV.gyro_x - (float)g0.gyro_x_cal) / 65.5f;
  float gy = ((float)gV.gyro_y - (float)g0.gyro_y_cal) / 65.5f;
  float gz = ((float)gV.gyro_z - (float)g0.gyro_z_cal) / 65.5f;

  angle_pitch += gx * dt;
  angle_roll  += gy * dt;

  float yawRad = gz * dt * PI / 180.0f;
  float pitch_before = angle_pitch;
  angle_pitch += angle_roll * sinf(yawRad);
  angle_roll  -= pitch_before * sinf(yawRad);

  float acc_total = sqrtf((float)acc_x * (float)acc_x +
                          (float)acc_y * (float)acc_y +
                          (float)acc_z * (float)acc_z);
  if (acc_total < 1.0f) return;

  angle_pitch_acc = asinf(clampFloat((float)acc_y / acc_total, -1.0f, 1.0f)) *
57.29578f;
  angle_roll_acc  = asinf(clampFloat((float)acc_x / acc_total, -1.0f, 1.0f)) * -
57.29578f;

  if (set_gyro_angles) {
    angle_pitch = angle_pitch * 0.9996f + angle_pitch_acc * 0.0004f;
    angle_roll  = angle_roll  * 0.9996f + angle_roll_acc  * 0.0004f;
  } else {
    angle_pitch = angle_pitch_acc;
    angle_roll  = angle_roll_acc;
```

```
    set_gyro_angles = true;
  }

  angle_pitch_output = angle_pitch_output * 0.90f + angle_pitch * 0.10f;
  angle_roll_output  = angle_roll_output  * 0.90f + angle_roll  * 0.10f;
}

bool readTilt(float &pitchDeg, float &rollDeg) {
  imuTick();
  if (!set_gyro_angles) return false;

  pitchDeg = angle_pitch_output;
  rollDeg  = angle_roll_output;
  return true;
}

void zeroLevelNow() { //This Function reads and adds up up to 20 IMU reads,
calculates the average and sets that as the new "zero" aka level. Also updates
the Web Page dot display to be level.
  float pitchDeg, rollDeg;
  float sx = 0.0f, sy = 0.0f;
  int count = 0;

  for (int i = 0; i < 20; i++) {
    if (readTilt(pitchDeg, rollDeg)) {
      sx += pitchDeg;
      sy += rollDeg;
      count++;
    }
    delay(10);
  }

  if (count > 0) {
    zeroX = sx / count;
    zeroY = sy / count;
    zeroCaptured = true;
  }
}

void setAutoLevelEnabled(bool enabled) {
  autoLevelEnabled = enabled;
  autoLevelLastMs = 0;

  // Start auto-level from the current platform position.
  if (enabled) {
    autoLevelTargetDeg = (float)PlatformCurrentDeg;
    PlatformTargetDeg = PlatformCurrentDeg;
  }
```

```
}

void updateAutoLevel() {
  if (!autoLevelEnabled || !imuOk || !zeroCaptured) return;

  unsigned long now = millis();
  if (now - autoLevelLastMs < AUTO_LEVEL_UPDATE_MS) return;
  autoLevelLastMs = now;

  float pitchDeg, rollDeg;
  if (!readTilt(pitchDeg, rollDeg)) return;

  // vehicle/front platform direction = MPU -X
  // vehicle right = MPU +Y
  float levelErrorDeg = rollDeg - zeroY;

  // If the platform is level enough, HOLD the current angle.
  if (fabs(levelErrorDeg) <= AUTO_LEVEL_DEADBAND_DEG) {
    autoLevelTargetDeg = (float)PlatformCurrentDeg;
    PlatformTargetDeg = PlatformCurrentDeg;
    return;
  }

  // Incremental correction.

  float correctionStep = AUTO_LEVEL_GAIN * levelErrorDeg;

  if (fabs(correctionStep) < AUTO_LEVEL_MIN_STEP_DEG) {
    correctionStep = (correctionStep >= 0.0f) ? AUTO_LEVEL_MIN_STEP_DEG : -
AUTO_LEVEL_MIN_STEP_DEG;
  }

  correctionStep *= AUTO_LEVEL_SIGN;

  // Limit each correction step so the platform does not jump.
  correctionStep = clampFloat(
    correctionStep,
    -AUTO_LEVEL_MAX_STEP_DEG,
    AUTO_LEVEL_MAX_STEP_DEG
  );

  autoLevelTargetDeg += correctionStep;
  autoLevelTargetDeg = clampFloat(autoLevelTargetDeg, 0.0f,
(float)PLATFORM_MAX_DEG);

  PlatformTargetDeg = (int)roundf(autoLevelTargetDeg);
}
```

```
// TEXT ENDPOINT
void sendCalibrateText(WiFiClient& client, const char* msg) { //Sends provided
message text
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/plain; charset=utf-8");
  client.println("Connection: close");
  client.println();
  client.print(msg);
}

// HTML PAGE
void sendPage(WiFiClient& client) {
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html; charset=utf-8");
  client.println("Connection: close");
  client.println();

  client.println(R"HTML(
<!DOCTYPE html>
<html lang="el">
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Rover Control</title>
<style>
  :root{
    --bg:#0b1020;
    --text:#eef2ff;
    --muted:#a7b0d6;
    --shadow: 0 10px 30px rgba(0,0,0,.35);
    --radius: 18px;
  }
  *{ box-sizing:border-box; }
  body{
    margin:0;
    min-height:100vh;
    display:flex;
    align-items:center;
    justify-content:center;
    background: rgba(0,0,0);
    color:var(--text);
    font-family: system-ui, -apple-system, Segoe UI, Roboto, Arial, sans-serif;
    padding: 20px;
  }
  .wrap{ width:min(720px, 100%); }
  .header{ text-align:center; margin-bottom: 14px; }
  .title{
    font-weight: 800;
    letter-spacing: .08em;
```

```
    text-transform: uppercase;
    font-size: clamp(22px, 4.5vw, 34px);
    margin: 0;
  }
  .subtitle{
    margin: 6px 0 0;
    font-size: clamp(14px, 2.6vw, 18px);
    color: var(--muted);
    letter-spacing: .06em;
    text-transform: uppercase;
  }
  .card{
    background: linear-gradient(180deg, rgba(255,255,255,.06),
rgba(255,255,255,.03));
    border: 1px solid rgba(255,255,255,.10);
    border-radius: var(--radius);
    box-shadow: var(--shadow);
    padding: 18px;
    margin-bottom: 14px;
  }
  .topRow{
    display:flex;
    gap: 12px;
    align-items: center;
    justify-content: space-between;
    flex-wrap: wrap;
    margin-bottom: 14px;
  }
  .badge{
    padding: 8px 12px;
    border-radius: 999px;
    background: rgba(124,58,237,.18);
    border: 1px solid rgba(124,58,237,.35);
    color: var(--text);
    font-size: 13px;
    letter-spacing: .03em;
  }
  .status{
    padding: 8px 12px;
    border-radius: 999px;
    background: rgba(255,255,255,.06);
    border: 1px solid rgba(255,255,255,.12);
    color: var(--muted);
    font-size: 13px;
  }
  .grid{
    display:grid;
    grid-template-columns: 1fr 1fr 1fr;
```

```
    gap: 12px;
    margin-top: 10px;
  }
  .btn{
    border: 1px solid rgba(255,255,255,.14);
    background: linear-gradient(180deg, rgba(255,255,255,.07),
rgba(255,255,255,.03));
    color: var(--text);
    border-radius: 16px;
    height: 72px;
    font-size: 18px;
    font-weight: 700;
    cursor: pointer;
    box-shadow: 0 8px 18px rgba(0,0,0,.25);
    transition: transform .06s ease, background .15s ease, border-color .15s
ease;
    user-select:none;
    -webkit-tap-highlight-color: transparent;
  }
  .btn:hover{ background: rgba(255,255,255,.09); border-color:
rgba(255,255,255,.22); }
  .btn:active{ transform: translateY(1px) scale(.99); }
  .btnStop{
    background: linear-gradient(180deg, rgba(239,68,68,.22),
rgba(239,68,68,.10));
    border-color: rgba(239,68,68,.35);
  }
  .btnFwd{
    background: linear-gradient(180deg, rgba(34,197,94,.22),
rgba(34,197,94,.10));
    border-color: rgba(34,197,94,.35);
  }
  .btnBack{
    background: linear-gradient(180deg, rgba(59,130,246,.22),
rgba(59,130,246,.10));
    border-color: rgba(59,130,246,.35);
  }
  .hint{
    margin-top: 12px;
    color: var(--muted);
    font-size: 13px;
    text-align:center;
  }
  .fwd { grid-column: 2; grid-row: 1; }
  .left{ grid-column: 1; grid-row: 2; }
  .stop{ grid-column: 2; grid-row: 2; }
  .right{ grid-column: 3; grid-row: 2; }
  .back{ grid-column: 2; grid-row: 3; }
```

```
  .PlatformRow{
    display:flex;
    gap:12px;
    align-items:center;
    flex-wrap:wrap;
    margin-top: 10px;
  }
  .PlatformInput{
    width: 140px;
    height: 52px;
    border-radius: 14px;
    border: 1px solid rgba(255,255,255,.14);
    background: rgba(255,255,255,.06);
    color: var(--text);
    font-size: 16px;
    font-weight: 800;
    padding: 0 12px;
    outline: none;
  }
  .PlatformRange{
    width: 100%;
    margin-top: 10px;
  }
  .levelButtons{
    display:flex;
    gap:12px;
    flex-wrap:wrap;
    margin-top:12px;
    justify-content:center;
  }
  .miniBtn{
    height: 52px;
    border-radius: 16px;
    border: 1px solid rgba(255,255,255,.14);
    background: linear-gradient(180deg, rgba(255,255,255,.07),
rgba(255,255,255,.03));
    color: var(--text);
    font-weight: 900;
    padding: 0 14px;
    cursor:pointer;
  }
</style>
</head>
<body>

