



ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
Τμ. ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ
Π.Μ.Σ. «ΜΗ ΕΠΑΝΔΡΩΜΕΝΑ ΑΥΤΟΝΟΜΑ ΚΑΙ
ΤΗΛΕΚΑΤΕΥΘΥΝΟΜΕΝΑ ΣΥΣΤΗΜΑΤΑ»

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
«ΜΗ ΕΠΑΝΔΡΩΜΕΝΑ ΑΥΤΟΝΟΜΑ ΚΑΙ
ΤΗΛΕΚΑΤΕΥΘΥΝΟΜΕΝΑ ΣΥΣΤΗΜΑΤΑ»

ΤΙΤΛΟΣ

*Σχεδίαση και προσομοίωση ΣμηΕΑ για σημασιολογική χαρτογράφηση
και εντοπισμό ατόμων με χρήση τεχνητής όρασης*

ΤΙΤΛΟΣ ΑΓΓΛΙΚΑ

*UAS Design and Simulation for Semantic Mapping and Human
Detection using Computer Vision*

Ονοματεπώνυμο Φοιτητή:

Μιχαήλ Ταμβακέρας

Ονοματεπώνυμο Υπεύθυνου Καθηγητή:

Δρ. Αβραάμ Χατζόπουλος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΑΝΟΥΑΡΙΟΣ 2026

Μέλη Εξεταστικής Επιτροπής



Digitally signed by
ABRAAM ΧΑΤΖΟΠΟΥΛΟΣ
Date: 2026.01.20 15:20:00
+02'00'



Digitally signed by
CHRISTOS DROSOS
Date: 2026.01.21
11:36:57 +02'00'



MICHAIL
ΠΑΡΟΥΤΣΙΔΑΚΗΣ
2026.01.22
12:20:26 +02'00'

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Μιχαήλ Ταμβακέρας του Παναγιώτη, με αριθμό μητρώου 8096675 φοιτητής/τρια του Προγράμματος Μεταπτυχιακών Σπουδών «Μη Επανδρωμένα Αυτόνομα και Τηλεκατευθυνόμενα Συστήματα» του Τμήματος Μηχανικών Βιομηχανικής Σχεδίασης και Παραγωγής της Σχολής Μηχανικών Πανεπιστημίου Δυτικής Αττικής, δηλώνω υπεύθυνα ότι: «Είμαι συγγραφέας αυτής της μεταπτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του διπλώματός μου».

Ο δηλών



TAMBAKERAS MICHAEL

Ημερομηνία

09/01/2026

**Σχεδίαση και Προσομοίωση ΣμηΜΕΑ για σημασιολογική Χαρτογράφηση και
Εντοπισμό Ατόμων με χρήση Τεχνητής Όρασης**

ΜΙΧΑΗΛ ΤΑΜΒΑΚΕΡΑΣ

**Μεταπτυχιακή Διπλωματική Εργασία που υποβάλλεται στο καθηγητικό σώμα για την
μερική εκπλήρωση των υποχρεώσεων απόκτησης του μεταπτυχιακού τίτλου του
Προγράμματος Μεταπτυχιακών Σπουδών «Μη Επανδρωμένα Αυτόνομα και
Τηλεκατευθυνόμενα Συστήματα» του Τμήματος Μηχανικών Βιομηχανικής Σχεδίασης
και Παραγωγής του Πανεπιστημίου Δυτικής Αττικής.**

Περίληψη

Η ταχεία πρόοδος των μη επανδρωμένων εναέριων οχημάτων (MEA - Unmanned Aerial Vehicles, UAVs) έχει διευρύνει σημαντικά το φάσμα εφαρμογών τους, όπως η επιτήρηση (surveillance), η χαρτογράφηση (Mapping) και οι επιχειρήσεις έρευνας και διάσωσης (search and rescue). Τα συστήματα αυτά δύνανται να λειτουργούν είτε υπό τηλεχειρισμό, είτε αυτόνομα, ανάλογα με τις απαιτήσεις της αποστολής. Η ενσωμάτωση προηγμένων αισθητήρων και αλγορίθμων τεχνητής νοημοσύνης ενισχύει τις δυνατότητες πλοήγησής τους, επιτρέποντας την εκτέλεση σύνθετων λειτουργιών χωρίς συνεχή ανθρώπινη παρέμβαση.

Αντικείμενο της παρούσας διπλωματικής εργασίας αποτελεί ο ψηφιακός σχεδιασμός, η μοντελοποίηση ενός εναέριου ρομποτικού συστήματος και η εξαγωγή σε μορφή συμβατή για σύγχρονα περιβάλλοντα προσομοίωσης. Η μελέτη επικεντρώνεται στην ανάπτυξη και ενσωμάτωση αλγορίθμων για τον εντοπισμό ανθρώπινης παρουσίας.

Η υλοποίηση βασίστηκε στις γλώσσες προγραμματισμού C++ και Python, αξιοποιώντας το ρομποτικό οικοσύστημα (ecosystem) Robot Operating System 2 (ROS 2) και τον προσομοιωτή Gazebo. Παράλληλα, χρησιμοποιήθηκαν εργαλεία όπως το RViz, το PX4 Control Stack και το QGroundControl. Η καθοδήγηση του οχήματος εκτελέστηκε μέσω χειροκίνητου ελέγχου με τη χρήση πληκτρολογίου. Για την περιβαλλοντική αντίληψη και την ασφαλή πλοήγηση στον χώρο, αξιοποιήθηκαν τεχνικές ταυτόχρονου εντοπισμού και χαρτογράφησης (Simultaneous Localization and Mapping - SLAM). Επιπλέον, η διαδικασία αναγνώρισης ατόμων βασίστηκε στην ενσωμάτωση του μοντέλου τεχνητής όρασης.

Τα πειραματικά αποτελέσματα καταδεικνύουν ικανοποιητική πτητική ευστάθεια, αξιόπιστη αποφυγή εμποδίων και ακριβή χαρτογράφηση σε πραγματικό χρόνο (real-time). Συνολικά, η συνδυαστική αξιοποίηση ψηφιακής μοντελοποίησης και τεχνικών τεχνητής νοημοσύνης συνθέτει ένα κλιμακούμενο (scalable) και αποδοτικό πλαίσιο για την ανάπτυξη μη επανδρωμένων εναέριων συστημάτων σε σύνθετες αποστολές μεγάλης κλίμακας.

Λέξεις Κλειδιά: Μη Επανδρωμένο Αεροσκάφος, Χαρτογράφηση, Προσομοίωση, Αισθητήρας Σάρωσης, Τεχνητή Όραση, PX4 Στοίβα Ελέγχου.

Abstract

The rapid advancement of Unmanned Aerial Vehicles (UAVs) has significantly expanded their application range, including surveillance, mapping, and search and rescue operations. These systems can operate either via remote control or autonomously, depending on mission requirements. The integration of advanced sensors and Artificial Intelligence (AI) algorithms enhances their navigation capabilities, enabling the execution of complex operations without continuous human intervention.

The objective of this thesis is the digital design and modeling of an aerial robotic system, and its export into a format compatible with modern simulation environments. The study focuses on the development and integration of algorithms for human detection.

Implementation was based on the C++ and Python programming languages, utilizing the Robot Operating System 2 (ROS 2) ecosystem and the Gazebo simulator. Additionally, tools such as RViz, the PX4 Control Stack, and QGroundControl were employed. Vehicle guidance was performed via manual keyboard control. For environmental perception and safe navigation, Simultaneous Localization and Mapping (SLAM) techniques were utilized. Furthermore, the person recognition process was based on the integration of a computer vision model.

Experimental results demonstrate satisfactory flight stability, reliable obstacle avoidance, and accurate real-time mapping. Overall, the combined utilization of digital modeling and AI techniques composes a scalable and efficient framework for the development of unmanned aerial systems in complex, large-scale missions.

Keywords: Unmanned Aerial Vehicle, Mapping, Simulation, LiDAR Sensor, Artificial Intelligence, PX4 Flight Controller.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Δρ. Αβραάμ Χατζόπουλο για την δυνατότητα που μου έδωσε και την εμπιστοσύνη που μου έδειξε, να εκπονήσω την παρούσα αυτή διατριβή.

Πίνακας Περιεχομένων

Περίληψη	v
Abstract.....	vi
Ευχαριστίες	vii
Πίνακας Περιεχομένων.....	viii
Κατάλογος Συντομογραφιών	7
1. Εισαγωγή	9
1.1 Επιστημονική και Πρακτική Σημασία της Έρευνας.....	10
1.2 Σκοπός και Στόχος Εργασίας.....	11
1.3 Σχετικές Επιστημονικές Εργασίες	12
2. Εισαγωγή στα Εναέρια Ρομποτικά Συστήματα (Aerial Robotics).....	14
2.1 Αρχιτεκτονική Υλικού και Λογισμικού.....	14
2.1.1 Υλικό (Hardware)	15
2.1.2 Λογισμικό (Software)	15
2.2 Εφαρμογές Εναέριων Ρομποτικών Συστημάτων	16
2.3 Διαφορές Συστημάτων ΜΕΑ και ΣμηΕΑ.....	17
2.4 Συγκριτική Ανάλυση Εναέριων και Επίγειων Ρομποτικών Συστημάτων ..	18
3. Ψηφιακή Κατασκευή ΜΕΑ και Εξαγωγή για Ρομποτικά Περιβάλλοντα	
Προσομοίωσης.....	20
3.1 Επιλογή Επιμέρους Υποσυστημάτων	21
3.1.1 Σκελετός Πλαισίου (Airframe)	21
3.1.2 Μπαταρία και Σύστημα Διανομής Ισχύος (Battery & Power	
Distribution)	23
3.1.3 Ηλεκτρονικός Ελεγκτής Ταχύτητας (Electronic Speed Controller, ESC)	
24	
3.1.4 Ηλεκτρικός Κινητήρας χωρίς Ψήκτρα (Brushless Motors).....	25
3.1.5 Σύστημα Πλοήγησης και Ελέγχου (Navigation System & Flight Control)	
27	
3.1.6 Μονάδα Τηλεμετρίας και Δορυφορικού Εντοπισμού (Telematic System	
& GNSS System).....	28
3.1.7 Συνοδευτικός Υπολογιστής (Companion Computer)	30

3.1.8	Κάμερα Βάθους (Depth Camera)	31
3.1.9	Αισθητήρας Σάρωσης (LiDAR Sensor).....	33
3.1.10	Έλικας (Propeller).....	34
3.1.11	Σύστημα Προσγείωσης (Landing Gear)	35
3.2	Σχεδίαση Εξαρτημάτων	36
3.2.1	Βάση Σταθεροποίησης Μπαταρίας.....	37
3.2.2	Ιμάντες Στερέωσης Μπαταρίας	38
3.3	Τοποθέτηση και Συναρμολόγηση Εξαρτημάτων.....	41
3.3.1	Συναρμολόγηση Σκελετού Πλαισίου.....	42
3.3.2	Τοποθέτηση Κινητήρων και Ελίκων στους Βραχίονες	43
3.3.3	Τοποθέτηση Εξαρτημάτων Κεντρικού Επιπέδου	46
3.3.4	Τοποθέτηση Εξαρτημάτων Πρώτου Επιπέδου	47
3.3.5	Τοποθέτηση Εξαρτημάτων Ανώτερου Επιπέδου	48
3.3.6	Τοποθέτηση Εξαρτημάτων στο Εμπρόσθιο και Οπίσθιο Τμήμα του Πλαισίου	49
3.3.7	Τοποθέτηση Εξαρτημάτων Κατώτερου Επιπέδου.....	50
3.3.8	Τελική Δομή και Ολοκληρωμένη Απεικόνιση του Οχήματος	51
3.4	Προσδιορισμός Κέντρου Βάρους, Ευθυγράμμιση και Επαλήθευση Συστήματος Αναφοράς	54
3.5	Εξαγωγή σε Πρότυπο Περιγραφής Ρομπότ	56
3.6	Επαλήθευση Εξαγόμενου Γεωμετρικού Μοντέλων	59
4.	Εγκατάσταση και Διαμόρφωση Υποδομής	62
4.1	Λειτουργικό Σύστημα και Εργαλεία Ανάπτυξης.....	62
4.1.1	Εγκατάσταση Λειτουργικού Συστήματος.....	63
4.1.2	Εγκατάσταση Ρομποτικού Οικοσυστήματος και Εργαλεία Ανάπτυξης.....	63
4.1.3	Εγκατάσταση Ρομποτικού Περιβάλλοντος Προσομοίωσης.....	65
4.1.4	Εγκατάσταση PX4 Στοίβα Ελέγχου (PX4 Control Stack).....	67
4.1.5	Εγκαθίδρυση Εικοινωνίας PX4 με ROS2.....	68
4.1.6	Ενσωμάτωση Λογισμικού Σταθμού Εδάφους και Ελέγχου (Ground Station).....	69
4.1.7	Εγκατάσταση Πακέτου Χαρτογράφησης (Mapping)	70
4.1.8	Εγκατάσταση Πακέτου Τεχνητής Όρασης (Computer Vision).....	72

5. Προγραμματισμός Ρομποτικού Συστήματος και Προσομοίωση MEA.....	74
5.1 Δημιουργία Χώρου Εργασίας (workspace)	75
5.1.1 Ανάπτυξη Πακέτου Περιγραφής Ρομπότ (uav_description)	75
5.1.2 Ανάπτυξη Πακέτου Προσομοίωσης (uav_simulation).....	80
5.1.3 Ανάπτυξη Πακέτου Εξωτερικού Ελέγχου Πτήσης (uav_offboard_control).....	82
5.1.4 Ανάπτυξη Πακέτου Πλοήγησης υπό Τηλεχειρισμό (uav_navigation) ...	90
5.1.5 Ανάπτυξη Πακέτου Χαρτογράφησης (uav_mapping).....	92
5.1.6 Ανάπτυξη Πακέτου Τεχνητής Όρασης (uav_vision).....	94
5.1.7 Ανάπτυξη Πακέτου Εκκίνησης Συστήματος (uav_bringup)	96
5.1.8 Αναπαράσταση Συνεργασίας Κόμβων	97
6. Προσομοίωση και Ανάλυση των Αποτελεσμάτων	99
6.1 Εκκίνηση Λογισμικού Σταθμού Εδάφους	99
6.2 Εκκίνηση Διεργασιών Προσομοίωσης και Προετοιμασία για την Απογείωση (Takeoff)	100
6.2.1 Προετοιμασία Προσομοίωσης	100
6.2.2 Εκκίνηση Προσομοίωσης	100
6.3 Εκκίνηση Κόμβου Εξωτερικού Ελέγχου	105
6.4 Εκκίνηση Κόμβου για τον Τηλεχειρισμό του MEA.....	107
6.5 Online Χαρτογράφηση μέσω RTAB-Map.....	108
6.6 Αναγνώριση Ατόμων	111
6.7 Ανάλυση Επικοινωνίας και Διασύνδεσης Συστατικών	112
7. Συμπεράσματα και Μελλοντικές Επεκτάσεις	117
Βιβλιογραφία	119
Παράρτημα Α: Διαδικασία Εγκατάσταση Λογισμικού	123
Παράρτημα Β: Οδηγίες Αναπαραγωγή Προσομοίωσης.....	128
Παράρτημα Γ: Πηγαίος Κώδικας και Αρχεία Διαμόρφωσης.....	129

Κατάλογος Εικόνων

Εικόνα 1: Εφαρμογή MEA στη γεωργία ακρίβειας [5]	9
Εικόνα 2: Χρήση MEA σε Search & Rescue αποστολή [19]	16
Εικόνα 3: Τρισδιάστατη (3D) απεικόνιση του πλαισίου Holybro X650	23
Εικόνα 4: Τρισδιάστατη (3D) απεικόνιση της μπαταρίας λιθίου πολυμερούς (LiPo)	24
Εικόνα 5: Τρισδιάστατη (3D) απεικόνιση του ηλεκτρονικού	25
Εικόνα 6: Τρισδιάστατη (3D) απεικόνιση του	27
Εικόνα 7: Τρισδιάστατη (3D) απεικόνιση του	28
Εικόνα 8: Τρισδιάστατη (3D) απεικόνιση μονάδας	29
Εικόνα 9: Τρισδιάστατη (3D) απεικόνιση συνοδευτικού υπολογιστή	31
Εικόνα 10: Τρισδιάστατη (3D) απεικόνιση της	33
Εικόνα 11: Τρισδιάστατη (3D) απεικόνιση του αισθητήρα	34
Εικόνα 12: Τρισδιάστατη (3D) απεικόνιση του έλικα CW (πάνω) και CCW (κάτω)	35
Εικόνα 13: Τρισδιάστατη (3D) απεικόνιση του συστήματος προσγείωσης	36
Εικόνα 14: Βήματα σχεδίασης βάσης μπαταρίας	38
Εικόνα 15: Τελικό αποτέλεσμα σχεδίασης βάσης μπαταρίας	38
Εικόνα 16: Βήματα σχεδίασης ιμάντα	40
Εικόνα 17: Τελικό αποτέλεσμα σχεδίασης ιμάντα	41
Εικόνα 18: Απεικόνιση του δεξιόστροφου συστήματος συντεταγμένων (right-hand-rule)	42
Εικόνα 19: Πλαίσιο X650	43
Εικόνα 20: Περιστροφή Κινητήρων: CW (Μπλε)	44
Εικόνα 21: Τοποθέτηση και συναρμολόγηση τμημάτων κινητήρα	46
Εικόνα 22: Τοποθέτηση εξαρτημάτων κεντρικού επιπέδου	47
Εικόνα 23: Τοποθέτηση εξαρτημάτων 1ου επιπέδου	48
Εικόνα 24: Τοποθέτηση εξαρτημάτων τελευταίου	49
Εικόνα 25: Τοποθέτηση εξαρτημάτων εμπρός/πίσω	50
Εικόνα 26: Τοποθέτηση εξαρτημάτων κατώτερου επιπέδου	50
Εικόνα 27: Μπροστινή όψη του MEA	51
Εικόνα 28: Δεξί πλαϊνή όψη του MEA	52
Εικόνα 29: Πάνω όψη του MEA	52

Εικόνα 30: Κάτω όψη του MEA.....	52
Εικόνα 31: Μπροστινή πανοραμική όψη του MEA	53
Εικόνα 32: Πίσω πανοραμική όψη του MEA.....	53
Εικόνα 33: Συνολική διάμετρος συμπεριλαμβανομένων και των ελίκων	54
Εικόνα 34: Αριστερά: Απόκλιση του μοντέλου από το κέντρο βάρους. Δεξιά: Διορθωμένες συντεταγμένες μετά την ευθυγράμμιση του κέντρου βάρους με το σημείο προέλευσης.	55
Εικόνα 35: Ρύθμιση για τον κύριο σύνδεσμο	57
Εικόνα 36: Ρύθμιση των αρθρώσεων	58
Εικόνα 37: Ρύθμιση των συνδέσμων	58
Εικόνα 38: Επαλήθευση συντεταγμένων αναφοράς στο MeshLab	60
Εικόνα 39: Επαλήθευση συντεταγμένων του έλικα στο MeshLab	60
Εικόνα 40: Επαλήθευση ορθής εγκατάστασης και έκδοσης Ubuntu	63
Εικόνα 41: Κενό περιβάλλον προσομοίωσης	66
Εικόνα 42: Περιβάλλον QGroundControl (QGC) [42]	70
Εικόνα 43: Είδη χαρτών από τον αλγόριθμο RTAB-Map [43]	71
Εικόνα 44: Απεικόνιση λειτουργίας του αλγόριθμου YOLO [45]	73
Εικόνα 45: Δημιουργία χώρου εργασίας για το έργο	75
Εικόνα 46: Εκτέλεση εντολών στο CLI.....	77
Εικόνα 47: Δημιουργία δομής μέσω urdf_to_graphviz	77
Εικόνα 48: Επικοινωνία των κόμβων σε γραφική αναπαράσταση rqt_graph	78
Εικόνα 49: Ανάλυση συστατικών του MEA στο πρόγραμμα RViz.....	79
Εικόνα 50: Πρόσθετα στο αρχείο του εικονικού κόσμου.....	81
Εικόνα 51: Πρόσθετα στο αρχείο του MEA.....	81
Εικόνα 52: Σύγκριση MEA ανάμεσα σε SolidWorks και Gazebo	82
Εικόνα 53: Ρεαλιστική αναπαράσταση φωτισμού και λεπτομερειών του MEA.....	82
Εικόνα 54: Γραφική αναπαράσταση πλαισίων αναφοράς NED (αριστερά) και ENU (δεξιά) [46].....	89
Εικόνα 55: Roll (αριστερά), Pitch (κέντρο), Yaw (δεξιά).....	92
Εικόνα 56: Ροή των δεδομένων και συγχρονισμός διεργασιών	98
Εικόνα 57: Πρόγραμμα QGC χωρίς διασύνδεση MEA.....	99
Εικόνα 58: Εκτέλεση PX4 SITL στο 1ο τερματικό.....	101

Εικόνα 59: Εκτέλεση XRCE-Agent στο 2ο τερματικό.....	101
Εικόνα 60: Εκτέλεση ROS-GZ-Bridge στο 3ο τερματικό.....	101
Εικόνα 61: Εκτέλεση RTAB-Map χαρτογράφηση στο 4ο τερματικό	102
Εικόνα 62: Εκτέλεση αναγνώριση ατόμων μέσω YOLO στο 5ο τερματικό.....	102
Εικόνα 63: Φόρτωση MEA μέσα στο πρόγραμμα QGC	103
Εικόνα 64: Διάταξη κινητήρων στο περιβάλλον QGC.....	103
Εικόνα 65: Έλεγχο φοράς κινητήρων στο QGC.....	104
Εικόνα 66: Παρουσίαση του MEA μέσα στο ρεαλιστικό εικονικό.....	105
Εικόνα 67: Εκτέλεση εξωτερικού ελέγχου	106
Εικόνα 68: Κατάσταση HOVER_MODE του MEA	106
Εικόνα 69: Μενού πλοήγησης για τον τηλεχειρισμό.....	107
Εικόνα 70: Πλοήγηση στο περιβάλλον Gazebo	108
Εικόνα 71: Οπτικοποίηση δεδομένων σάρωσης του αισθητήρα LiDAR.....	109
Εικόνα 72: Προβολή δημιουργίας του χάρτη στο RViz.....	110
Εικόνα 73: Απεικόνιση δεδομένων στο rtabmap_viz.....	111
Εικόνα 74: Αναγνώριση ατόμων (1).....	112
Εικόνα 75: Αναγνώριση ατόμων (2).....	112

Κατάλογος Σχημάτων

Σχήμα 1: Αρχιτεκτονική επικοινωνίας μέσω Micro XRCE-DDS Agent [41].....	68
Σχήμα 2: Γραφική αναπαράσταση δομής μέσω του εργαλείου urdf_to_graphviz.....	78
Σχήμα 3: Ροή εκτέλεσης μηχανής πεπερασμένων καταστάσεων (1)	86
Σχήμα 4: Ροή εκτέλεσης μηχανής πεπερασμένων καταστάσεων (2)	87
Σχήμα 5: Γραφική αναπαράσταση της αναθεωρημένης δομής μέσω του εργαλείου urdf_to_graphviz.....	94
Σχήμα 6: Ανάλυση επικοινωνίας του κόμβου robot_state_publisher.....	113
Σχήμα 7: Ανάλυση επικοινωνίας του κόμβου gazebo_bridge.....	114
Σχήμα 8: Ανάλυση επικοινωνίας του κόμβου TF.....	114
Σχήμα 9: Ανάλυση επικοινωνίας του κόμβου px4_ctrl.....	115
Σχήμα 10: Ανάλυση επικοινωνίας του κόμβου px4_prop_js_publisher	115
Σχήμα 11: Ανάλυση επικοινωνίας του κόμβου keyboard_teleop.....	115
Σχήμα 12: Ανάλυση επικοινωνίας του κόμβου odom_to_tf.....	115
Σχήμα 13: Ανάλυση επικοινωνίας του κόμβου rtabmap	116
Σχήμα 14: Ανάλυση επικοινωνίας του κόμβου rtabmap_viz	116
Σχήμα 15: Ανάλυση επικοινωνίας του κόμβου object_detection.....	116

Κατάλογος Πινάκων

Πίνακας 1: Υπολογιστικό σύστημα σχεδίασης και ανάπτυξης λογισμικού	20
Πίνακας 2: Τεχνικά χαρακτηριστικά πλαισίου X650 [23]	22
Πίνακας 3: Τεχνικά χαρακτηριστικά μπαταρίας [24].....	24
Πίνακας 4: Τεχνικά χαρακτηριστικά ηλεκτρονικού ελεγκτή ταχύτητας [25]	25
Πίνακας 5: Τεχνικά χαρακτηριστικά κινητήρα χωρίς Ψήκτρα [26]	26
Πίνακας 6: Τεχνικά χαρακτηριστικά του συστήματος πλοήγησης [27]	28
Πίνακας 7: Τεχνικά χαρακτηριστικά μονάδα τηλεμετρίας [28]	29
Πίνακας 8: Τεχνικά χαρακτηριστικά δορυφορικής μονάδας GPS [29].....	29
Πίνακας 9: Τεχνικά χαρακτηριστικά Pixhawk Jetson Baseboard [32].....	31
Πίνακας 10: Τεχνικά χαρακτηριστικά κάμερα βάθους [33]	32
Πίνακας 11: Τεχνικά χαρακτηριστικά αισθητήρα σάρωσης [34].....	34
Πίνακας 12: Τεχνικά χαρακτηριστικά του έλικα [35]	35
Πίνακας 13: Τεχνικά χαρακτηριστικά συστήματος προσγείωσης [23]	36
Πίνακας 14: Ενδεικτικά βήματα της διαδικασίας σχεδίασης	37
Πίνακας 15: Βήματα σχεδίασης μάντων στερέωσης.....	39
Πίνακας 16: Βήματα συναρμολόγησης και τοποθέτησης του κινητήρα CW	45
Πίνακας 17: Διαδικασία για την επίτευξη της απεικόνισης	46
Πίνακας 18: Βήματα για τον προσδιορισμό κέντρου βάρους και επιδιόρθωση απόκλισης	55
Πίνακας 19: Βήματα ρύθμισης για την εξαγωγή σε URDF (1).....	57
Πίνακας 20: Βήματα ρύθμισης για την εξαγωγή σε URDF (2).....	57
Πίνακας 21: Βήματα ρύθμισης για την εξαγωγή σε URDF (3).....	58
Πίνακας 22: Βήματα ρύθμισης για την εξαγωγή σε URDF (4).....	59
Πίνακας 23: Απαιτήσεις συστήματος για Ubuntu 24.04 LTS [37]	63
Πίνακας 24: Πρόσθετα πακέτα και εργαλεία του ROS 2	65
Πίνακας 25: Σύγκριση Gazebo Sim με άλλα περιβάλλοντα ρομποτικής	67
Πίνακας 26: Διαδικασία εγκατάστασης PX4-Autopilot.....	68
Πίνακας 27: Εγκαθίδρυση εικοινωνίας PX4 με ROS 2.....	69
Πίνακας 28: Σειρά ανάπτυξης πακέτων.....	74

Πίνακας 29: Διαδικασία εκτέλεσης για τη δημιουργία του πακέτου uav_description	76
Πίνακας 30: Διαδικασία εκτέλεσης για τη δημιουργία του πακέτου uav_simulation	81
Πίνακας 31: Διαδικασία εκτέλεσης για τη δημιουργία του πακέτου uav_offboard_ctrl	84
Πίνακας 32: Συστήματα αναφοράς PX4 και ROS 2.....	89
Πίνακας 33: Διαδικασία εκτέλεσης για τη δημιουργία του πακέτου uav_navigation	91
Πίνακας 34: Διαδικασία εκτέλεσης για τη δημιουργία του πακέτου uav_mapping...	93
Πίνακας 35: Διαδικασία εκτέλεσης για τη δημιουργία του πακέτου uav_vision	96
Πίνακας 36: Μεταβάσεις κατά την εκτέλεση της FSM.....	106

Κατάλογος Συντομογραφιών

CAD	Computer Aided Design
UAV	Unmanned Aerial Vehicle
UAS	Unmanned Aircraft System
UGV	Unmanned Ground Systems
VTOL	Vertical Take-Off and Landing
MEA	Μη Επανδρωμένο Αεροσκάφος
ΣμηEA	Σύστημα Μη Επανδρωμένων Αεροσκαφών
SITL	Software In The Loop
ROS	Robot Operating System
CoM	Κέντρο Μάζας
TN	Τεχνητή Νοημοσύνη
URDF	Unified Robot Description Format
Xacro	Extensible Markup Language (XML) Macros
LiPo	Lithium Polymer Battery
SLAM	Simultaneous Localization and Mapping
LiDAR	Light Detection and Ranging
GPS	Global Positioning System
LTS	Long Term Support
STP / STEP	Standard for the Exchange of Product Model Data
SLDPRT	SolidWorks Part File
SLDASM	SolidWorks Assembly File
AI	Artificial Intelligence
CUDA	Compute Unified Device Architecture
IMU	Inertial Measurement Unit
ESC	Electronic Speed Controller
PDB	Power Distribution Board
CW	Clockwise
CCW	Counter-Clockwise
CoM	Center of Mass
RCL	ROS Client Library

DDS	Data Distribution Service
MEMS	Micro-Electro-Mechanical Systems
STL	Stereolithography
OpenCV	Open Source Computer Vision Library
CLI	Command Line Interface
FSM	Finite State Machine
SDF	Simulation Description Format

1. Εισαγωγή

Η ραγδαία εξέλιξη των καινοτόμων τεχνολογιών κατά τα τελευταία έτη έχει οδηγήσει στην ανάπτυξη εξελιγμένων εναέριων ρομποτικών συστημάτων. Τα συστήματα αυτά αποτελούν πλέον θεμελιώδη εργαλεία σε ένα εκτεταμένο φάσμα εφαρμογών, το οποίο περιλαμβάνει τη βιομηχανία, την επιστημονική έρευνα, την εθνική άμυνα και την εξερεύνηση του διαστήματος.

Παράλληλα, τα οχήματα αυτά έχουν εδραιωθεί σε ποικίλες γήινες εφαρμογές με σημαντικό κοινωνικό και οικονομικό αντίκτυπο. Στον τομέα της ασφάλειας, χρησιμοποιούνται για την επιτήρηση συνόρων, τον έλεγχο υποδομών ζωτικής σημασίας και την παρακολούθηση φυσικών καταστροφών, παρέχοντας δεδομένα σε πραγματικό χρόνο χωρίς την έκθεση προσωπικού σε κινδύνους. Στο εμπορικό επίπεδο, αξιοποιούνται στην αεροφωτογράφιση (aerial photography) [1], τη χαρτογράφηση εκτάσεων (land cover) [2], την γεωργία ακρίβειας (precision agriculture) [3] και την επιθεώρηση ενεργειακών μονάδων, όπως οι ανεμογεννήτριες και τα φωτοβολταϊκά πάρκα (photovoltaic farms) [4]. Οι εν λόγω αποστολές, που παραδοσιακά απαιτούσαν δαπανηρές και επικίνδυνες ανθρώπινες παρεμβάσεις, υλοποιούνται πλέον με μειωμένο κόστος και αυξημένη ασφάλεια.



Εικόνα 1: Εφαρμογή ΜΕΑ στη γεωργία ακρίβειας [5]

Η εκτεταμένη διείσδυση των συστημάτων αυτών αποδίδεται σε κρίσιμα τεχνολογικά επιτεύγματα, όπως η ενίσχυση της υπολογιστικής ισχύος, η εξέλιξη των αισθητήρων

και η ενσωμάτωση αλγορίθμων Τεχνητής Νοημοσύνης (Artificial Intelligence, AI). Οι δυνατότητες αυτόνομης πλοήγησης και λήψης αποφάσεων μετατρέπουν τα συστήματα αυτά σε ενεργές μονάδες δράσης σε δυναμικά περιβάλλοντα. Έτσι η παγκόσμια αγορά τους αναμένεται να παρουσιάσει σημαντική οικονομική ανάπτυξη στα επόμενα χρόνια.

Η πολυμορφία των εφαρμογών έχει οδηγήσει στη δημιουργία διαφορετικών κατηγοριών, οι οποίες ταξινομούνται με βάση το μέγιστο βάρος απογείωσης, την εμβέλεια και τον τύπο πτήσης. Διακρίνονται σε πολυκόπτερα, που προσφέρουν ευελιξία και δυνατότητα κάθετης απογείωσης και προσγείωσης (Vertical Take-Off and Landing, VTOL), σε αεροσκάφη σταθερών πτερύγων για αποστολές μεγάλης διάρκειας, καθώς και σε υβριδικές πλατφόρμες. Η χρήση καμερών βάθους και αισθητήρων όρασης επιτρέπει την ακριβή τρισδιάστατη αντίληψη του χώρου, ενισχύοντας την αυτόνομη κίνηση.

Στην παρούσα μελέτη, η ερευνητική διαδικασία επικεντρώθηκε στην αρχιτεκτονική ανάπτυξη και την πειραματική αξιολόγηση ενός μη επανδρωμένου συστήματος μέσω προηγμένων ψηφιακών εργαλείων και περιβάλλον προσομοίωσης.