  <div class="wrap">
    <div class="header">
```

```
      <h1 class="title">ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ</h1>
      <div class="subtitle">ΦΑΝΟΥΡΓΑΚΗΣ ΦΡΟΝΙΜΟΣ</div>
    </div>

    <div class="card">
      <div class="topRow">
        <div class="badge">Rover Control Panel</div>
        <div id="status" class="status">Status: STOP</div>
      </div>

      <div class="grid">
        <button class="btn btnFwd fwd" onclick="send('f')">▲ ΜΠΡΟΣΤΑ</button>
        <button class="btn left" onclick="send('l')">◀ ΑΡΙΣΤΕΡΑ</button>
        <button class="btn btnStop stop" onclick="send('s')">■ STOP</button>
        <button class="btn right" onclick="send('r')">ΔΕΞΙΑ ▶</button>
        <button class="btn btnBack back" onclick="send('b')">▼ ΠΙΣΩ</button>
      </div>

    <div class="card">
      <div class="topRow">
        <div class="badge">Platform Control (FT5320M)</div>
        <div id="PlatformStatus" class="status">Platform: 0° / 35°</div>
      </div>

      <div class="PlatformRow">
        <input id="PlatformDeg" class="PlatformInput" type="number" min="0"
max="35" value="0">
        <button class="btn" style="height:56px;"
onclick="applyPlatform()">APPLY</button>
      </div>

      <input id="PlatformSlider" class="PlatformRange" type="range" min="0"
max="35" value="0"
             oninput="PlatformDeg.value=this.value;
updatePlatformLabelPreview(this.value);">

    <div class="card">
      <div class="topRow">
        <div class="badge">Auto Level</div>
        <div id="levelStatus" class="status">Auto-level: OFF</div>
      </div>

      <div class="levelButtons">
        <button class="miniBtn" onclick="calibrateLevel()">CALIBRATE</button>
        <button class="miniBtn" onclick="zeroLevel()">ZERO NOW</button>
        <button class="miniBtn" id="btnAutoLevel"
onclick="enableAutoLevel()">ENABLE AUTO LEVEL</button>
```

```
        <button class="miniBtn" onclick="disableAutoLevel()">DISABLE AUTO
LEVEL</button>
      </div>

      <div class="hint">Auto-level stays OFF until you press ZERO NOW and then
ENABLE AUTO LEVEL.</div>
    </div>
  </div>

<script>
const statusEl = document.getElementById('status');
const PlatformStatusEl = document.getElementById('PlatformStatus');
const PlatformDegEl = document.getElementById('PlatformDeg');
const PlatformSliderEl = document.getElementById('PlatformSlider');
const levelStatusEl = document.getElementById('levelStatus');
const btnAutoLevelEl = document.getElementById('btnAutoLevel');
const PLATFORM_MAX_DEG_UI = 35;

function label(cmd){
  switch(cmd){
    case 'f': return 'FORWARD';
    case 'b': return 'BACK';
    case 'l': return 'LEFT';
    case 'r': return 'RIGHT';
    case 's': return 'STOP';
    default:  return 'STOP';
  }
}

async function send(cmd) {
  statusEl.textContent = 'Status: ' + label(cmd);

  try {
    const r = await fetch('/' + cmd, { cache: 'no-store' });
    await r.text();
  } catch (e) {
    statusEl.textContent = 'Status: Connection error';
  }
}
function clampDeg(v){
  v = Number(v);
  if (isNaN(v)) v = 0;
  v = Math.round(v);

  if (v < 0) v = 0;
  if (v > PLATFORM_MAX_DEG_UI) v = PLATFORM_MAX_DEG_UI;

  return v;
```

```
}

function updatePlatformLabelPreview(deg){
  deg = clampDeg(deg);
  PlatformStatusEl.textContent = 'Platform: ' + deg + '° / 35°';
}

function applyPlatform(){
  const deg = clampDeg(PlatformDegEl.value);
  PlatformDegEl.value = deg;
  PlatformSliderEl.value = deg;
  updatePlatformLabelPreview(deg);

  fetch('/Platform?deg=' + encodeURIComponent(deg))
    .then(r => r.text())
    .then(txt => { levelStatusEl.textContent = txt; btnAutoLevelEl.textContent =
'ENABLE AUTO LEVEL'; })
    .catch(() => { PlatformStatusEl.textContent = 'Platform: Connection error';
});
}

async function zeroLevel(){
  try{
    levelStatusEl.textContent = 'ZEROING...';
    const r = await fetch('/zero', {cache:'no-store'});
    const txt = await r.text();
    levelStatusEl.textContent = txt;
    btnAutoLevelEl.textContent = 'ENABLE AUTO LEVEL';
  }catch(e){
    levelStatusEl.textContent = 'Connection error';
  }
}

async function calibrateLevel(){
  try{
    levelStatusEl.textContent = 'CALIBRATING...';
    const r = await fetch('/calibrate', {cache:'no-store'});
    const txt = await r.text();
    levelStatusEl.textContent = txt;
  }catch(e){
    levelStatusEl.textContent = 'Connection error';
  }
}

async function enableAutoLevel(){
  try{
    const r = await fetch('/autolevel_on', {cache:'no-store'});
    const txt = await r.text();
```

```
    levelStatusEl.textContent = txt;
    if (txt === 'AUTO_LEVEL_ON') btnAutoLevelEl.textContent = 'AUTO LEVEL
ENABLED';
  }catch(e){
    levelStatusEl.textContent = 'Connection error';
  }
}

async function disableAutoLevel(){
  try{
    const r = await fetch('/autolevel_off', {cache:'no-store'});
    const txt = await r.text();
    levelStatusEl.textContent = txt;
    btnAutoLevelEl.textContent = 'ENABLE AUTO LEVEL';
  }catch(e){
    levelStatusEl.textContent = 'Connection error';
  }
}

window.addEventListener('load', () => {
  send('s');
  updatePlatformLabelPreview(PlatformSliderEl.value);
});

</script>

</body>
</html>
)HTML");
}

// SETUP
void setup() {
  Serial.begin(115200);
  delay(1500);

  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH);

  matrix.begin();
  matrix.loadFrame(ALL_ON_FRAME);

  Serial.println("Turning on LEDs and Booting...");

  Wire.begin();
  Wire.setClock(100000);

  pwm.begin();
```

```
pwm.setPWMFreq(50);
delay(10);

stopRover();

PlatformCurrentDeg = PlatformLastDeg;
PlatformTargetDeg  = PlatformLastDeg;
applyPlatformMirrorDeg(PlatformLastDeg);

Serial.println("Servos initialized (STOP + Platform).");

configureMPU6050();
imuOk = true;
Serial.println("IMU ready.");

resetImuFilterState();
imuLoopTimerUs = micros();

for (int i = 0; i < 240; i++) {
  imuTick();
  delay(4);
}

IPAddress ip(192,168,1,10);
IPAddress gateway(192,168,1,254);
IPAddress subnet(255,255,255,0);
IPAddress dns(192,168,1,254);
WiFi.config(ip, dns, gateway, subnet);

Serial.print("Connecting to WiFi");
WiFi.begin(ssid, pass);

unsigned long start = millis();
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
  if (millis() - start > 20000) {
    Serial.println("\nWiFi timeout");

    digitalWrite(LED_BUILTIN, LOW);
    matrix.clear();
    matrix.loadFrame(ALL_ON_FRAME);
    matrix.clear();

    return;
  }
}
```

```
  IPAddress ipLocal;
  do {
    delay(200);
    ipLocal = WiFi.localIP();
  } while (ipLocal[0] == 0);

  Serial.println("\nWiFi connected!");
  Serial.print("IP address: ");
  Serial.println(ipLocal);

  server.begin();
  Serial.println("Web server started. Turning LEDs off.");

  digitalWrite(LED_BUILTIN, LOW);
  matrix.clear();
}

// LOOP
void loop() {
  imuTick();

  updateAutoLevel();

  updatePlatformSmooth();

  WiFiClient client = server.available();
  if (!client) return;

  client.setTimeout(50);

  String reqLine = client.readStringUntil('\r');
  client.flush();

  String path = getPathFromReqLine(reqLine);

  if (path == "/favicon.ico") {
    client.println("HTTP/1.1 204 No Content");
    client.println("Connection: close");
    client.println();
    client.stop();
    return;
  }

  if (path == "/zero") {
    zeroLevelNow();
    if (zeroCaptured) {
      setAutoLevelEnabled(false);
      sendCalibrateText(client, "ZERO_SET_AUTO_OFF");
```

```
    } else {
      sendCalibrateText(client, "ZERO_FAILED");
    }
    client.stop();
    return;
  }

  if (path == "/calibrate") {
    if (imuOk) {
    calibrateGyroOffsets();
    sendCalibrateText(client, "CALIBRATED");
    } else {
    sendCalibrateText(client, "IMU_NOT_READY");
    }
    client.stop();
    return;
  }

  if (path == "/autolevel_on") {
    if (!zeroCaptured) {
      setAutoLevelEnabled(false);
      sendCalibrateText(client, "SET_ZERO_FIRST");
    } else {
      setAutoLevelEnabled(true);
      sendCalibrateText(client, "AUTO_LEVEL_ON");
    }
    client.stop();
    return;
  }

  if (path == "/autolevel_off") {
    setAutoLevelEnabled(false);
    sendCalibrateText(client, "AUTO_LEVEL_OFF");
    client.stop();
    return;
  }

  if (path == "/") {
    stopRover();
    sendPage(client);
    client.stop();
    return;
  }
  //Direction isn't wrong, Backward and Forward movement were swapped late into
development, so I just reversed the commands
  if      (path == "/f") { moveBackward();  sendCalibrateText(client, "BACK"); }
  else if (path == "/b") { moveForward(); sendCalibrateText(client, "FORWARD"); }
  else if (path == "/l") { turnLeft();     sendCalibrateText(client, "LEFT"); }
```

```
  else if (path == "/r") { turnRight();    sendCalibrateText(client, "RIGHT"); }
  else if (path == "/s") { stopRover();    sendCalibrateText(client, "STOP"); }
  else if (path.startsWith("/Platform?")) {
    setAutoLevelEnabled(false); // manual platform command takes control back
from auto-level

    int deg = getQueryInt(path, "deg", PlatformLastDeg);
    deg = clampInt(deg, 0, PLATFORM_MAX_DEG);

    PlatformTargetDeg = deg;

    // If the requested angle is already the current angle, re-send it
immediately.
    // This makes pressing APPLY useful even if the platform was idle.
    if (PlatformCurrentDeg == PlatformTargetDeg) {
      applyPlatformMirrorDeg(PlatformCurrentDeg);
      PlatformLastRefreshMs = millis();
    }
    sendCalibrateText(client, "PLATFORM_MANUAL_AUTO_OFF");
  }else {
    stopRover();
    sendCalibrateText(client, "UNKNOWN_COMMAND_STOPPED");
  }

  client.stop();
}
```

8.3 ArUco Scripts

8.3.1 ChArUco Generator - Python

```
import cv2

# BOARD SETTINGS
SQUARES_X = 7
SQUARES_Y = 5
SQUARE_LENGTH_M = 0.040   # 4 cm
MARKER_LENGTH_M = 0.030   # 3 cm
DICT_NAME = cv2.aruco.DICT_4X4_50

# Output image size in pixels for printing
IMG_W = 2000
IMG_H = 1400
MARGIN = 50

dictionary = cv2.aruco.getPredefinedDictionary(DICT_NAME)

# Modern OpenCV style
board = cv2.aruco.CharucoBoard(
```

```
    (SQUARES_X, SQUARES_Y),
    SQUARE_LENGTH_M,
    MARKER_LENGTH_M,
    dictionary
)

board_img = board.generateImage((IMG_W, IMG_H), marginSize=MARGIN)

cv2.imwrite("charuco_board.png", board_img)
print("Saved charuco_board.png")
```

8.3.2 Calibration Image Capture - Python

```
import os
import time
import cv2
import numpy as np
import requests

ESP32_BASE = "http://192.168.2.10"
CAPTURE_JPG_URL = f"{ESP32_BASE}/capture.jpg"

SAVE_DIR = "calib_images"
os.makedirs(SAVE_DIR, exist_ok=True)

SQUARES_X = 7
SQUARES_Y = 5
SQUARE_LENGTH_M = 0.036
MARKER_LENGTH_M = 0.028
DICT_NAME = cv2.aruco.DICT_4X4_50

dictionary = cv2.aruco.getPredefinedDictionary(DICT_NAME)
board = cv2.aruco.CharucoBoard(
    (SQUARES_X, SQUARES_Y),
    SQUARE_LENGTH_M,
    MARKER_LENGTH_M,
    dictionary
)

detector_params = cv2.aruco.DetectorParameters()
detector_params.cornerRefinementMethod = cv2.aruco.CORNER_REFINE_SUBPIX

if hasattr(cv2.aruco, "ArucoDetector"):
    aruco_detector = cv2.aruco.ArucoDetector(dictionary, detector_params)
else:
    aruco_detector = None

capture_count = 0
```

```
def get_frame():
    try:
        r = requests.get(CAPTURE_JPG_URL, timeout=5)
        r.raise_for_status()
        arr = np.frombuffer(r.content, dtype=np.uint8)
        frame = cv2.imdecode(arr, cv2.IMREAD_COLOR)
        return frame
    except Exception as e:
        print(f"[WARN] get_frame failed: {e}")
        return None

def detect_charuco(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    if aruco_detector is not None:
        marker_corners, marker_ids, _ = aruco_detector.detectMarkers(gray)
    else:
        marker_corners, marker_ids, _ = cv2.aruco.detectMarkers(
            gray, dictionary, parameters=detector_params
        )

    charuco_corners = None
    charuco_ids = None

    if marker_ids is not None and len(marker_ids) > 0:
        # Newer OpenCV path
        if hasattr(cv2.aruco, "CharucoDetector"):
            charuco_detector = cv2.aruco.CharucoDetector(board)
            charuco_corners, charuco_ids, marker_corners, marker_ids =
charuco_detector.detectBoard(gray)
        else:
            # Older OpenCV path
            _, charuco_corners, charuco_ids =
cv2.aruco.interpolateCornersCharuco(
                marker_corners, marker_ids, gray, board
            )

    return marker_corners, marker_ids, charuco_corners, charuco_ids

while True:
    frame = get_frame()
    if frame is None:
        time.sleep(0.5)
        continue

    display = frame.copy()
```

```
    marker_corners, marker_ids, charuco_corners, charuco_ids =
detect_charuco(frame)

    if marker_ids is not None and len(marker_ids) > 0:
        cv2.aruco.drawDetectedMarkers(display, marker_corners, marker_ids)

    num_charuco = 0
    if charuco_ids is not None and len(charuco_ids) > 0:
        num_charuco = len(charuco_ids)
        try:
            cv2.aruco.drawDetectedCornersCharuco(display, charuco_corners,
charuco_ids)
        except Exception:
            pass

    cv2.putText(display, f"Detected ChArUco corners: {num_charuco}",
                (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)

    cv2.putText(display, "Press C to save frame, ESC to quit",
                (10, 65), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255), 2)

    cv2.imshow("ESP32-CAM Calibration Capture", display)
    key = cv2.waitKey(1) & 0xFF

    if key == ord('c'):
        # Save only if enough corners are visible
        if num_charuco >= 8:
            filename = os.path.join(SAVE_DIR, f"calib_{capture_count:03d}.jpg")
            cv2.imwrite(filename, frame)
            print(f"Saved {filename}")
            capture_count += 1
        else:
            print("Not enough ChArUco corners visible; move board and try
again.")

    elif key == 27:
        break

cv2.destroyAllWindows()
```

8.3.3 Camera Calibration - Python

```
import os
import glob
import cv2
import numpy as np

IMAGE_DIR = "calib_images"
```

```
OUT_FILE = "cam_calib_esp32cam.npz"

SQUARES_X = 7
SQUARES_Y = 5
SQUARE_LENGTH_M = 0.036
MARKER_LENGTH_M = 0.028
DICT_NAME = cv2.aruco.DICT_4X4_50

dictionary = cv2.aruco.getPredefinedDictionary(DICT_NAME)
board = cv2.aruco.CharucoBoard(
    (SQUARES_X, SQUARES_Y),
    SQUARE_LENGTH_M,
    MARKER_LENGTH_M,
    dictionary
)

detector_params = cv2.aruco.DetectorParameters()
detector_params.cornerRefinementMethod = cv2.aruco.CORNER_REFINE_SUBPIX

if hasattr(cv2.aruco, "ArucoDetector"):
    aruco_detector = cv2.aruco.ArucoDetector(dictionary, detector_params)
else:
    aruco_detector = None

image_paths = sorted(glob.glob(os.path.join(IMAGE_DIR, "*.jpg")))
if not image_paths:
    raise RuntimeError("No calibration images found.")

all_charuco_corners = []
all_charuco_ids = []
image_size = None

def detect_charuco(gray):
    if aruco_detector is not None:
        marker_corners, marker_ids, _ = aruco_detector.detectMarkers(gray)
    else:
        marker_corners, marker_ids, _ = cv2.aruco.detectMarkers(
            gray, dictionary, parameters=detector_params
        )

    if marker_ids is None or len(marker_ids) == 0:
        return None, None

    if hasattr(cv2.aruco, "CharucoDetector"):
        charuco_detector = cv2.aruco.CharucoDetector(board)
        charuco_corners, charuco_ids, _, _ = charuco_detector.detectBoard(gray)
        return charuco_corners, charuco_ids
    else:
```

```
        _, charuco_corners, charuco_ids = cv2.aruco.interpolateCornersCharuco(
            marker_corners, marker_ids, gray, board
        )
        return charuco_corners, charuco_ids

valid_count = 0

for path in image_paths:
    img = cv2.imread(path)
    if img is None:
        continue

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    image_size = gray.shape[::-1]

    charuco_corners, charuco_ids = detect_charuco(gray)

    if charuco_ids is not None and len(charuco_ids) >= 8:
        all_charuco_corners.append(charuco_corners)
        all_charuco_ids.append(charuco_ids)
        valid_count += 1
        print(f"[OK] {os.path.basename(path)} -> {len(charuco_ids)} corners")
    else:
        print(f"[SKIP] {os.path.basename(path)} -> insufficient corners")

if valid_count < 10:
    raise RuntimeError(f"Only {valid_count} valid images. Capture more varied
images.")