1.1 Επιστημονική και Πρακτική Σημασία της Έρευνας

Σε επιστημονικό επίπεδο, η παρούσα έρευνα συμβάλλει στη διεύρυνση της γνώσης σχετικά με την ανάλυση δεδομένων αισθητήρων σε πραγματικό χρόνο. Η συνδυαστική μελέτη της αεροδυναμικής, της μηχανικής πτήσης και των μεθόδων υπολογιστικής όρασης προσφέρει σημαντικές προοπτικές βελτίωσης της λειτουργικής αποδοτικότητας των συστημάτων αυτών σε δυναμικά περιβάλλοντα.

Η πρακτική αξία της μελέτης έγκειται στην ανάδειξη της ψηφιακής μοντελοποίησης (digital modeling) και της προσομοίωσης ως κρίσιμων προ-κατασκευαστικών σταδίων. Η χρήση προτύπων περιγραφής ρομπότ επιτρέπει τον ακριβή έλεγχο της τοποθέτησης υποσυστημάτων και της κατανομής μάζας, περιορίζοντας τα πιθανά σφάλματα κατά τη φυσική υλοποίηση. Μέσω του οικοσυστήματος ROS 2, του προσομοιωτή Gazebo και του εργαλείου οπτικοποίησης RViz, καθίσταται δυνατή η

αξιολόγηση των αλγορίθμων εναέριας ρομποτικής, χωρίς την ανάγκη για δαπανηρές δοκιμές με φυσικά πρωτότυπα.

Συμπερασματικά, η διεπιστημονική αυτή προσέγγιση προάγει την ασφάλεια και τη μείωση του κόστους παραγωγής. Η εργασία δεν περιορίζεται στην ανάπτυξη ενός ερευνητικού πρωτοτύπου, αλλά διαμορφώνει ένα επεκτάσιμο πλαίσιο σχεδίασης με άμεση εφαρμογή σε βιομηχανικούς, επιστημονικούς και κοινωνικούς τομείς.

1.2 Σκοπός και Στόχος Εργασίας

Ο πρωταρχικός σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη και η πειραματική αξιολόγηση ενός ολοκληρωμένου εναέριου ρομποτικού συστήματος σε περιβάλλον προσομοίωσης υψηλής πιστότητας. Η έρευνα εστιάζει στη δημιουργία ενός ψηφιακού MEA, το οποίο επιτρέπει τη μελέτη σύνθετων διεργασιών, όπως ο αυτόνομος εντοπισμός και η παρακολούθηση ανθρώπινης παρουσίας, εντός ενός ελεγχόμενου εικονικού πλαισίου.

Επιμέρους αντικειμενικοί στόχοι της εργασίας αποτελούν:

- Η ακριβής τρισδιάστατη σχεδίαση και παραμετροποίηση του σκελετού και των υποσυστημάτων του οχήματος, διασφαλίζοντας την πλήρη συμβατότητα με πρότυπα περιγραφής ρομπότ (Unified Robot Description Format - URDF).
- Η ενοποίηση του οικοσυστήματος ROS 2 με τη στοίβα ελέγχου PX4, επιτυγχάνοντας τον απρόσκοπτο συγχρονισμό μεταξύ των αλγορίθμων υψηλού επιπέδου και του εικονικού υλικού πτήσης.
- Η εφαρμογή μεθόδων ταυτόχρονου εντοπισμού και χαρτογράφησης SLAM (Simultaneous Localization and Mapping) [6] για την πλοήγηση του οχήματος σε δυναμικά περιβάλλοντα.
- Η ενσωμάτωση αρχιτεκτονικών βαθιάς μάθησης (You Only Look Once - YOLO) για την αναγνώριση ατόμων σε πραγματικό χρόνο μέσω της ανάλυσης οπτικών δεδομένων από ειδικές κάμερες.

Τελική επιδίωξη της μελέτης είναι η διαμόρφωση ενός επεκτάσιμου πλαισίου (framework) προσομοίωσης, το οποίο θα λειτουργήσει ως υπόβαθρο για τη

μελλοντική ανάπτυξη και δοκιμή προηγμένων εφαρμογών εναέριας επιτήρησης και αυτόνομης λήψης αποφάσεων.

1.3 Σχετικές Επιστημονικές Εργασίες

Στο πλαίσιο της ανάπτυξης συστημάτων ΜΕΑ, ιδιαίτερο ενδιαφέρον παρουσιάζουν οι εργασίες που επικεντρώνονται στη διασύνδεση προηγμένων περιβαλλόντων προσομοίωσης με αλγορίθμους τεχνητής όρασης και χαρτογράφησης.

Συγκεκριμένα, ο Enrico [7] στην εργασία του αναλύει τη μεθοδολογία ελέγχου offboard σε περιβάλλοντα ROS 2 και PX4, καταδεικνύοντας πώς η χρήση του επιπέδου αφαίρεσης του ROS επιτρέπει τον ακριβή συντονισμό μεταξύ του αυτόματου πιλότου και εξωτερικών αλγορίθμων πλοήγησης. Η έρευνά του δίνει ιδιαίτερη έμφαση στη χρήση του Gazebo για την προσομοίωση SITL, επιτρέποντας την αξιολόγηση εντολών στάσης και ταχύτητας πριν από την πραγματική πτητική εφαρμογή, διασφαλίζοντας έτσι τη σταθερότητα της επικοινωνίας.

Παράλληλα, οι Hofmann et al. [8] παρουσιάζουν ένα σύστημα σημασιολογικής χαρτογράφησης (semantic mapping) υψηλής απόδοσης που συνδυάζει τον αλγόριθμο YOLOv8 με το RTAB-Map. Στα πειράματά τους απέδειξαν ότι η ενσωμάτωση σημασιολογικών πληροφοριών από την τεχνητή όραση στον τρισδιάστατο χάρτη επιτρέπει στο ΜΕΑ να αναγνωρίζει και να ταξινομεί αντικείμενα του περιβάλλοντος σε πραγματικό χρόνο. Επιπλέον, η προσέγγισή τους βελτιώνει την ακρίβεια του SLAM σε δυναμικά περιβάλλοντα, καθώς επιτρέπει το φιλτράρισμα κινούμενων εμποδίων που συχνά προκαλούν σφάλματα στην οδομετρία.

Επιπροσθέτως, οι Wang κ.ά. [9] εστιάζουν στη βελτίωση του YOLOv8 (μοντέλο FFG-YOLO) για τον εντοπισμό στόχων από ελαφριά ΜΕΑ. Η εργασία τους αναδεικνύει πώς η επεξεργασία της ροής εικόνας μπορεί να οδηγήσει σε αποτελεσματικό εντοπισμό ακόμα και μικρών αντικειμένων ή ατόμων, βελτιώνοντας την ικανότητα του drone για αυτόνομη λήψη αποφάσεων σε κρίσιμες αποστολές. Η χρήση προηγμένων τεχνικών εξαγωγής χαρακτηριστικών επιτρέπει στο σύστημα να λειτουργεί με υψηλή ακρίβεια, παρά τους περιορισμένους υπολογιστικούς πόρους των onboard συστημάτων.

Ακόμη, οι Fanta-Jende et al. [10] περιγράφουν μια ολοκληρωμένη αρχιτεκτονική για την online χαρτογράφηση και τον εντοπισμό αντικειμένων σε πραγματικό χρόνο. Χρησιμοποιώντας τεχνικές vSLAM και σημασιολογική επισήμανση (semantic labelling), το σύστημά τους κατασκευάζει χάρτες που επιτρέπουν στο ΜΕΑ να αντιλαμβάνεται τη δομή του χώρου. Οι μετρήσεις τους έδειξαν ότι ο συνδυασμός οπτικών δεδομένων και αλγορίθμων μηχανικής μάθησης παρέχει ένα περιβάλλον υψηλής πιστότητας για την κατανόηση του περιβάλλοντος, μειώνοντας την ανάγκη για χρονοβόρα μετα-επεξεργασία των δεδομένων.

Τέλος, ο Labbé [11] τεκμηριώνει την καθολική ισχύ του RTAB-Map ως μια βιβλιοθήκη ανοικτού κώδικα ικανή να διαχειρίζεται δεδομένα από LiDAR και κάμερες βάθους για τη δημιουργία χαρτών μεγάλης κλίμακας, προσφέροντας τη βάση για τη σταθερή πλοήγηση που απαιτείται σε σύγχρονες εφαρμογές.

2. Εισαγωγή στα Εναέρια Ρομποτικά Συστήματα (Aerial Robotics)

Η εναέρια ρομποτική αποτελεί έναν διεπιστημονικό κλάδο που συνδυάζει την αεροναυπηγική, την πληροφορική και τις θετικές επιστήμες, εστιάζοντας σε πτητικές πλατφόρμες με δυνατότητα αυτόνομης ή τηλεκατευθυνόμενης λειτουργίας. Τα συστήματα αυτά υπερβαίνουν τον συμβατικό έλεγχο πτήσης, ενσωματώνοντας αισθητήρες υψηλής πιστότητας και μηχανισμούς επεξεργασίας πληροφοριών σε πραγματικό χρόνο (real-time processing), επιτρέποντας τη συλλογή δεδομένων και την εκτέλεση σύνθετων αποστολών μέσω της αλληλεπίδρασης με το περιβάλλον.

Η ιστορική εξέλιξη των εναέριων συστημάτων εκκινεί από τις στρατιωτικές εφαρμογές των αρχών του 19ού αιώνα [12] και επεκτείνεται σε πολιτικά πεδία, όπως η χαρτογράφηση, μετά τη δεκαετία του 1990 [13]. Η μετάβαση αυτή υποστηρίχθηκε από κρίσιμα ορόσημα, όπως η σμίκρυνση των μικροεπεξεργαστών, η ανάπτυξη μπαταριών λιθίου και η χρήση αισθητήρων τύπου MEMS (Micro-Electro-Mechanical Systems).

Στη σύγχρονη εποχή, η ενσωμάτωση τεχνικών μηχανικής μάθησης (machine learning) και βαθιάς μάθησης (deep learning) έχει καταστήσει εφικτές λειτουργίες αιχμής, όπως η αυτόνομη αποφυγή εμποδίων (obstacle avoidance) [14], η αναγνώριση αντικειμένων και η ρομποτική σμηνών (swarm robotics) [15]. Οι εφαρμογές τους εκτείνονται από τη γεωργία (agriculture) [16] και την έρευνα και διάσωση (search and rescue) [17] έως την εξωπλανητική εξερεύνηση, με χαρακτηριστικό παράδειγμα την αποστολή του ελικοπτέρου Ingenuity στον Άρη το 2021 [18]. Η εξέλιξη αυτή επιβεβαιώνει τη μεταμόρφωση των μη επανδρωμένων αεροχημάτων (Unmanned Aerial Vehicles, UAVs) σε πολυδιάστατες τεχνολογικές πλατφόρμες με ευρύ κοινωνικό και οικονομικό αντίκτυπο.

2.1 Αρχιτεκτονική Υλικού και Λογισμικού

Η λειτουργική αρτιότητα ενός εναέριου ρομποτικού συστήματος βασίζεται στη συντονισμένη αλληλεπίδραση μεταξύ των δομικών υποσυστημάτων του υλικού και των επιπέδων του λογισμικού. Η αρχιτεκτονική αυτή διασφαλίζει την επεξεργασία

των δεδομένων από τους αισθητήρες και την παραγωγή των κατάλληλων εντολών ελέγχου για την εκτέλεση της πτήσης.

2.1.1 Υλικό (Hardware)

Το υλικό μέρος ενός μη επανδρωμένου συστήματος αποτελείται από το πλαίσιο (airframe) και το σύστημα πρόωσης, το οποίο περιλαμβάνει τους κινητήρες, τους ηλεκτρονικούς ελεγκτές ταχύτητας (electronic speed controllers, ESC) και τους έλικες. Κεντρικό στοιχείο αποτελεί η μονάδα ελέγχου πτήσης (flight controller), η οποία ενσωματώνει αισθητήρες όπως η αδρανειακή μονάδα μέτρησης (inertial measurement unit, IMU), το βαρόμετρο και το σύστημα δορυφορικού εντοπισμού (Global Navigation Satellite System, GNSS).

Για την εκτέλεση προηγμένων διεργασιών, όπως η υπολογιστική όραση και η αυτόνομη χαρτογράφηση, χρησιμοποιείται συχνά ένας συνοδός υπολογιστής (companion computer). Αυτή η μονάδα διαθέτει υψηλή υπολογιστική ισχύ και συνδέεται με εξειδικευμένους αισθητήρες αντίληψης, όπως κάμερες βάθους (depth cameras) και συστήματα LiDAR, επιτρέποντας στο όχημα να αναγνωρίζει το περιβάλλον του σε τρεις διαστάσεις.

2.1.2 Λογισμικό (Software)

Το λογισμικό ενός εναέριου ρομπότ οργανώνεται σε ιεραρχικά επίπεδα, ξεκινώντας από το υλικολογισμικό (firmware) του ελεγκτή πτήσης, το οποίο διαχειρίζεται τη σταθερότητα και τους βρόχους ελέγχου σε πραγματικό χρόνο. Στα συστήματα υψηλότερου επιπέδου, η χρήση του λειτουργικού συστήματος Linux και ειδικότερα της διανομής Ubuntu αποτελεί το πρότυπο για την ανάπτυξη ρομποτικών εφαρμογών.

Το ενδιάμεσο λογισμικό (middleware) ROS 2 διευκολύνει την επικοινωνία μεταξύ των διαφόρων διεργασιών, επιτρέποντας την ανταλλαγή μηνυμάτων μεταξύ των αισθητήρων και των αλγορίθμων πλοήγησης. Η αρχιτεκτονική συμπληρώνεται από το λογισμικό σταθμού εδάφους (ground control station), το οποίο παρέχει τη διεπαφή διαχείρισης των αποστολών και την οπτικοποίηση της κατάστασης του οχήματος κατά τη διάρκεια της πτήσης.

2.2 Εφαρμογές Εναέριων Ρομποτικών Συστημάτων

Η πολυμορφία δυνατοτήτων των εναέριων ρομποτικών συστημάτων επιτρέπει την ενσωμάτωσή τους σε ένα ευρύ φάσμα τομέων με σημαντική κοινωνική και οικονομική επίδραση. Η ικανότητά τους να προσεγγίζουν δυσπρόσιτες περιοχές και να συλλέγουν δεδομένα υψηλής ανάλυσης τα καθιστά αναπόσπαστο κομμάτι της σύγχρονης βιομηχανίας και της επιστημονικής έρευνας.

Στον τομέα της πολιτικής προστασίας και της ασφάλειας, τα μη επανδρωμένα αεροχήματα χρησιμοποιούνται εκτενώς σε αποστολές έρευνας και διάσωσης, όπου ο χρόνος απόκρισης είναι κρίσιμος. Η χρήση θερμικών καμερών και αισθητήρων επιτρέπει τον εντοπισμό αγνοουμένων σε περιβάλλοντα με περιορισμένη ορατότητα ή επικίνδυνες συνθήκες. Παράλληλα, η επιτήρηση συνόρων και η παρακολούθηση κρίσιμων υποδομών, όπως οι αγωγοί ενέργειας και τα δίκτυα μεταφορών, πραγματοποιούνται με αυξημένη αποτελεσματικότητα και μειωμένο κόστος σε σύγκριση με τις παραδοσιακές μεθόδους.



Εικόνα 2: Χρήση MEA σε Search & Rescue αποστολή [19]

Στο εμπορικό και βιομηχανικό επίπεδο, η γεωργία ακριβείας αποτελεί έναν από τους ταχύτερα αναπτυσσόμενους κλάδους, με τα εναέρια συστήματα να συμβάλλουν στη βελτιστοποίηση της παραγωγής μέσω της πολυφασματικής απεικόνισης και του ψεκασμού καλλιεργειών.

Τέλος, η εναέρια ρομποτική διαδραματίζει καθοριστικό ρόλο στην τοπογραφία και τη χαρτογράφηση (mapping). Η παραγωγή τρισδιάστατων μοντέλων εδάφους και η

ψηφιακή αποτύπωση αρχαιολογικών χώρων ή αστικών περιοχών επιτυγχάνονται με υψηλή ακρίβεια μέσω φωτογραμμετρικών μεθόδων [20]. Οι εφαρμογές αυτές αναδεικνύουν τη μετάβαση από την απλή παρατήρηση στην ενεργό διαχείριση και ανάλυση χωρικών δεδομένων σε πραγματικό χρόνο.

2.3 Διαφορές Συστημάτων ΜΕΑ και ΣμηΕΑ

Η διάκριση μεταξύ των ΜΕΑ και των Συστημάτων μη Επανδρωμένων Αεροσκαφών (Unmanned Aircraft Systems - UAS) είναι θεμελιώδης για την ακριβή περιγραφή της λειτουργίας τους σε ακαδημαϊκό και τεχνικό επίπεδο. Παρόλο που οι όροι χρησιμοποιούνται συχνά ως ταυτόσημοι στην καθομιλουμένη, στην επιστημονική βιβλιογραφία αντιπροσωπεύουν διαφορετικές έννοιες της εναέριας ρομποτικής.

Ο όρος ΜΕΑ αναφέρεται αποκλειστικά στην πτητική πλατφόρμα, δηλαδή στο φυσικό όχημα που φέρει τους κινητήρες, τους έλικες, τους αισθητήρες και τον ελεγκτή πτήσης. Πρόκειται για το υποσύστημα που εκτελεί την κίνηση στον τρισδιάστατο χώρο και αποτελεί το αντικείμενο της ψηφιακής μοντελοποίησης και του δομικού σχεδιασμού.

Αντιθέτως, ο όρος ΣμηΕΑ υιοθετήθηκε για να περιγράψει το σύνολο των στοιχείων που είναι απαραίτητα για την ολοκλήρωση μιας αποστολής. Ένα πλήρες σύστημα περιλαμβάνει, εκτός από το αεροκάφος ή ΜΕΑ, τον σταθμό εδάφους (ground control station, GCS), τις τηλεπικοινωνιακές ζεύξεις δεδομένων (data links), καθώς και το απαραίτητο λογισμικό για τον σχεδιασμό και την εποπτεία της πτήσης. Η συστημική αυτή προσέγγιση αναγνωρίζει ότι η πτήση δεν είναι μια μεμονωμένη ενέργεια του οχήματος, αλλά το αποτέλεσμα της συνεχούς αλληλεπίδρασης μεταξύ της πτητικής μονάδας και της υποδομής υποστήριξης.

Στο πλαίσιο της παρούσας εργασίας, η εστίαση δίνεται στο ΣμηΕΑ, καθώς η έρευνα δεν περιορίζεται στον σχεδιασμό και προσομοίωση του οχήματος, αλλά επεκτείνεται στην ενσωμάτωση του λογισμικού ελέγχου και των συστημάτων επικοινωνίας.

2.4 Συγκριτική Ανάλυση Εναέριων και Επίγειων Ρομποτικών Συστημάτων

Η εξέλιξη της ρομποτικής επιστήμης έχει οδηγήσει σε ποικίλες πλατφόρμες, με τα εναέρια και επίγεια Οχήματα (Unmanned Ground Vehicles, UGVs) να αποτελούν τις πλέον διαδεδομένες κατηγορίες. Παρότι τα συστήματα αυτά συγκλίνουν σε θεμελιώδεις λειτουργικές αρχές, όπως η αυτόνομη πλοήγηση (autonomous navigation), η ενσωμάτωση αισθητήρων και η αξιοποίηση αλγορίθμων τεχνητής νοημοσύνης, παρουσιάζουν ουσιώδεις αποκλίσεις. Οι διαφοροποιήσεις αυτές εντοπίζονται στη δυναμική της κίνησης, στις απαιτήσεις του δομικού σχεδιασμού και στο εύρος των επιχειρησιακών εφαρμογών τους.

Η παρούσα συγκριτική ανάλυση αποτυπώνεται συνοπτικά στον παρακάτω πίνακα, ο οποίος επισημαίνει τα κύρια τεχνικά και επιχειρησιακά χαρακτηριστικά που διαχωρίζουν τις δύο κατηγορίες.

Κριτήριο	MEA (UAV)	MEE0 (UGV)
Κινητικότητα και Χώρος Λειτουργίας	Τρισδιάστατη (3D) ελευθερία κίνησης, υπέρβαση φυσικών εμποδίων και δυνατότητα κάθετης απογείωσης.	Δισδιάστατη (2D) κίνηση σε άμεση σύνδεση με το έδαφος και τις κλίσεις της επιφάνειας.
Αισθητήρες και Αντίληψη Περιβάλλοντος	Χρήση GNSS, IMU και LiDAR για πλοήγηση στον τρισδιάστατο χώρο και έλεγχο υψομέτρου.	Παρόμοια συστήματα με έμφαση στην ανάλυση της τοπογραφίας και τον εντοπισμό εδάφινων εμποδίων.
Ενεργειακές Απαιτήσεις	Συνεχής κατανάλωση ισχύος για τη διατήρηση της πτήσης, γεγονός που περιορίζει την αυτονομία.	Υψηλότερη ενεργειακή αποδοτικότητα λόγω στήριξης στο έδαφος και δυνατότητα μεταφοράς μεγαλύτερου φορτίου.
Σχεδιαστικές και Λειτουργικές Προκλήσεις	Αντιμετώπιση δυναμικών αεροδυναμικών φαινομένων που απαιτούν προηγμένους ελεγκτές ευστάθειας.	Ανάγκη για στιβαρό μηχανικό σχεδιασμό αναρτήσεων και συστημάτων κίνησης για ανώμαλα εδάφη.

Πεδίο Εφαρμογών	Εναέρια επιτήρηση, φωτογραμμετρία, γεωργία ακριβείας και αποστολές ταχείας απόκρισης.	Μεταφορά βαρέων φορτίων, επιθεώρηση εσωτερικών χώρων και αποστολές μακράς παραμονής στο πεδίο.
Συνολική Αξιολόγηση	Υπερέχουν σε ταχύτητα και πρόσβαση σε απρόσιτα σημεία, προσφέροντας ευρεία οπτική κάλυψη.	Προσφέρουν μεγαλύτερη αντοχή σε χρονικά εκτεταμένες επιχειρήσεις και σταθερότητα σε βαριές εργασίες.

Πίνακας 1: Σύγκριση εναέρια και επίγεια Οχήματα.

3. Ψηφιακή Κατασκευή ΜΕΑ και Εξαγωγή για Ρομποτικά Περιβάλλοντα Προσομοίωσης

Η ψηφιακή κατασκευή αποτελεί το αρχικό στάδιο ανάπτυξης ενός ΜΕΑ, συνενώνοντας τον σχεδιασμό και τη μοντελοποίηση σε μια ενιαία διεργασία. Η μέθοδος αυτή εξασφαλίζει τον ακριβή προσδιορισμό των διαστάσεων και της χωροθέτησης όλων των υποσυστημάτων πριν από την υλική υλοποίηση. Με τη χρήση λογισμικού σχεδίασης (Computer-Aided Design, CAD), επιτυγχάνεται η ρεαλιστική απόδοση της γεωμετρίας, ο υπολογισμός κρίσιμων μηχανικών ιδιοτήτων και η προετοιμασία του μοντέλου για τα ψηφιακά περιβάλλοντα.

Το υπολογιστικό σύστημα που χρησιμοποιήθηκε τόσο για την ψηφιακή σχεδίαση και συναρμολόγηση, όσο και για τον προγραμματισμό και την προσομοίωση είχε τις εξής ιδιότητες:

Laptop Alienware m16 R1	
CPU	13th Gen Intel Core i9
RAM	32 GB system memory
GPU	NVIDIA GeForce RTX 4080 12GB
HDD	1TB of free hard drive space
OS	Dual Boot Windows 11 Home, Ubuntu 24.04 LTS

Πίνακας 1: Υπολογιστικό σύστημα σχεδίασης και ανάπτυξης λογισμικού

Στην παρούσα ενότητα αναλύεται η επιλογή των δομικών εξαρτημάτων του οχήματος με βάση τις τεχνικές προδιαγραφές της έρευνας. Περιγράφεται η διαδικασία ψηφιακής σύνθεσης, η οποία περιλαμβάνει τη συναρμολόγηση και τη δημιουργία νέων εξαρτημάτων όπου αυτό κρίθηκε απαραίτητο. Ιδιαίτερη έμφαση αποδίδεται στον ορθό καθορισμό του σημείου αναφοράς (origin), στην εφαρμογή του κανόνα του δεξιούφ χεριού (right-hand rule) και στον ακριβή εντοπισμό του κέντρου μάζας του οχήματος (Center of Mass, CoM).

Η εξαγωγή του μοντέλου πραγματοποιήθηκε μέσω πρόσθετου εργαλείου, στο πρόγραμμα, παράγοντας αρχεία μορφής Stereolithography (STL) και συνοδευτικά

αρχεία για τη ρομποτική προσομοίωση. Ακολούθησε στάδιο επαλήθευσης για τη διασφάλιση της γεωμετρικής ακρίβειας και της συνέπειας των κινηματικών χαρακτηριστικών. Τυχόν αποκλίσεις διορθώθηκαν άμεσα ώστε το ψηφιακό μοντέλο να ανταποκρίνεται πλήρως στις απαιτήσεις της προσομοίωσης, διασφαλίζοντας την αξιοπιστία της μετέπειτα αξιολόγησης του συστήματος σε εικονικούς χώρους.

3.1 Επιλογή Επιμέρους Υποσυστημάτων

Η συγκρότηση ενός εναέριου ρομποτικού συστήματος προϋποθέτει τη σχολαστική αξιολόγηση και τον συνδυασμό επιμέρους υλικών υποσυστημάτων, η συμβολή των οποίων είναι καθοριστική για τη λειτουργική ευστάθεια και την επιχειρησιακή απόδοση. Στην παρούσα μελέτη, χρησιμοποιήθηκαν εικονικά εξαρτήματα αντί φυσικών, προκειμένου να διαμορφωθεί μια ολοκληρωμένη ψηφιακή αναπαράσταση. Τα επιλεγμένα στοιχεία αντιστοιχούν σε πραγματικά προϊόντα που διατίθενται στο εμπόριο, διασφαλίζοντας έτσι την ακριβή αποτύπωση του οχήματος και τη δυνατότητα μετάβασης από το στάδιο της προσομοίωσης στη φυσική παραγωγή με τις ελάχιστες δυνατές τροποποιήσεις.

Τα τρισδιάστατα μοντέλα που ενσωματώθηκαν προέρχονται από την επίσημη βιβλιοθήκη της εταιρείας Holybro [21], η οποία ειδικεύεται στον σχεδιασμό εξαρτημάτων για ρομποτικά συστήματα, καθώς και από την ανοικτή πλατφόρμα διαμοιρασμού αρχείων GrabCAD [22]. Τα αρχεία αντλήθηκαν σε τυποποιημένες μορφές, όπως STP και STEP, οι οποίες κατά την εισαγωγή τους στο SolidWorks μετατράπηκαν αυτόματα σε μορφές SLDPART ή SLDASM. Η διαδικασία αυτή επιτρέπει την άμεση επεξεργασία, την παραμετροποίηση και την οργανική ενσωμάτωση των έτοιμων εξαρτημάτων στο συνολικό μοντέλο, χωρίς την ανάγκη περίπλοκων αναδιαμορφώσεων.

3.1.1 Σκελετός Πλαισίου (Airframe)

Ο σκελετός αποτελεί το θεμελιώδες δομικό στοιχείο του οχήματος, καθώς πάνω σε αυτόν ενσωματώνονται όλα τα υποσυστήματα, ενώ παράλληλα καθορίζεται η γεωμετρία, η μηχανική ανθεκτικότητα και η πτητική ευστάθεια. Η χρήση ανθρακονημάτων (carbon fiber) ως κύριο υλικό κατασκευής κρίνεται απαραίτητη, προκειμένου να διασφαλιστεί η βέλτιστη αναλογία αντοχής προς βάρος και η

αποτελεσματική απόσβεση των κραδασμών. Η επιλογή του κατάλληλου πλαισίου επηρεάζει άμεσα τη συνολική μάζα, τη φέρουσα ικανότητα και τη δυναμική συμπεριφορά του συστήματος κατά την πτήση.

Στο πλαίσιο της εργασίας, επιλέχθηκε το πλαίσιο X650 της εταιρείας Holybro. Κατά τη διαδικασία αξιολόγησης εξετάστηκε αρχικά η εναλλακτική λύση του πλαισίου X500, ωστόσο το X650 προκρίθηκε λόγω των αυξημένων διαστάσεων του και του μεγαλύτερου ωφέλιμου χώρου που διαθέτει για την εγκατάσταση πρόσθετου εξοπλισμού. Η δομή του υποστηρίζει την ενσωμάτωση βαρύτερων συσσωρευτών και ισχυρότερων κινητήρων, στοιχείο που ενισχύει την επιχειρησιακή αυτονομία και την ευελιξία του συστήματος.

Αντιθέτως, το μικρότερο πλαίσιο X500, αν και προσφέρει αυξημένη ευελιξία λόγω χαμηλότερου βάρους, παρουσιάζει περιορισμούς ως προς το μέγιστο ωφέλιμο φορτίο και τη χωροθέτηση των αισθητήρων. Η υιοθέτηση του X650 εξασφαλίζει την απαραίτητη ισορροπία μεταξύ δομικής αντοχής και επεκτασιμότητας, καθιστώντας το ιδανικό για την αξιόπιστη εκτέλεση των πειραματικών διεργασιών σε πραγματικές και εικονικές συνθήκες.

Feature	Detail
Wheelbase	650 mm
Motor Mount Pattern	16 – 25 mm
Upper & Battery Plate	120 × 128 mm
Body	160 × 160 × 2mm
Flight Time	30 min (Hover with 6S 10000 mAh Battery, no additional payload)
Full Kit Weight	2000 g (without battery)

Πίνακας 2: Τεχνικά χαρακτηριστικά πλαισίου X650 [23]



Εικόνα 3: Τρισδιάστατη (3D) απεικόνιση του πλαισίου Holybro X650

3.1.2 Μπαταρία και Σύστημα Διανομής Ισχύος (Battery & Power Distribution)

Η παροχή ενέργειας αποτελεί καθοριστικό παράγοντα για τη λειτουργία κάθε εναέριου συστήματος, με τις μπαταρίες λιθίου πολυμερούς (lithium polymer, LiPo) να αποτελούν την επικρατούσα επιλογή λόγω της υψηλής ενεργειακής τους πυκνότητας. Τα στοιχεία αυτά διαθέτουν την ικανότητα παροχής μεγάλων ρευμάτων εκφόρτισης, καλύπτοντας επαρκώς τις αυξημένες απαιτήσεις των κινητήρων και των ηλεκτρονικών υποσυστημάτων. Για τη διασφάλιση της σταθερής τροφοδοσίας, χρησιμοποιείται ένα σύστημα διανομής ισχύος (power distribution board, PDB), το οποίο επιτρέπει την ασφαλή και οργανωμένη μεταφορά της ηλεκτρικής ενέργειας από τον συσσωρευτή προς όλα τα επιμέρους τμήματα του οχήματος.

Στην παρούσα μελέτη, επιλέχθηκε ο συσσωρευτής Tattu 6S 9000 mAh 25 C, ο οποίος αποδίδει ονομαστική ενέργεια 200 Wh ($22.2\text{ V} \times 9\text{ Ah}$). Με την ενσωμάτωση ενός ορίου ασφαλείας της τάξης του 20% για τη διαδικασία αυτόματης επιστροφής στη βάση, η πραγματική ωφέλιμη ενέργεια διαμορφώνεται περίπου στα 160 Wh. Η μέγιστη συνεχής παροχή ρεύματος του συγκεκριμένου μοντέλου ανέρχεται κοντά στα 225 A, τιμή που κρίνεται υπερ-επαρκής για την υποστήριξη του πλαισίου X650 και του συνόλου των αισθητήρων του.

Με βάση την ωφέλιμη ενέργεια και υπό συνθήκες ήρεμης πτήσης με εκτιμώμενη μέση κατανάλωση 400-500 W, η αναμενόμενη επιχειρησιακή αυτονομία του οχήματος κυμαίνεται μεταξύ 19 και 24 λεπτών. Επισημαίνεται ότι η διάρκεια αυτή είναι άμεσα εξαρτώμενη από τις ατμοσφαιρικές συνθήκες. Για παράδειγμα, η

παρουσία ανέμου αυξάνει τις ενεργειακές απαιτήσεις για τη διατήρηση της ευστάθειας, οδηγώντας σε ταχύτερη εξάντληση των αποθεμάτων ισχύος.