# Preferred direct function if available
used_direct_charuco = False
if hasattr(cv2.aruco, "calibrateCameraCharuco"):
    try:
        ret, camera_matrix, dist_coeffs, rvecs, tvecs =
cv2.aruco.calibrateCameraCharuco(
            all_charuco_corners,
            all_charuco_ids,
            board,
            image_size,
            None,
            None
        )
        used_direct_charuco = True
    except Exception as e:
        print(f"[WARN] calibrateCameraCharuco unavailable/failed: {e}")

if not used_direct_charuco:
    # Fallback: build object/image point sets and use cv2.calibrateCamera
```

```
    object_points = []
    image_points = []

    for corners, ids in zip(all_charuco_corners, all_charuco_ids):
        ids_flat = ids.flatten().tolist()
        chessboard_corners = board.getChessboardCorners()

        obj = []
        imgp = []

        for i, cid in enumerate(ids_flat):
            obj.append(chessboard_corners[cid])
            imgp.append(corners[i][0])

        object_points.append(np.array(obj, dtype=np.float32))
        image_points.append(np.array(imgp, dtype=np.float32))

    ret, camera_matrix, dist_coeffs, rvecs, tvecs = cv2.calibrateCamera(
        object_points,
        image_points,
        image_size,
        None,
        None
    )

print("\n=== CALIBRATION RESULT ===")
print(f"Reprojection error: {ret}")
print("Camera matrix:")
print(camera_matrix)
print("Distortion coefficients:")
print(dist_coeffs.ravel())

np.savez(
    OUT_FILE,
    camera_matrix=camera_matrix,
    dist_coeffs=dist_coeffs,
    reprojection_error=ret,
    image_size=np.array(image_size)
)

print(f"\nSaved calibration to {OUT_FILE}")
```

8.3.4 ArUco Landing_Target - Python

```
import time
import math
import os
```

```
# Force pymavlink to load MAVLink2 + ArduPilot dialect before pymavlink import.
os.environ.setdefault("MAVLINK20", "1")
os.environ.setdefault("MAVLINK_DIALECT", "ardupilotmega")

from typing import Dict, List, Tuple, Optional

import cv2
import numpy as np
import requests
from pymavlink import mavutil

# USER CONFIG
# Camera
ESP32_CAM_BASE = "http://192.168.2.10"
CAPTURE_JPG_URL = f"{ESP32_CAM_BASE}/capture.jpg"
HEALTH_URL = f"{ESP32_CAM_BASE}/health"

# MAVLink
MAVLINK_CONNECTION = "udpout:192.168.2.1:14550"

# Companion-computer MAVLink IDs
COMPANION_SYSID = 191
COMPANION_COMPID = mavutil.mavlink.MAV_COMP_ID_ONBOARD_COMPUTER

# Camera calibration
CALIB_FILE = "cam_calib_esp32cam.npz"

# ArUco dictionary
ARUCO_DICT_ID = cv2.aruco.DICT_4X4_50

# LANDING PAD GEOMETRY
# Define every marker by:
# - its real printed side length in metres
# - its centre position in the landing-pad frame, relative to the
#   TRUE touchdown point (0, 0)
# - optional yaw_deg if a marker is rotated on the pad
#
# Landing-pad frame convention used here:
# - X_pad: right on the platform
# - Y_pad: forward on the platform
# - Z_pad: upward from the pad plane

MARKER_LAYOUT: Dict[int, Dict[str, float]] = {
    # Big marker
    7: {
        "size_m": 0.1445,
        "center_x_m": 0.000,
        "center_y_m": 0.110,
```

```
        "yaw_deg": 0.0,
    },

    # Three small markers
    20: {
        "size_m": 0.050,
        "center_x_m": -0.055,
        "center_y_m": 0.000,
        "yaw_deg": 0.0,
    },
    21: {
        "size_m": 0.050,
        "center_x_m": 0.000,
        "center_y_m": 0.000,
        "yaw_deg": 0.0,
    },
    22: {
        "size_m": 0.050,
        "center_x_m": 0.055,
        "center_y_m": 0.000,
        "yaw_deg": 0.0,
    },
}

# Marker IDs grouped for logging/debug
BIG_MARKER_IDS = {7}
SMALL_MARKER_IDS = {20, 21, 22}

# Timing
HTTP_TIMEOUT_S = 5
HEARTBEAT_PERIOD_S = 1.0
LANDING_TARGET_PERIOD_S = 0.08   # about 12.5 Hz target
LOG_PERIOD_S = 0.5

# Detection
MIN_VISIBLE_MARKERS = 1
SHOW_DEBUG_WINDOW = True

# LOGGING

def log(msg: str) -> None:
    ts = time.strftime("%H:%M:%S")
    print(f"[{ts}] {msg}")

# CAMERA + MAVLINK SETUP

def load_calibration(path: str) -> Tuple[np.ndarray, np.ndarray]:
    data = np.load(path)
```

```
    return data["camera_matrix"], data["dist_coeffs"]

def make_http_session() -> requests.Session:
    s = requests.Session()
    s.headers.update({"Connection": "keep-alive"})
    return s

def camera_health_check(session: requests.Session) -> bool:
    try:
        r = session.get(HEALTH_URL, timeout=HTTP_TIMEOUT_S)
        r.raise_for_status()
        log("ESP32-CAM reachable")
        return True
    except Exception as e:
        log(f"[ERROR] Camera health check failed: {e}")
        return False

def get_frame(session: requests.Session) -> Optional[np.ndarray]:
    try:
        r = session.get(CAPTURE_JPG_URL, timeout=HTTP_TIMEOUT_S)
        r.raise_for_status()
        arr = np.frombuffer(r.content, dtype=np.uint8)
        frame = cv2.imdecode(arr, cv2.IMREAD_COLOR)
        return frame
    except Exception as e:
        log(f"[WARN] get_frame failed: {e}")
        return None

def open_mavlink(conn_str: str) -> mavutil.mavfile:
    master = mavutil.mavlink_connection(
        conn_str,
        source_system=COMPANION_SYSID,
        source_component=COMPANION_COMPID,
    )
    log(f"MAVLink opened on {conn_str}")
    return master

def send_companion_heartbeat(master: mavutil.mavfile) -> None:
    master.mav.heartbeat_send(
        mavutil.mavlink.MAV_TYPE_ONBOARD_CONTROLLER,
        mavutil.mavlink.MAV_AUTOPILOT_INVALID,
        0,
        0,
        mavutil.mavlink.MAV_STATE_ACTIVE,
    )

# ARUCO DETECTION
```

```
ARUCO_DICT = cv2.aruco.getPredefinedDictionary(ARUCO_DICT_ID)
DETECTOR_PARAMS = cv2.aruco.DetectorParameters()
DETECTOR_PARAMS.cornerRefinementMethod = cv2.aruco.CORNER_REFINE_SUBPIX
DETECTOR_PARAMS.adaptiveThreshWinSizeMin = 3
DETECTOR_PARAMS.adaptiveThreshWinSizeMax = 53
DETECTOR_PARAMS.adaptiveThreshWinSizeStep = 4

def detect_markers(gray: np.ndarray):
    if hasattr(cv2.aruco, "ArucoDetector"):
        detector = cv2.aruco.ArucoDetector(ARUCO_DICT, DETECTOR_PARAMS)
        corners, ids, rejected = detector.detectMarkers(gray)
    else:
        corners, ids, rejected = cv2.aruco.detectMarkers(
            gray,
            ARUCO_DICT,
            parameters=DETECTOR_PARAMS
        )
    return corners, ids, rejected

# GEOMETRY HELPERS

def rotated_marker_corners(
    center_x: float,
    center_y: float,
    size_m: float,
    yaw_deg: float
) -> np.ndarray:
    half = size_m / 2.0

    # Marker local corners in pad frame, assuming "top" of marker faces +Y_pad
    local = np.array([
        [-half, +half, 0.0],  # top-left
        [+half, +half, 0.0],  # top-right
        [+half, -half, 0.0],  # bottom-right
        [-half, -half, 0.0],  # bottom-left
    ], dtype=np.float32)

    yaw = math.radians(yaw_deg)
    c = math.cos(yaw)
    s = math.sin(yaw)

    rot = np.array([
        [c, -s, 0.0],
        [s,  c, 0.0],
        [0.0, 0.0, 1.0],
    ], dtype=np.float32)

    rotated = (rot @ local.T).T
```

```
    rotated[:, 0] += center_x
    rotated[:, 1] += center_y
    return rotated.astype(np.float32)

def build_correspondences(
    corners,
    ids
) -> Tuple[Optional[np.ndarray], Optional[np.ndarray], List[int]]:
    if ids is None or len(ids) == 0:
        return None, None, []

    object_points: List[np.ndarray] = []
    image_points: List[np.ndarray] = []
    used_ids: List[int] = []

    ids_flat = ids.flatten().tolist()

    for idx, marker_id in enumerate(ids_flat):
        if marker_id not in MARKER_LAYOUT:
            continue

        marker_info = MARKER_LAYOUT[marker_id]

        obj = rotated_marker_corners(
            center_x=marker_info["center_x_m"],
            center_y=marker_info["center_y_m"],
            size_m=marker_info["size_m"],
            yaw_deg=marker_info.get("yaw_deg", 0.0),
        )

        img = corners[idx].reshape(4, 2).astype(np.float32)

        object_points.append(obj)
        image_points.append(img)
        used_ids.append(marker_id)

    if len(used_ids) < MIN_VISIBLE_MARKERS:
        return None, None, []

    obj_pts = np.concatenate(object_points, axis=0).reshape(-1, 3)
    img_pts = np.concatenate(image_points, axis=0).reshape(-1, 2)

    return obj_pts, img_pts, used_ids

def estimate_pad_pose(
    object_points: np.ndarray,
    image_points: np.ndarray,
    camera_matrix: np.ndarray,
```

```
    dist_coeffs: np.ndarray
) -> Tuple[bool, Optional[np.ndarray], Optional[np.ndarray]]:
    if object_points is None or image_points is None:
        return False, None, None

    ok, rvec, tvec = cv2.solvePnP(
        object_points,
        image_points,
        camera_matrix,
        dist_coeffs,
        flags=cv2.SOLVEPNP_ITERATIVE
    )

    if not ok:
        return False, None, None

    return True, rvec, tvec

def camera_to_body_frd(
    tvec_cam: np.ndarray
) -> np.ndarray:

    x_cam, y_cam, z_cam = tvec_cam.flatten().astype(float)

    x_body =  y_cam
    y_body = -x_cam
    z_body =  z_cam

    return np.array([x_body, y_body, z_body], dtype=np.float32)

def compute_angles_from_body(body_xyz: np.ndarray) -> Tuple[float, float]:
    x, y, z = body_xyz.astype(float)
    if z <= 1e-6:
        return 0.0, 0.0
    angle_x = math.atan2(y, z)
    angle_y = math.atan2(x, z)
    return angle_x, angle_y