Feature	Detail
Capacity	9000 mAh
Voltage	22.2 V (6S, 6 cells)
Discharge Rate	25 C
Weight	1173 g
Size	208 × 73 × 37 mm

Πίνακας 3: Τεχνικά χαρακτηριστικά μπαταρίας [24]



Εικόνα 4: Τρισδιάστατη (3D) απεικόνιση της μπαταρίας λιθίου πολυμερούς (LiPo) και του συστήματος διανομής ισχύος

3.1.3 Ηλεκτρονικός Ελεγκτής Ταχύτητας (Electronic Speed Controller, ESC)

Το σύστημα πρόωσης του MEA βασίζεται στη λειτουργία του ηλεκτρονικού ελεγκτή ταχύτητας, ο οποίος αποτελεί τον κρίσιμο ενδιάμεσο κρίκο για τη διαχείριση της ηλεκτρικής ισχύος μεταξύ του συσσωρευτή και των κινητήρων. Η συγκεκριμένη μονάδα μετατρέπει τις εντολές που λαμβάνει από τον ελεγκτή πτήσης σε ακριβείς παλμούς τάσης, ρυθμίζοντας με υψηλή συχνότητα την ταχύτητα περιστροφής κάθε κινητήρα. Αυτός ο μηχανισμός διασφαλίζει τη διατήρηση σταθερής ώσης και την άμεση απόκριση του συστήματος στις εντολές καθοδήγησης. Παράλληλα, οι σύγχρονες υλοποιήσεις ενσωματώνουν πρωτόκολλα προστασίας έναντι θερμικής καταπόνησης, υπερφόρτωσης και εσφαλμένης πολικότητας, ενισχύοντας τη δομική αξιοπιστία και τη μακροζωία του συστήματος.

Για το υπό μελέτη όχημα επιλέχθηκε η μονάδα Tekko32 F4, η οποία αξιοποιεί το υλικολογισμικό AM32. Η επιλογή αυτή υπαγορεύτηκε από τις ενεργειακές απαιτήσεις του πλαισίου X650 και την πλήρη συμβατότητα με τον ελεγκτή πτήσης

Pixhawk 6X. Η αρχιτεκτονική τεσσάρων σε ένα (4-in-1) επιτρέπει τον συγκεντρωτικό έλεγχο όλων των κινητήρων από μία ενιαία πλακέτα, γεγονός που συμβάλλει στη μείωση του συνολικού βάρους και στην απλοποίηση της καλωδίωσης. Σε συνδυασμό με την επιλεγμένη μπαταρία, ο ελεγκτής προσφέρει την απαραίτητη ισορροπία μεταξύ της μέγιστης συνεχούς παροχής ρεύματος, η οποία μπορεί να ανέλθει έως τα 225 A συνολικά, και της ενεργειακής αποδοτικότητας. Η συγκεκριμένη διαμόρφωση εγγυάται την αξιόπιστη λειτουργία του συστήματος πρόωσης ακόμη και σε ιδιαίτερα απαιτητικά πτητικά σενάρια.

Feature	Detail
Provides Continuous Current	50 A $\times$ 4
Supports	4-6S battery
Dimensions	48 $\times$ 37 $\times$ 6 mm
Weight	13.8 g

Πίνακας 4: Τεχνικά χαρακτηριστικά ηλεκτρονικού ελεγκτή ταχύτητας [25]



Εικόνα 5: Τρισδιάστατη (3D) απεικόνιση του ηλεκτρονικού ελεγκτή ταχύτητας

3.1.4 Ηλεκτρικός Κινητήρας χωρίς Ψήκτρα (Brushless Motors)

Οι ηλεκτρικοί κινητήρες χωρίς ψήκτρες (brushless) αποτελούν την κύρια πηγή παραγωγής ώσης στα πολυκόπτερα, μετατρέποντας την ηλεκτρική ενέργεια σε μηχανικό έργο περιστροφής που μεταδίδεται στους έλικες. Σε σύγκριση με τους συμβατικούς κινητήρες, οι διατάξεις αυτές διακρίνονται για τον υψηλό βαθμό απόδοσης, τις ελάχιστες τριβές και την εκτεταμένη διάρκεια ζωής τους. Η ταχεία απόκρισή τους στις μεταβολές της τάσης επιτρέπει την ακαριαία ρύθμιση της ισχύος, διασφαλίζοντας την πτητική ευστάθεια ακόμη και υπό αντίξοες αεροδυναμικές συνθήκες.

Για τις ανάγκες του συστήματος επιλέχθηκαν κινητήρες τύπου 4014-330 KV, οι οποίοι προσφέρουν την απαιτούμενη ισχύ για το πλαίσιο X650 διατηρώντας παράλληλα χαμηλή ενεργειακή κατανάλωση. Η συγκεκριμένη περιγραφή προσδιορίζει τις γεωμετρικές διαστάσεις του στάτορα και τη σταθερά ταχύτητας (velocity constant). Η διάταξη περιλαμβάνει μονάδες με δεξιόστροφη (Clockwise, CW) και αριστερόστροφη (Counter-Clockwise, CCW) φορά περιστροφής, στοιχείο απαραίτητο για την εξισορρόπηση των ροπών στρέψης και τον έλεγχο της εκτροπής του οχήματος.

Οι εν λόγω κινητήρες χαρακτηρίζονται από ευνοϊκή αναλογία ώσης προς βάρος και την ικανότητα περιστροφής ελίκων διαμέτρου 15 ιντσών σε σχετικά χαμηλές στροφές. Η παραγωγή υψηλής ροπής στρέψης (torque) σε αυτό το καθεστώς λειτουργίας περιορίζει τη θερμική καταπόνηση και τον παραγόμενο θόρυβο, ενισχύοντας ταυτόχρονα την ενεργειακή οικονομία. Η επιλογή αυτή καθιστά την πλατφόρμα ικανή για παρατεταμένες πτήσεις και ακριβή έλεγχο, υποστηρίζοντας αποτελεσματικά τις απαιτήσεις της αυτόνομης πλοήγησης και των προηγμένων εφαρμογών ρομποτικής.

Feature	Detail
KV	330
Stator Diameter	40 mm
Stator Length	14 mm
Shaft Diameter	4 mm
Max Continuous Current (A)	25 A
Max Continuous Power (W)	750 W
Motor Dimensions (Dia.*Len)	Φ 44.8 × 34.5 mm
Weight	149 g

Πίνακας 5: Τεχνικά χαρακτηριστικά κινητήρα χωρίς Ψήκτρα [26]



Εικόνα 6: Τρισδιάστατη (3D) απεικόνιση του ηλεκτρικού κινητήρα χωρίς ψήκτρα

3.1.5 Σύστημα Πλοήγησης και Ελέγχου (Navigation System & Flight Control)

Ο ελεγκτής πτήσης αποτελεί το κεντρικό υποσύστημα πλοήγησης και ελέγχου, εκτελώντας λειτουργίες αυτόματου πιλότου για το αερόχημα. Το σύστημα επεξεργάζεται δεδομένα από πολλαπλές μονάδες αδρανειακής μέτρησης, γυροσκόπια και επιταχυνσιόμετρα, ενώ ενσωματώνει αλγορίθμους πρόβλεψης που επιτρέπουν την ασφαλή λειτουργία ακόμη και υπό την επίδραση έντονων κραδασμών ή αεροδυναμικών διαταραχών.

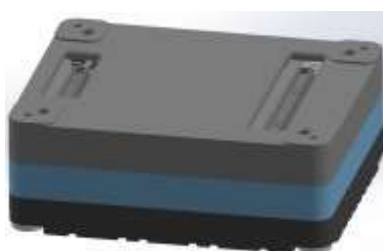
Η επιλογή του Pixhawk 6X κρίθηκε ιδανική, ιδιαίτερα λόγω της δυνατότητας συνεργασίας με την υπολογιστική μονάδα Jetson Nano (Jetson Nano). Επιπλέον, η συγκεκριμένη μονάδα υποστηρίζει τη σύνδεση με προηγμένα συστήματα εντοπισμού θέσης (Global Positioning System, GPS), καθώς και αισθητήρες οπτικής ροής (optical flow), διευρύνοντας τις δυνατότητες του οχήματος για ακριβή αυτόνομη κίνηση.

Ο συνδυασμός αυτών των δύο στοιχείων δημιουργεί μια ολοκληρωμένη αρχιτεκτονική που γεφυρώνει την ασφαλή πτήση με τις δυνατότητες τεχνητής νοημοσύνης. Η συγκεκριμένη συνεργασία μετατρέπει το όχημα σε μια ευέλικτη πλατφόρμα πειραματισμού για την ανάπτυξη εφαρμογών στην ανίχνευση αντικειμένων και τη συνεργατική ρομποτική (collaborative robotics). Η επιλογή αυτή εξασφαλίζει τη βέλτιστη ισορροπία μεταξύ σταθερότητας, επεκτασιμότητας και υπολογιστικής ισχύος, καλύπτοντας πλήρως τις ερευνητικές απαιτήσεις της παρούσας εργασίας.

Feature	Detail
FMU Processor	STM32H753

IO Processor	STM32F103
On-board Sensors	Accel/Gyro, Barometer, Mag
Max Input voltage	6V
PWM Signal Voltage	3.3V (Adjustable to 5V)
Dimensions	38.8 × 31.8 × 16.8 mm
Weight	31.3 g

Πίνακας 6: Τεχνικά χαρακτηριστικά του συστήματος πλοήγησης [27]



Εικόνα 7: Τρισδιάστατη (3D) απεικόνιση του ελεγκτή πτήσης

3.1.6 Μονάδα Τηλεμετρίας και Δορυφορικού Εντοπισμού (Telematic System & GNSS System)

Η απρόσκοπτη λειτουργία του συστήματος προϋποθέτει την ύπαρξη αξιόπιστων διαύλων επικοινωνίας με τον επίγειο σταθμό ελέγχου και τη δυνατότητα ακριβούς προσδιορισμού των γεωγραφικών συντεταγμένων. Η μονάδα τηλεμετρίας (telemetry) αναλαμβάνει τη συνεχή αναμετάδοση κρίσιμων παραμέτρων, όπως το βαρομετρικό ύψος, η ταχύτητα κίνησης και το επίπεδο φόρτισης του συσσωρευτή, επιτρέποντας την εποπτεία της αποστολής σε πραγματικό χρόνο.

Για τις ανάγκες της επικοινωνίας επιλέχθηκε το σύστημα SiK Telemetry Radio V3, μια ελαφριά και ενεργειακά αποδοτική διάταξη που εξασφαλίζει σταθερή ζεύξη σε μεγάλες αποστάσεις. Η συγκεκριμένη μονάδα παρουσιάζει πλήρη συμβατότητα με τον ελεγκτή Pixhawk, διευκολύνοντας την ενσωμάτωση στο υφιστάμενο δίκτυο και ενισχύοντας την επιχειρησιακή αξιοπιστία. Παράλληλα, ο δορυφορικός εντοπισμός υλοποιείται μέσω του δέκτη PX4 Neo-M8N, ο οποίος υποστηρίζει πολλαπλά συστήματα, όπως το GLONASS και το Galileo. Η δυνατότητα αυτή περιορίζει σημαντικά την πιθανότητα απώλειας σήματος και βελτιώνει την ακρίβεια θέσης σε εξωτερικά περιβάλλοντα.

Ο συνδυασμός των παραπάνω τεχνολογιών διασφαλίζει τον ακριβή γεωεντοπισμό και τον υπολογισμό της πορείας. Η γρήγορη απόκριση του δέκτη και το περιορισμένο βάρος του συμβάλλουν στη διατήρηση της πτητικής αυτονομίας, ενώ παράλληλα καθιστούν εφικτή την ασφαλή επιστροφή του οχήματος στο σημείο απογείωσης σε περίπτωση ανάγκης.

Feature	Detail
Communication	2-way full-duplex
Protocol Framing	MAVLink
Supply Voltage	5 V
Transmit Current	100 mA at 20 dBm
Receive Current	25 mA
Dimensions	28 × 53 × 10.7 mm (without antenna)
Weight	23.5g (without antenna)

Πίνακας 7: Τεχνικά χαρακτηριστικά μονάδα τηλεμετρίας [28]

Feature	Detail
Module	Ublox Neo-M8N
Current consumption	less than 150 mA
Input voltage	5 V
Current consumption	less than 150 mA
Dimensions	Φ 50 mm
Weight	32 g

Πίνακας 8: Τεχνικά χαρακτηριστικά δορυφορικής μονάδας GPS [29]



Εικόνα 8: Τρισδιάστατη (3D) απεικόνιση μονάδας τηλεμετρίας και δορυφορικής

3.1.7 Συνοδευτικός Υπολογιστής (Companion Computer)

Τα μη επανδρωμένα συστήματα που απαιτούν προηγμένη επεξεργασία δεδομένων, όπως η υπολογιστική όραση και η δισδιάστατη ή τρισδιάστατη χαρτογράφηση, επιβάλλουν τη χρήση επιπλέον υπολογιστικής ισχύος πέρα από τις δυνατότητες του βασικού ελεγκτή πτήσης.

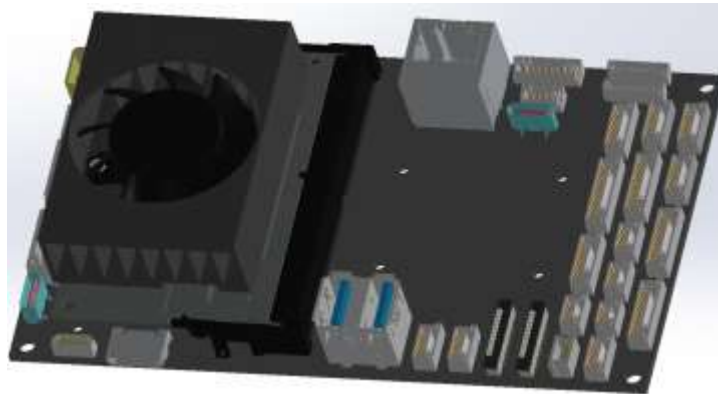
Για την κάλυψη αυτών των αναγκών, στο όχημα ενσωματώθηκε η πλακέτα διασύνδεσης Pixhawk Jetson Baseboard, η οποία επιτρέπει τη συνεργασία του Pixhawk 6X με την υπολογιστική μονάδα NVIDIA Jetson Nano. Το Jetson Nano εξασφαλίζει υψηλή υπολογιστική απόδοση με περιορισμένη ενεργειακή κατανάλωση, υποστηρίζοντας εξειδικευμένες βιβλιοθήκες όπως το Open Source Computer Vision Library (OpenCV) [30], για εφαρμογές τεχνητής νοημοσύνης και ανάλυσης εικόνας. Στην αρχιτεκτονική αυτή, ο Pixhawk 6X παραμένει υπεύθυνος για τις κρίσιμες λειτουργίες πτήσης, τη σταθεροποίηση και τη διαχείριση των κινητήρων. Αντιθέτως, ο συνοδός υπολογιστής αναλαμβάνει την εκτέλεση αλγορίθμων υψηλού επιπέδου, συμπεριλαμβανομένης της αναγνώρισης εμποδίων και την ενσωμάτωση του ROS 2.

Η ανταλλαγή δεδομένων μεταξύ των δύο μονάδων πραγματοποιείται μέσω του πρωτοκόλλου MAVLink [31], το οποίο διασφαλίζει την ασφαλή μετατροπή των αποτελεσμάτων της επεξεργασίας σε εντολές κίνησης. Λόγω του χαμηλού βάρους και της αποδοτικής διαχείρισης ισχύος, ο συνδυασμός αυτός προσδίδει στο αερόχημα ισχυρές υπολογιστικές ικανότητες, χωρίς να επηρεάζει αρνητικά την πτητική αυτονομία. Με τον τρόπο αυτό, επιτυγχάνεται η βέλτιστη ισορροπία μεταξύ της σταθερότητας του συστήματος και των αυξημένων απαιτήσεων που επιβάλλουν οι σύγχρονες ρομποτικές εφαρμογές.

Feature	Detail
Baseboard	Pixhawk Nvidia Jetson
RAM	4 GB
Number of Connectors	25
Output Voltage	6.0 V / 7.2 V / 8.0 V /9.2 V (recommend

	7.2 V if supplying power to Jetson Board)
Output Current	12 A (Continuous), 24 A (Burst)
Dimensions	126 × 80 × 38 mm
Weight	175 g

Πίνακας 9: Τεχνικά χαρακτηριστικά Pixhawk Jetson Baseboard [32]



Εικόνα 9: Τρισδιάστατη (3D) απεικόνιση συνοδευτικού υπολογιστή

3.1.8 Κάμερα Βάθους (Depth Camera)

Οι κάμερες βάθους αποτελούν μια εξειδικευμένη κατηγορία αισθητήρων όρασης που επεκτείνουν τις δυνατότητες της συμβατικής απεικόνισης, καθώς ενσωματώνουν πληροφορία απόστασης για κάθε εικονοστοιχείο παράλληλα με το χρωματικό πρότυπο (Red-Green-Blue, RGB). Η συγκεκριμένη τεχνολογία επιτρέπει την πιστή αποτύπωση της γεωμετρίας του χώρου, προσφέροντας στα εναέρια ρομποτικά συστήματα τη δυνατότητα τρισδιάστατης αντίληψης, ακριβούς εκτίμησης αποστάσεων και αποτελεσματικής αποφυγής εμποδίων.

Στην παρούσα εργασία επιλέχθηκε η κάμερα Intel RealSense D455 [33], η οποία διακρίνεται για την ενισχυμένη γεωμετρική ακρίβεια σε μεσαίες και μεγάλες αποστάσεις. Η βελτίωση αυτή οφείλεται στη διευρυμένη απόσταση μεταξύ των στερεοσκοπικών φακών της, επιτρέποντας την ανίχνευση αντικειμένων σε βάθος που υπερβαίνει τα δέκα μέτρα. Παράλληλα, ο αισθητήρας παρέχει ευρύ οπτικό πεδίο (field of view) και καταγραφή χρώματος υψηλής ευκρίνειας, παραμένοντας ταυτόχρονα συμπαγής και ελαφρύς.

Η ενσωμάτωση του συγκεκριμένου εξοπλισμού στο πλαίσιο του οχήματος βασίστηκε στον συνδυασμό της χαμηλής κατανάλωσης ισχύος και της πλήρους συμβατότητας με τα περιβάλλοντα ROS 2 και Gazebo. Τα κριτήρια επιλογής του συγκεκριμένου αισθητήρα συνοψίζονται στα ακόλουθα:

- Ικανότητα παραγωγής χάρτη βάθους (depth map) με υψηλή πιστότητα σε αποστάσεις που ξεπερνούν τα δέκα μέτρα.
- Ταυτόχρονη παροχή οπτικής πληροφορίας και δεδομένων απόστασης σε πραγματικό χρόνο (real-time).
- Διευρυμένο οπτικό πεδίο, κατάλληλο για την πλοήγηση σε σύνθετα και πυκνά περιβάλλοντα.
- Περιορισμένη μάζα και συμπαγείς διαστάσεις, στοιχεία κρίσιμα για τη διατήρηση της πτητικής ευστάθειας.

Με τα χαρακτηριστικά αυτά, η κάμερα βάθους καθίσταται ένα ζωτικής σημασίας υποσύστημα, ενισχύοντας την αυτόνομη λήψη αποφάσεων και την ασφαλή λειτουργία του αεροχήματος σε δυναμικά μεταβαλλόμενα περιβάλλοντα.

Feature	Detail
Depth Technology	Stereo
Operating Range (Min-Max)	0.6-6 m
Depth Resolution and FPS	1280 × 720 px up to 90FPS
Depth Field of View	86 × 57 px
Power Supply	5 V
Current	700 mA
Dimensions	124 × 29 mm × 26 mm
Weight	116 g

Πίνακας 10: Τεχνικά χαρακτηριστικά κάμερα βάθους [33]



Εικόνα 10: Τρισδιάστατη (3D) απεικόνιση της κάμερας βάθους

3.1.9 Αισθητήρας Σάρωσης (LiDAR Sensor)

Ο αισθητήρας σάρωσης LiDAR αποτελεί μια προηγμένη τεχνολογία τηλεπισκόπησης (remote sensing) που βασίζεται στην εκπομπή παλμών λέιζερ για τον προσδιορισμό των αποστάσεων από τα περιβάλλοντα αντικείμενα. Η διεργασία αυτή επιτρέπει τη δημιουργία ενός λεπτομερούς νέφους σημείων (point cloud), αποδίδοντας με υψηλή ακρίβεια τη γεωμετρία του χώρου. Στο πεδίο της εναέριας ρομποτικής, η συγκεκριμένη τεχνολογία είναι θεμελιώδης για την αναγνώριση εμποδίων, τη χαρτογράφηση και την υποστήριξη αλγορίθμων SLAM, επιτρέποντας στο όχημα να αντιλαμβάνεται τη θέση του σε άγνωστα περιβάλλοντα.

Για την παρούσα εργασία επιλέχθηκε ο δισδιάστατος σαρωτής RPLiDAR A2M8, ο οποίος προσφέρει πλήρη κάλυψη 360 μοιρών. Ο συγκεκριμένος αισθητήρας αποτελεί μια ισορροπημένη λύση για πτητικές πλατφόρμες, καθώς συνδυάζει τα εξής χαρακτηριστικά:

- Περιορισμένη μάζα και συμπαγής σχεδιασμός, που δεν επιβαρύνουν το κέντρο βάρους του οχήματος.
- Χαμηλή κατανάλωση ηλεκτρικής ισχύος, συμβάλλοντας στη διατήρηση της πτητικής αυτονομίας.
- Δυνατότητα σάρωσης πλήρους κύκλου για την ολοκληρωμένη αντίληψη του περιβάλλοντος χώρου.
- Πλήρης συμβατότητα με το ROS 2, διευκολύνοντας την ανάπτυξη και την προσομοίωση αλγορίθμων πλοήγησης.

Η ενσωμάτωση του RPLiDAR A2M8 εξασφαλίζει τη σταθερή ροή δεδομένων απόστασης, καθιστώντας τον αισθητήρα ένα αξιόπιστο εργαλείο για την αυτόνομη κίνηση και τη χαρτογράφηση σε πραγματικές συνθήκες πτήσης.

Feature	Detail
Measuring Range	0.2-12m
Sampling Frequency	8000
Angular Resolution	0.45°
Power Supply	5 V
Current	400-500 mA
Dimensions	$\Phi 76 \times 41$ mm
Weight	190 g

Πίνακας 11: Τεχνικά χαρακτηριστικά αισθητήρα σάρωσης [34]



Εικόνα 11: Τρισδιάστατη (3D) απεικόνιση του αισθητήρα σάρωσης για δισδιάστατη (2D) σάρωση

3.1.10 Έλικας (Propeller)

Οι έλικες συνιστούν ζωτικό τμήμα του συστήματος πρόωσης, καθώς μετατρέπουν τη μηχανική ροπή των κινητήρων σε αεροδυναμική ώση (thrust), επηρεάζοντας άμεσα τη σταθερότητα και την ενεργειακή αποδοτικότητα. Παράμετροι όπως η διάμετρος, την κλίση από τα πτερύγια και το υλικό κατασκευής καθορίζουν τη δυναμική συμπεριφορά του οχήματος, την κατανάλωση ισχύος και την ικανότητα εκτέλεσης ελιγμών.

Για τον σκελετό πλαίσιο χρησιμοποιήθηκαν οι πτυσσόμενοι έλικες TAROT 1555. Η κατασκευή τους από ανθρακονήματα προσφέρει εξαιρετική ακαμψία και περιορισμένη μάζα, εξασφαλίζοντας υψηλή αντοχή στις μηχανικές καταπονήσεις. Η διάμετρος των 15 ιντσών σε συνδυασμό με το αεροδυναμικό προφίλ επιτρέπει την παραγωγή μεγάλης ώσης με μειωμένες ενεργειακές απαιτήσεις.

Ο πτυσσόμενος μηχανισμός ενισχύει την πρακτικότητα του συστήματος, διευκολύνοντας τη μεταφορά και προστατεύοντας τα πτερύγια από τυχόν φθορές

κατά την αποθήκευση. Επιπλέον, η ισορροπημένη περιστροφή τους συμβάλλει στον περιορισμό του θορύβου και των δονήσεων, παράγοντες που είναι κρίσιμοι για την ομαλή λειτουργία των αισθητήρων και τη συνολική αξιοπιστία της αυτόνομης πτήσης. Η επιλογή αυτή κρίνεται ως η πλέον αποδοτική για τις πειραματικές ανάγκες της παρούσας μελέτης, προσφέροντας ανθεκτικότητα και σταθερότητα σε ποικίλα επιχειρησιακά σενάρια.

Feature	Detail
Length	173 mm
Diameter	15 inches
Diameter of Center hole	3 mm
Assembly Thickness	3.95 mm
Weight	8 g

Πίνακας 12: Τεχνικά χαρακτηριστικά του έλικα [35]



Εικόνα 12: Τρισδιάστατη (3D) απεικόνιση του έλικα CW (πάνω) και CCW (κάτω)

3.1.11 Σύστημα Προσγείωσης (Landing Gear)

Το σύστημα προσγείωσης αποτελεί τον δομικό σύνδεσμο μεταξύ του MEA και του εδάφους, επιτελώντας καθοριστικό ρόλο στην ασφάλεια κατά τις φάσεις της απογείωσης και της προσέγγισης. Η κύρια λειτουργία του εστιάζεται στην προστασία των ευαίσθητων ηλεκτρονικών υποσυστημάτων, όπως οι κάμερες και οι μονάδες αντίληψης, αποτρέποντας την άμεση επαφή τους με την επιφάνεια του εδάφους και αποσβένοντας τις δυνάμεις κρούσης.

Για το μοντέλο επιλέχθηκε σύστημα από ανθρακονήματα, το οποίο εξασφαλίζει την απαραίτητη μηχανική στιβαρότητα χωρίς να επιβαρύνει σημαντικά τη συνολική μάζα. Η συγκεκριμένη διάταξη προσφέρει αυξημένη απόσταση από το έδαφος, στοιχείο που είναι ζωτικής σημασίας για την ασφαλή τοποθέτηση του εξοπλισμού

στο κάτω μέρος του πλαισίου. Η στιβαρότητα της κατασκευής επιτρέπει τη σταθερή έδραση του οχήματος, προστατεύοντας τα υποσυστήματα από κραδασμούς και πιθανές αστοχίες κατά τη διάρκεια απότομων προσγειώσεων. Η υιοθέτηση αυτής της λύσης συμβάλλει στη μακροχρόνια αξιοπιστία της πλατφόρμας, διασφαλίζοντας την ακεραιότητα του συστήματος σε ποικίλα μορφολογικά ανάγλυφα.

Feature	Detail
Material	Carbon Fiber with plastic connectors
Height	300 mm
Length	320 mm
Weight	70 g

Πίνακας 13: Τεχνικά χαρακτηριστικά συστήματος προσγείωσης [23]



Εικόνα 13: Τρισδιάστατη (3D) απεικόνιση του συστήματος προσγείωσης

3.2 Σχεδίαση Εξαρτημάτων

Η τρισδιάστατη σχεδίαση ενός εναέριου οχήματος μπορεί να πραγματοποιηθεί μέσω διαφόρων λογισμικών μοντελοποίησης, όπως τα Fusion 360, Inventor και FreeCAD. Στο πλαίσιο της παρούσας εργασίας επιλέχθηκε το SolidWorks, καθώς αποτελεί ένα προηγμένο επαγγελματικό εργαλείο σχεδίασης για περιβάλλον Windows. Το συγκεκριμένο λογισμικό προσφέρει όλες τις απαραίτητες λειτουργίες για την επίτευξη υψηλής ακρίβειας κατά την ψηφιακή κατασκευή των εξαρτημάτων.

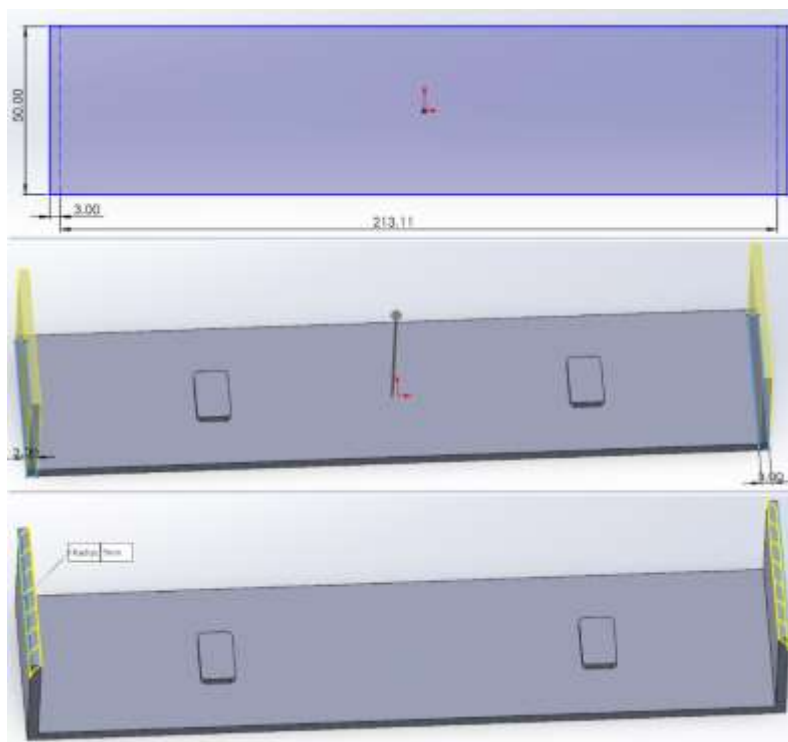
Η χρήση του κρίθηκε επιβεβλημένη τόσο για τη συνολική μοντελοποίηση όσο και για τον σχεδιασμό ειδικών μερών, όπως η βάση στήριξης του συσσωρευτή, τα οποία δεν εντοπίστηκαν στα διαθέσιμα ψηφιακά αποθετήρια που προαναφέρθηκαν. Με αυτόν τον τρόπο, αντιμετωπίστηκαν οι ελλείψεις στα έτοιμα ψηφιακά μοντέλα και δημιουργήθηκε ένα συνεκτικό σύνολο εξαρτημάτων, έτοιμο να ενσωματωθεί στην τελική δομή του οχήματος.

3.2.1 Βάση Σταθεροποίησης Μπαταρίας

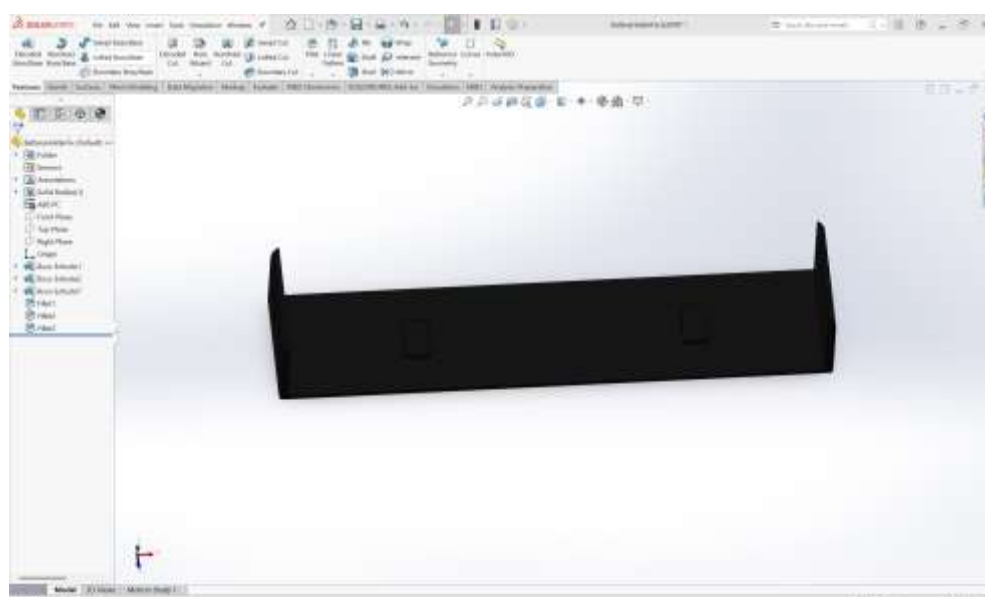
Η βάση της μπαταρίας που τοποθετήθηκε στο κάτω μέρος του πλαισίου X650 σχεδιάστηκε σε δύο στάδια. Στη συνέχεια παρουσιάζονται ορισμένα ενδεικτικά βήματα της διαδικασίας σχεδίασης, καθώς και το τελικό αποτέλεσμα.

Βήμα	Ενέργεια	Εκτέλεση
1	Δημιουργία Πλατφόρμας	Καρτέλα Sketch → Corner Rectangle → (50×213.11) → Extruded Boss/Base με πάχος 3 mm.
2	Δημιουργία δύο στηλών	Δεξί κλικ στην πλατφόρμα → Sketch → Corner Rectangle → (10×17) , τοποθέτηση σε απόσταση 110 mm πλάτος και 24.32 mm ύψος από το κέντρο της πλατφόρμας → Extruded Boss/Base με πάχος 2 mm.
3	Διαμόρφωση άκρων βάσης	Νέο Sketch στην πλατφόρμα → ορισμός άκρων με πλάτος 3 mm → Extruded Boss/Base με πάχος 35 mm.
4	Στρογγυλοποίηση (Fillet)	Καρτέλα Features → Fillet με ακτίνες 2 mm, 3 mm και 7 mm για τη στρογγυλοποίηση των άκρων.
5	Οπτική	Δεξί μενού Appearances → Plastic → Medium Gloss → Black Medium Gloss Plastic για τον τελικό χρωματισμό.

Πίνακας 14: Ενδεικτικά βήματα της διαδικασίας σχεδίασης



Εικόνα 14: Βήματα σχεδίασης βάσης μπαταρίας



Εικόνα 15: Τελικό αποτέλεσμα σχεδίασης βάσης μπαταρίας

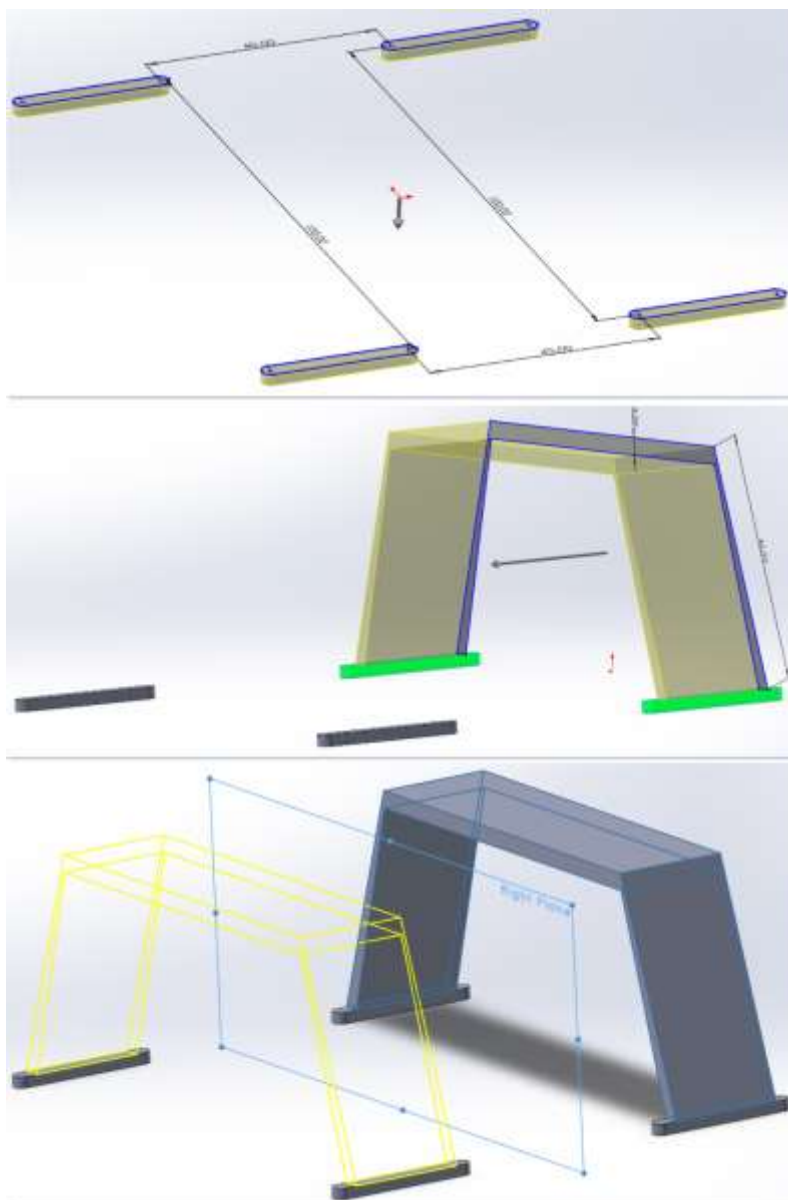
3.2.2 Ιμάντες Στερέωσης Μπαταρίας

Για την ασφαλή συγκράτηση της μπαταρίας κατά τη διάρκεια της πτήσης κρίθηκε απαραίτητη η χρήση δύο ειδικών ιμάντων στερέωσης. Οι ιμάντες σχεδιάστηκαν έτσι

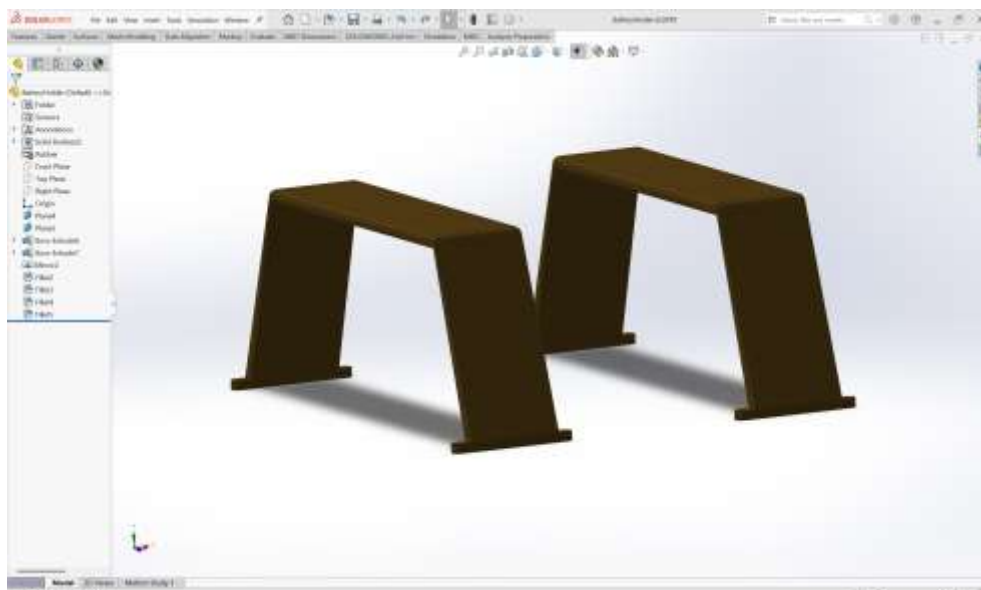
ώστε να εφαρμόζουν σφιχτά γύρω από το σώμα του συσσωρευτή, εξασφαλίζοντας την ακινητοποίησή του ακόμη και σε συνθήκες έντονων κραδασμών ή απότομων ελιγμών. Η χρήση τους θεωρείται κρίσιμη για την προστασία των ηλεκτρονικών υποσυστημάτων και τη διασφάλιση της επιχειρησιακής αξιοπιστίας του οχήματος. Ακολουθούν τα βήματα σχεδίασης:

Βήμα	Ενέργεια	Εκτέλεση
1	Σημεία σύνδεσης πλαισίου	Δεξί κλικ στο κάτω μέρος του πλαισίου → Sketch → επιλογή σημείων → Convert Entities → αντιγραφή/επικόλληση σε νέο Sketch.
2	Δημιουργία σημείων σύνδεσης μάντα	Extruded Boss/Base με πάχος 2 mm.
3	Καμπύλη μάντα	Extruded Boss/Base με πλάτος 20 mm, ύψος 42,93 mm και πάχος 3mm.
4	Καθρεφτισμός (Mirroring)	Features → Mirror → Mirror Face/Plane → Right Plane, επιλογή του πρώτου μάντα για αντιγραφή.
5	Στρογγυλοποίηση γωνιών και ακμών (Fillet)	Features → Fillet με ακτίνα 1.5 mm για τις γωνίες και ακμές. Η διαδικασία επαναλαμβάνεται με τιμές 1.5 mm και 0.4 mm.
6	Οπτική	Δεξί μενού Appearances → Plastic → Medium Gloss → RGB(105, 77, 26) για τον τελικό χρωματισμό.

Πίνακας 15: Βήματα σχεδίασης μάντων στερέωσης



Εικόνα 16: Βήματα σχεδίασης μάντα



Εικόνα 17: Τελικό αποτέλεσμα σχεδίασης ιμάντας

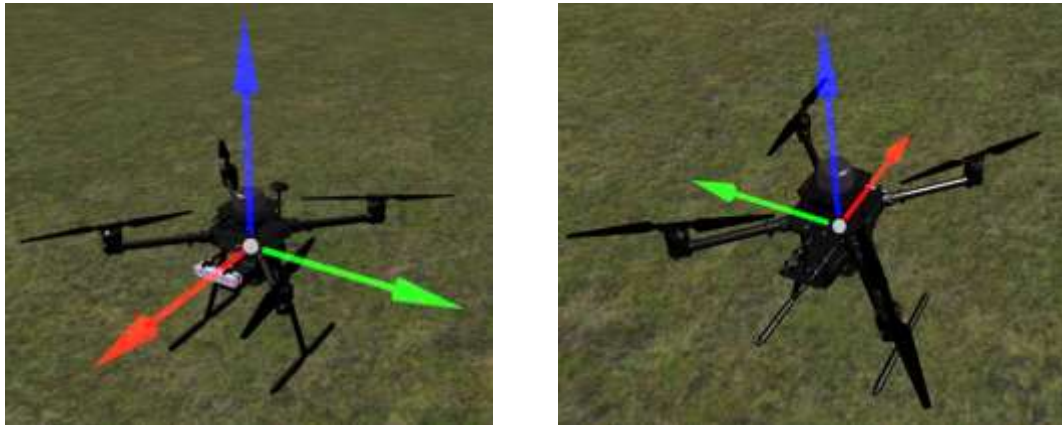
3.3 Τοποθέτηση και Συναρμολόγηση Εξαρτημάτων

Η χωροθέτηση των επιμέρους εξαρτημάτων στην ψηφιακή συναρμολόγηση πρέπει να υλοποιείται με τέτοιο τρόπο ώστε το σημείο αναφοράς (origin) να ταυτίζεται με τη θέση μηδέν του συστήματος συντεταγμένων. Το σημείο αυτό αποτελεί την τομή των τριών αξόνων X, Y και Z, δηλαδή τις συντεταγμένες (0,0,0). Οποιαδήποτε απόκλιση από αυτή τη σύμβαση μπορεί να προκαλέσει σφάλματα στους συνδέσμους (links) και τις αρθρώσεις (joints) κατά τη μεταφορά του μοντέλου στο περιβάλλον προσομοίωσης, καθιστώντας αδύνατη την πιστή αναπαράσταση της φυσικής συμπεριφοράς του οχήματος.

Ιδιαίτερη έμφαση δόθηκε στην τήρηση του δεξιόστροφου συστήματος συντεταγμένων (right-hand rule), το οποίο αποτελεί το πρότυπο στη ρομποτική και την αεροναυπηγική. Κατά τη διαδικασία της ψηφιακής σύνθεσης, η διάταξη ορίστηκε ως εξής:

- Άξονας X (κόκκινο): Αντιστοιχεί στην μπροστινή όψη (forward) του οχήματος.
- Άξονας Y (πράσινο): Αντιστοιχεί στην αριστερή πλευρά (left).
- Άξονας Z (μπλέ): Αντιστοιχεί στο πάνω μέρος (up) του οχήματος.

Η αυστηρή τήρηση αυτής της αρχής είναι θεμελιώδης, καθώς τα λογισμικά προσομοίωσης και οι αλγόριθμοι πλοήγησης βασίζονται σε αυτούς τους άξονες για τον υπολογισμό της κίνησης και του προσανατολισμού. Η λανθασμένη ευθυγράμμιση των αξόνων θα οδηγούσε σε εσφαλμένη ερμηνεία των δεδομένων από τον ελεγκτή πτήσης, καθιστώντας το MEA μη λειτουργικό στο εικονικό περιβάλλον.



Εικόνα 18: Απεικόνιση του δεξιόστροφου συστήματος συντεταγμένων (right-hand-rule)

Η ορθή τοποθέτηση των εξαρτημάτων στο πλαίσιο αποτέλεσε καθοριστικό παράγοντα για τη διατήρηση της μηχανικής ισορροπίας, της σταθερότητας πτήσης και της αξιόπιστης διασύνδεσης των επιμέρους στοιχείων. Η διαδικασία ακολούθησε μια ιεραρχική λογική, βασισμένη στα δομικά επίπεδα του σκελετού. Με γνώμονα τη βέλτιστη κατανομή βάρους, την ευκολία συντήρησης και τον περιορισμό των ηλεκτρομαγνητικών παρεμβολών που θα προέκυπταν σε μια φυσική κατασκευή, κάθε υποσύστημα χωροθετήθηκε σε συγκεκριμένη θέση στο αντίστοιχο επίπεδο.

Η συνολική κατασκευή, πλήρως συναρμολογημένη και με όλα τα επιμέρους υποσυστήματα τοποθετημένα στις αντίστοιχες θέσεις τους, παρουσιάζεται στο τέλος του κεφαλαίου. Μέσα από μια σειρά απεικονίσεων, αποδίδεται το ψηφιακό μοντέλο από διαφορετικές οπτικές γωνίες, αναδεικνύοντας τη δομική συνοχή και την ετοιμότητα του συστήματος για το στάδιο της προσομοίωσης.

3.3.1 Συναρμολόγηση Σκελετού Πλαισίου

Η επεξεργασία του μοντέλου στο SolidWorks περιορίστηκε στα εξής στάδια:

- Φόρτωση και Καθορισμός Υλικού: Ως υλικό επιλέχθηκε το ανθρακόνημα, διασφαλίζοντας ότι οι φυσικές ιδιότητες του μοντέλου προσομοιάζουν την πραγματική κατασκευή.
- Εφαρμογή Χρωματισμού: Ο χρωματισμός αποδόθηκε σύμφωνα με την πραγματική εμφάνιση του X650, ενισχύοντας τη ρεαλιστικότητα της οπτικής
- Αποθήκευση: Το τελικό αρχείο αποθηκεύτηκε σε μορφή SLDPRT.

Η ίδια διαδικασία ακολουθήθηκε και για τα υπόλοιπα εξαρτήματα, προκειμένου να διατηρηθεί οπτική και σχεδιαστική ομοιομορφία στο τελικό εικονικό μοντέλο.

Επιπλέον, για τη διευκόλυνση της τοποθέτησης των ελίκων και τη βελτίωση της ιεραρχικής οργάνωσης, οι κινητήρες απομονώθηκαν από το κύριο πλαίσιο και ορίστηκαν ως ανεξάρτητες οντότητες. Η κίνηση αυτή απλοποιεί σημαντικά τη διαχείριση και την εξαγωγή του μοντέλου στο ρομποτικό περιβάλλον, επιτρέποντας την αυτόνομη ανάθεση κινηματικών ιδιοτήτων σε κάθε περιστρεφόμενο μέρος.

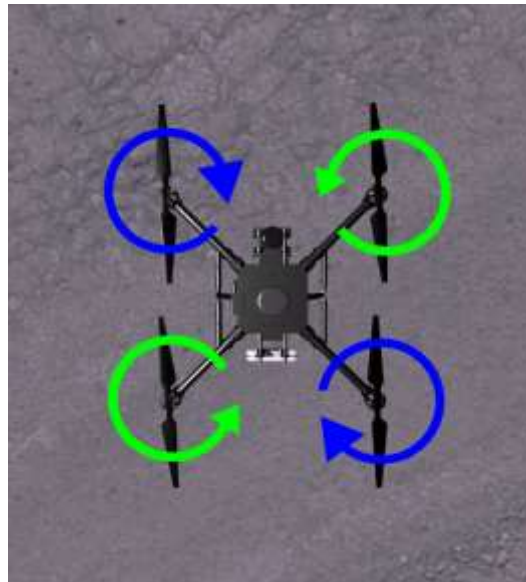


Εικόνα 19: Πλαίσιο X650

3.3.2 Τοποθέτηση Κινητήρων και Ελίκων στους Βραχίονες

Οι κινητήρες, μαζί με τους αντίστοιχους έλικες, τοποθετήθηκαν στα άκρα κάθε βραχίονα του οχήματος. Ιδιαίτερη προσοχή δόθηκε στην τήρηση της αρχής λειτουργίας ενός τετρακόπτερου, σύμφωνα με την οποία οι κινητήρες και οι έλικες

τύπου CW τοποθετούνται στο δεξί εμπρόσθιο και αριστερό οπίσθιο τμήμα του πλαισίου, ενώ οι κινητήρες και έλικες τύπου CCW τοποθετούνται στο αριστερό εμπρόσθιο και δεξί οπίσθιο τμήμα. Με τον τρόπο αυτό διασφαλίζεται η απαιτούμενη ισορροπία ροπής και η σταθερότητα κατά την πτήση.



Εικόνα 20: Περιστροφή Κινητήρων: CW (Μπλε) και CCW (Πράσινο)

Για παράδειγμα, η συναρμολόγηση και τοποθέτηση του κινητήρα CW ακολούθησε τα εξής βήματα:

Βήμα	Ενέργεια	Εκτέλεση
1	Δημιουργία συναρμολόγησης (assembly)	Από το μενού File → New → Επιλογή Assembly για νέα συναρμολόγησης.
2	Εισαγωγή εξαρτημάτων	Καρτέλα Assembly → Insert Components → Insert Component → Επιλογή κινητήρα και έλικα τύπου CW.
3	Τοποθέτηση και προσανατολισμός	Καρτέλα Assembly → Move Component → Επιλογή Move Component/Rotate Component έως ότου τα εξαρτήματα τοποθετηθούν και ευθυγραμμιστούν σωστά, όπως φαίνεται στην παρακάτω Εικόνα.

4	Συναρμολόγηση εξαρτημάτων	Καρτέλα Assembly → Mate → Επιλογή τύπου συναρμολόγησης και καθορίζονται τα αντίστοιχα σημεία των εξαρτημάτων για την ολοκλήρωση της σύνδεσης.
---	----------------------------------	---

Πίνακας 16: Βήματα συναρμολόγησης και τοποθέτησης του κινητήρα CW

Για την ακριβέστερη και πιο κατανοητή απεικόνιση της τοποθέτησης των εξαρτημάτων, το πρόγραμμα προσφέρει τη δυνατότητα δημιουργίας εκρηκτικής απεικόνισης (Exploded View). Στην προβολή αυτή, κάθε εξάρτημα παρουσιάζεται ελαφρώς απομακρυσμένο με διακεκομμένες γραμμές από τη θέση σύνδεσής του, επιτρέποντας την οπτική κατανόηση της διάταξης και της σειράς συναρμολόγησης.

Η διαδικασία που εφαρμόστηκε για την επίτευξη της απεικόνισης ήταν η εξής:

Βήμα	Ενέργεια	Εκτέλεση
1	Απομόνωση εξαρτημάτων	Επιλογή των τμημάτων και δεξί κλικ → Επιλογή Isolate (Απομόνωση) → Εμφανίζονται μόνο τα επιλεγμένα τμήματα.
2	Αποθήκευση απεικόνισης απομόνωσης	Διάλογος Isolate (εμφανίζεται αυτόματα μετά την απομόνωση) → Εισαγωγή όνομα απομόνωσης → Αποθήκευση μέσω δισκέτας.
3	Δημιουργία εκρηκτικής απεικόνισης	Καρτέλα Assembly → Exploded View → Εισαγωγή ονόματος στο Exploded Step Name πεδίο → Επιλογή εξαρτημάτων για εκρηκτική διατύπωση → Επιλογή και ρύθμιση Auto-space components για ορισμό απόστασης από κάθε εξάρτημα, ώστε να επιτευχθεί ομοιόμορφη κατανομή.
4	Επόμενη επιλογή εκρηκτικής απεικόνισης	Στη λίστα Explode Steps → Τράβηγμα γραμμής προς τα πάνω, για να ενεργοποιηθεί η επόμενη εκρηκτική απεικόνιση → Αυτό επαναλαμβάνετε για κάθε επιπλέον απεικόνιση.
5	Ολοκλήρωση	Επιλογή OK στο πάνω μέρος.
6	Επιλογή προβολής εκρηκτικής απεικόνισης	Στο αριστερό μενού καρτέλα Configuration Manager → Με το

7	Σχεδίαση εκρηκτικών γραμμών	ποντίκι τράβηγμα μπλε γραμμής προς τα πάνω (για την επιστροφή στην εκριτική απεικόνιση προς τα κάτω). Configuration Manager → 3DExplode1 → Sketch Color → Επιλογή χρώματος.
---	------------------------------------	--

Πίνακας 17: Διαδικασία για την επίτευξη της απεικόνισης

Όλες οι παραπάνω διαδικασίες επαναλήφθηκαν για κάθε βραχίονα, διασφαλίζοντας ότι οι άξονες περιστροφής είναι απόλυτα κάθετοι στο επίπεδο XY του οχήματος και ευθυγραμμισμένοι με τον άξονα Z. Η ακρίβεια σε αυτό το στάδιο είναι κρίσιμη, καθώς η παραμικρή κλίση θα προκαλούσε ανεπιθύμητες δυνάμεις εκτροπής κατά την προσομοίωση.

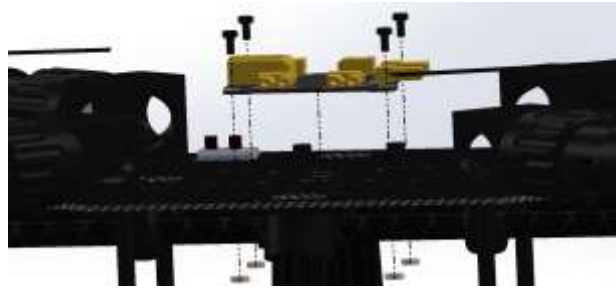


Εικόνα 21: Τοποθέτηση και συναρμολόγηση τμημάτων κινητήρα

3.3.3 Τοποθέτηση Εξαρτημάτων Κεντρικού Επιπέδου

Ως κεντρικό επίπεδο θεωρήθηκε εκείνο στο οποίο συνδέονται οι βραχίονες με τον σκελετό του οχήματος. Σε αυτό το σημείο τοποθετήθηκε το σύστημα διανομής ισχύος, καθώς αποτελεί το πλέον κατάλληλο σημείο για τη διασύνδεση των κινητήρων.

Η συγκεκριμένη θέση επιλέχθηκε ώστε να ελαχιστοποιηθεί το συνολικό μήκος των καλωδιώσεων προς τους κινητήρες. Η προσέγγιση αυτή συμβάλλει καθοριστικά στη μείωση των απωλειών ισχύος λόγω ηλεκτρικής αντίστασης, ενώ παράλληλα βελτιώνει την καλωδιακή οργάνωση στο εσωτερικό του πλαισίου, διευκολύνοντας τη συντήρηση και διασφαλίζοντας ότι δεν υπάρχουν παρεμβολές με τα κινητά μέρη ή τους αισθητήρες.



Εικόνα 22: Τοποθέτηση εξαρτημάτων κεντρικού επιπέδου

3.3.4 Τοποθέτηση Εξαρτημάτων Πρώτου Επιπέδου

Στο πρώτο επίπεδο του πλαισίου εγκαταστάθηκαν τα κρίσιμα ηλεκτρονικά συστήματα που ελέγχουν τη λειτουργία και την επικοινωνία του ΜΕΑ. Η διάταξη αυτή σχεδιάστηκε με γνώμονα τη βέλτιστη διασύνδεση και την ελαχιστοποίηση του ηλεκτρονικού θορύβου.

Συγκεκριμένα, τα εξαρτήματα χωροθετήθηκαν ως εξής:

- Ηλεκτρονικός Ελεγκτής Ταχύτητας (ESC): Η τοποθέτηση σε αυτό το επίπεδο εξασφαλίζει άμεση και συμμετρική σύνδεση με τους κινητήρες. Η προσέγγιση αυτή μειώνει την ηλεκτρική αντίσταση των καλωδίων και περιορίζει τις ηλεκτρομαγνητικές παρεμβολές προς τους ευαίσθητους αισθητήρες.
- Υπολογιστική Μονάδα και Σύστημα Πλοήγησης: Η τοποθέτηση του Jetson Nano και του Pixhawk 6X στο ίδιο δομικό επίπεδο εξασφαλίζει την αμεσότητα στην επικοινωνία τους. Αυτή η γειτνίαση βελτιώνει τον συγχρονισμό μεταξύ του ελέγχου πτήσης, των αλγορίθμων τεχνητής νοημοσύνης και της επεξεργασίας δεδομένων από τους αισθητήρες.
- Μονάδα Τηλεμετρίας: Η θέση της επιλέχθηκε ώστε να διευκολύνεται η απρόσκοπτη ανταλλαγή δεδομένων με τον επίγειο σταθμό ελέγχου, διασφαλίζοντας τη μέγιστη δυνατή εμβέλεια και σταθερότητα σήματος.

Η συγκέντρωση αυτών των μονάδων στην κεντρική περιοχή του πλαισίου συμβάλλει στη συγκέντρωση της μάζας κοντά στο κέντρο βάρους, γεγονός που βελτιώνει τη δυναμική απόκριση του οχήματος κατά τις απότομες μεταβολές κατεύθυνσης.



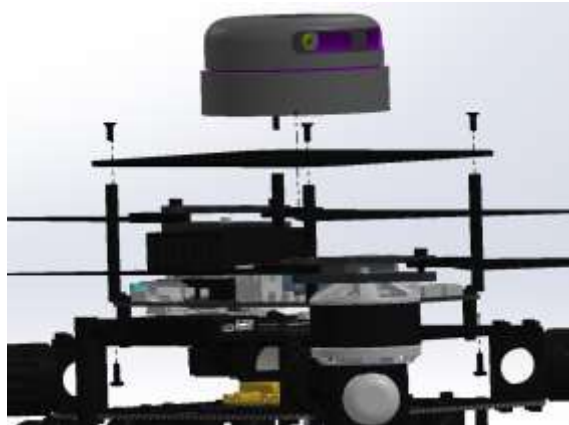
Εικόνα 23: Τοποθέτηση εξαρτημάτων 1ου επιπέδου

3.3.5 Τοποθέτηση Εξαρτημάτων Ανώτερου Επιπέδου

Ο αισθητήρας LiDAR τοποθετήθηκε στο ανώτατο επίπεδο του πλαισίου, ώστε να εξασφαλίζεται ανεμπόδιστη εκπομπή και λήψη των ακτίνων λέιζερ χωρίς παρεμβολές ή ανακλάσεις από τα επιμέρους τμήματα του ίδιου του συστήματος (όπως οι βραχίονες ή οι έλικες).

Η επιλογή της συγκεκριμένης θέσης είναι στρατηγικής σημασίας, καθώς:

- Επιτρέπει πλήρη περιμετρική ορατότητα 360°, εξασφαλίζοντας ότι ο αισθητήρας «βλέπει» προς όλες τις κατευθύνσεις του οριζόντιου επιπέδου.
- Διευκολύνει την ακριβή χαρτογράφηση του περιβάλλοντος χώρου σε πραγματικό χρόνο.
- Ενισχύει την αξιόπιστη ανίχνευση εμποδίων, στοιχείο απαραίτητο για την ασφάλεια των πτήσεων.



Εικόνα 24: Τοποθέτηση εξαρτημάτων τελευταίου ανωτέρου επιπέδου

3.3.6 Τοποθέτηση Εξαρτημάτων στο Εμπρόσθιο και Οπίσθιο Τμήμα του Πλαισίου

Στο εμπρόσθιο μέρος του οχήματος τοποθετήθηκε η κάμερα βάθους, ενώ στο οπίσθιο, σε κάθετο στύλο στήριξης, εγκαταστάθηκε η μονάδα GPS.

Η θέση τοποθέτησης της κάμερας εξασφαλίζει ανεμπόδιστο πεδίο λήψης (field of view) και βέλτιστη ορατότητα, στοιχεία απαραίτητα για την τρισδιάστατη αντίληψη του χώρου. Αντίστοιχα, η μονάδα GPS τοποθετήθηκε σε υπερυψωμένο σημείο προκειμένου να διασφαλίζεται η απρόσκοπτη δορυφορική λήψη σήματος και να ελαχιστοποιούνται οι παρεμβολές από τα ηλεκτρονικά ισχύος του πλαισίου. Η συγκεκριμένη διάταξη εξασφαλίζει:

- Αξιόπιστη πλοήγηση μέσω σταθερής λήψης GNSS.
- Ακριβή γεωεντοπισμό σε πραγματικό χρόνο.
- Μέγιστη απόδοση των αλγορίθμων αποφυγής εμποδίων λόγω της καθαρής οπτικής γωνίας της κάμερας.



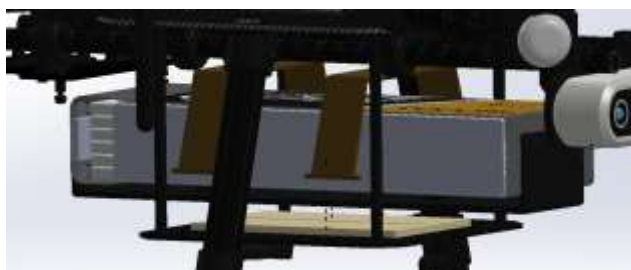
Εικόνα 25: Τοποθέτηση εξαρτημάτων εμπρός/πίσω

3.3.7 Τοποθέτηση Εξαρτημάτων Κατώτερου Επιπέδου

Το τελευταίο υποσύστημα που ενσωματώθηκε στο πλαίσιο είναι η μπαταρία, συνοδευόμενη από τα απαραίτητα εξαρτήματα σταθεροποίησης. Δεδομένου ότι η μπαταρία αποτελεί το βαρύτερο εξάρτημα του οχήματος, η ορθή τοποθέτησή της είναι κρίσιμη για τη διατήρηση του κέντρου βάρους σε χαμηλό σημείο, ενισχύοντας την πτητική ευστάθεια.

Για την ασφαλή στερέωσή της χρησιμοποιήθηκαν τα εξής:

- **Ειδική Βάση Στήριξης:** Σχεδιασμένη (όπως περιγράφηκε στην ενότητα 3.2.1) για να συγκρατεί τον συσσωρευτή στη θέση του, αποσβένοντας τους κραδασμούς και ανθιστάμενη στις έντονες επιταχύνσεις.
- **Ιμάντες Συγκράτησης:** Οι δύο ιμάντες (όπως περιγράφηκε στην ενότητα 3.2.2) εξασφαλίζουν τη σφιχτή πρόσδεση της μπαταρίας πάνω στη βάση. Η διάταξη αυτή αποτρέπει οποιαδήποτε μετακίνηση ή χαλάρωση κατά τη διάρκεια των ελιγμών, διασφαλίζοντας την επιχειρησιακή αξιοπιστία του ΜΕΑ.



Εικόνα 26: Τοποθέτηση εξαρτημάτων κατώτερου επιπέδου

Με την ολοκλήρωση της διαδικασίας όλων των υποσυστημάτων, το ψηφιακό μοντέλο είναι πλέον πλήρες

3.3.8 Τελική Δομή και Ολοκληρωμένη Απεικόνιση του Οχήματος

Τέλος, παρουσιάζονται οι απεικονίσεις της τελικής διάταξης, οι οποίες προβάλλουν το όχημα από διάφορες οπτικές γωνίες. Οι συγκεκριμένες προβολές αναδεικνύουν τη χωρική διάταξη των επιμέρους υποσυστημάτων, τη μηχανική ισορροπία και την αισθητική ολοκλήρωση του τελικού μοντέλου, επιβεβαιώνοντας τη σωστή εφαρμογή των αρχών σχεδιασμού. Η ολοκληρωμένη ψηφιακή αναπαράσταση επιτρέπει τον λεπτομερή έλεγχο των αποστάσεων όπως η συνολική διάμετρος και μεταξύ των εξαρτημάτων, διασφαλίζοντας ότι δεν υπάρχουν επικαλύψεις ή δομικές αστοχίες. Παράλληλα, η οπτικοποίηση αυτή αποτελεί το τελικό στάδιο πριν από την εξαγωγή των δεδομένων προς το περιβάλλον προσομοίωσης, όπου η γεωμετρία του μοντέλου θα συνδυαστεί με τις φυσικές του ιδιότητες.



Εικόνα 27: Μπροστινή όψη του MEA



Εικόνα 28: Δεξιά πλαϊνή όψη του MEA



Εικόνα 29: Πάνω όψη του MEA



Εικόνα 30: Κάτω όψη του MEA



Εικόνα 31: Μπροστινή πανοραμική όψη του ΜΕΑ



Εικόνα 32: Πίσω πανοραμική όψη του ΜΕΑ



Εικόνα 33: Συνολική διάμετρος συμπεριλαμβανομένων και των ελίκων

3.4 Προσδιορισμός Κέντρου Βάρους, Ευθυγράμμιση και Επαλήθευση Συστήματος Αναφοράς

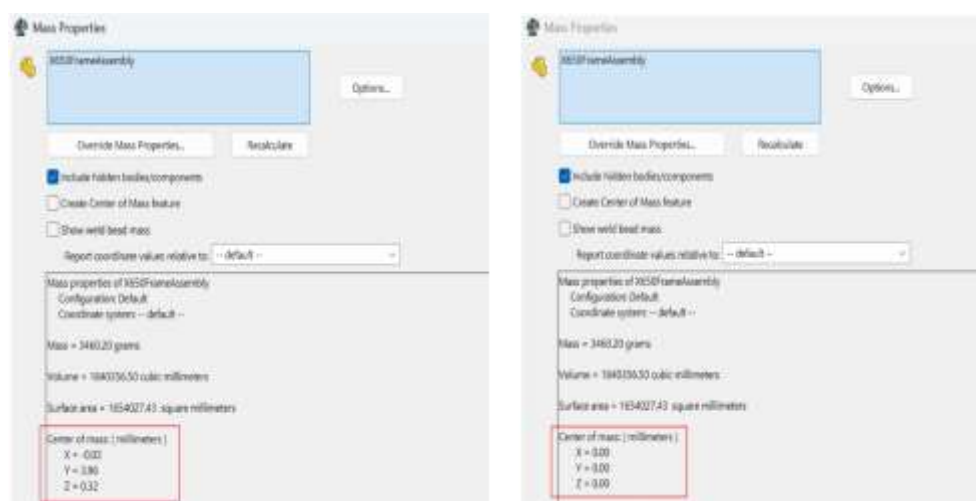
Κατά τη διαδικασία τοποθέτησης και συναρμολόγησης των εξαρτημάτων, το μοντέλο του ΜΕΑ ενδέχεται να είχε μετατοπιστεί από το αρχικό σημείο προέλευσης (0,0,0), λόγω επαναλαμβανόμενων προσαρμογών και αλλαγών στη σχεδίαση. Για τον λόγο αυτό, πριν από την τελική εξαγωγή, κρίθηκε απαραίτητη η επαναφορά και η ευθυγράμμιση όλων των στοιχείων ως προς το σημείο αναφοράς, ώστε το παγκόσμιο σύστημα συντεταγμένων (global coordinate system) να ταυτίζεται με το κέντρο βάρους του οχήματος. Το κέντρο βάρους (Center of Mass, CoM) αποτέλεσε το φυσικό σημείο ισορροπίας του συστήματος και η επιλογή του ως σημείο αναφοράς συνέβαλε στην ακριβή και σταθερή προσομοίωση.