# MAVLINK LANDING_TARGET-

def send_landing_target(
    master: mavutil.mavfile,
    body_xyz: np.ndarray
) -> None:
    x, y, z = [float(v) for v in body_xyz]
    distance = float(np.linalg.norm(body_xyz))
    angle_x, angle_y = compute_angles_from_body(body_xyz)
```

```
    target_type = getattr(
        mavutil.mavlink,
        "LANDING_TARGET_TYPE_VISION_FIDUCIAL",
        2
    )

    time_usec = int(time.time() * 1e6)
    frame = mavutil.mavlink.MAV_FRAME_BODY_FRD

    # Preferred MAVLink2 LANDING_TARGET with extension fields:
    # x, y, z, q, target_type, position_valid.
    try:
        master.mav.landing_target_send(
            time_usec,              # time_usec
            0,                      # target_num
            frame,                  # MAV_FRAME_BODY_FRD
            angle_x,                # angle_x
            angle_y,                # angle_y
            distance,               # distance
            0.0,                    # size_x
            0.0,                    # size_y
            x,                      # x forward
            y,                      # y right
            z,                      # z down
            [1.0, 0.0, 0.0, 0.0],   # q, identity quaternion
            target_type,            # type
            1                       # position_valid
        )
        return

    except TypeError as e:
        msg = str(e)
        if "positional arguments" not in msg and "unexpected keyword" not in msg:
            raise

        print(
            "[WARN] Old pymavlink LANDING_TARGET signature. "
            "Falling back to angle-only message. "
            "Recommended fix: pip install --upgrade pymavlink"
        )

        master.mav.landing_target_send(
            time_usec,
            0,
            frame,
            angle_x,
            angle_y,
            distance,
```

```
            0.0,
            0.0
        )

# DEBUG DRAWING

def draw_debug(
    frame: np.ndarray,
    corners,
    ids,
    used_ids: List[int],
    rvec: Optional[np.ndarray],
    tvec: Optional[np.ndarray],
    body_xyz: Optional[np.ndarray]
) -> np.ndarray:
    out = frame.copy()

    if ids is not None and len(ids) > 0:
        cv2.aruco.drawDetectedMarkers(out, corners, ids)

    lines = []

    if used_ids:
        lines.append(f"Used IDs: {used_ids}")

        big_seen = [mid for mid in used_ids if mid in BIG_MARKER_IDS]
        small_seen = [mid for mid in used_ids if mid in SMALL_MARKER_IDS]

        if big_seen and not small_seen:
            lines.append("Mode: BIG marker")
        elif small_seen and not big_seen:
            lines.append("Mode: SMALL markers")
        elif big_seen and small_seen:
            lines.append("Mode: MIXED markers")
    else:
        lines.append("Used IDs: []")

    if body_xyz is not None:
        x, y, z = [float(v) for v in body_xyz]
        d = float(np.linalg.norm(body_xyz))
        lines.append(f"BODY_FRD x={x:.3f} y={y:.3f} z={z:.3f} m")
        lines.append(f"distance={d:.3f} m")

    if rvec is not None and tvec is not None:
        cv2.drawFrameAxes(
            out,
            CAMERA_MATRIX,
            DIST_COEFFS,
```

```
            rvec,
            tvec,
            0.05
        )

    y0 = 25
    for line in lines:
        cv2.putText(
            out,
            line,
            (10, y0),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.65,
            (0, 255, 0),
            2,
            cv2.LINE_AA
        )
        y0 += 28

    return out

# MAIN

CAMERA_MATRIX, DIST_COEFFS = load_calibration(CALIB_FILE)

def main():
    log("Camera calibration loaded")
    session = make_http_session()
    camera_health_check(session)

    master = open_mavlink(MAVLINK_CONNECTION)

    last_heartbeat = 0.0
    last_target_send = 0.0
    last_log = 0.0
    last_used_ids: List[int] = []

    while True:
        now = time.time()

        if now - last_heartbeat >= HEARTBEAT_PERIOD_S:
            send_companion_heartbeat(master)
            last_heartbeat = now

        frame = get_frame(session)
        if frame is None:
            time.sleep(0.2)
            continue
```

```
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        corners, ids, _ = detect_markers(gray)
        obj_pts, img_pts, used_ids = build_correspondences(corners, ids)
        pose_ok = False
        rvec = None
        tvec = None
        body_xyz = None

        if obj_pts is not None and img_pts is not None:
            pose_ok, rvec, tvec = estimate_pad_pose(
                obj_pts,
                img_pts,
                CAMERA_MATRIX,
                DIST_COEFFS
            )

        if pose_ok and tvec is not None:
            body_xyz = camera_to_body_frd(tvec)

            if now - last_target_send >= LANDING_TARGET_PERIOD_S:
                send_landing_target(master, body_xyz)
                last_target_send = now

        if used_ids != last_used_ids:
            if used_ids:
                log(f"Visible landing markers: {used_ids}")
            else:
                log("Landing target lost")
            last_used_ids = used_ids.copy()

        if pose_ok and body_xyz is not None and now - last_log >= LOG_PERIOD_S:
            x, y, z = [float(v) for v in body_xyz]
            log(f"LANDING_TARGET x={x:.3f} y={y:.3f} z={z:.3f} m |
ids={used_ids}")
            last_log = now

        if SHOW_DEBUG_WINDOW:
            dbg = draw_debug(frame, corners, ids, used_ids, rvec, tvec, body_xyz)
            cv2.imshow("Precision Landing Target Sender", dbg)
            key = cv2.waitKey(1) & 0xFF
            if key == 27:
                log("ESC pressed, exiting")
                break

    cv2.destroyAllWindows()
if __name__ == "__main__":
    main()
```

8.3.5 Final ArUco Generator - Python

```python
import os
import base64
import cv2
import numpy as np

# Generates a real-scale 16 x 22 cm landing-pad layout.
#
# Final output:
#   landing_pad_aruco_16x22cm.png
#
# IMPORTANT FOR PRINTING:
# Print at 100% / Actual Size.
# Do NOT use "Fit to page", "Scale to fit", or similar options.
# After printing, measure the marker sides with a ruler.

# ARUCO SETTINGS
DICT_NAME = cv2.aruco.DICT_4X4_50
BORDER_BITS = 1

# IDs used by Aruco_Python_v2.py
BIG_ID = 7
SMALL_LEFT_ID = 20
SMALL_MIDDLE_ID = 21
SMALL_RIGHT_ID = 22

# REAL DIMENSIONS
PLATFORM_WIDTH_CM = 16.0       # left-right
PLATFORM_LENGTH_CM = 22.0      # front-back

BIG_MARKER_SIZE_CM = 11.5
SMALL_MARKER_SIZE_CM = 4.0

# The required true touchdown / camera target point:
# centre of the middle small marker = 6 cm from the front,
# and 8 cm from each side, i.e. centred across the 16 cm width.
SMALL_MIDDLE_CENTER_X_CM = 8.0
SMALL_CENTER_Y_FROM_FRONT_CM = 6.0

# Same spacing as current MARKER_LAYOUT:
# small markers at x = -4.5 cm, 0 cm, +4.5 cm around the middle marker.
SMALL_CENTER_SPACING_CM = 4.5

# Big marker is 9 cm behind the small middle marker,
# matching center_y_m = 0.090
BIG_CENTER_OFFSET_BEHIND_SMALL_CM = 9.0

# OUTPUT SETTINGS
```

```
DPI = 300
PX_PER_CM = DPI / 2.54

OUT_CLEAN_PNG = "landing_pad_aruco_16x22cm.png"
OUT_DEBUG_PNG = "landing_pad_aruco_16x22cm_debug.png"
OUT_SVG = "landing_pad_aruco_16x22cm.svg"
OUT_INFO = "landing_pad_marker_layout_info.txt"


def cm_to_px(cm: float) -> int:
    return int(round(cm * PX_PER_CM))


def platform_to_pixel(x_cm: float, y_from_front_cm: float) -> tuple[int, int]:
    x_px = cm_to_px(x_cm)
    y_px = cm_to_px(PLATFORM_LENGTH_CM - y_from_front_cm)
    return x_px, y_px


def generate_marker(dictionary, marker_id: int, size_cm: float) -> np.ndarray:
    side_px = cm_to_px(size_cm)

    if hasattr(cv2.aruco, "generateImageMarker"):
        marker = cv2.aruco.generateImageMarker(
            dictionary,
            marker_id,
            side_px,
            borderBits=BORDER_BITS
        )
    else:
        marker = np.zeros((side_px, side_px), dtype=np.uint8)
        cv2.aruco.drawMarker(
            dictionary,
            marker_id,
            side_px,
            marker,
            borderBits=BORDER_BITS
        )

    return marker


def paste_marker(canvas: np.ndarray, marker: np.ndarray, center_x_cm: float,
center_y_from_front_cm: float) -> None:
    cx, cy = platform_to_pixel(center_x_cm, center_y_from_front_cm)
    h, w = marker.shape[:2]

    x0 = int(round(cx - w / 2))
    y0 = int(round(cy - h / 2))
```

```
    x1 = x0 + w
    y1 = y0 + h

    if x0 < 0 or y0 < 0 or x1 > canvas.shape[1] or y1 > canvas.shape[0]:
        raise ValueError(
            f"Marker does not fit on platform: "
            f"center=({center_x_cm:.2f}, {center_y_from_front_cm:.2f}) cm, "
            f"size=({w / PX_PER_CM:.2f}, {h / PX_PER_CM:.2f}) cm"
        )

    canvas[y0:y1, x0:x1] = marker


def draw_marker_label(img: np.ndarray, text: str, center_x_cm: float,
center_y_from_front_cm: float, dy_cm: float = -0.4) -> None:
    x, y = platform_to_pixel(center_x_cm, center_y_from_front_cm + dy_cm)
    cv2.putText(
        img,
        text,
        (x - 60, y),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.65,
        0,
        2,
        cv2.LINE_AA
    )


def draw_crosshair(img: np.ndarray, center_x_cm: float, center_y_from_front_cm:
float) -> None:
    x, y = platform_to_pixel(center_x_cm, center_y_from_front_cm)
    s = cm_to_px(0.35)
    cv2.line(img, (x - s, y), (x + s, y), 0, 2)
    cv2.line(img, (x, y - s), (x, y + s), 0, 2)


def save_png_with_dpi(path: str, img: np.ndarray) -> None:
    try:
        from PIL import Image
        Image.fromarray(img).save(path, dpi=(DPI, DPI))
    except Exception:
        cv2.imwrite(path, img)


def png_data_uri(marker_img: np.ndarray) -> str:
    ok, buf = cv2.imencode(".png", marker_img)
    if not ok:
        raise RuntimeError("Could not encode marker PNG for SVG.")
    b64 = base64.b64encode(buf.tobytes()).decode("ascii")
```