Για τον προσδιορισμό του κέντρου βάρους και την επιδιόρθωση της απόκλισης, ακολουθήθηκαν τα παρακάτω βήματα:

Βήμα	Ενέργεια	Εκτέλεση
1	Υπολογισμός κέντρου βάρους	Καρτέλα Evaluate → Mass Properties. Αν παρατηρηθεί απόκλιση της τιμής του CoM από το μηδέν, έστω και σε έναν άξονα, ακολουθούνται τα παρακάτω βήματα για τη διόρθωση της θέσης του υποσυνόλου.

2	Δημιουργία υποσυνόλου (Subassembly)	Επιλογή εξαρτημάτων για τη δημιουργία υποσυνόλου → Μενού Insert → Component → Assembly from [Selected] Components...
3	Επιδιόρθωση θέσης	Επιλογή υποσυνόλου → Καρτέλα Assembly → Move Component → Επιλογή By Delta XYZ → Επιδιόρθωση διαφοράς, ώστε όλοι οι άξονες να έχουν τιμή μηδέν π.χ. Αν $Y = 3.96$, επιδιόρθωση με -3.96 .

Πίνακας 18: Βήματα για τον προσδιορισμό κέντρου βάρους και επιδιόρθωση απόκλισης



Εικόνα 34: Αριστερά: Απόκλιση του μοντέλου από το κέντρο βάρους. Δεξιά: Διορθωμένες συντεταγμένες μετά την ευθυγράμμιση του κέντρου βάρους με το σημείο προέλευσης.

Αρχικά όπως διατυπώνει η παραπάνω Εικόνα παρατηρήθηκε απόκλιση του κέντρου βάρους από το σημείο προέλευσης, με μικρές μετατοπίσεις κατά μήκος των αξόνων X, Y και Z ($X = -0,02$ mm, $Y = 3,96$ mm, $Z = 0,32$ mm). Μετά τη διορθωτική μετατόπιση του υποσυνόλου, οι συντεταγμένες του κέντρου βάρους μηδενίστηκαν.

Το αποτέλεσμα επιβεβαιώνει ότι το κέντρο βάρους ταυτίζεται πλέον με το παγκόσμιο σημείο αναφοράς (0,0,0), εξασφαλίζοντας συμμετρική αναφορά και ορθή προετοιμασία του μοντέλου για ακριβή αναπαράσταση στο ρομποτικό περιβάλλον προσομοίωσης. Σημειώνεται ότι, μετά τη μετατόπιση, οι μαζικές και αδρανειακές ιδιότητες επανυπολογίστηκαν ώστε να διατηρηθεί η φυσική ορθότητα του μοντέλου.

3.5 Εξαγωγή σε Πρότυπο Περιγραφής Ρομπότ

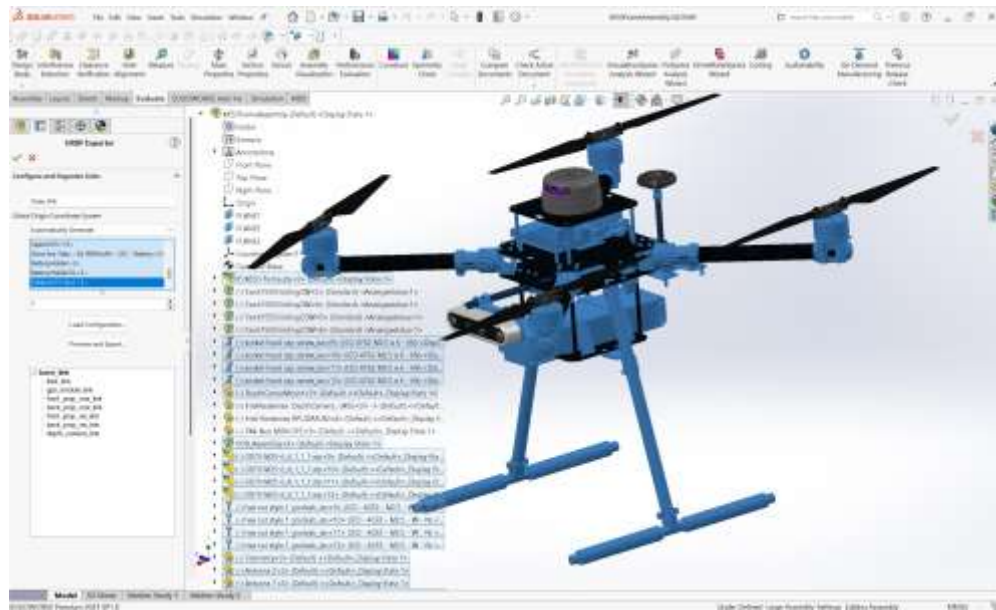
Η ακριβής και ολοκληρωμένη σχεδίαση και συναρμολόγηση του οχήματος, σύμφωνα με τις προδιαγραφές που προηγήθηκαν, αποτελεί κρίσιμη προϋπόθεση για την επιτυχή εξαγωγή του μοντέλου σε μορφή URDF (Unified Robot Description Format). Για τον σκοπό αυτό, το λογισμικό σχεδίασης χρησιμοποιεί το πρόσθετο (plugin) URDF Exporter, μέσω του οποίου καθορίζονται οι σύνδεσμοι (links), οι αρθρώσεις (joints) και οι σχετικές φυσικές και κινηματικές τους ιδιότητες πριν από την τελική εξαγωγή.

Η διαδικασία ρύθμισης και εξαγωγής ακολούθησε συγκεκριμένα βήματα, όπως περιγράφονται στη συνέχεια:

Βήμα	Ενέργεια	Εκτέλεση
1	Εξάλειψη του υποσυνόλου	Δεξί κλικ στο υποσύνολο → Dissolve Subassembly.
2	Επιλογή εξαγωγή σε URDF	Επιλογή μενού Tools → Export as URDF.
3	Επιλογή των εξαρτημάτων	Η διαδικασία ξεκινά με ιεραρχική επιλογή των οντοτήτων, αρχίζοντας από τον κύριο σύνδεσμο (base_link). Στον σύνδεσμο εντάσσονται όλες οι μη κινούμενες οντότητες που δεν συμμετέχουν ενεργά στο περιβάλλον προσομοίωσης, ώστε να αποτελέσουν τη σταθερή αναφορά του μοντέλου.
4	Ορισμός ονόματος άρθρωσης και συνδέσμου	Εισαγωγή στα πρώτα δύο πεδία → Ονομασία συνδέσμου <...>_link, Ονομασία άρθρωσης <...>_joint.
5	Συντεταγμένες αναφοράς	Referenced Coordinate System και Reference Axis πρέπει να παραμείνουν στην επιλογή Automatically Generate. Σε διαφορετική περίπτωση οι συντεταγμένες αναφοράς δε θα συμπίπτουν με το όχημα, το οποίο θα προκαλέσει προβληματισμό στην προσομοίωση στο ρομποτικό περιβάλλον.
6	Ορισμός αρθρώσεων	Joint Type → Για κινούμενες οντότητες επιλέγεται ο κατάλληλος τύπος άρθρωσης (Continuous, Revolute ή Prismatic)

ανάλογα με τη φύση και τον βαθμό ελευθερίας της κίνησης. Αντίθετα, για στατικές οντότητες επιλέγεται η επιλογή Fixed, ώστε να παραμένουν αμετακίνητες κατά την προσομοίωση.

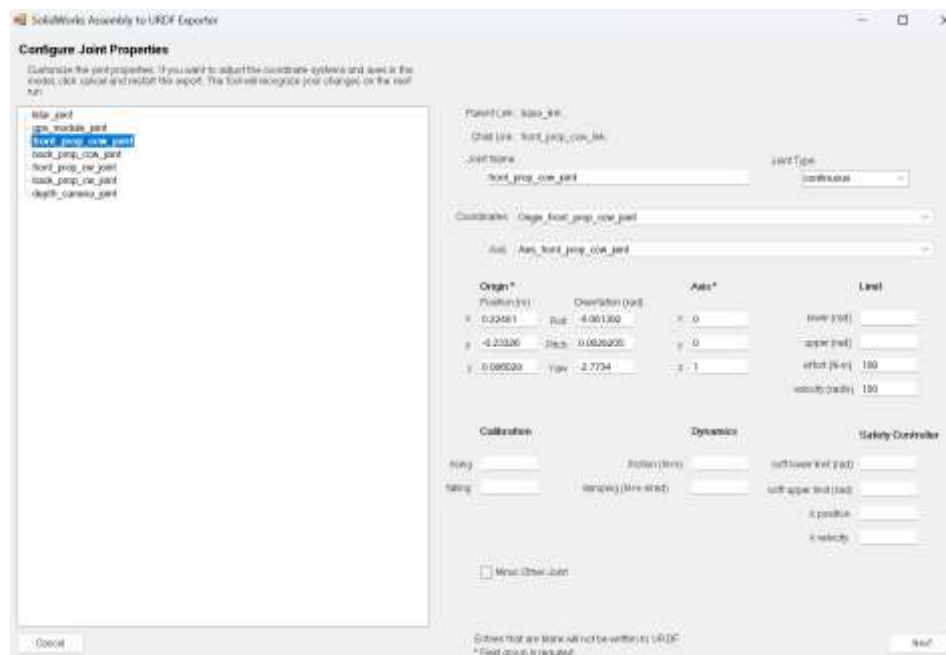
Πίνακας 19: Βήματα ρύθμισης για την εξαγωγή σε URDF (1)



Εικόνα 35: Ρύθμιση για τον κύριο σύνδεσμο

Βήμα	Ενέργεια	Συνέχεια Εκτέλεση
7	Προεπισκόπηση και Εξαγωγή	Επιλογή Preview and Export..
8	Ρύθμιση αρθρώσεων	Για τις σταθερές οντότητες επιλέγεται τύπος άρθρωσης (Joint Type) Fixed, ώστε να παραμένουν αμετακίνητες κατά την προσομοίωση. Αντίθετα, για τις έλικες επιλέγεται Continuous, προκειμένου να επιτρέπεται η περιστροφική κίνηση χωρίς περιορισμούς. Για τις έλικες καθορίζεται επίσης ο άξονας περιστροφής, με τιμές: Axis (X = 0, Y = 0, Z = 1) και ορίζονται οι δυναμικές παράμετροι: Effort = 100 και Velocity = 100.

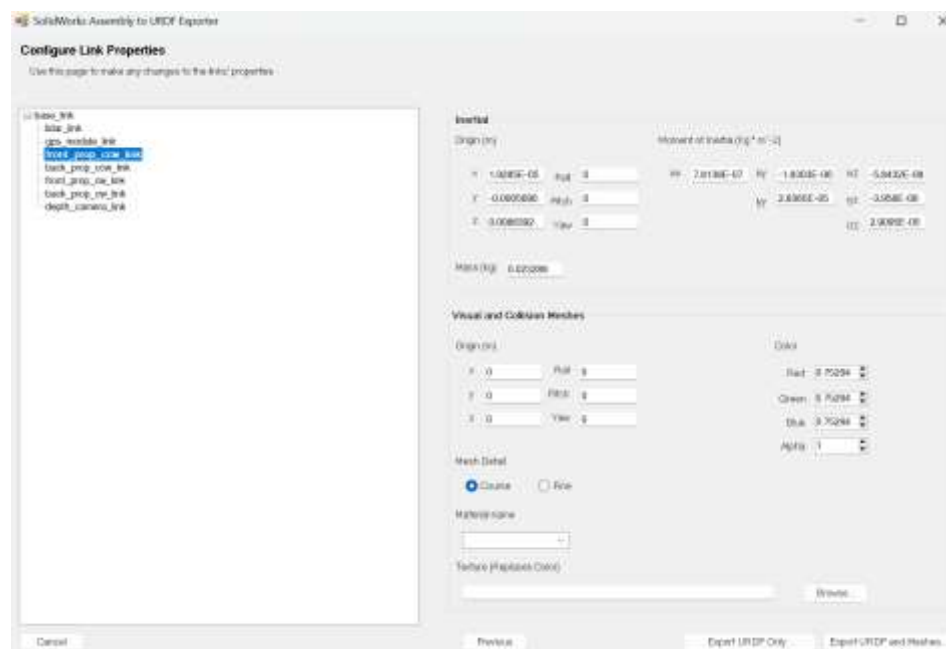
Πίνακας 20: Βήματα ρύθμισης για την εξαγωγή σε URDF (2)



Εικόνα 36: Ρύθμιση των αρθρώσεων

Βήμα	Ενέργεια	Συνέχεια Εκτέλεση
9	Ρύθμιση συνδέσμων	Εδώ τα στοιχεία δεν τροποποιούνται.

Πίνακας 21: Βήματα ρύθμισης για την εξαγωγή σε URDF (3)



Εικόνα 37: Ρύθμιση των συνδέσμων

Βήμα	Ενέργεια	Συνέχεια Εκτέλεση
10	Ολοκλήρωση ρυθμίσεων και εξαγωγή	Επιλογή Export URDF and Meshes → Επιλογή θέση αποθήκευσης → Save.
11	Επιβεβαίωση δημιουργίας φακέλου	Όταν εμφανιστεί ο διάλογος → OK.

Πίνακας 22: Βήματα ρύθμισης για την εξαγωγή σε URDF (4)

Η ορθή παραμετροποίηση κατά την εξαγωγή διασφαλίζει ότι το ρομποτικό περιβάλλον θα αναγνωρίσει σωστά τη γεωμετρία, τη μάζα και τις κινηματικές αλυσίδες του MEA. Με την ολοκλήρωση αυτής της διαδικασίας, παράγεται ένα πακέτο αρχείων που περιλαμβάνει το αρχείο .urdf, καθώς και τα απαραίτητα αρχεία πλέγματος (meshes) για την οπτική αναπαράσταση.

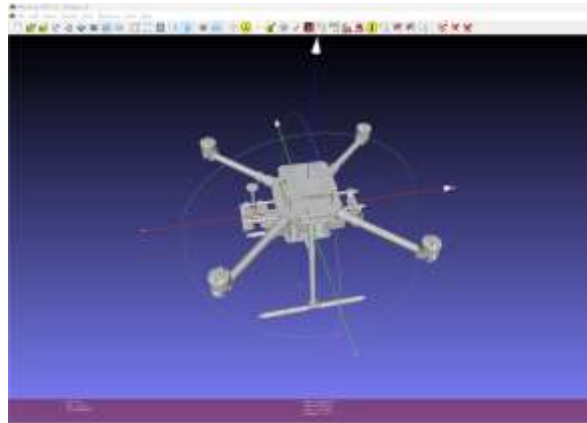
3.6 Επαλήθευση Εξαγόμενου Γεωμετρικού Μοντέλων

Η δημιουργία των αρχείων URDF δεν συνεπάγεται αυτόματα και την ορθή εξαγωγή τους. Απαιτείται επιπλέον έλεγχος και διαδικασία επαλήθευσης, ώστε να διασφαλιστεί ότι τα παραγόμενα τρισδιάστατα δεδομένα αποτυπώνουν με ακρίβεια τη γεωμετρία του συστήματος και ότι οι συντεταγμένες αναφοράς έχουν οριστεί ορθά. Η φάση αυτή είναι ιδιαίτερα κρίσιμη, καθώς σφάλματα όπως ο λανθασμένος υπολογισμός του κέντρου βάρους, η ανακριβής ευθυγράμμιση αξόνων ή ο εσφαλμένος ορισμός αρθρώσεων μπορούν να οδηγήσουν σε μη ρεαλιστική προσομοίωση και ανεπιθύμητη συμπεριφορά του MEA στο ρομποτικό περιβάλλον.

Για τη διαδικασία επαλήθευσης αξιοποιήθηκε το εργαλείο MeshLab [36], ένα λογισμικό ανοιχτού κώδικα που επιτρέπει την ανάλυση και τον έλεγχο τρισδιάστατων πλεγμάτων (3D meshes). Μέσω του MeshLab κατέστη εφικτό να:

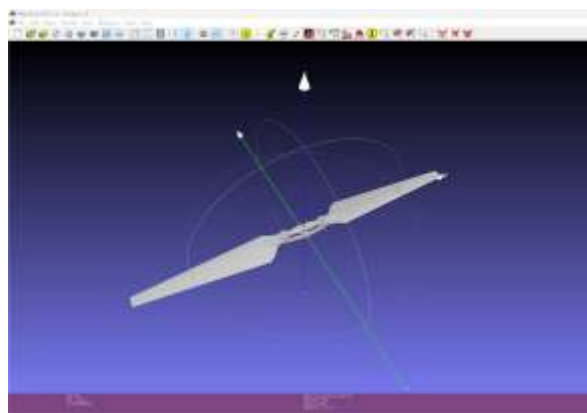
- Επιβεβαιωθεί η σωστή χωρική τοποθέτηση των γεωμετρικών δεδομένων.
- Ελεγχθεί η ακεραιότητα των εξαγόμενων επιφανειών (meshes).
- Επαληθευτεί ότι το σημείο προέλευσης κάθε εξαρτήματος συμπίπτει με τις προδιαγραφές του URDF.

Σε περίπτωση που σε ένα ή περισσότερα εξαρτήματα οι άξονες αναφοράς παρουσίαζαν απόκλιση από το επιθυμητό σύστημα συντεταγμένων, απαιτήθηκε επανάληψη της διαδικασίας που περιγράφηκε στο Κεφάλαιο 3.5 και διόρθωση των λανθασμένων ρυθμίσεων. Η επαναληπτική αυτή διαδικασία διασφαλίζει ότι το μοντέλο που θα εισαχθεί στον εξομοιωτή θα είναι απόλυτα πιστό στις φυσικές και γεωμετρικές ιδιότητες του πραγματικού οχήματος.



Εικόνα 38: Επαλήθευση συντεταγμένων αναφοράς στο MeshLab

Σημαντική Σημείωση Ιδιαίτερα κρίσιμη ήταν η εξάλειψη οποιασδήποτε απόκλισης στους άξονες αναφοράς των ελίκων. Σε αντίθετη περίπτωση, η περιστροφή τους στο ρομποτικό περιβάλλον θα αποδιδόταν με λανθασμένο προσανατολισμό. Μια τέτοια αστοχία θα προκαλούσε ασυμμετρία στην παραγόμενη ώση, με αποτέλεσμα το MEA να αδυνατεί να διατηρήσει σταθερή πτήση ή να παρουσιάζει απρόβλεπτη δυναμική συμπεριφορά.



Εικόνα 39: Επαλήθευση συντεταγμένων του έλικα στο MeshLab

Με την επιτυχή ολοκλήρωση της γεωμετρικής και φυσικής επαλήθευσης, το ψηφιακό μοντέλο του X650 μεταπίπτει από το στάδιο της στατικής σχεδίασης σε εκείνο της δυναμικής προσομοίωσης. Η διαδικασία αυτή διασφαλίζει ότι το «ψηφιακό δίδυμο» (digital twin) που δημιουργήθηκε, δεν είναι απλώς μια οπτική αναπαράσταση, αλλά ένα πλήρες φυσικό σύστημα έτοιμο να ανταποκριθεί σε πραγματικούς νόμους της μηχανικής.

4. Εγκατάσταση και Διαμόρφωση Υποδομής

Η προσομοίωση σε εικονικά περιβάλλοντα αποτελεί θεμέλιο λίθο στην ανάπτυξη των ΣμηΕΑ. Πριν από την εφαρμογή σε πραγματικές συνθήκες, επιτρέπει τη διεξοδική μελέτη της λειτουργίας και την αξιολόγηση της συμπεριφοράς τους σε ποικίλα σενάρια. Με τον τρόπο αυτό, εντοπίζονται έγκαιρα τυχόν αστοχίες χωρίς την ανάγκη δαπανηρών φυσικών πρωτοτύπων, ενώ διασφαλίζεται η ακεραιότητα του οχήματος και η προστασία ατόμων. Επιπρόσθετα, το ελεγχόμενο περιβάλλον προσομοίωσης είναι ιδανικό για την εκπαίδευση και βελτιστοποίηση αλγορίθμων τεχνητής νοημοσύνης, ενισχύοντας τις δυνατότητες αυτόνομης πλοήγησης και αντίληψης. Η προσέγγιση αυτή ελαχιστοποιεί το κόστος και τους κινδύνους των πειραματικών δοκιμών.

Στην παρούσα εργασία, υλοποιήθηκε η απαραίτητη υποδομή και αναπτύχθηκε ένα ολοκληρωμένο εικονικό περιβάλλον. Η διαδικασία αυτή διαρθρώθηκε σε συγκεκριμένες φάσεις, οι οποίες αναλύονται διεξοδικά στη συνέχεια.

4.1 Λειτουργικό Σύστημα και Εργαλεία Ανάπτυξης

Η προετοιμασία της υπολογιστικής υποδομής ξεκίνησε με την εγκατάσταση του λειτουργικού συστήματος, πάνω στο οποίο δομήθηκε το ρομποτικό οικοσύστημα ROS 2, ενσωματώνοντας όλες τις απαραίτητες βιβλιοθήκες ανοιχτού κώδικα. Στη συνέχεια, προστέθηκε το περιβάλλον προσομοίωσης για την πιστή αναπαράσταση της δυναμικής του ΜΕΑ, ενώ ακολούθησε η εγκατάσταση του λογισμικού ελέγχου πτήσης και η παραμετροποίηση της κρίσιμης αμφίδρομης επικοινωνίας με το ROS 2. Η υποδομή ολοκληρώθηκε με την ενσωμάτωση του επίγειου σταθμού ελέγχου, των κατάλληλων πακέτων (packets) χαρτογράφησης και προηγμένων αλγορίθμων υπολογιστικής όρασης (computer vision). Η συνδυαστική χρήση αυτών των εργαλείων επιτρέπει τη μετατροπή του ΜΕΑ σε ένα ευφυές σύστημα με πλήρη αντίληψη του χώρου και δυνατότητα αυτόνομης αναγνώρισης αντικειμένων.

Οι αναλυτικές οδηγίες για την εγκατάσταση του συνόλου των παραπάνω εργαλείων παρατίθενται στο Παράρτημα Α. Αξίζει να σημειωθεί ότι όλα τα προαναφερθέντα πακέτα διατίθενται δωρεάν υπό άδειες ανοικτού κώδικα, αποτελώντας ισχυρά

εργαλεία που αξιοποιούνται ευρέως τόσο στην ακαδημαϊκή κοινότητα όσο και σε επαγγελματικές εφαρμογές ρομποτικής υψηλών απαιτήσεων.

4.1.1 Εγκατάσταση Λειτουργικού Συστήματος

Η εγκατάσταση του λειτουργικού συστήματος Ubuntu 24.04 LTS [37] αποτέλεσε το θεμέλιο για τη διαμόρφωση του υπολογιστικού περιβάλλοντος. Η επιλογή της συγκεκριμένης έκδοσης βασίστηκε στη σταθερότητα που προσφέρει η μακροχρόνια υποστήριξη (LTS), την πλήρη συμβατότητα με το ROS 2 Jazzy και την ευρεία αποδοχή της από τη ρομποτική κοινότητα. Πριν από την έναρξη της διαδικασίας, διασφαλίστηκε ότι το υλικό ακολουθεί τις απαιτήσεις για την απρόσκοπτη εκτέλεση της προσομοίωσης και των εργαλείων ανάπτυξης. Οι προδιαγραφές αυτές, όπως αναφέρονται στην επίσημη τεκμηρίωση, συνοψίζονται στον παρακάτω πίνακα:

System Requirements	
CPU	2 GHz dual-core processor or better
RAM	4 GB system memory
HDD	25 GB of free hard drive space
Installer media	Either a USB or a DVD drive

Πίνακας 23: Απαιτήσεις συστήματος για Ubuntu 24.04 LTS [37]

```
$ /lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 24.04.2 LTS
Release:       24.04
Codename:      noble
```

Εικόνα 40: Επαλήθευση ορθής εγκατάστασης και έκδοσης Ubuntu

4.1.2 Εγκατάσταση Ρομποτικού Οικοσυστήματος και Εργαλεία Ανάπτυξης

Η εγκατάσταση του ROS 2 στην έκδοση Jazzy (Jazzy Jalisco) [38] αποτέλεσε το επόμενο στάδιο στη διαμόρφωση της υποδομής, λειτουργώντας ως η κεντρική πλατφόρμα ανάπτυξης για το λογισμικό του MEA. Η επιλογή της συγκεκριμένης έκδοσης αποτέλεσε την ενδεδειγμένη λύση για το λειτουργικό σύστημα, ακολουθώντας πιστά τις προδιαγραφές επίσημης τεκμηρίωσης της εκάστοτε έκδοσης ROS 2. Η ταύτιση αυτή διασφαλίζει την πλήρη συμβατότητα των βιβλιοθηκών, την

απρόσκοπτη εγκατάσταση των εξαρτήσεων και την ελαχιστοποίηση σφαλμάτων κατά τη διαδικασία μεταγλώττισης του λογισμικού.

Το ROS 2 δεν αποτελεί αυτόνομο λειτουργικό σύστημα, αλλά ένα σύγχρονο, ανοιχτού κώδικα οικοσύστημα που αποτελείτε από μια πληθώρα από πακέτα, εργαλείων και βιβλιοθηκών που υποστηρίζουν τη δημιουργία σύνθετων και αρθρωτών ρομποτικών συστημάτων διαφόρων κατηγοριών. Προσφέρει προηγμένους μηχανισμούς επικοινωνίας μεταξύ κόμβων, δυνατότητα ενσωμάτωσης αισθητήρων και πλήρη υποστήριξη για περιβάλλοντα προσομοίωσης, εξασφαλίζοντας μια σταθερή βάση για την αξιολόγηση του συστήματος.

Κύρια εργαλεία που συνοδεύονται στην έκδοση αυτή περιλαμβάνουν:

Κατάλογος Βασικών Πρόσθετων και Εργαλείων του ROS 2	
ROS 2 core	Περιλαμβάνει τα βασικά εργαλεία διαχείρισης κόμβων (nodes), κανάλια επικοινωνίας (topics), υπηρεσίες (services) και ενεργείες (actions).
colcon	Απαραίτητο εργαλείο για τη δημιουργία και διαχείριση πακέτων.
rosdep	Για την αυτόματη επίλυση εξαρτήσεων (dependencies) των πακέτων.
RViz	Τρισδιάστατη οπτικοποίηση δεδομένων και κατάστασης του ρομπότ.
rqt και το rqt_graph	Παρακολούθηση της επικοινωνίας και της δομής των κόμβων του συστήματος.
tf2	Διαχείριση πλαισίων αναφοράς και τον μετασχηματισμό συντεταγμένων μεταξύ διαφορετικών μερών του ρομπότ.
Προκαθορισμένα πακέτα	Πακέτα geometry_msgs, sensor_msgs και nav_msgs, τα οποία παρέχουν μηνύματα για γεωμετρικά δεδομένα, πληροφορίες αισθητήρων και δεδομένα πλοήγησης αντίστοιχα.
ROS Client Library (RCL)	Αποτελεί τον μηχανισμό επικοινωνίας του ROS 2 με τις γλώσσες προγραμματισμού και διαχωρίζεται στις βιβλιοθήκες rclcpp για τον προγραμματισμό σε C++ και rclpy για Python, παρέχοντας ευελιξία στην αξιοποίηση

διαφορετικών γλωσσών προγραμματισμού, ανάλογα με τις ανάγκες της εκάστοτε εφαρμογής.

Πίνακας 24: Πρόσθετα πακέτα και εργαλεία του ROS 2

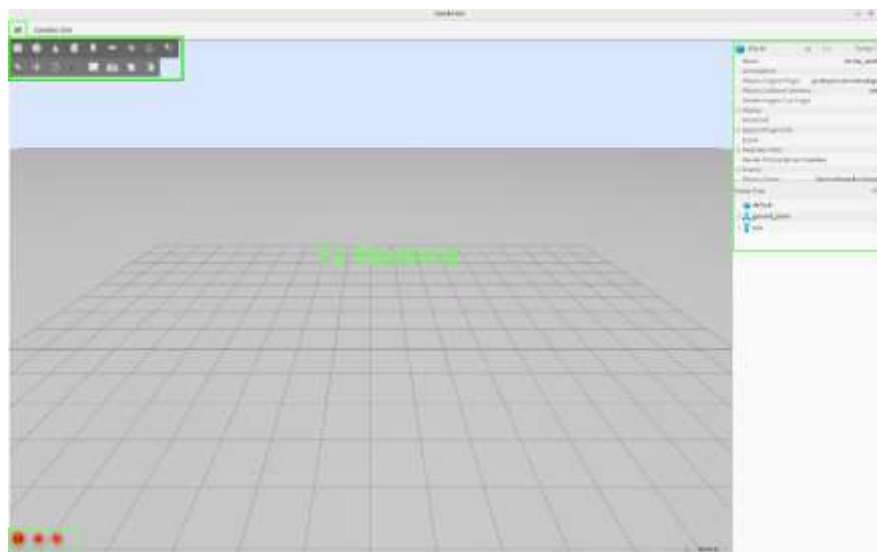
4.1.3 Εγκατάσταση Ρομποτικού Περιβάλλοντος Προσομοίωσης

Για τις ανάγκες της δυναμικής ανάλυσης του οχήματος σε περιβάλλον προσομοίωσης, εγκαταστάθηκε το Gazebo Sim στην έκδοση Harmonic [39], το οποίο αποτελεί μια σύγχρονη επιλογή του δημοφιλούς περιβάλλοντος προσομοίωσης ανοικτού κώδικα, πλήρως βελτιστοποιημένο για το ROS 2. Το Gazebo προσφέρει υψηλής ακρίβειας φυσική προσομοίωση, επιτρέποντας την πιστή αναπαράσταση της δυναμικής συμπεριφοράς του οχήματος σε συνθήκες που προσεγγίζουν το πραγματικό περιβάλλον.

Το περιβάλλον εργασίας του Gazebo Harmonic οργανώνεται ως εξής:

- Κεντρικό Παράθυρο: Τρισδιάστατος χώρος προβολής όπου ο χρήστης αλληλεπιδρά με τη σκηνή, μετακινώντας ή περιστρέφοντας αντικείμενα.
- Εργαλεία Χειρισμού (Επάνω Γραμμή): Περιλαμβάνουν λειτουργίες επιλογής, μετατόπισης και περιστροφής, καθώς και επιλογές για την εισαγωγή γεωμετρικών σχημάτων.
- Πάνελ Ιδιοτήτων και Δέντρο Ενοτήτων (Entity Tree - Δεξιά Πλευρά): Προβάλλουν τα χαρακτηριστικά των μοντέλων, τα ενεργά plugins, καθώς και την ιεραρχική δομή όλων των οντοτήτων (μοντέλα, φωτισμός).
- Έλεγχος Χρόνου (Κάτω Αριστερά): Παρέχει τα κουμπιά εκκίνησης, παύσης και επαναφοράς της προσομοίωσης.
- Διαχείριση Σκηνής (Επάνω Αριστερά): Προσφέρει επιλογές για την αποθήκευση ή τη φόρτωση αρχείων κόσμου .sdf (SDF, Simulation Description Format).

Η συγκεκριμένη διεπαφή επιτρέπει τον πλήρη έλεγχο και την παρακολούθηση των φυσικών παραμέτρων του MEA κατά την εκτέλεση της προσομοίωσης.



Εικόνα 41: Κενό περιβάλλον προσομοίωσης

Προκειμένου να δικαιολογηθεί η επιλογή του περιβάλλοντος, πραγματοποιήθηκε αξιολόγηση έναντι άλλων δημοφιλών προσομοιωτών.

Στον παρακάτω πίνακα παρουσιάζονται τα εναλλακτικά περιβάλλοντα προσομοίωσης και οι βασικές τους διαφορές σε σχέση με το Gazebo Sim:

Ιδιότητα / Χαρακτηριστικό	Gazebo Sim	Webots	NVIDIA Isaac Sim
Ρεαλισμός φυσικής	Πολύ υψηλός, με ακριβή μοντελοποίηση φυσικών φαινομένων	Μέτριος προς υψηλός	Εξαιρετικά υψηλός, με προηγμένα μοντέλα φυσικής
Παραμετροποίηση	Εκτεταμένη, με δυνατότητα προσαρμογής σε επίπεδο λεπτομέρειας	Περιορισμένη σε σχέση με Gazebo	Πολύ εκτεταμένη, με έμφαση σε ρεαλισμό και ακρίβεια
Υποστήριξη αισθητήρων	Πλούσια γκάμα αισθητήρων και ενεργοποιητών	Βασική υποστήριξη	Προηγμένη, με υψηλής ακρίβειας προσομοιώσεις
Υποστήριξη πολλαπλών ρομπότ	Ναι, πλήρης	Περιορισμένη	Πλήρης, με βελτιστοποιημένη απόδοση
Ενσωμάτωση με	Πλήρης μέσω	Υπάρχει, αλλά	Υποστηρίζεται,

ROS 2	ros_gz_bridge	με πρόσθετες ρυθμίσεις	με επιπλέον διαδικασίες
Απόδοση	Πολύ καλή σε τυπικά συστήματα	Ελαφριά πλατφόρμα	Πολύ υψηλές απαιτήσεις σε υπολογιστικούς πόρους
Ευχρηστία / Καμπύλη εκμάθησης	Απαιτεί περισσότερη εξοικείωση	Εύκολη για αρχάριους	Δύσκολη, απαιτεί εξειδικευμένες γνώσεις
Κοινότητα και υποστήριξη	Πολύ μεγάλη, ενεργή	Ικανοποιητική	Περιορισμένη σε σχέση με Gazebo, αλλά αναπτυσσόμενη

Πίνακας 25: Σύγκριση Gazebo Sim με άλλα περιβάλλοντα ρομποτικής

Η υπεροχή του Gazebo Sim για την παρούσα διπλωματική εργασία έγκειται στην εγγενή υποστήριξη του πρωτοκόλλου μεταφοράς δεδομένων του ROS 2, γεγονός που ελαχιστοποιεί την καθυστέρηση (latency) στην ανταλλαγή μηνυμάτων μεταξύ των αισθητήρων και των αλγορίθμων ελέγχου. Παράλληλα, η δυνατότητα εναλλαγής μεταξύ διαφορετικών κινητήρων φυσικής (physics engines) επιτρέπει την προσαρμογή της προσομοίωσης στις ιδιαίτερες απαιτήσεις της πτήσης ενός πολυκοπτέρου.

4.1.4 Εγκατάσταση PX4 Στοίβα Ελέγχου (PX4 Control Stack)

Για τη λειτουργία του ελεγκτή πτήσης Pixhawk 6X, εγκαταστάθηκε το PX4 Control Stack [40], το οποίο αποτελεί τον δομικό πυρήνα του αυτόματου πιλότου (PX4 Autopilot). Το λογισμικό αυτό είναι υπεύθυνο για τις κρίσιμες λειτουργίες σταθεροποίησης, τον έλεγχο των ενεργοποιητών (κινητήρων) και την πλοήγηση, ενώ παράλληλα επιτρέπει την απρόσκοπτη επικοινωνία με το ROS 2.

Η εγκατάσταση όπως διατυπώνετε στον παρακάτω πίνακα πραγματοποιήθηκε μέσω του επίσημου αποθετηρίου και παραμετροποιήθηκε κατάλληλα για τη λειτουργία λογισμικού σε κύκλωμα (Software-In-The-Loop, SITL). Στη διαμόρφωση αυτή, ο κώδικας του αυτόματου πιλότου εκτελείται στον υπολογιστή και επικοινωνεί με το

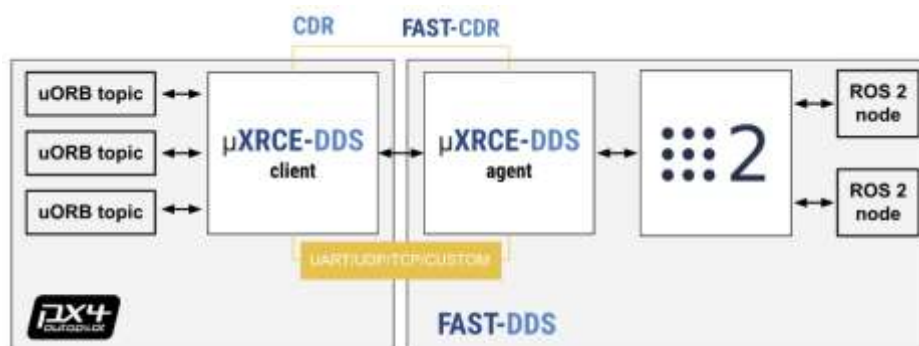
περιβάλλον προσομοίωσης, επιτρέποντας την εκτέλεση πτητικών δοκιμών σε ένα απόλυτα ελεγχόμενο ψηφιακό περιβάλλον πριν από τη χρήση του φυσικού υλικού.

Βήμα	Ενέργεια
1	Κλωνοποίηση του επίσημου αποθετηρίου (repository) PX4-Autopilot από το GitHub στο τοπικό σύστημα.
2	Μεταγλώττιση αποθετηρίου για το προκαθορισμένο μοντέλο X500.
3	Δημιουργία προσαρμοσμένου σκελετού αεροσκάφους (airframe) σε υποφάκελο ROMFS και ρύθμιση παραμέτρων.
4	Αντιγραφή του αρχείου σε υποφάκελο του rootfs.
5	Εξαγωγή μεταβλητής περιβάλλοντος (environment variable) GZ_SIM_RESOURCE_PATH, για την κοινοποίηση διαδρομής του μοντέλου στο περιβάλλον προσομοίωσης.

Πίνακας 26: Διαδικασία εγκατάστασης PX4-Autopilot

4.1.5 Εγκαθίδρυση Εικονισμίας PX4 με ROS2

Παράλληλα με τη στοίβα ελέγχου, κρίθηκε απαραίτητη η εγκατάσταση του Micro XRCE-DDS Agent [41]. Ο πράκτορας αυτός αποτελεί το κρίσιμο στοιχείο της αρχιτεκτονικής, καθώς επιτρέπει την αμφίδρομη επικοινωνία μεταξύ του PX4 και του ROS 2 μέσω του πρωτοκόλλου DDS (Data Distribution Service). Με τη χρήση αυτού του ενδιάμεσου λογισμικού (middleware), διασφαλίζεται η μετάδοση δεδομένων τηλεμετρίας, η ανάγνωση των αισθητήρων και η αποστολή εντολών πτήσης σε πραγματικό χρόνο με ελάχιστη καθυστέρηση.



Σχήμα 1: Αρχιτεκτονική επικοινωνίας μέσω Micro XRCE-DDS Agent [41]

Η διαδικασία εγκατάστασης και παραμετροποίησης υλοποιήθηκε μέσω των εξής ενεργειών:

Βήμα	Ενέργεια
1	Κλωνοποίηση του επίσημου αποθετηρίου Micro-XRCE-DDS-Agent και px4_msgs από το GitHub στο τοπικό σύστημα.
2	Μεταγλώττιση αποθετηρίου XRCE-DDS-Agent.
3	Ενημέρωση της μνήμης cache, ώστε το σύστημα να αναγνωρίζει τις νέες βιβλιοθήκες του πράκτορα (Agent).

Πίνακας 27: Εγκαθίδρυση επικοινωνίας PX4 με ROS 2

Η συγκεκριμένη εδραίωση επικοινωνίας είναι θεμελιώδης, καθώς μετατρέπει το PX4 από έναν κλειστό αυτόματο πιλότο σε ένα ανοιχτό ρομποτικό σύστημα, πλήρως ελεγχόμενο από το ROS 2.

4.1.6 Ενσωμάτωση Λογισμικού Σταθμού Εδάφους και Ελέγχου (Ground Station)

Όπως προαναφέρθηκε στο Κεφάλαιο 2.3 ένα ΣμηΕΑ εφαρμόζει και έναν επίγειο σταθμό, για την επικοινωνία και τον έλεγχο με το ΜΕΑ. Για αυτό το λόγο επιλέχθηκε το πρόγραμμα QGroundControl (QGC) [42]. Η επιλογή του QGC βασίστηκε, πέρα από τη δωρεάν διάθεσή του, στην πληθώρα δυνατοτήτων που προσφέρει για την παραμετροποίηση και τον έλεγχο των πτητικών λειτουργιών. Η διαισθητική γραφική διεπαφή του επιτρέπει τη διαχείριση σύνθετων αποστολών, τη λεπτομερή βαθμονόμηση των παραμέτρων του ελεγκτή πτήσης και την παρακολούθηση της τηλεμετρίας σε πραγματικό χρόνο. Με τον τρόπο αυτό, ο χρήστης αποκτά πλήρη έλεγχο και εποπτεία του συστήματος, διασφαλίζοντας την ορθή λειτουργία του ΜΕΑ κατά τη διάρκεια της προσομοίωσης.



Εικόνα 42: Περιβάλλον QGroundControl (QGC) [42]

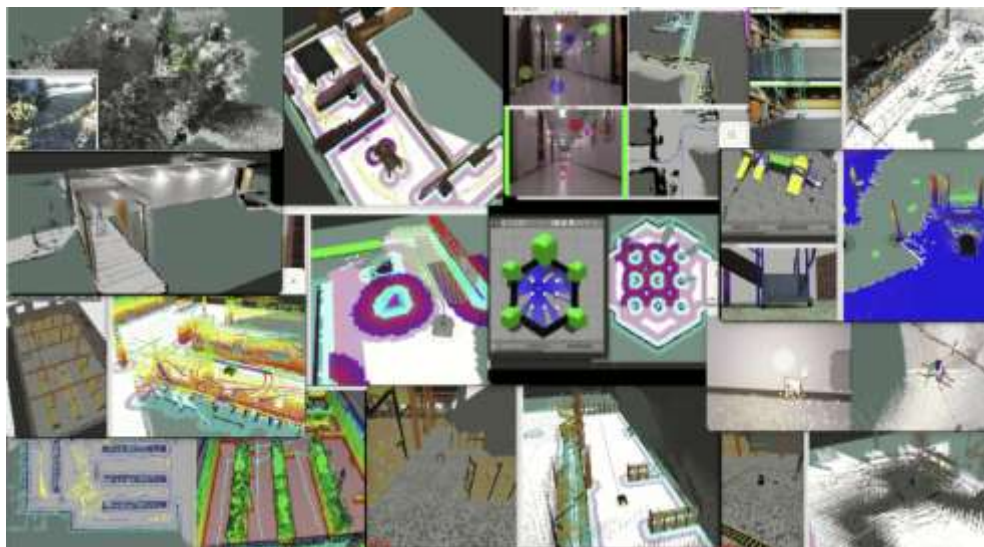
Μετά τη λήψη του AppImage από την επίσημη ιστοσελίδα, εκχωρήθηκαν δικαιώματα εκτέλεσης, επιτρέποντας στο πρόγραμμα να συνδεθεί αυτόματα με το ενεργό περιβάλλον PX4 SITL κατά την εκκίνηση του προγράμματος.

4.1.7 Εγκατάσταση Πακέτου Χαρτογράφησης (Mapping)

Η χαρτογράφηση αποτελεί μία από τις κρίσιμότερες λειτουργίες για ένα αυτόνομο ή τηλεκατευθυνόμενο ΜΕΑ, καθώς επιτρέπει στο όχημα να αντιλαμβάνεται τη γεωμετρία του περιβάλλοντος και να δημιουργεί ένα ψηφιακό αντίγραφο αυτού σε πραγματικό χρόνο. Η χαρτογράφηση είναι απαραίτητη για την παροχή χωρικής επίγνωσης στον χειριστή, επιτρέποντας την ασφαλή πλοήγηση σε σύνθετα περιβάλλοντα όπου η απλή οπτική επαφή δεν επαρκεί για την ακριβή εκτίμηση αποστάσεων.

Για την υλοποίηση αυτής της λειτουργίας, χρησιμοποιήθηκε το πακέτο RTAB-Map (Real-Time Appearance-Based Mapping) [43]. Πρόκειται για έναν αλγόριθμο SLAM που βασίζεται σε γραφήματα. Η συγκεκριμένη μέθοδος επιτρέπει στο ΜΕΑ να επιλύει ταυτόχρονα το πρόβλημα της δημιουργίας του χάρτη και του αυτοπροσδιορισμού της θέσης του μέσα σε αυτόν.

Στη συνέχεια, παρατίθενται ενδεικτικά παραδείγματα δισδιάστατων (2D) και τρισδιάστατων (3D) χαρτών τους οποίους δύναται να κατασκευάσει ο αλγόριθμος.



Εικόνα 43: Είδη χαρτών από τον αλγόριθμο RTAB-Map [43]

Η υλοποίηση στην εργασία βασίστηκε στη συνδυαστική χρήση δεδομένων από δύο κύριους αισθητήρες:

- 2D LiDAR: Χρησιμοποιείται για την ακριβή ανίχνευση εμποδίων στο οριζόντιο επίπεδο και τη δημιουργία δισδιάστατων χαρτών κατάληψης (occupancy grids).
- RGB-D Κάμερα: Επιτρέπει την εξαγωγή πληροφορίας βάθους και τη δημιουργία πυκνών τρισδιάστατων νεφών σημείων (Point Clouds), προσφέροντας πλούσια οπτική πληροφορία για το περιβάλλον.

Η απεικόνιση και ο έλεγχος των δεδομένων πραγματοποιείται μέσω δύο συμπληρωματικών εργαλείων. Το RViz2 αποτελεί το κύριο εργαλείο οπτικοποίησης του ROS 2. Μέσω της παραμετροποίησης πεδίων όπως το PointCloud2, το Map, καθώς και των MapGraph και MapCloud, ο χρήστης αποκτά μια πλήρη εικόνα για την πλοήγηση. Παράλληλα, χρησιμοποιήθηκε το rtabmap_viz, ένα εξειδικευμένο περιβάλλον διεπαφής του ίδιου του πακέτου RTAB-Map.

Οι αναλυτικές οδηγίες για την εγκατάσταση παρατίθενται στο Παράρτημα Α. Πρόκειται για πακέτα ανοικτού κώδικα που διατίθενται δωρεάν και χρησιμοποιούνται ευρέως τόσο στην έρευνα όσο και σε επαγγελματικές εφαρμογές.

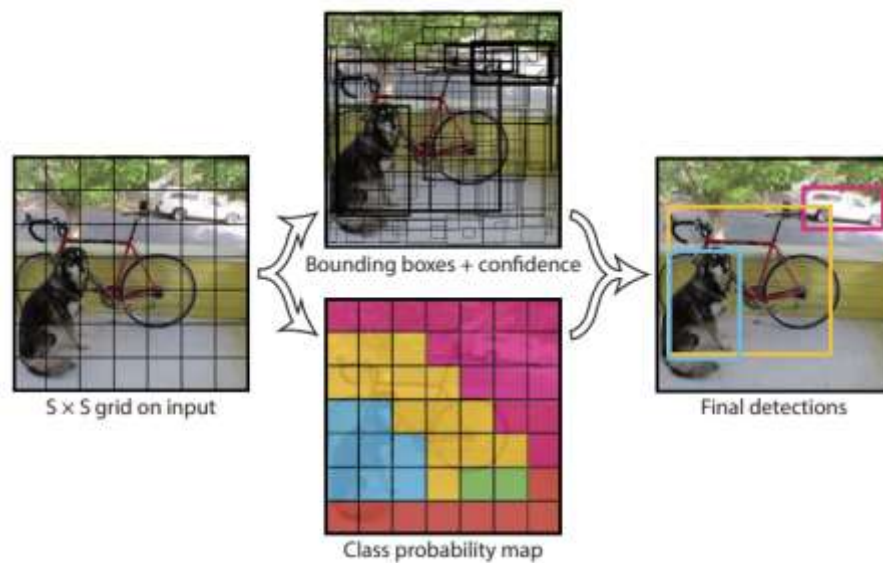
4.1.8 Εγκατάσταση Πακέτου Τεχνητής Όρασης (Computer Vision)

Η ενσωμάτωση δυνατοτήτων τεχνητής όρασης αποτελεί το επόμενο κρίσιμο στάδιο, επιτρέποντάς στο ΜΕΑ όχι μόνο να χαρτογραφεί τον χώρο, αλλά και να αντιλαμβάνεται την ταυτότητα των αντικειμένων μέσα σε αυτόν. Η υποδομή βασίστηκε στον συνδυασμό των βιβλιοθηκών OpenCV και Ultralytics [44]. Ενσωματώνουν προηγμένους αλγόριθμους, προσφέροντας την απαραίτητη υπολογιστική ισχύ για την επεξεργασία οπτικών δεδομένων σε πραγματικό χρόνο.

Η OpenCV χρησιμοποιήθηκε ως το βασικό εργαλείο για τη διαχείριση της ροής βίντεο από την εικονική κάμερα του Gazebo. Συγκεκριμένα, η βιβλιοθήκη είναι υπεύθυνη για τη λήψη των πλαισίων (frames), την προ-επεξεργασία τους, καθώς και τη δημιουργία και δυναμική ενημέρωση των παραθύρων απεικόνισης, όπου προβάλλονται τα αποτελέσματα της ανίχνευσης στον χρήστη.

Για το Ultralytics πρόκειται για ένα σύγχρονο, υψηλής απόδοσης πλαίσιο (framework) λογισμικού, σχεδιασμένο για την εκπαίδευση και την εφαρμογή μοντέλων υπολογιστικής όρασης. Το Ultralytics παρέχει μια βελτιστοποιημένη διεπαφή για την εκτέλεση του αλγορίθμου YOLO (You Only Look Once) [45], εξασφαλίζοντας υψηλή ταχύτητα επεξεργασίας ακόμη και σε συστήματα με περιορισμένους υπολογιστικούς πόρους. Η YOLO αποτελεί έναν από τους πιο προηγμένους αλγορίθμους ανίχνευσης αντικειμένων σε πραγματικό χρόνο. Σε αντίθεση με παλαιότερες μεθόδους που εξέταζαν μια εικόνα σε πολλαπλά στάδια, ο αλγόριθμος προσεγγίζει την ανίχνευση ως ένα ενιαίο πρόβλημα παλινδρόμησης, επεξεργαζόμενος ολόκληρο το πλαίσιο σε ένα μόνο πέρασμα από το νευρωνικό δίκτυο. Αυτό επιτρέπει στο όχημα να εντοπίζει και να ταξινομεί ταχύτατα ταυτόχρονα πολλαπλά αντικείμενα.

Στην εικόνα που ακολουθεί παρουσιάζετε γραφικά η λειτουργία του αλγορίθμου YOLO.



Εικόνα 44: Απεικόνιση λειτουργίας του αλγόριθμου YOLO [45]

Ο αλγόριθμος λειτουργεί αντιμετωπίζοντας τον εντοπισμό αντικειμένων ως ένα ενιαίο πρόβλημα παλινδρόμησης, επεξεργαζόμενος ολόκληρη την εικόνα σε ένα μόνο πέρασμα από το νευρωνικό δίκτυο. Αρχικά, χωρίζει την εικόνα σε ένα πλέγμα κελιών. Κάθε κελί προβλέπει ταυτόχρονα τα πλαίσια οριοθέτησης (bounding boxes), την εμπιστοσύνη για την ύπαρξη αντικειμένου και τις πιθανότητες κατηγορίας (π.χ. αν πρόκειται για άνθρωπο). Αυτή η παράλληλη επεξεργασία του επιτρέπει να είναι εξαιρετικά γρήγορος και κατάλληλος για εφαρμογές σε πραγματικό χρόνο, όπως ο εντοπισμός ατόμων από το ΜΕΑ, τροφοδοτώντας άμεσα τον χάρτη με σημασιολογική πληροφορία.

Οι αναλυτικές οδηγίες για την εγκατάσταση των πακέτων και των εξαρτήσεών τους παρατίθενται στο Παράρτημα Α.

5. Προγραμματισμός Ρομποτικού Συστήματος και Προσομοίωση MEA

Η αρχιτεκτονική λογισμικού του MEA αναπτύχθηκε σε Ubuntu μέσω του περιβάλλοντος ανάπτυξης VS Code και έχοντας ως κεντρική πλατφόρμα το οικοσύστημα του ROS 2. Για την υλοποίηση αξιοποιήθηκαν οι γλώσσες προγραμματισμού C++ και Python, ένας συνδυασμός που προσφέρει υπολογιστική απόδοση και ευελιξία. Η επικοινωνία μεταξύ των υποσυστημάτων βασίστηκε στη βιβλιοθήκη ROS Client Library (RCL). Συγκεκριμένα, χρησιμοποιήθηκε η βιβλιοθήκη rclcpp (C++) για τη δημιουργία κόμβων που απαιτούν υψηλή ταχύτητα εκτέλεσης και λειτουργία σε πραγματικό χρόνο, όπως ο έλεγχος offboard. Παράλληλα, αξιοποιήθηκε η rclpy (Python) για τον προγραμματισμό ανώτερου επιπέδου, τη διαχείριση των αρχείων εκκίνησης (launch files) και την ενσωμάτωση αλγορίθμων τεχνητής νοημοσύνης. Αυτή η υβριδική προσέγγιση επιτρέπει τη βέλτιστη διαχείριση των πόρων του συστήματος, διασφαλίζοντας ότι οι κρίσιμες για την πτήση λειτουργίες εκτελούνται με ελάχιστη καθυστέρηση, ενώ οι εφαρμογές ευφυΐας επωφελούνται από την επεκτασιμότητα των βιβλιοθηκών της Python.

Η υποδομή οργανώθηκε σε πακέτα με την εκτέλεση εντολών γραμμής (CLI), εξασφαλίζοντας μια τυποποιημένη δομή και διαλειτουργικότητα. Τα πακέτα αναπτύχθηκαν με την εξής σειρά:

Σειρά	Πακέτο	Σκοπός
1	uav_description	Περιγραφή ρομπότ
2	uav_simulation	Ρύθμιση εικονικού περιβάλλοντος
3	uav_offboard_control	Εξωτερικός έλεγχος του MEA
4	uav_navigation	Πλοήγηση με τηλεχειρισμό
5	uav_mapping	Χαρτογράφηση
6	uav_vision	Εφαρμογή αλγορίθμων τεχνητής όρασης
7	uav_bringup	Συντονισμός συστήματος

Πίνακας 28: Σειρά ανάπτυξης πακέτων

Ο πλήρης πηγαίος κώδικας καθώς και οι σχετικές παραμετροποιήσεις του συστήματος παρατίθενται στο Παράρτημα Γ. Στην ενότητα που ακολουθεί, παρουσιάζεται η αναλυτική περιγραφή της δομής και των λειτουργιών κάθε επιμέρους πακέτου λογισμικού που αναπτύχθηκε. Το κεφάλαιο ολοκληρώνεται με σχετικό διάγραμμα που απεικονίζει τη διασύνδεση και την επικοινωνία μεταξύ των κόμβων στο συνολικό σύστημα.

5.1 Δημιουργία Χώρου Εργασίας (workspace)

Ο χώρος εργασίας (workspace) με την ονομασία `uav_project_ws` αποτελεί τη θεμελιώδη δομή του έργου, εντός της οποίας φιλοξενείται ο υποφάκελος `src` για τον πηγαίο κώδικα των πακέτων. Η προετοιμασία του περιβάλλοντος πραγματοποιήθηκε μέσω της γραμμής εντολών (CLI), ακολουθώντας την τυποποιημένη ροή εργασίας του ROS 2.

Αρχικά, δημιουργήθηκαν οι απαραίτητοι κατάλογοι και στη συνέχεια εκτελέστηκε η αρχική μεταγλώττιση με τη χρήση του εργαλείου `colcon`, ώστε να δημιουργηθούν οι φάκελοι `build`, `install` και `log`. Οι σχετικές εντολές τερματικού, όπως αποτυπώνονται στην ακόλουθη εικόνα, εξασφάλισαν την ορθή δόμηση και αναγνώριση των πακέτων από το σύστημα.



```
$ project mkdir -p uav_project_ws/src
$ project cd uav_project_ws/
$ uav_project_ws ls
src
```

Εικόνα 45: Δημιουργία χώρου εργασίας για το έργο

5.1.1 Ανάπτυξη Πακέτου Περιγραφής Ρομπότ (`uav_description`)

Στο πρώτο στάδιο υλοποίησης αναπτύχθηκε το πακέτο `uav_description`, το οποίο περιλαμβάνει την πλήρη περιγραφή της γεωμετρίας, των συνδέσμων, των αρθρώσεων και των αδρανειακών χαρακτηριστικών του οχήματος. Τα στοιχεία αυτά είναι θεμελιώδη για την ακριβή αναπαράσταση και προσομοίωση του ΜΕΑ στο ρομποτικό περιβάλλον. Η διαδικασία ανάπτυξης ξεκίνησε εντός του φακέλου `src` του χώρου εργασίας, μέσω εντολών που εκτελέστηκαν σε τερματικό. Η μεθοδολογία αυτή ακολουθήθηκε συστηματικά και για την υλοποίηση όλων των υπόλοιπων πακέτων του συστήματος, διασφαλίζοντας ομοιογένεια στη δομή τους.

Στον επόμενο πίνακα παρατίθενται οι ενέργειες για τη δημιουργία και τη παραμετροποίηση του πακέτου. Μετά, παρουσιάζονται οι εντολές CLI που εκτελέστηκαν κατά τη δημιουργία της αρχικής δομής.

Βήμα	Ενέργεια	Περιγραφή
1	Δημιουργία πακέτου	Εκτέλεση της εντολής <code>ros2 pkg create</code> στον κατάλογο <code>src</code> με τις απαραίτητες εξαρτήσεις και του ονόματος του πακέτου.
2	Δημιουργία δομής	Μεταφορά φακέλων <code>urdf</code> και <code>meshes</code> στο πακέτο <code>uav_description</code> , που δημιουργήθηκαν μέσω του URDF Exporter του SolidWorks. Δημιουργία υποφακέλων <code>launch</code> και <code>rviz</code> , για την ενσωμάτωση αρχείων εκτέλεσης (<code>launch.py</code>) και αποθήκευσης ρυθμίσεων του RViz.
3	Τροποποίηση αρχείων για συμβατότητα με το ROS 2	Τα αρχεία του URDF Exporter είναι συμβατά μόνο με το ROS 1, απαιτώντας ανασχεδιασμό για το ROS 2. Οι παρεμβάσεις περιέλαβαν τη διόρθωση των διαδρομών των πλεγμάτων, την επικαιροποίηση της σύνταξης XML
4	Δημιουργία αρχείου εκκίνησης (Python)	Δημιουργία του αρχείου <code>display.launch.py</code> , για την εκκίνηση του κόμβου <code>robot_state_publisher</code> , ο οποίος επεξεργάζεται το αρχείο URDF για τη δημοσίευση των μετασχηματισμών (TF) του MEA. Παράλληλα, ενεργοποιεί το RViz με προκαθορισμένες ρυθμίσεις, επιτρέποντας την τρισδιάστατη οπτικοποίηση του μοντέλου και των δεδομένων των αισθητήρων σε πραγματικό χρόνο προσομοίωσης.
5	Ρύθμιση εξαρτήσεων	Ενημέρωση των αρχείων <code>package.xml</code> και <code>CMakeLists.txt</code> για την αναγνώριση των νέων φακέλων.
6	Μεταγλώττιση	Εκτέλεση <code>colcon build</code> για την ενσωμάτωση του πακέτου στο χώρο εργασίας.

Πίνακας 29: Διαδικασία εκτέλεσης για τη δημιουργία του πακέτου `uav_description`

```

$ uav project ws cd src/
$ src rosb2 pkg create uav_description
going to create a new package
package name: uav_description
destination directory: /mnt/e/TEMP/project/uav_project_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainers: ['mike «mikegtdo.todo»']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: []
Creating folder ./uav_description
creating ./uav_description/package.xml
creating source and include folder
creating folder ./uav_description/src
creating folder ./uav_description/include/uav_description
creating ./uav_description/CMakeLists.txt

[WARNING]: Unknown license 'TODO: License declaration'. This has been set in the package.xml, but no LICENSE file has been created.
It is recommended to use one of the ament license identifiers:
Apache-2.0
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
GPL-3.0-only
MIT
MIT-0
$ src cd ..
$ uav project ws colcon build
Starting >>> uav_description
Finished <<< uav_description [8.65s]

Summary: 1 package finished [8.83s]
$ uav project ws cd src/
$ src tree -L 3
.
├── uav_description
│   ├── CMakeLists.txt
│   ├── include
│   │   └── uav_description
│   ├── package.xml
│   └── src
└── 3 directories, 2 files

```

Εικόνα 46: Εκτέλεση εντολών στο CLI

Για την επαλήθευση της ορθότητας της περιγραφής του ρομπότ, χρησιμοποιήθηκε το βοηθητικό εργαλείο `urdf_to_graphviz`. Το εργαλείο αυτό αναλύει το αρχείο URDF και παράγει μια σχηματική αναπαράσταση της ιεραρχίας του MEA σε μορφή PDF, επιτρέποντας τον οπτικό έλεγχο της σύνδεσης μεταξύ των συνδέσμων (links) και των αρθρώσεων (joints).

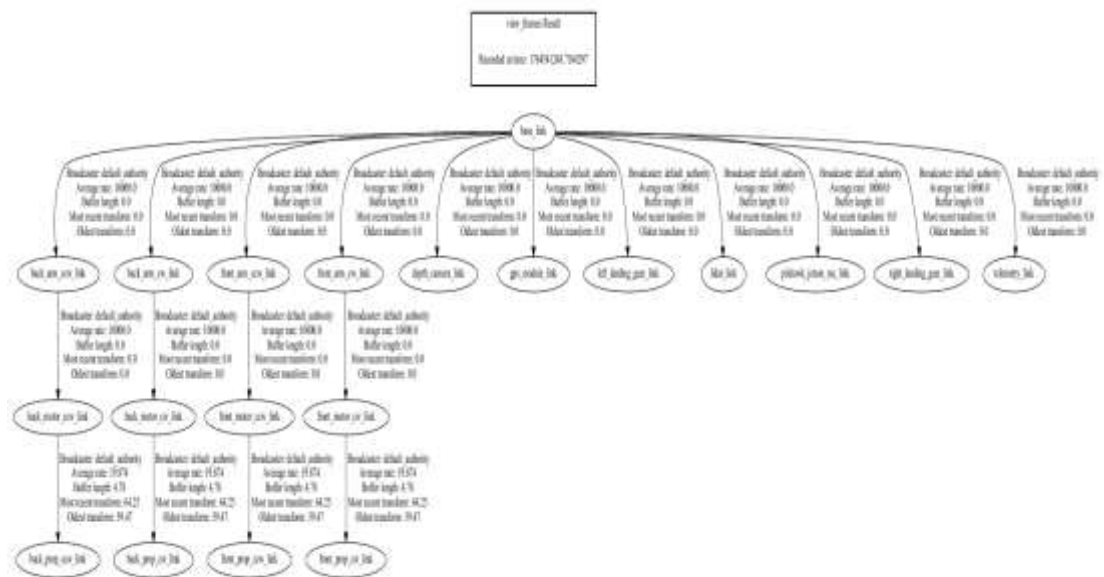
```

$ uav project ws cd src/uav_description/urdf/
$ urdf urdf to graphviz uav.urdf.xacro
WARNING: OUTPUT not given. This type of usage is deprecated!Usage:
gv 5 OUTPUT.pdf
Created file UAV_X650.gv
Created file UAV_X650.pdf
$ urdf evince UAV_X650.pdf

```

Εικόνα 47: Δημιουργία δομής μέσω `urdf_to_graphviz`

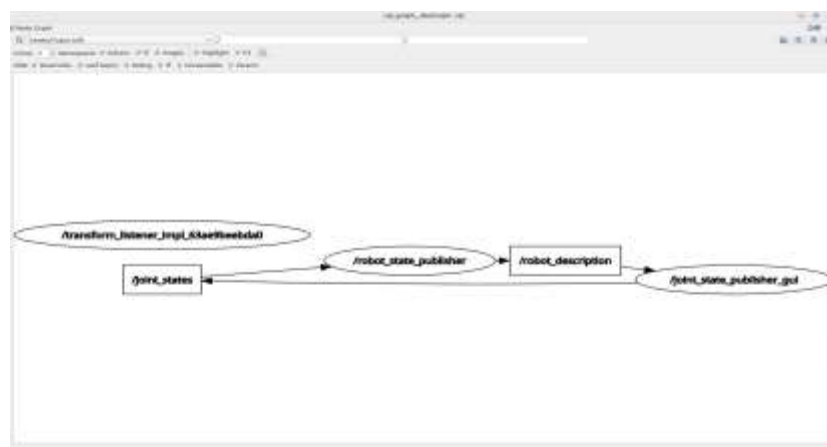
Στο παρακάτω Σχήμα αποτυπώνονται ιεραρχικά οι σχέσεις μεταξύ των συνδέσμων (links) και των αρθρώσεων (joints), με σημείο αναφοράς τον κύριο σύνδεσμο (base_link), το οποίο λειτουργεί ως ρίζα του μοντέλου.