```
    return f"{b64}"


def write_svg(markers: list[dict]) -> None:
    lines = [
        f'<svg xmlns="http://www.w3.org/2000/svg" width="{PLATFORM_WIDTH_CM}cm"
height="{PLATFORM_LENGTH_CM}cm" viewBox="0 0 {PLATFORM_WIDTH_CM}
{PLATFORM_LENGTH_CM}">',
        '<rect x="0" y="0" width="100%" height="100%" fill="white"/>',
    ]

    for m in markers:
        size = m["size_cm"]
        x = m["x_cm"] - size / 2
        # SVG y=0 is top, while our platform y=0 is front/bottom.
        y_top = PLATFORM_LENGTH_CM - m["y_from_front_cm"] - size / 2
        uri = png_data_uri(m["marker_img"])
        lines.append(
            f'<image href="{uri}" x="{x:.4f}" y="{y_top:.4f}" '
            f'width="{size:.4f}" height="{size:.4f}" image-
rendering="pixelated"/>'
        )

    lines.append("</svg>")

    with open(OUT_SVG, "w", encoding="utf-8") as f:
        f.write("\n".join(lines))


def main() -> None:
    dictionary = cv2.aruco.getPredefinedDictionary(DICT_NAME)

    platform_w_px = cm_to_px(PLATFORM_WIDTH_CM)
    platform_h_px = cm_to_px(PLATFORM_LENGTH_CM)

    clean = np.full((platform_h_px, platform_w_px), 255, dtype=np.uint8)

    marker_defs = [
        {
            "id": BIG_ID,
            "size_cm": BIG_MARKER_SIZE_CM,
            "x_cm": SMALL_MIDDLE_CENTER_X_CM,
            "y_from_front_cm": SMALL_CENTER_Y_FROM_FRONT_CM +
BIG_CENTER_OFFSET_BEHIND_SMALL_CM,
            "name": "BIG",
        },
        {
            "id": SMALL_LEFT_ID,
            "size_cm": SMALL_MARKER_SIZE_CM,
```

```
            "x_cm": SMALL_MIDDLE_CENTER_X_CM - SMALL_CENTER_SPACING_CM,
            "y_from_front_cm": SMALL_CENTER_Y_FROM_FRONT_CM,
            "name": "SMALL LEFT",
        },
        {
            "id": SMALL_MIDDLE_ID,
            "size_cm": SMALL_MARKER_SIZE_CM,
            "x_cm": SMALL_MIDDLE_CENTER_X_CM,
            "y_from_front_cm": SMALL_CENTER_Y_FROM_FRONT_CM,
            "name": "SMALL MIDDLE / TOUCHDOWN",
        },
        {
            "id": SMALL_RIGHT_ID,
            "size_cm": SMALL_MARKER_SIZE_CM,
            "x_cm": SMALL_MIDDLE_CENTER_X_CM + SMALL_CENTER_SPACING_CM,
            "y_from_front_cm": SMALL_CENTER_Y_FROM_FRONT_CM,
            "name": "SMALL RIGHT",
        },
    ]

    for m in marker_defs:
        m["marker_img"] = generate_marker(dictionary, m["id"], m["size_cm"])
        paste_marker(clean, m["marker_img"], m["x_cm"], m["y_from_front_cm"])

    save_png_with_dpi(OUT_CLEAN_PNG, clean)

    debug = clean.copy()
    cv2.rectangle(debug, (0, 0), (platform_w_px - 1, platform_h_px - 1), 0, 3)

    cv2.putText(
        debug,
        "REAR",
        (cm_to_px(0.7), cm_to_px(1.0)),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.8,
        0,
        2,
        cv2.LINE_AA
    )
    cv2.putText(
        debug,
        "FRONT",
        (cm_to_px(0.7), platform_h_px - cm_to_px(0.7)),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.8,
        0,
        2,
        cv2.LINE_AA
```

```
    )

    for m in marker_defs:
        draw_marker_label(
            debug,
            f'ID {m["id"]}',
            m["x_cm"],
            m["y_from_front_cm"],
            dy_cm=-(m["size_cm"] / 2 + 0.3)
        )

    draw_crosshair(debug, SMALL_MIDDLE_CENTER_X_CM, SMALL_CENTER_Y_FROM_FRONT_CM)
    cv2.putText(
        debug,
        "TOUCHDOWN / CAMERA TARGET",
        (cm_to_px(3.1), platform_h_px - cm_to_px(3.0)),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.55,
        0,
        2,
        cv2.LINE_AA
    )

    save_png_with_dpi(OUT_DEBUG_PNG, debug)
    write_svg(marker_defs)

    with open(OUT_INFO, "w", encoding="utf-8") as f:
        f.write("Final landing pad ArUco layout\n")
        f.write("==============================\n\n")
        f.write(f"Platform: {PLATFORM_WIDTH_CM:.1f} x {PLATFORM_LENGTH_CM:.1f}
cm\n")
        f.write(f"Dictionary: DICT_4X4_50\n")
        f.write(f"Big marker: ID {BIG_ID}, side {BIG_MARKER_SIZE_CM:.1f} cm\n")
        f.write(f"Small markers: IDs {SMALL_LEFT_ID}, {SMALL_MIDDLE_ID},
{SMALL_RIGHT_ID}, side {SMALL_MARKER_SIZE_CM:.1f} cm\n\n")
        f.write("Physical positions, measured from platform front-left
corner:\n")
        for m in marker_defs:
            f.write(
                f'  ID {m["id"]:>2} ({m["name"]}): '
                f'centre x={m["x_cm"]:.1f} cm from left, '
                f'y={m["y_from_front_cm"]:.1f} cm from front, '
                f'size={m["size_cm"]:.1f} cm\n'
            )

        f.write("\nMiddle small marker / target point:\n")
        f.write(f"  x = {SMALL_MIDDLE_CENTER_X_CM:.1f} cm from left\n")
        f.write(f"  y = {SMALL_CENTER_Y_FROM_FRONT_CM:.1f} cm from front\n")
```

```
        f.write(f"  This is also 8.0 cm from each side on a 16 cm wide
platform.\n\n")

        f.write("Aruco_Python_v2.py MARKER_LAYOUT equivalent, relative to middle
small marker:\n")
        f.write("MARKER_LAYOUT = {\n")
        f.write(f"    {BIG_ID}: {{'size_m': {BIG_MARKER_SIZE_CM / 100:.3f},
'center_x_m': 0.000, 'center_y_m': {BIG_CENTER_OFFSET_BEHIND_SMALL_CM / 100:.3f},
'yaw_deg': 0.0}},\n")
        f.write(f"    {SMALL_LEFT_ID}: {{'size_m': {SMALL_MARKER_SIZE_CM /
100:.3f}, 'center_x_m': {-SMALL_CENTER_SPACING_CM / 100:.3f}, 'center_y_m':
0.000, 'yaw_deg': 0.0}},\n")
        f.write(f"    {SMALL_MIDDLE_ID}: {{'size_m': {SMALL_MARKER_SIZE_CM /
100:.3f}, 'center_x_m': 0.000, 'center_y_m': 0.000, 'yaw_deg': 0.0}},\n")
        f.write(f"    {SMALL_RIGHT_ID}: {{'size_m': {SMALL_MARKER_SIZE_CM /
100:.3f}, 'center_x_m': {SMALL_CENTER_SPACING_CM / 100:.3f}, 'center_y_m': 0.000,
'yaw_deg': 0.0}},\n")
        f.write("}\n\n")
        f.write("Print note: print at 100% / Actual Size. Do not use Fit to
page.\n")

    print("Generated:")
    print(" ", os.path.abspath(OUT_CLEAN_PNG))
    print(" ", os.path.abspath(OUT_DEBUG_PNG))
    print(" ", os.path.abspath(OUT_SVG))
    print(" ", os.path.abspath(OUT_INFO))