Σχήμα 2: Γραφική αναπαράσταση δομής μέσω του εργαλείου urdf_to_graphviz

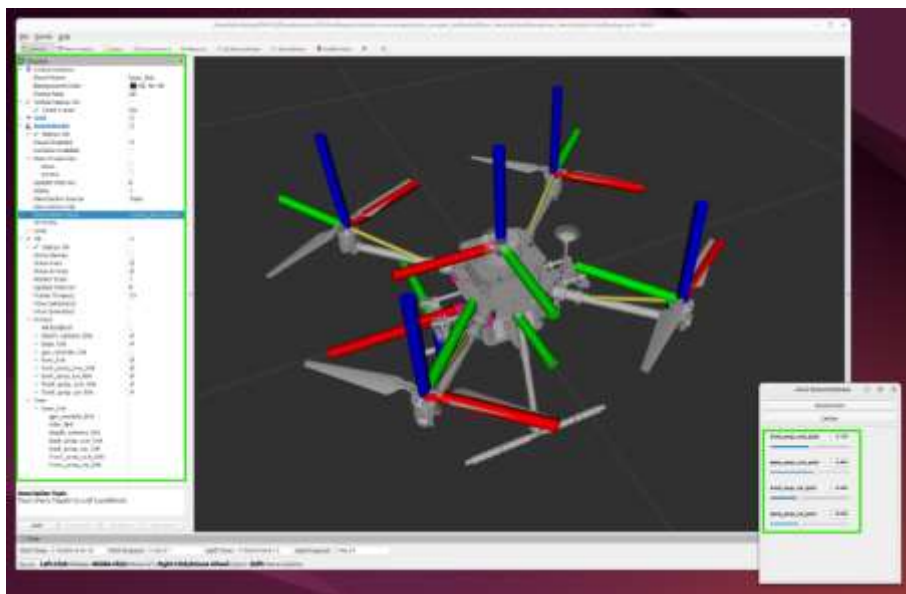
Μέσω του εργαλείου, επιβεβαιώθηκε η σωστή δομή της κινηματικής αλυσίδας του MEA (έλικες, LiDAR, GPS, κάμερα βάθους), διασφαλίζοντας ότι κάθε τμήμα του είναι σωστά προσαρτημένο στο γονικό του στοιχείο (parent link). Οι αναγραφόμενες τιμές διατυπώνουν τη μετατόπιση (xyz), τη περιστροφή (rpy) και επαληθεύουν τον ακριβή χωρικό προσανατολισμό των εξαρτημάτων.

Η επαλήθευση της σωστής λειτουργίας των κόμβων ROS 2 πραγματοποιήθηκε με τη χρήση του εργαλείου rqt. Αποτελείται από μια σειρά εργαλείων ελέγχου και παρέχει, μέσω του πρόσθετου rqt_graph, μια γραφική αναπαράσταση των ενεργών κόμβων και των μεταξύ τους επικοινωνιών (topics).



Εικόνα 48: Επικοινωνία των κόμβων σε γραφική αναπαράσταση rqt_graph

Η επιτυχής επικοινωνία μεταξύ των κόμβων, διασφαλίζει τον πλήρη συγχρονισμό των δεδομένων του MEA. Συγκεκριμένα, ο κόμβος `joint_state_publisher_gui` αντλεί πληροφορίες από το `/robot_description` και δημοσιεύει τις γωνίες των αρθρώσεων στο κανάλι `/joint_states`. Η πληροφορία αυτή ενημερώνεται δυναμικά μέσω των ρυθμιστικών (sliders) του γραφικού περιβάλλοντος RViz, επιτρέποντας τον άμεσο έλεγχο του μοντέλου. Ακολούθως, ο κόμβος `robot_state_publisher` επεξεργάζεται αυτά τα δεδομένα για την ενημέρωση του συστήματος TF (Transforms). Το TF αποτελεί μια κρίσιμη υποδομή του ROS 2 που διαχειρίζεται τη θέση και τον προσανατολισμό κάθε τμήματος του ρομπότ σε πραγματικό χρόνο. Με αυτόν τον τρόπο, κάθε αισθητήρας αναπαρίσταται με ακρίβεια ως προς ένα κοινό σύστημα αναφοράς, διευκολύνοντας την πλοήγηση και την απεικόνιση στο προσομοιωτικό περιβάλλον.



Εικόνα 49: Ανάλυση συστατικών του MEA στο πρόγραμμα RViz

Η ομαλή αυτή ροή πληροφορίας επιβεβαιώνει τη λειτουργική ετοιμότητα του πακέτου και θέτει πλέον τα θεμέλια για το επόμενο στάδιο, που αφορά την ενσωμάτωση του MEA στο περιβάλλον προσομοίωσης και τη διασύνδεσή του με τον ελεγκτή πτήσης.

5.1.2 Ανάπτυξη Πακέτου Προσομοίωσης (uav_simulation)

Η ολοκλήρωση του προαναφερθέντος πακέτου αποτέλεσε τη συνέχεια για την ενσωμάτωση του MEA στο εικονικό περιβάλλον. Για το σκοπό αυτό, αναπτύχθηκε το πακέτο uav_simulation, το οποίο λειτουργεί ως συνδετική γέφυρα μεταξύ του λογισμικού ελέγχου και του περιβάλλοντος προσομοίωσης. Ο κύριος ρόλος του πακέτου είναι η διαμόρφωση του εικονικού κόσμου και η διαχείριση της αμφίδρομης ροής δεδομένων μεταξύ του Gazebo και του ROS 2. Μέσω αυτού, εξασφαλίζεται ότι οι εντολές κίνησης μεταφέρονται στο προσομοιωμένο όχημα, ενώ ταυτόχρονα τα δεδομένα των εικονικών αισθητήρων (LiDAR, κάμερα, IMU) επιστρέφουν στο ROS 2 για περαιτέρω επεξεργασία.

Βήμα	Ενέργεια	Περιγραφή
1	Δημιουργία πακέτου	Εκτέλεση της εντολής <code>ros2 pkg create</code> στον κατάλογο <code>src</code> με τις απαραίτητες εξαρτήσεις και του ονόματος του πακέτου.
2	Δημιουργία δομής	Δημιουργία υποφακέλων <code>launch</code> , <code>config</code> και <code>worlds</code> .
3	Δημιουργία αρχείου ρύθμισης της γέφυρας	Σύνταξη του αρχείου <code>ros_gz_bridge.yaml</code> , για την παραμετροποίηση της γέφυρας επικοινωνίας μεταξύ Gazebo και ROS 2. Το αρχείο είναι καθοριστικό, καθώς επιτρέπει τη διασύνδεση και την ανταλλαγή δεδομένων μεταξύ των δύο διαφορετικών οντοτήτων.
4	Δημιουργία αρχείου εκκίνησης (Python)	Ανάπτυξη του αρχείου <code>ros_gz_bridge.launch.py</code> , για την ενεργοποίηση της γέφυρας επικοινωνίας και τη μετατροπή της οδομετρίας σε μετασχηματισμούς (<code>odom</code> προς <code>base_link</code>). Παράλληλα, εκτελείται ο κόμβος συγχρονισμού των αρθρώσεων, εξασφαλίζοντας τη ρεαλιστική κίνηση των ελίκων του μοντέλου στο RViz.
5	Τροποποίηση αρχείου XACRO	Ενσωμάτωση πρόσθετων (<code>plugins</code>) στο αρχείο <code>uav.gazebo.xacro</code> , για την ενεργοποίηση των αισθητήρων στο περιβάλλον προσομοίωσης και την επικοινωνία με ROS 2.
	Δημιουργία SDF αρχείου MEA	Δημιουργία του αρχείου <code>model.sdf</code> και ενσωμάτωση πρόσθετων, για αισθητήρες και κινητήρες του MEA. Αυτά ενεργοποιούν τα δεδομένα από τους αισθητήρες και παρέχουν ρεαλιστική περιστροφή των ελίκων του MEA στο εικονικό περιβάλλον.

6	Δημιουργία SDF αρχείου κόσμου	Δημιουργία του αρχείου baylands.sdf, για το ρεαλιστικό περιβάλλον Baylands στο Gazebo. Περιλαμβάνει ρυθμίσεις φυσικής, φωτισμού και τρισδιάστατα αντικείμενα, ενώ ενσωματώνει plugins για τη διαχείριση της βαρύτητας και των δυναμικών αλληλεπιδράσεων, διασφαλίζοντας ένα έγκυρο πλαίσιο δοκιμών για το MEA.
7	Ρύθμιση εξαρτήσεων	Ενημέρωση των αρχείων package.xml και CMakeLists.txt για την αναγνώριση των νέων φακέλων.
8	Μεταγλώττιση	Εκτέλεση colcon build για την ενσωμάτωση του πακέτου στο χώρο εργασίας.

Πίνακας 30: Διαδικασία εκτέλεσης για τη δημιουργία του πακέτου uav_simulation

Στις επόμενες Εικόνες παρουσιάζονται τα πρόσθετα που χρησιμοποιήθηκαν για τη ρεαλιστική προσομοίωση του MEA και των δεδομένων από τους αισθητήρες στο εικονικό περιβάλλον.

```
<plugin filename="gz-sim-physics-system" name="gz::sim::systems::Physics"/>
<plugin filename="gz-sim-user-commands-system" name="gz::sim::systems::UserCommands"/>
<plugin filename="gz-sim-scene-broadcaster-system" name="gz::sim::systems::SceneBroadcaster"/>
<plugin filename="gz-sim-sensors-system" name="gz::sim::systems::Sensors"/>
<plugin filename="gz-sim-imu-system" name="gz::sim::systems::Imu"/>
<plugin filename="gz-sim-navsat-system" name="gz::sim::systems::NavSat"/>
```

Εικόνα 50: Πρόσθετα στο αρχείο του εικονικού κόσμου

```
<plugin filename="gz-sim-contact-system" name="gz::sim::systems::Contact"/>
<plugin filename="gz-sim-imu-system" name="gz::sim::systems::Imu"/>
<plugin filename="gz-sim-air-pressure-system" name="gz::sim::systems::AirPressure"/>
<plugin filename="gz-sim-air-speed-system" name="gz::sim::systems::AirSpeed"/>
<plugin filename="gz-sim-apply-link-wrench-system" name="gz::sim::systems::ApplyLinkWrench"/>
<plugin filename="gz-sim-navsat-system" name="gz::sim::systems::NavSat"/>
<plugin filename="gz-sim-magnetometer-system" name="gz::sim::systems::Magnetometer"/>
```

Εικόνα 51: Πρόσθετα στο αρχείο του MEA

Έπειτα ακολούθησε μια δοκιμαστική εκτέλεση, για την επαλήθευση απεικόνισης του MEA στο εικονικό περιβάλλον, με στόχο τον έγκαιρο εντοπισμό γεωμετρικών αστοχιών ή προβλημάτων στις οπτικές ιδιότητες. Η λειτουργική ενσωμάτωση και η παραμετροποίηση της συμπεριφοράς του οχήματος υλοποιήθηκαν σε επόμενο στάδιο της ανάπτυξης.

Η παρακάτω εικόνα παρουσιάζει την οπτική σύγκριση του MEA μεταξύ του περιβάλλοντος σχεδίασης SolidWorks (δεξιά) και του προσομοιωτή Gazebo (αριστερά), επιβεβαιώνοντας την επιτυχή μεταφορά της γεωμετρίας και των υλικών.



Εικόνα 52: Σύγκριση MEA ανάμεσα σε SolidWorks και Gazebo



Εικόνα 53: Ρεαλιστική αναπαράσταση φωτισμού και λεπτομερειών του MEA

5.1.3 Ανάπτυξη Πακέτου Εξωτερικού Ελέγχου Πτήσης (uav_offboard_control)

Για τη διαχείριση των λειτουργιών πτήσης και τη διασύνδεση με τον ελεγκτή PX4, αναπτύχθηκε το πακέτο uav_offboard_control. Το πακέτο υλοποιεί εκτελέσιμα

αρχεία που έχουν δημιουργηθεί μέσω C++ και το οποίο αξιοποιεί τη στοίβα ελέγχου του PX4 για την επικοινωνία χαμηλού επιπέδου με τον ελεγκτή πτήσης και τους υπόλοιπους κόμβους του συστήματος.

Η διαδικασία της ανάπτυξης του πακέτου περιγράφεται στον πίνακα που ακολουθεί.

Βήμα	Ενέργεια	Περιγραφή
1	Δημιουργία πακέτου	Εκτέλεση της εντολής <code>ros2 pkg create</code> στον κατάλογο <code>src</code> με τις απαραίτητες εξαρτήσεις και του ονόματος του πακέτου.
2	Δημιουργία δομής	Δημιουργία υποφακέλων <code>launch</code> , <code>include</code> και <code>src</code> (<code>include</code> και <code>src</code> δημιουργούνται αυτόματα με την επιλογή μεταγλωττιστή <code>ament_cmake</code>).
3	Δημιουργία λογισμικού διαχείρισης ελεγκτή πτήσης (C++/Header)	Ανάπτυξη του λογισμικού <code>px4_ctrl.cpp</code> , για τον εξωτερικό έλεγχο (<code>offboard control</code>), το οποίο εξασφαλίζει τη διασύνδεση με τη στοίβα ελέγχου του PX4. Κεντρικό στοιχείο της εφαρμογής αποτελεί η Μηχανή Πεπερασμένων Καταστάσεων (FSM), η οποία διαχειρίζεται αυτόνομα τις διαδοχικές πτητικές φάσεις του MEA.
4	Δημιουργία λογισμικού διαχείρισης αρθρώσεων (C++/Header)	Ανάπτυξη του λογισμικού <code>px4_prop_js_publisher.cpp</code> , για τη δημοσίευση των καταστάσεων αρθρώσεων. Υλοποίηση κώδικα για την εξαγωγή δεδομένων κίνησης σε πραγματικό χρόνο, διασφαλίζοντας τη ρεαλιστική απεικόνιση της περιστροφής των ελίκων στο περιβάλλον RViz.
5	Δημιουργία αρχείου εκκίνησης εξωτερικού ελέγχου (Python)	Δημιουργία του αρχείου <code>offboard_ctrl.launch.py</code> , για την εκκίνηση κόμβου εξωτερικού ελέγχου του MEA.
6	Δημιουργία αρχείου εκκίνησης PX4-Autopilot (Python)	Δημιουργία του αρχείου <code>px4_autopilot.launch.py</code> , για την ταυτόχρονη εκκίνηση του MEA X650 και του αυτόματου πιλότου PX4 σε περιβάλλον προσομοίωσης SITL. Δεδομένου ότι η προεπιλεγμένη ρύθμιση εκτελεί το μοντέλο X500, πραγματοποιήθηκε παραμετροποίηση των αρχείων συστήματος για την ορθή φόρτωση του προσαρμοσμένου οχήματος (όπως αναλύεται στη συνέχεια).

7	Δημιουργία αρχείου εκκίνησης Micro-XRCE-DDS-Agent (Python)	Δημιουργία του αρχείου <code>xrce_dds_agent.launch.py</code> , για την εκκίνηση του πράκτορα, που λειτουργεί ως ενδιάμεσος (middleware), μετατρέποντας τα μηνύματα του πρωτοκόλλου uORB από τον αυτόματο πιλότο σε θέματα (topics) του ROS 2. Η λειτουργία του είναι απαραίτητη για την ανταλλαγή δεδομένων τηλεμετρίας και εντολών ελέγχου σε πραγματικό χρόνο μεταξύ του PX4 και των κόμβων του ROS 2.
8	Ρύθμιση εξαρτήσεων	Ενημέρωση των αρχείων <code>package.xml</code> και <code>CMakeLists.txt</code> για την αναγνώριση των νέων φακέλων.
9	Μεταγλώττιση	Εκτέλεση <code>colcon build</code> για την ενσωμάτωση του πακέτου στο χώρο εργασίας.

Πίνακας 31: Διαδικασία εκτέλεσης για τη δημιουργία του πακέτου `uav_offboard_ctrl`

Στη συνέχεια παρατίθενται οι καταστάσεις της Μηχανής Πεπερασμένων Καταστάσεων (Finite State Machine, FSM), συνοδευόμενες από τη λειτουργική τους περιγραφή, όπως αυτές υλοποιήθηκαν στον κώδικα του ελεγκτή:

1. INIT_MODE:
 - Η αρχική κατάσταση κατά την εκκίνηση του συστήματος, όπου ο κόμβος αναμένει το σήμα οπλισμού από τον χρήστη (`teleop armed`) για να ξεκινήσει η διαδικασία.
2. OFFBOARD_MODE:
 - Το στάδιο όπου αποστέλλεται αίτημα στον αυτόματο πιλότο για ανάληψη του ελέγχου από εξωτερικό λογισμικό (`offboard control`).
3. ARM_MODE:
 - Η φάση ενεργοποίησης (οπλισμού) των κινητήρων του MEA, εφόσον η μετάβαση σε Offboard mode ήταν επιτυχής.
4. TAKEOFF_MODE:
 - Η διαδικασία αυτόνομης ανόδου του σκάφους μέχρι το προκαθορισμένο ύψος ασφαλείας (5 μέτρα στο πλαίσιο της εργασίας).
5. HOVER_MODE:
 - Η κατάσταση σταθερής αιώρησης σε συγκεκριμένο σημείο, διατηρώντας το ύψος και τη θέση του οχήματος.

6. TELEOP_MODE:

- Η λειτουργία κατά την οποία το MEA δέχεται και εκτελεί εντολές κίνησης (με βάση της ταχύτητας) σε πραγματικό χρόνο από τον χειριστή.

7. RTL_MODE:

- Κατάσταση ασφαλείας κατά την οποία το MEA διακόπτει την τρέχουσα αποστολή και επιστρέφει αυτόματα στο προηγούμενο σημείο απογείωσης. Ενεργοποιείται είτε χειροκίνητα σε περίπτωση ανάγκης είτε αυτόματα αν εντοπιστεί χαμηλή στάθμη μπαταρίας ή απώλεια σύνδεσης.

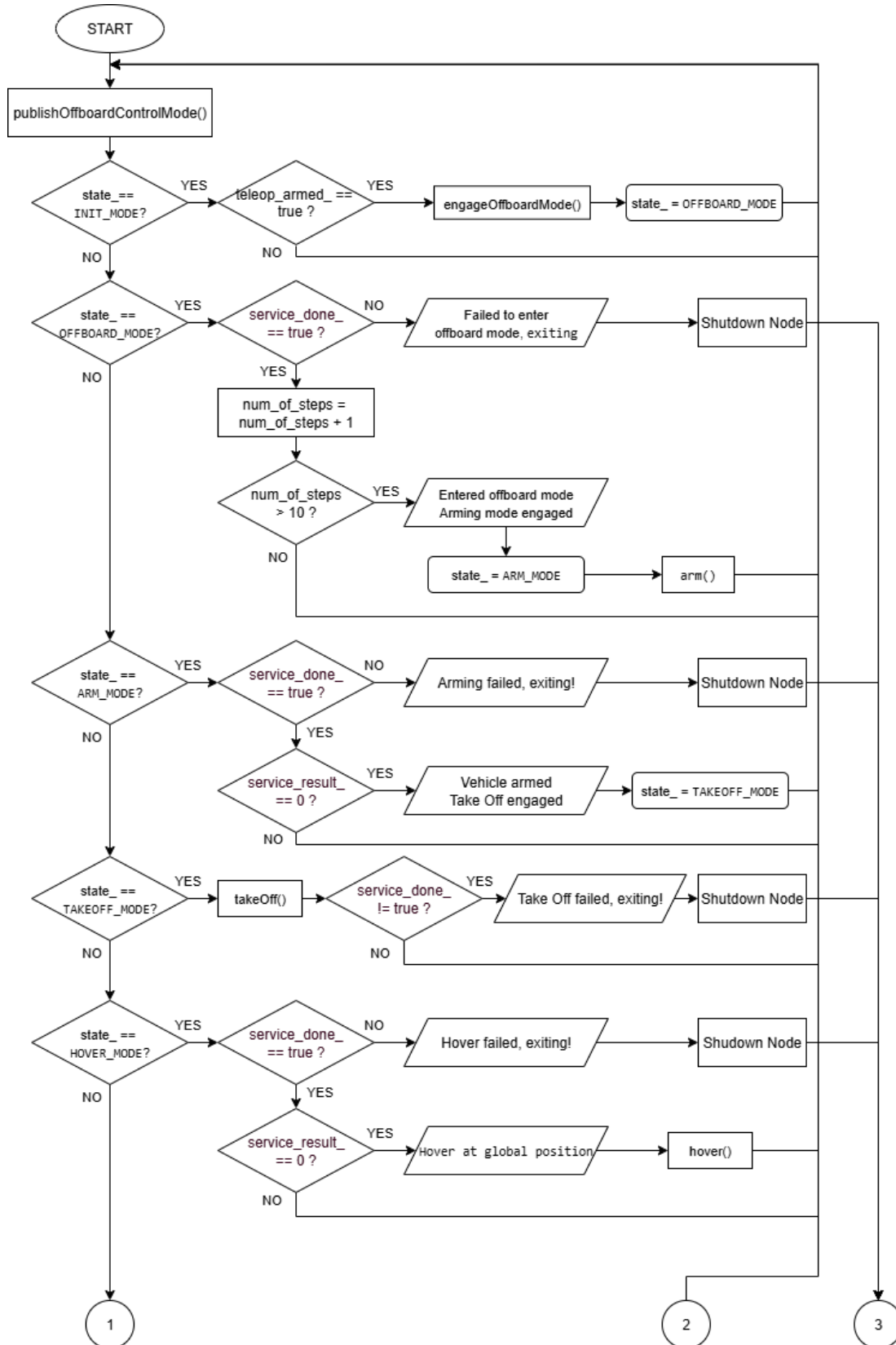
8. LAND_MODE:

- Η διαδικασία ελεγχόμενης καθόδου και προσγείωσης, η οποία διαχειρίζεται πλήρως από τον αυτόματο πιλότο για μέγιστη ακρίβεια.

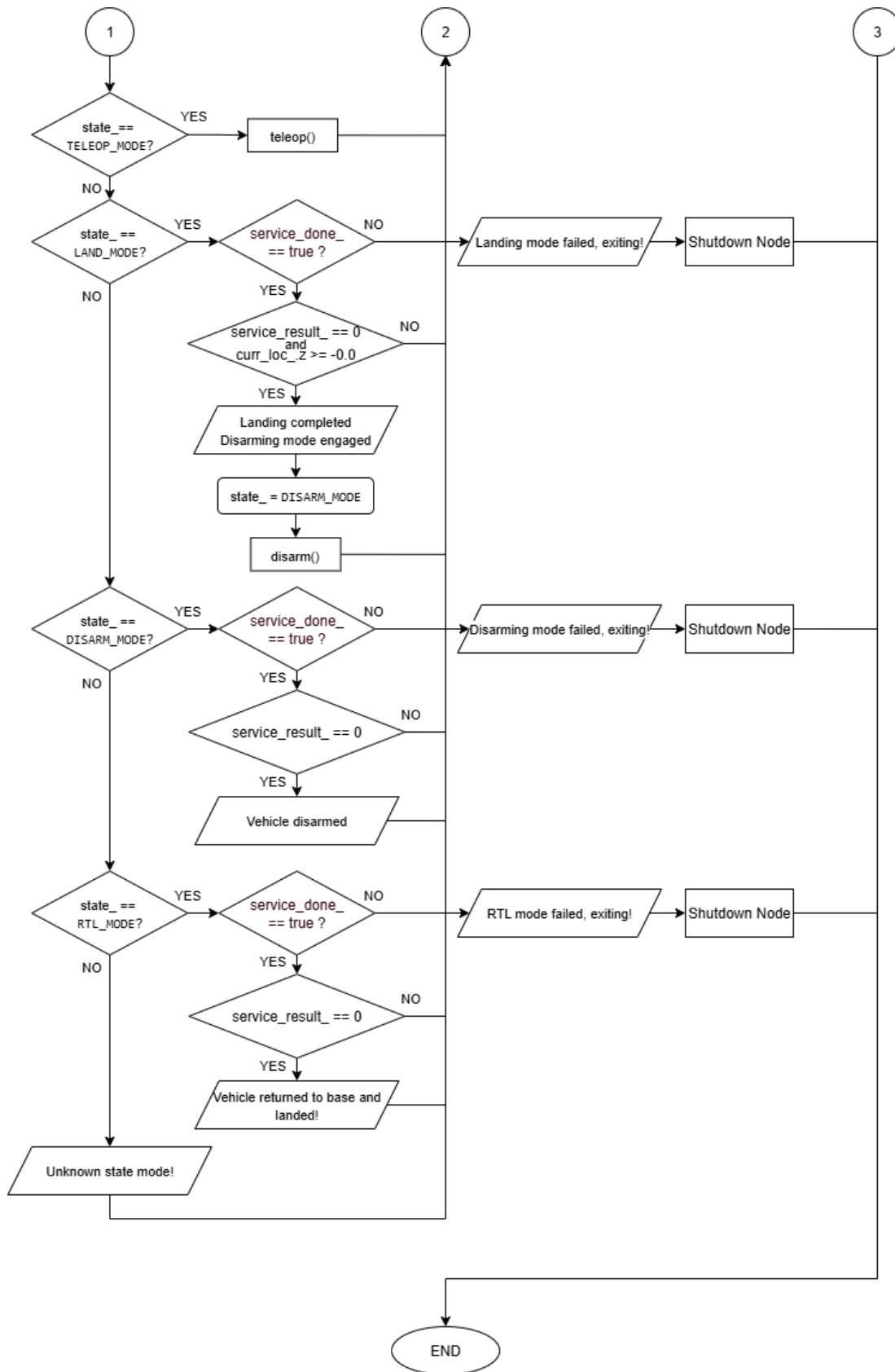
9. DISARM_MODE:

- Η τελική κατάσταση όπου απενεργοποιούνται οι κινητήρες αμέσως μετά την επιβεβαίωση της επαφής με το έδαφος.

Στα παρακάτω διαγράμματα απεικονίζεται η ροή της μηχανής καταστάσεων, η οποία εκκινεί από την αρχική κατάσταση και εξελίσσεται βάσει των επιβεβαιώσεων από τον αυτόματο πιλότο PX4 και των εντολών του χρήστη. Λόγω του μεγάλου όγκου του, το διάγραμμα διαχωρίζεται σε δύο τμήματα για την καλύτερη αναγνωσιμότητά του.



Σχήμα 3: Ροή εκτέλεσης μηχανής πεπερασμένων καταστάσεων (1)



Σχήμα 4: Ροή εκτέλεσης μηχανής πεπερασμένων καταστάσεων (2)

Για τη διατήρηση του ελέγχου σε κατάσταση Offboard, απαιτείται η συνεχής και αδιάλειπτη δημοσίευση των μηνυμάτων OffboardControlMode και TrajectorySetpoint. Στο μήνυμα OffboardControlMode υφίσταται η απαραίτητη ρύθμιση που καθορίζει το είδος του ελέγχου που θα ασκεί το λογισμικό στο MEA (για παράδειγμα, έλεγχος θέσης, ταχύτητας ή στάσης), ενεργοποιώντας τις αντίστοιχες εισόδους του αυτόματου πιλότου. Παράλληλα, μέσω του TrajectorySetpoint, αποστέλλονται τα δεδομένα των σημείων ρύθμισης της τροχιάς (setpoints), όπως οι επιθυμητές συντεταγμένες ή οι διανυσματικές ταχύτητες κίνησης. Ο αυτόματος πιλότος PX4 απαιτεί την αποστολή αυτών των δεδομένων σε τακτά διαστήματα, συγκεκριμένα κάθε 20 - 100 ms. Οποιαδήποτε διακοπή στη ροή ή καθυστέρηση (blocking) του κώδικα μπορεί να οδηγήσει σε απώλεια του ελέγχου και αυτόματη έξοδο από τη λειτουργία Offboard για λόγους ασφαλείας, καθώς το σύστημα θεωρεί ότι ο εξωτερικός ελεγκτής έπαψε να αποκρίνεται.

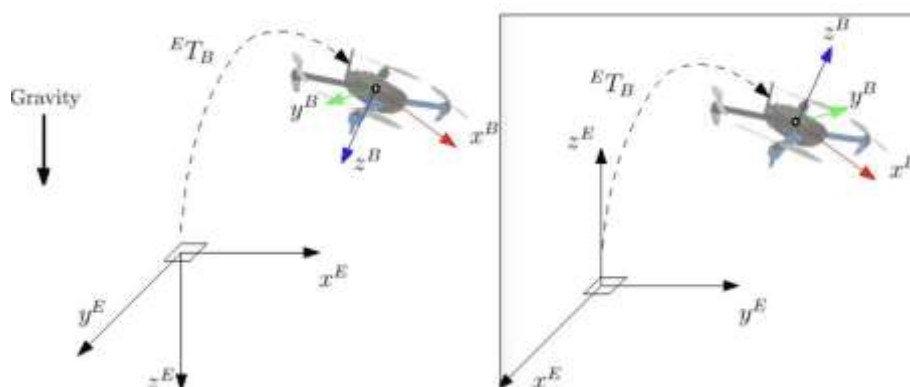
Επισημαίνεται ότι οι διαδικασίες της προσγείωσης και της επιστροφής στη βάση (RTL) αναλαμβάνονται εξ ολοκλήρου από τους εσωτερικούς αλγορίθμους του PX4 και όχι από το εξωτερικό λογισμικό. Ο ελεγκτής αποστέλλει, ανάλογα με την περίπτωση, είτε την εντολή VEHICLE_CMD_NAV_LAND (για άμεση προσγείωση) είτε την εντολή VEHICLE_CMD_NAV_RETURN_TO_LAUNCH (για αυτόνομη επιστροφή στο σημείο απογείωσης). Από εκείνο το σημείο και έπειτα, ο αυτόματος πιλότος διαχειρίζεται αυτόνομα την κάθοδο και την επαφή με το έδαφος, διασφαλίζοντας τη μέγιστη δυνατή ασφάλεια της πτήσης.

Συνεχίζοντας την ανάλυση των τεχνικών προδιαγραφών, η ορθή διαχείριση των συστημάτων αναφοράς είναι απαραίτητη για τη διαλειτουργικότητα μεταξύ των διαφορετικών υποσυστημάτων. Στον τομέα των MEA χρησιμοποιούνται δύο βασικές συμβάσεις για τον προσδιορισμό των συντεταγμένων, οι οποίες επηρεάζουν άμεσα την οπτικοποίηση και την κίνηση:

Πλαίσιο	PX4	ROS 2
Σώμα (Body)	FRD (X Forward, Y Right, Z Down)	FLU (X Forward, Y Left, Z Up), usually named base_link
Γη	FRD or NED (X North, Y East, Z Down)	FLU or ENU (X East, Y North, Z Up),

(World)	Z Down)	with the naming being odom or map
Χρήση (Use)	Χρησιμοποιείται από το PX4 Control Stack. Ο άξονας Z δείχνει προς τα κάτω.	Χρησιμοποιείται από το Gazebo και το ROS 2. Σε αυτό, ο άξονας Z έχει θετική φορά προς τα πάνω, γεγονός που διευκολύνει τις γραφικές αναπαραστάσεις και τη συμβατότητα με τα πρότυπα του ROS 2.

Πίνακας 32: Συστήματα αναφοράς PX4 και ROS 2



Εικόνα 54: Γραφική αναπαράσταση πλαισίων αναφοράς NED (αριστερά) και ENU (δεξιά) [46]

Η διαφορά αυτή έχει ως αποτέλεσμα την αντιστροφή της οπτικής περιστροφής. Για παράδειγμα, μια αριστερόστροφη περιστροφή (CCW) στο σύστημα NED του PX4 εμφανίζεται ως δεξιόστροφη (CW) στο περιβάλλον ENU του Gazebo. Αυτή η απόκλιση προκύπτει αποκλειστικά από την αντίθετη κατεύθυνση του κατακόρυφου άξονα και λαμβάνεται υπόψη κατά τη μετατροπή των δεδομένων στον κώδικα.

Συγκεκριμένα, στον κώδικα `px4_ctrl.cpp`, οι μετασχηματισμοί αυτοί εφαρμόζονται στις εξής περιπτώσεις:

- **Μετατροπή Ύψους:** Το ύψος απογείωσης ορίζεται ως -5.0 στο σύστημα NED για να αντιστοιχεί σε 5 μέτρα πάνω από το έδαφος στο σύστημα ROS/Gazebo.
- **Μετατροπή Ταχυτήτων:** Στη συνάρτηση `velocityCallback`, η ταχύτητα στον άξονα Z αντιστρέφεται για να εναρμονιστεί η εντολή "Up" του ROS με την εντολή "Down" του PX4.
- **Υπολογισμός Yaw:** Ο υπολογισμός της γωνίας εκτροπής (`true_yaw`) περιλαμβάνει αρνητικό πρόσημο για τη διόρθωση της φοράς περιστροφής μεταξύ των δύο συστημάτων.

Μια επιπλέον κρίσιμη τεχνική λεπτομέρεια αφορά τη διαφοροποίηση στην αρίθμηση των κινητήρων μεταξύ των δύο συστημάτων. Το Gazebo, ακολουθώντας τις συμβάσεις της γλώσσας C++, χρησιμοποιεί μηδενική αρίθμηση (zero-based indexing), αναγνωρίζοντας τους κινητήρες ως Motor 0 έως 3. Αντιθέτως, το PX4 χρησιμοποιεί αρίθμηση που ξεκινά από τη μονάδα (Motor 1 έως 4), η οποία είναι ευθυγραμμισμένη με τη φυσική συνδεσμολογία του υλικού.