if __name__ == "__main__":
    main()
```

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	Ελληνικός όρος
Autonomous landing	Αυτόνομη προσγείωση
Mobile platform	Κινητή πλατφόρμα
Landing platform	Πλατφόρμα προσγείωσης
Landing target	Στόχος προσγείωσης
Quadcopter	Τετρακόπτερο
Multirotor	Πολυκόπτερο
Unmanned Aerial Vehicle	Μη Επανδρωμένο Εναέριο Όχημα
Unmanned Ground Vehicle	Μη Επανδρωμένο Επίγειο Όχημα
Machine vision	Μηχανική όραση
Computer vision	Υπολογιστική όραση

Fiducial marker	Δείκτης αναφοράς
ArUco marker	Δείκτης ArUco
ChArUco board	Πίνακας ChArUco
Camera calibration	Βαθμονόμηση κάμερας
Camera matrix	Μητρώο κάμερας
Distortion coefficients	Συντελεστές παραμόρφωσης
Reprojection error	Σφάλμα επαναπροβολής
Pose estimation	Εκτίμηση θέσης και προσανατολισμού
Perspective-n-Point	Πρόβλημα προοπτικής n σημείων
Pinhole camera model	Μοντέλο κάμερας οπής
Precision Landing	Προσγείωση ακριβείας
Precision Loiter	Αιώρηση ακριβείας
Flight controller	Ελεγκτής πτήσης
Electronic Speed Controller	Ηλεκτρονικός ρυθμιστής ταχύτητας
Power Distribution Board	Πίνακας διανομής ισχύος
Firmware	Υλικολογισμικό
Custom firmware	Προσαρμοσμένο υλικολογισμικό
Access Point	Σημείο πρόσβασης
Endpoint	Τελικό σημείο επικοινωνίας
Serial-to-Wi-Fi bridge	Γέφυρα σειριακής επικοινωνίας προς Wi-Fi
Telemetry	Τηλεμετρία
Hovering	Αιώρηση
Vertical Take-Off and Landing	Κάθετη απογείωση και προσγείωση
Roll	Γωνία κλίσης
Pitch	Γωνία πρόνευσης
Yaw	Γωνία εκτροπής
Euler angles	Γωνίες Euler
Inertial reference frame	Αδρανειακό σύστημα αναφοράς
Body reference frame	Σύστημα αναφοράς σώματος
Rotation matrix	Πίνακας περιστροφής
State-space model	Μοντέλο χώρου καταστάσεων
Linearisation	Γραμμικοποίηση
Model Predictive Control	Προγνωστικός έλεγχος μοντέλου
Linear Quadratic Regulator	Γραμμικός τετραγωνικός ρυθμιστής
Differential drive	Διαφορική οδήγηση

Kinematics	Κινηματική
Non-holonomic constraints	Μη ολόνομοι περιορισμοί
Instantaneous Center of Curvature	Στιγμιαίο κέντρο καμπυλότητας
Forward kinematics	Ευθύ κινηματικό πρόβλημα
Inverse kinematics	Αντίστροφο κινηματικό πρόβλημα
Inertial Measurement Unit	Μονάδα αδρανειακής μέτρησης
Accelerometer	Επιταχυνσιόμετρο
Gyroscope	Γυροσκόπιο
Complementary filter	Συμπληρωματικό φίλτρο
Servomotor	Σερβοκινητήρας
Continuous rotation servomotor	Σερβοκινητήρας συνεχούς περιστροφής
Position servomotor	Σερβοκινητήρας θέσης
Pulse Width Modulation	Διαμόρφωση πλάτους παλμού
Three-dimensional printing	Τρισδιάστατη εκτύπωση
Polycarbonate	Πολυανθρακικό υλικό
Polylactic Acid	Πολυγαλακτικό οξύ
Brushless motor	Κινητήρας χωρίς ψήκτρες
Propeller	Προπέλα
Rotor	Ρότορας
Thrust	Ώση
Hover thrust	Ώση αιώρησης
Throttle	Ποσοστό εντολής ισχύος κινητήρων
Harmonic Notch filter	Φίλτρο αρμονικής εγκοπής
Simulation	Προσομοίωση
Software-in-the-Loop	Προσομοίωση λογισμικού στον βρόχο
State machine	Μηχανή καταστάσεων
Guided mode	Λειτουργία καθοδήγησης
Land mode	Λειτουργία προσγείωσης

ΣΥΝΤΜΗΣΕΙΣ - ΑΡΚΤΙΚΟΛΕΞΑ - ΑΚΡΩΝΥΜΙΑ

Σύντμηση / Ακρωνύμιο	Πλήρης ανάπτυξη
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
VTOL	Vertical Take-Off and Landing
FC	Flight Controller
ESC	Electronic Speed Controller
PDB	Power Distribution Board
IMU	Inertial Measurement Unit
PWM	Pulse Width Modulation
I2C	Inter-Integrated Circuit
SDA	Serial Data
SCL	Serial Clock
UART	Universal Asynchronous Receiver-Transmitter
RX	Receive
TX	Transmit
MAVLink	Micro Air Vehicle Link
SITL	Software-in-the-Loop
AP	Access Point
UDP	User Datagram Protocol
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
IDE	Integrated Development Environment
EEPROM	Electrically Erasable Programmable Read-Only Memory
LED	Light Emitting Diode
LiPo	Lithium Polymer
DShot	Digital Shot
GPS	Global Positioning System
GNSS	Global Navigation Satellite System
MPC	Model Predictive Control
LQR	Linear Quadratic Regulator
PID	Proportional-Integral-Derivative
ICC	Instantaneous Center of Curvature
PnP	Perspective-n-Point

HFOV	Horizontal Field of View
FRD	Front-Right-Down
BODY_FRD	Body Front-Right-Down
OpenCV	Open Source Computer Vision Library
ArUco	Augmented Reality University of Cordoba
ChArUco	Chessboard ArUco
ΕΚΠΑ	Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
NKUA	National and Kapodistrian University of Athens
RCCL	Robotics, Automatic Control and Cyber-Physical Systems Laboratory
PC	Polycarbonate
PLA	Polylactic Acid
XT60	XT60 connector
KV	Revolutions per minute per volt
V	Volt
A	Ampere
mAh	Milliampere-hour
Hz	Hertz
FPS	Frames Per Second
m	Metre
cm	Centimetre
kg	Kilogram
N	Newton
Rad	Radian
EKF	Extended Kalman Filter

ΑΝΑΦΟΡΕΣ

[1] A. Arif , H. Wang, H. Castañeda και Y. Wang , «Finite-time tracking of moving platform with single camera for quadrotor autonomous landing,» *IEEE Transactions on Circuits and Systems I: Regular Papers,* 2023.

[2] I. M. P. Castelo, «Autonomous landing of an aerial robot on a moving platform,» *Master's thesis, Universidade de Lisboa,* 2021.

[3] S. K. Cangiz και L. Ucum, «Kinematic models for mobile robots,» *Mobile robots: Navigation, control and sensing, surface robots and AUVs,* pp. 5-12, 2020.

[4] D. Falanga, A. Zanchettin, A. Simovic, J. Delmerico και D. Scaramuzza, «Vision-based autonomous quadrotor landing on a moving platform,» 2017.

[5] Y. Feng, C. Zhang, S. Baek , S. Rawashdeh και A. Mohammadi, «Autonomous landing of a UAV on a moving platform using model predictive control,» *Drones,* pp. 2(4), 34, 2018.

[6] L. Garegnani, «Autonomous landing of a UAV on a moving UGV platform using cooperative MPC,» *Master's thesis, KTH Royal Institute of Technology,* 2021.

[7] J. Ghommam και M. Saad, «Autonomous landing of a quadrotor on a moving platform,» *IEEE Transactions on Aerospace and Electronic Systems,* pp. 53, 1504–1519, 2017.

[8] G. L. Goh, G. D. Goh και Z. W. Zhong , «Outdoor autonomous landing of a quadcopter on a moving platform using off-board computer vision,» *Journal of Mechatronics and Optics,* 2019.

[9] K. Guo, P. Tang , H. Wang, D. Lin και X. Cui, «Autonomous landing of a quadrotor on a moving platform via model predictive control,» *Aerospace,* pp. 9, 34, 2022.

[10] T. Hellström, «Kinematics equations for differential drive and articulated steering,» *Department of Computing Science, Umeå University,* 2011.

[11] A. Paris, B. T. Lopez και J. P. How, «Dynamic landing of an autonomous quadrotor on a moving platform in turbulent wind conditions,» *IEEE International Conference on Robotics and Automation (ICRA),* 2020.

[12] Y. Qi, J. Jiang, J. Wu, J. Wang, C. Wang και J. Shan, «Autonomous landing solution of low-cost quadrotor on a moving platform,» *Robotics and Autonomous Systems,* pp. 119, 64–76, 2019.

[13] R. Siegwart, I. R. Nourbakhsh και D. Scaramuzza, «Introduction to autonomous mobile robots,» *MIT Press,* 2011.

[14] S. G. Tzafestas, «Mobile robot kinematics,» *Introduction to mobile robot control, Elsevier,* pp. 31-67, 2014.

[15] P. Wang, C. Wang, J. Wang και M. Q.-H. Meng, «Quadrotor autonomous landing on moving platform,» *Procedia Computer Science,* pp. 209, 40–49, 2022.

[16] H. Wu, W. Wang, T. Wang και S. Suzuki, «High-precision landing on a moving platform based on drone vision using YOLO algorithm,» *Drones,* pp. 9, 261, 2025.

[17] N. Xuan-Mung, N. P. Nguyen, T. Nguyen, D. B. Pham, M. T. Vu, H. L. N. N. Thanh και S. K. Hong, «Quadcopter precision landing on moving targets via disturbance observer-based controller and autonomous landing planner,» *IEEE Access,* 2022.

[18] ArduPilot, «MethodicConfigurator,» 2024. [Ηλεκτρονικό]. Available: https://github.com/ArduPilot/MethodicConfigurator.

[19] ArduPilot, «CustomBuild,» 2020. [Ηλεκτρονικό]. Available: https://github.com/ArduPilot/CustomBuild/.

[20] OpenCV, «ArUco,» [Ηλεκτρονικό]. Available: https://docs.opencv.org/4.x/index.html.

[21] ArduPilot, «SITL and Gazebo,» [Ηλεκτρονικό]. Available: https://ardupilot.org/dev/docs/sitl-with-gazebo.html.

[22] ArduPilot, «Mission Planner,» [Ηλεκτρονικό]. Available: https://ardupilot.org/planner/.

[23] ArduPilot, «Copter Firmware,» [Ηλεκτρονικό]. Available: https://ardupilot.org/copter/index.html.

[24] F. Kouboulis και N. Kouvakas, «Kinimatic Analysis Of Differential-Drive Vehicles,» NKUA, 2022.

[25] Bird Sanctuary, «Bluejay,» [Ηλεκτρονικό]. Available: https://github.com/bird-sanctuary/bluejay.

[26] stylesuxx, «ESC Configurator,» 2021. [Ηλεκτρονικό]. Available: https://github.com/stylesuxx/esc-configurator.

[27] seeul8er, «DroneBridge for ESP32,» [Ηλεκτρονικό]. Available: https://github.com/DroneBridge/ESP32.

[28] Arduino, «UNO R4 WiFi,» [Ηλεκτρονικό]. Available: https://docs.arduino.cc/hardware/uno-r4-wifi/.

[29] Last Minute Engineers, «How to Control Multiple Servo Motors with a PCA9685 Module and Arduino,» [Ηλεκτρονικό]. Available: https://lastminuteengineers.com/pca9685-multiple-servo-motor-control-arduino-tutorial/.

[30] Last Minute Engineers, «Interface MPU6050 Accelerometer and Gyroscope Sensor with Arduino,» [Ηλεκτρονικό]. Available: https://lastminuteengineers.com/mpu6050-accel-gyro-arduino-tutorial/.

[31] Mateksys, «(EOL) Flight Controller F405-WMN,» [Ηλεκτρονικό]. Available: https://www.mateksys.com/?portfolio=f405-wmn#tab-id-3.

[32] Hawk's Work, «2212 Brushless Motor 920KV 2-4S,» [Ηλεκτρονικό]. Available: https://www.hawks-work.com/pages/brushless-motor-2212.

[33] S. Yeom, "Vision-Only Localization of Drones with Optimal Window Velocity Fusion," *Electronics,* vol. 15, no. 3, p. 637, 2026.

[34] M. H. Teh, G. Lou, N. Ralston, Z. Zhao, B. Fan and T. Li, "Drone-based human motion capture: A review," *Intelligent Sports and Health,* vol. 2, no. 1, p. 24–38, 2026.

[35] M. G. Skarpetis, F. N. Koumboulis and P. Papanikolaou, "Vehicle lateral control using a robust tracking controller based on vision look ahead system," in *2017 IEEE 21st International Conference on Intelligent Engineering Systems (INES)*, Larnaca, Cyprus, 2017.

[36] M. G. Skarpetis and F. N. Koumboulis, "Robust triangular decoupling velocity varying controllers for four wheel steering autonomous vehicles," in *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Padova, Italy, 2024.

[37] M. G. Skarpetis, F. N. Koumboulis and N. S. Roussos, "Robust output command tracking for linear systems with nonlinear uncertain structure with application to flight control," in *Proceedings of the 44th IEEE Conference on Decision and Control*, 2005.

[38] M. G. Skarpetis, F. N. Koumboulis and A. S. Ntellis, "Robust multi-condition flight controllers," in *Proceedings of the 16th Mediterranean Conference on Control and Automation*, Ajaccio, France, 2008.

[39] M. G. Skarpetis and F. N. Koumboulis, "Robust manoeuvres for the lateral motion of an aircraft," in *Proceedings of an IET Control Conference*, 1996.

[40] M. G. Skarpetis, F. N. Koumboulis, A. S. Ntellis and T. E. Tsimos, "Robust Lane Keeping for a Tractor-Trailer," in *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, Patras, Greece, 2007.

[41] M. G. Skarpetis and F. N. Koumboulis, "Robust control of the lateral motion of an aircraft with actuator failure," in *Proceedings of the 8th IEEE Mediterranean Conference on Control and Automation*, 2000.

[42] M. G. Skarpetis, F. N. Koumboulis and T. G. Koussiouris, "Pitch and flight path control with simultaneous forward gust rejection," in *Proceedings of the International Conference on Computers and Computational Engineering in Control*, 1999.

[43] M. G. Skarpetis, F. N. Koumboulis and X. Loutas, "Missile autopilot design using a robust asymptotic tracking controller," in *Proceedings of the IEEE International Conference on Intelligent Engineering Systems (INES)*, 2018.

[44] M. G. Skarpetis, F. N. Koumboulis and A. S. Ntellis, "Longitudinal flight multi-condition control using robust PID controllers," in *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2011.

[45] M. G. Skarpetis, F. N. Koumboulis and T. Ladopoulos, "Independent control of the lateral motion of an aircraft," in *Proceedings of the 8th IEEE Mediterranean Conference on Control and Automation*, 2000.

[46] M. G. Skarpetis, "Disturbance rejection for an aircraft flying in atmospheric disturbances," *Dynamics and Control,* vol. 5, p. 389–400, 1995.

[47] M. G. Skarpetis and F. N. Koumboulis, "Decoupling of the longitudinal modes of advanced aircraft," *Journal of Guidance, Control, and Dynamics,* vol. 19, no. 5, p. 1184–1186, 1996.

[48] M. G. Skarpetis and F. N. Koumboulis, "Controller design for lateral manoeuvres in electromagnetic wind tunnels," in *Proceedings of the Third International Conference on Electronics, Circuits, and Systems*, 1996.

[49] M. G. Skarpetis, "Control of Linear Systems with Structured Uncertainties," 1999.

[50] M. G. Skarpetis, F. N. Koumboulis and P. Papanikolaou, "Control and Visualization of Mobile Robot Formation," in *2018 IEEE 22nd International Conference on Intelligent Engineering Systems (INES)*, Las Palmas de Gran Canaria, Spain, 2018.

[51] M. G. Skarpetis, "Aircraft Flight Control via Decoupling Techniques (Έλεγχος Πτήσης Αεροσκαφών με Τεχνικές Αποσύζευξης)," 1991.

[52] M. G. Skarpetis, F. N. Koumboulis and V. Tsigkopoulos, "A fuzzy control scheme for asymptotic command tracking of self-balancing robots," in *International Conference on Robot Systems and Applications*, 2023.

[53] M. G. Skarpetis and F. N. Koumboulis, "2-level hierarchical control for longitudinal flight envelopes," in *Proceedings of the European Workshop on Intelligent Forecasting, Diagnosis and Decision Support Systems*, 2001.

[54] W. N. Singh, D. Sharma, A. Dutta and T. Roy, "Drone Based Face Recognition System using MTCNN and Facenet in ArduPilot Software Platform," *Computación y Sistemas,* vol. 30, no. 1, 2026.

[55] J. Sigalas, M. G. Skarpetis, F. N. Koumboulis, D. Benetos and N. D. Kouvakas, "Design and Implementation of a 3D Printed Robotic Vision System Connected to an Ontology-Based Editor for Manuscript Transcription and Annotation," in *International Conference on Artificial Intelligence and Ethics*, 2023.

[56] C. Ranjan, "Next-Generation Computer Vision for Autonomous Drone Navigation: Cutting-Edge Techniques and Real-World Applications," in *Innovative Machine Learning Applications in the Aerospace Industry*, IGI Global Scientific Publishing, 2026, p. 87–110.

[57] A. Pekias, G. S. Maraslidis, M. G. Tsipouras, F. N. Koumboulis and G. F. Fragulis, "Power supply technologies for drones and machine vision applications: A comparative analysis and future trends," *Telecom,* vol. 4, no. 3, p. 459–476, 2023.

[58] A. Pekias, G. S. Maraslidis, M. G. Tsipouras, F. N. Koumboulis and G. F. Fragulis, "An overview of power supply technologies for unmanned aerial vehicles (UAVs) and machine vision applications," in *Proceedings of the South-East Europe Design Automation, Computer Engineering Conference*, 2022.

[59] K.-T. Lai, Y.-T. Chung, J.-J. Su, C.-H. Lai and Y.-H. Huang, "AI Wings: An AIoT Drone System for Commanding ArduPilot UAVs," *IEEE Systems Journal,* vol. 17, no. 2, p. 2213–2224, 2022.

[60] N. D. Kouvakas and F. N. Koumboulis, "Trim Point Transitions via I/O Decoupling Controllers for 6DOF Quadrotors," in *International Conference on Unmanned Aircraft Systems*, 2021.

[61] F. N. Koumboulis and N. D. Kouvakas, "Wireless Longitudinal Motion Controller Design for Quadrotors," in *International Conference on Unmanned Aircraft Systems*, 2020.

[62] F. N. Koumboulis, M. G. Skarpetis and M. K. Griparis, "Rotary gust rejection for helicopters," in *Proceedings of the IEEE International Conference on Control Applications*, 1998.

[63] F. N. Koumboulis and M. G. Skarpetis, "Robust triangular decoupling with application to 4WS cars," *IEEE Transactions on Automatic Control,* vol. 45, no. 2, p. 344–352, Feb. 2000.

[64] F. N. Koumboulis, M. G. Skarpetis and M. P. Tzamtzi, "Robust PID controller design with application to a flight actuator," in *Proceedings of the 32nd Annual Conference of the IEEE Industrial Electronics Society (IECON)*, 2006.

[65] F. N. Koumboulis and M. G. Skarpetis, "Robust control of cars with front and rear wheel steering," *IEE Proceedings - Control Theory and Applications,* vol. 149, no. 5, p. 377–382, 2002.

[66] F. N. Koumboulis, M. G. Skarpetis and M. K. Griparis, "Noninteracting control for helicopter in straight horizontal flight," in *Proceedings of the IEEE International Conference on Control Applications*, 1998.

[67] F. N. Koumboulis and N. D. Kouvakas, "Common Noninteracting Control with Simultaneous Common Partial Zeroing with Application to a Tracked UGV," *Machines,* vol. 11, no. 1, p. 113, 2023.

[68] T. Audronis, Designing Purpose-Built Drones for ArduPilot Pixhawk 2.1: Build Drones with ArduPilot, Packt Publishing Ltd, 2017.

[69] S. A. Fronimos και D. Fanourgakis, «Undergraduate Thesis Project,» 2026. [Ηλεκτρονικό]. Available: https://github.com/sumurtugu/Undergraduate-Thesis-Project.