Η διατήρηση της σωστής αντιστοίχισης (mapping) μεταξύ των αναγνωριστικών των κινητήρων στο αρχείο SDF και των παραμέτρων του PX4 είναι ζωτικής σημασίας. Οποιαδήποτε αναντιστοιχία στις εντολές των κινητήρων οδηγεί σε εσφαλμένη απόκριση στις ροπές, προκαλώντας αστάθεια στον άξονα εκτροπής (yaw) ή ολοκληρωτική αποτυχία απογείωσης. Με την οριστικοποίηση αυτών των ρυθμίσεων, το όχημα απέκτησε την τελική του οπτική μορφή, έχοντας διασφαλίσει ταυτόχρονα την ορθή λειτουργική του συμπεριφορά στο εικονικό περιβάλλον.

5.1.4 Ανάπτυξη Πακέτου Πλοήγησης υπό Τηλεχειρισμό (uav_navigation)

Με στόχο την υλοποίηση της κίνησης και την πλοήγηση του MEA, αναπτύχθηκε το πακέτο uav_navigation, το οποίο ενσωματώνει διαδικασίες τηλεκατευθυνόμενου χειρισμού μέσω πληκτρολογίου. Η αρχιτεκτονική του πακέτου σχεδιάστηκε με πρόβλεψη για τη μελλοντική ενσωμάτωση αλγορίθμων αυτόνομης πλοήγησης σε μεταγενέστερο στάδιο, πέραν των ορίων της παρούσας διπλωματικής εργασίας. Το πακέτο υιοθετεί μια υβριδική προσέγγιση, συνδυάζοντας την υπολογιστική ισχύ της C++ με την ευελιξία της Python. Η επιλογή αυτή επιτρέπει τον βέλτιστο προγραμματισμό, καθώς η C++ διαχειρίζεται τις διεργασίες πραγματικού χρόνου και τη σταθερότητα της επικοινωνίας με τον αυτόματο πιλότο, ενώ η Python διευκολύνει την ταχεία ανάπτυξη περιφερειακών λειτουργιών, όπως η διασύνδεση με το πληκτρολόγιο και ο πειραματισμός με αλγορίθμους υψηλότερου επιπέδου.

Ακολουθεί η αναλυτική περιγραφή των βημάτων υλοποίησης, κατ' αντιστοιχία με τη δομή που ακολουθήθηκε στα προηγούμενα πακέτα:

Βήμα	Ενέργεια	Περιγραφή
1	Δημιουργία πακέτου	Εκτέλεση της εντολής <code>ros2 pkg create</code> στον κατάλογο <code>src</code> με τις απαραίτητες εξαρτήσεις και του

		ονόματος του πακέτου.
2	Δημιουργία δομής	Δημιουργία υποφακέλων config, launch, scripts, και include, src.
3	Δημιουργία λογισμικού τηλεχειρισμού (Python)	Ανάπτυξη του λογισμικού keyboard_teleop.py, για τον τηλεχειρισμό με το πληκτρολόγιο.
4	Ρύθμιση εξαρτήσεων	Ενημέρωση των αρχείων package.xml και CMakeLists.txt για την αναγνώριση των νέων φακέλων και την υποστήριξη υβριδικού προγραμματισμού.
5	Μεταγλώττιση	Εκτέλεση colcon build για την ενσωμάτωση του πακέτου στο χώρο εργασίας.

Πίνακας 33: Διαδικασία εκτέλεσης για τη δημιουργία του πακέτου uav_navigation

Στον κατάλογο που έπεται παρουσιάζεται αναλυτικά το μενού λειτουργιών του συστήματος τηλεχειρισμού, όπου περιγράφετε η αντιστοίχιση κάθε πλήκτρου με την αντίστοιχη κίνηση ή εντολή. Παρακάτω ακολουθεί εικόνα που παρουσιάζει τις κινήσεις κύλισης (Roll), κλίσης (Pitch) και εκτροπής γωνίας (Yaw) στο περιβάλλον προσομοίωσης:

- **Κίνηση και Προσανατολισμός:** Τα πλήκτρα w και s ελέγχουν την άνοδο και κάθοδο (Up/Down), ενώ τα a και d τη μεταβολή της εκτροπής γωνίας (Yaw Left/Right).
- **Μεταφορική Κίνηση:** Τα βέλη του πληκτρολογίου αναλαμβάνουν τον έλεγχο των αξόνων κλίσης (Pitch Forward/Backward) και κύλισης (Roll Left/Right).
- **Ειδικές Λειτουργίες:** Το πλήκτρο Space χρησιμοποιείται για τον οπλισμό ή αφοπλισμό των κινητήρων (Arm/Disarm), το πλήκτρο R ενεργοποιεί την αυτόματη επιστροφή στη βάση (Return to Base/RTL), ενώ το πλήκτρο ESC τερματίζει την εφαρμογή.

Η διασύνδεση αυτή επιτρέπει στον χρήστη να στέλνει δεδομένα ταχύτητας σε πραγματικό χρόνο, οι οποίες μεταφράζονται στον κώδικα px4_ctrl.cpp σε κατάλληλα εντολές για τον αυτόματο πιλότο. Συγκεκριμένα, η λήψη της εντολής για την

προσγείωση ή επιστροφή στη βάση μέσω του πληκτρολογίου, πυροδοτεί τη μετάβαση της Μηχανής Πεπερασμένων Καταστάσεων στην αντίστοιχη λειτουργία.



Εικόνα 55: Roll (αριστερά), Pitch (κέντρο), Yaw (δεξιά)

5.1.5 Ανάπτυξη Πακέτου Χαρτογράφησης (uav_mapping)

Η χαρτογράφηση σε εσωτερικούς και εξωτερικούς χώρους αποτελεί μια θεμελιώδη διαδικασία στη ρομποτική, επιτρέποντας τη συλλογή χωρικών δεδομένων από πολλαπλές γωνίες θέασης. Στην παρούσα εργασία, η διαδικασία εκτελείται αποκλειστικά σε πραγματικό χρόνο (online mapping), προσφέροντας άμεση ανατροφοδότηση για τη δομή του περιβάλλοντος.

Στη συνέχεια περιγράφεται η δημιουργία του πακέτου uav_mapping και έπειτα θα ακολουθήσει η περιγραφή της διαδικασίας και πώς επιτυγχάνεται η αναπαράσταση του χώρου σε δισδιάστατη και τρισδιάστατη μορφή. Το πακέτο αναπτύχθηκε με σκοπό τη συγκέντρωση όλων των απαραίτητων ρυθμίσεων και αρχείων εκκίνησης για τη διαδικασία, ή οποία περιλαμβάνει:

Βήμα	Ενέργεια	Περιγραφή
1	Δημιουργία πακέτου	Εκτέλεση της εντολής <code>ros2 pkg create</code> στον κατάλογο <code>src</code> με τις απαραίτητες εξαρτήσεις και του ονόματος του πακέτου.
2	Δημιουργία δομής	Δημιουργία υποφακέλων <code>rviz</code> , <code>launch</code> .
3	Δημιουργία αρχείο εκκίνησης χαρτογράφησης (Python)	Δημιουργία του αρχείου <code>rtabmap_online.launch.py</code> , το οποίο αποτελεί τον κεντρικό πυλώνα για την εκτέλεση της online χαρτογράφησης και την οπτικοποίηση των δεδομένων στο RViz.

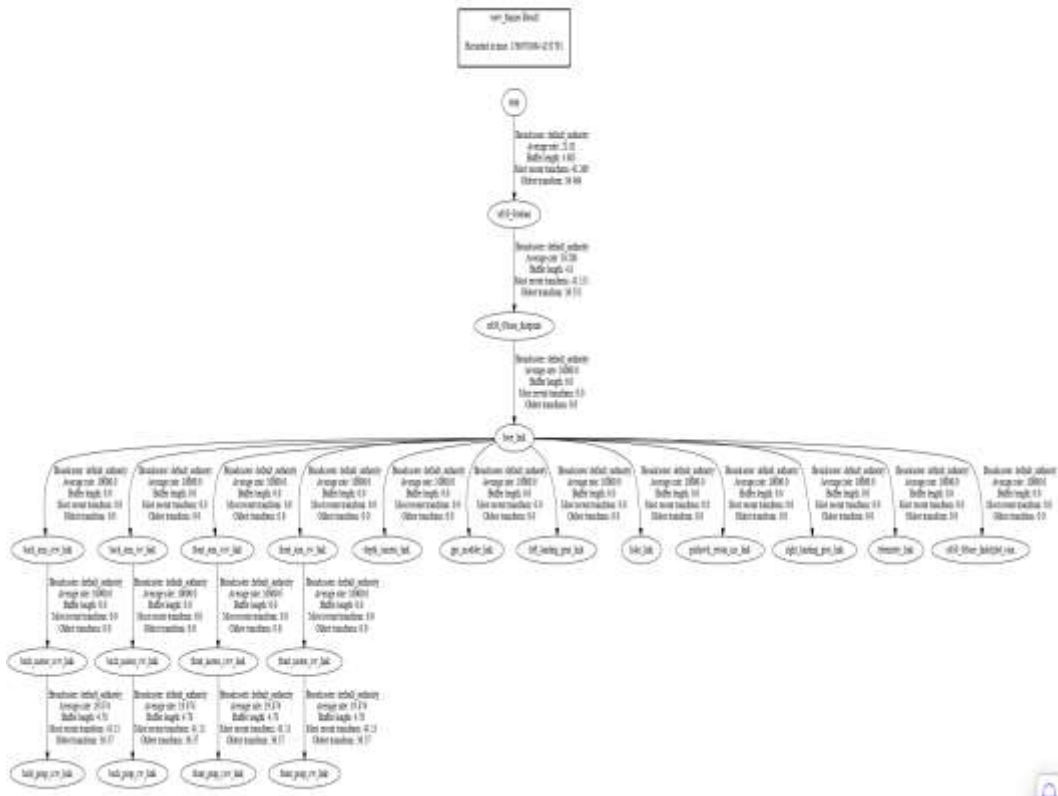
4	Ρύθμιση εξαρτήσεων	Ενημέρωση του αρχείου package.xml με τις απαραίτητες εξαρτήσεις για τη λειτουργία του κώδικα. Σημειώνεται ότι στο συγκεκριμένο πακέτο δεν υφίσταται το αρχείο CMakeLists.txt, καθώς η ανάπτυξη βασίζεται αποκλειστικά σε Python, χρησιμοποιώντας το σύστημα δόμησης ament_python.
5	Μεταγλώττιση	Εκτέλεση colcon build για την ενσωμάτωση του πακέτου στο χώρο εργασίας.

Πίνακας 34: Διαδικασία εκτέλεσης για τη δημιουργία του πακέτου uav_mapping

Για την online χαρτογράφηση χρησιμοποιείται το πακέτο RTAB-Map, το οποίο παραμετροποιείται μέσω του αρχείου rtabmap_online.launch.py. Το πακέτο εφαρμόζει τις ακόλουθες λειτουργίες:

- Συγχρονισμός Αισθητήρων: Το σύστημα συνδυάζει δεδομένα από κάμερα βάθους (RGB-D) μέσω των καναλιών επικοινωνίας /camera/image και /camera/depth/image, εξασφαλίζοντας πλούσια οπτική πληροφορία.
- Χρήση 2D LiDAR: Για τη βελτίωση της ακρίβειας και την ανίχνευση εμποδίων, ενσωματώνονται δεδομένα από αισθητήρα 2D LiDAR μέσω του καναλιού /scan.
- Διαχείριση Οδομετρίας: Η κίνηση του MEA παρακολουθείται online μέσω της οδομετρίας που παρέχει το Gazebo στο κανάλι /odom.
- Στατικοί Μετασχηματισμοί (TF): Διασφαλίζεται η ορθή γεωμετρική συσχέτιση μεταξύ του σώματος του MEA (base_link), της κάμερας (rgb_d_cam) και του base_footprint για την ακριβή τοποθέτηση των δεδομένων στον χάρτη.

Στο παρακάτω Σχήμα παρουσιάζεται η δομή των TF's, η οποία είναι καθοριστική για τη σωστή εφαρμογή της χαρτογράφησης. Το δέντρο των μετασχηματισμών απεικονίζει τη συνεχή ροή δεδομένων και τη γεωμετρική συσχέτιση μεταξύ των διαφορετικών πλαισίων αναφοράς:



Σχήμα 5: Γραφική αναπαράσταση της αναθεωρημένης δομής μέσω του εργαλείου urdf_to_graphviz

Η προσθήκη των συγκεκριμένων στατικών μετασχηματισμών (TFs) πριν από το base_link κρίθηκε απαραίτητη για να γεφυρωθεί το χάσμα μεταξύ των διαφορετικών συστημάτων ονοματοδοσίας και προσανατολισμού που χρησιμοποιούν το Gazebo και το RTAB-Map. Μέσω αυτών, το σύστημα ορίζει τη γεωμετρική σχέση του σώματος του MEA με το έδαφος και, κυρίως, ευθυγραμμίζει την κάμερα βάθους με το κεντρικό πλαίσιο αναφοράς του σκάφους. Χωρίς αυτές τις ρυθμίσεις, τα δεδομένα από το LiDAR και την κάμερα δεν θα μπορούσαν να συσχετιστούν σωστά με την κίνηση του οχήματος, καθιστώντας αδύνατη την ακριβή τοποθέτηση των αντικειμένων στον χάρτη κατά τη διάρκεια της online χαρτογράφησης.

5.1.6 Ανάπτυξη Πακέτου Τεχνητής Όρασης (uav_vision)

Η ενσωμάτωση δυνατοτήτων υπολογιστικής όρασης επιτρέπει στο MEA να εκτελεί σύνθετες εργασίες, όπως ο εντοπισμός αντικειμένων, η παρακολούθηση στόχων και η οπτική επιτήρηση. Μέσω της επεξεργασίας δεδομένων σε πραγματικό χρόνο, το όχημα μετατρέπεται από ένα απλό τηλεκατευθυνόμενο μέσο σε ένα «έξυπνο»

σύστημα, ικανό να κατηγοριοποιεί οντότητες στο οπτικό του πεδίο και να παρέχει κρίσιμες πληροφορίες στον χειριστή χωρίς καθυστέρηση.

Στο πλαίσιο αυτό, αναπτύχθηκε το πακέτο `uav_vision`, το οποίο συγκεντρώνει αλγόριθμους τεχνητής όρασης, ενώ η αρχιτεκτονική του είναι πλήρως επεκτάσιμη για την ενσωμάτωση μελλοντικών λειτουργιών. Η κεντρική εφαρμογή του πακέτου βασίζεται στον αλγόριθμο YOLOv8 (You Only Look Once), έναν από τους πιο σύγχρονους και αποδοτικούς αλγόριθμους ανίχνευσης αντικειμένων παγκοσμίως. Ο YOLOv8 αποτελεί την εξέλιξη των αρχιτεκτονικών βαθιάς μάθησης (Deep Learning) και διαφοροποιείται από άλλους αλγόριθμους λόγω της ταχύτητας και της ακρίβειάς του. Οι βασικές ιδιότητες του αλγόριθμου είναι:

- **Single-Stage Detection:** Σε αντίθεση με παλαιότερες μεθόδους που εξέταζαν την εικόνα σε δύο στάδια, ο YOLOv8 επεξεργάζεται ολόκληρο το πλαίσιο (frame) της εικόνας με ένα μόνο πέρασμα από το νευρωνικό δίκτυο. Αυτό επιτρέπει την επίτευξη εξαιρετικά υψηλών ρυθμών ανανέωσης (FPS), καθιστώντας τον ιδανικό για χρήση σε MEA.
- **Spatial Attention:** Ο αλγόριθμος χωρίζει την εικόνα σε ένα πλέγμα (grid) και για κάθε κελί προβλέπει ταυτόχρονα τα πλαίσια οριοθέτησης (bounding boxes) και τις πιθανότητες για την κλάση του αντικειμένου που περιέχεται σε αυτά.
- **Anchor-Free Detection:** Η έκδοση 8 εισάγει μια προσέγγιση χωρίς προκαθορισμένα "anchors", γεγονός που του επιτρέπει να εντοπίζει με μεγαλύτερη ακρίβεια αντικείμενα διαφορετικών μεγεθών και σχημάτων, μειώνοντας ταυτόχρονα την υπολογιστική πολυπλοκότητα.

Η διαδικασία για την υλοποίηση και ενσωμάτωση του πακέτου ακολούθησε τα εξής στάδια:

Βήμα	Ενέργεια	Περιγραφή
1	Δημιουργία πακέτου	Εκτέλεση της εντολής <code>ros2 pkg create</code> στον κατάλογο <code>src</code> με τις απαραίτητες εξαρτήσεις και του ονόματος του πακέτου.
2	Δημιουργία δομής	Δημιουργία φακέλου <code>models</code> , για την αποθήκευση του νευρωνικού δικτύου YOLOv8n (<code>yolov8n.pt</code>).

3	Δημιουργία λογισμικού εντοπισμού ατόμων (Python)	Ανάπτυξη λογισμικού object_detection.py, το οποίο αξιοποιεί τη βιβλιοθήκη Ultralytics για την ενσωμάτωση του αλγορίθμου YOLOv8. Ο κώδικας επιτρέπει τον εντοπισμό και την ταξινόμηση αντικειμένων σε πραγματικό χρόνο, επεξεργαζόμενος τη ροή δεδομένων από την κάμερα του MEA.
4	Ρύθμιση εξαρτήσεων	Ενημέρωση του αρχείου package.xml με τις απαραίτητες εξαρτήσεις για τη λειτουργία του κώδικα.
5	Μεταγλώττιση	Εκτέλεση colcon build για την ενσωμάτωση του πακέτου στο χώρο εργασίας.

Πίνακας 35: Διαδικασία εκτέλεσης για τη δημιουργία του πακέτου uav_vision

5.1.7 Ανάπτυξη Πακέτου Εκκίνησης Συστήματος (uav_bringup)

Η ανάγκη για την ταυτόχρονη εκτέλεση πολλαπλών υποσυστημάτων σε διακριτά τερματικά οδήγησε στη δημιουργία του πακέτου uav_bringup, το οποίο συγκεντρώνει όλες τις απαραίτητες ρυθμίσεις για το περιβάλλον ROS 2. Κεντρικό ρόλο στην αυτοματοποίηση της διαδικασίας παίζει το αρχείο open_ros_terminals.sh, το οποίο αποτελεί τον ενορχηστρωτή της προσομοίωσης, εκκινώντας ταυτόχρονα όλες τις κρίσιμες διεργασίες σε πέντε ξεχωριστά παράθυρα τερματικού.

Η δομή και η συνδυαστική λειτουργία του πακέτου αναλύονται ως εξής:

- Τερματικό 1 (start_sitl.launch.py): Εκκινεί τον πυρήνα της προσομοίωσης, φορτώνοντας τον αυτόματο πιλότο PX4 και το μοντέλο του MEA στο περιβάλλον SITL.
- Τερματικό 2 (xrce_dds_agent.launch.py): Ενεργοποιεί τον πράκτορα Micro-XRCE-DDS για τη διασφάλιση της επικοινωνίας χαμηλού επιπέδου μεταξύ του PX4 και του ROS 2.
- Τερματικό 3 (ros_gz_bridge.launch.py): Δημιουργεί τη γέφυρα επικοινωνίας για τη μεταφορά δεδομένων αισθητήρων και εντολών μεταξύ του ROS 2 και του προσομοιωτή Gazebo.
- Τερματικό 4 (rtabmap_online.launch.py): Εκκινεί τη διαδικασία της online χαρτογράφησης και του SLAM σε πραγματικό χρόνο.

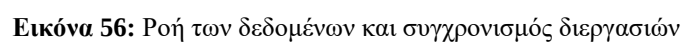
- Τερματικό 5 (object_detection): Εκτελεί τον κόμβο τεχνητής όρασης για τον εντοπισμό αντικειμένων, ολοκληρώνοντας τη στοίβα λογισμικού του συστήματος.

Η προσέγγιση αυτή μέσω του `open_ros_terminals.sh` εξασφαλίζει ότι όλα τα υποσυστήματα εκκινούν ταυτόχρονα και παραμένουν διακριτά, διευκολύνοντας την παρακολούθηση και την αποσφαλμάτωση (debugging), καθώς ο χρήστης έχει άμεση οπτική επαφή με τα μηνύματα εξόδου κάθε κόμβου ξεχωριστά.

Η αναλυτική παρουσίαση των βημάτων υλοποίησης παραλείπεται εδώ, καθώς η δομή του πακέτου είναι πολύ απλή και ακολούθησε την ίδια διαδικασία όπως και για τα προηγούμενα πακέτα.

5.1.8 Αναπαράσταση Συνεργασίας Κόμβων

Μετά την επιτυχή ανάπτυξη και παραμετροποίηση των επιμέρους υποσυστημάτων, παρουσιάζεται η συνολική αρχιτεκτονική συνεργασίας των κόμβων. Η αλληλεπίδραση μεταξύ των δομικών στοιχείων του λογισμικού αναδεικνύει τη ροή των δεδομένων και τον συγχρονισμό των διεργασιών, όπως αυτά αποτυπώνονται στο ακόλουθο διάγραμμα επικοινωνίας.



6. Προσομοίωση και Ανάλυση των Αποτελεσμάτων

Στο παρόν κεφάλαιο περιγράφονται και αναλύονται όλες οι διαδικασίες που εκτελέστηκαν για την προσομοίωση, όπως και η αξιολόγηση των αποτελεσμάτων. Αρχικά, αναλύεται η εκτέλεση του αρχείου `open_ros_terminals.sh`, το οποίο εκκινεί τους απαραίτητους κόμβους για την εκκίνηση της προσομοίωσης.

6.1 Εκκίνηση Λογισμικού Σταθμού Εδάφους

Στο πρώτο στάδιο εκτελέστηκε το πρόγραμμα του σταθμού εδάφους QGroundControl (QGC), το οποίο αποτελεί τη βασική διεπαφή επικοινωνίας με το MEA και παρέχει την άδεια για την απογείωση (arming). Χωρίς αυτήν οι κινητήρες παραμένουν ανενεργοί. Μέσω της εφαρμογής πραγματοποιείτε η παρακολούθηση του οχήματος σε πραγματικό χρόνο. Σε αυτό συμπεριλαμβάνονται μεταξύ άλλων η παγκόσμια θέση στον διαδραστικό χάρτη, η στάθμη της μπαταρίας και η κατάσταση των αισθητήρων. Κατά την εκκίνηση του προγράμματος εμφανίζετε μόνο ο κενός χάρτης, αν δεν είναι συνδεδεμένο MEA.



Εικόνα 57: Πρόγραμμα QGC χωρίς διασύνδεση MEA

6.2 Εκκίνηση Διεργασιών Προσομοίωσης και Προετοιμασία για την Απογείωση (Takeoff)

Στις ενότητες που ακολουθούν, αναλύονται διεξοδικά οι υπολογιστικές διεργασίες που εκτελέστηκαν για την προσομοίωση του MEA στο εικονικό περιβάλλον Baylands. Η ανάλυση ξεκινά από τον καθορισμό της διαδρομής φακέλου με τα τρισδιάστατα μοντέλα του Gazebo.

6.2.1 Προετοιμασία Προσομοίωσης

Για να μπορέσει να γίνει η φόρτωση όλων των τρισδιάστατων μοντέλων στο περιβάλλον προσομοίωσης και να ενεργοποιηθεί η άδεια για το MEA στο QGC, έπρεπε πρώτα να γίνει η εξαγωγή της διαδρομής του φακέλου models στη μεταβλητή GZ_SIM_RESOURCE_PATH του συστήματος (bashrc). Έτσι διασφαλίζετε ότι ο εξομοιωτής είναι σε θέση να εντοπίσει και να φορτώσει τα τρισδιάστατα μοντέλα και τα παρελκόμενα στο περιβάλλον του.

6.2.2 Εκκίνηση Προσομοίωσης

Έπειτα ακολούθησε η εκτέλεση του αρχείου open_ros_terminals.sh σε τερματικό. Το αρχείο αναλαμβάνει την ταυτόχρονη εκκίνηση όλων των απαραίτητων διεργασιών και κόμβων για την έναρξη της προσομοίωσης. Κάθε λειτουργία οργανώνεται σε διακριτό τερματικό όπως περιγράφηκε στο κεφάλαιο 5.7.1. Οι διεργασίες που εκτελέστηκαν ταυτόχρονα στα τερματικά διατυπώνονται στις εικόνες που ακολουθούν.


```
Object Detection
/home/mikelap/.ros/log/2026-01-03-04-13-08-A56040-mikelap-aliemware-293
882' => "/home/mikelap/.ros/log/latest"

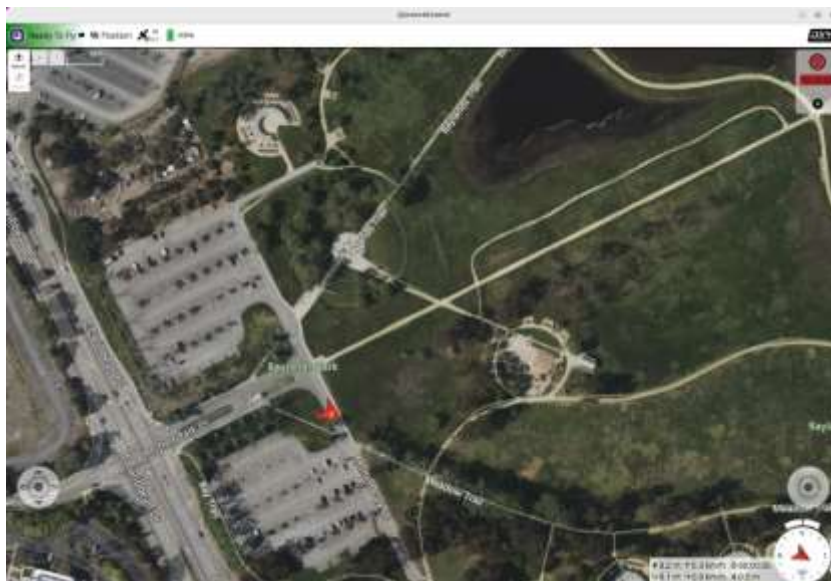
warnings.warn
[INFO] [launch]: All log files can be found below /home/mikelap/.ros/log/2026-01-03-04-13-08-A56040-mikelap-aliemware-293882
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [rtabmap-1]: process started with pid [293139]
[INFO] [rtabmap_viz-2]: process started with pid [293140]
[INFO] [static_transform_publisher-3]: process started with pid [293141]
[INFO] [static_transform_publisher-4]: process started with pid [293142]
[INFO] [rviz2-5]: process started with pid [293143]
[static_transform_publisher-3] [WARN] [1767406388.604324963] []: Old-style arguments are deprecated; see --help for a
new-style arguments
[static_transform_publisher-4] [WARN] [1767406388.608961689] []: Old-style arguments are deprecated; see --help for a
new-style arguments
[static_transform_publisher-3] [INFO] [1767406388.615843082] [base_footprint_to_base_link]: Spinning until stopped -
publishing transform
[static_transform_publisher-3] Translation: ('0.000000', '0.000000', '0.000000')
[static_transform_publisher-3] rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
[static_transform_publisher-3] from 'x650_0/base_footprint' to 'base_link'
[static_transform_publisher-4] [INFO] [1767406388.620525192] [base_link_to_camera]: Spinning until stopped - publish
ing transform
[static_transform_publisher-4] Translation: ('0.160000', '0.000000', '0.000000')
[static_transform_publisher-4] rotation: ('0.500000', '0.499992', '0.500000', '0.500398')
[static_transform_publisher-4] from 'base_link' to 'x650_0/base_link/cam'
[rtabmap-1] [INFO] [1767406388.620825906] [rtabmap.rtabmap]: rtabmap[maps]: latch = true
[rtabmap-1] [INFO] [1767406388.620126604] [rtabmap.rtabmap]: rtabmap[maps]: map_filter_radius = 0.000000
[rtabmap-1] [INFO] [1767406388.620113990] [rtabmap.rtabmap]: rtabmap[maps]: map_filter_angle = 30.000000
[rtabmap-1] [INFO] [1767406388.620338236] [rtabmap.rtabmap]: rtabmap[maps]: map_cleanup = true
[rtabmap-1] [INFO] [1767406388.620341169] [rtabmap.rtabmap]: rtabmap[maps]: map_always_update = false
[rtabmap-1] [INFO] [1767406388.620144412] [rtabmap.rtabmap]: rtabmap[maps]: map_empty_ray_tracing = true
[rtabmap-1] [INFO] [1767406388.620147829] [rtabmap.rtabmap]: rtabmap[maps]: map_cache_loaded_on_init = true
[rtabmap-1] [INFO] [1767406388.620150625] [rtabmap.rtabmap]: rtabmap[maps]: cloud_output_voxelized = true
[rtabmap-1] [INFO] [1767406388.620153761] [rtabmap.rtabmap]: rtabmap[maps]: cloud_subtract_filtering = false
[rtabmap-1] [INFO] [1767406388.620157955] [rtabmap.rtabmap]: rtabmap[maps]: cloud_subtract_filtering_min_neighbours =
3
[rtabmap-1] [INFO] [1767406388.620167829] [rtabmap.rtabmap]: rtabmap[maps]: octomap_tree_depth = 10
[rtabmap-1] [INFO] [1767406388.620172100] [rtabmap.rtabmap]: rtabmap: frame_id = "base_link"
```

Εικόνα 61: Εκτέλεση RTAB-Map χαρτογράφηση στο 4ο τερματικό

```
Object Detection
[INFO] [1767403570.235437961] [object_detection]: Node started. Waiting for sync
hronized RGB and Depth frames...
```

Εικόνα 62: Εκτέλεση αναγνώριση ατόμων μέσω YOLO στο 5ο τερματικό

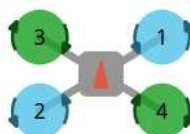
Μαζί με τις παραπάνω διεργασίες, φορτώνεται και το MEA μέσα στο πρόγραμμα QGC και ενεργοποιείται η άδεια για την απογείωση.



Εικόνα 63: Φόρτωση MEA μέσα στο πρόγραμμα QGC

Πριν όμως πραγματοποιηθεί η απογείωση, πρέπει να γίνει οπτική επιβεβαίωση και έλεγχο της ορθής φοράς περιστροφής των ελίκων, η οποία είναι κρίσιμη για την ευστάθεια του σκάφους. Έτσι σύμφωνα με τις προδιαγραφές του αυτόματου πιλότου PX4, επιβεβαιώθηκε η εξής διάταξη, η οποία παρουσιάζετε με γραφικό τρόπο παρακάτω:

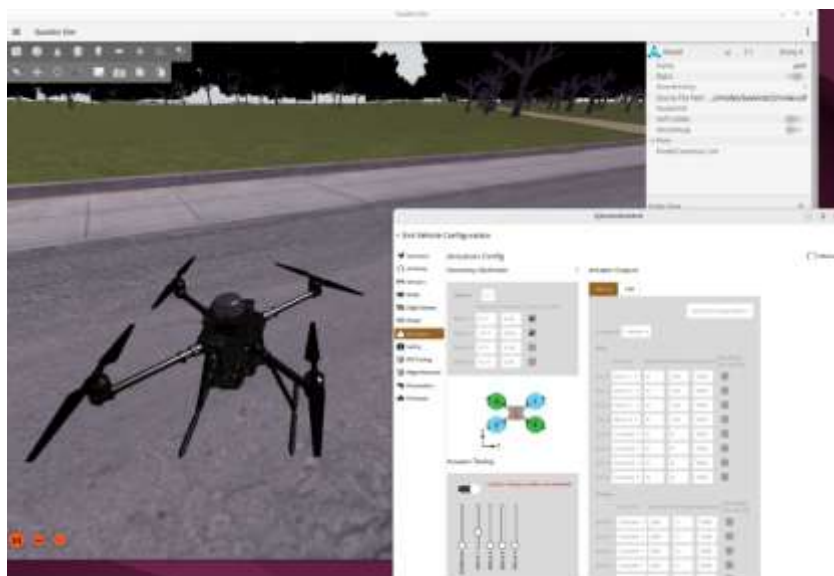
- Εμπρόσθιος δεξιός (front-right): Αριστερόστροφη κίνηση (μπλε, CCW).
- Εμπρόσθιος αριστερός (front-left): Δεξιόστροφη κίνηση (πράσινο, CW).
- Οπίσθιος δεξιός (back-right): Δεξιόστροφη κίνηση (CW).
- Οπίσθιος αριστερός (back-left): Αριστερόστροφη κίνηση (CCW).



Εικόνα 64: Διάταξη κινητήρων στο περιβάλλον QGC

Η επαλήθευση της φοράς κάθε κινητήρα έγινε με την βοήθεια του πρόσθετου Vehicle Configuration στο πρόγραμμα μέσω των ρυθμιστικών (sliders). Μέσω της επιλογής Identify & Assign Motors εκκινείται η διαδικασία αναγνώρισης των κινητήρων, η οποία μεταφράζεται στο περιβάλλον Gazebo. Σε περίπτωση

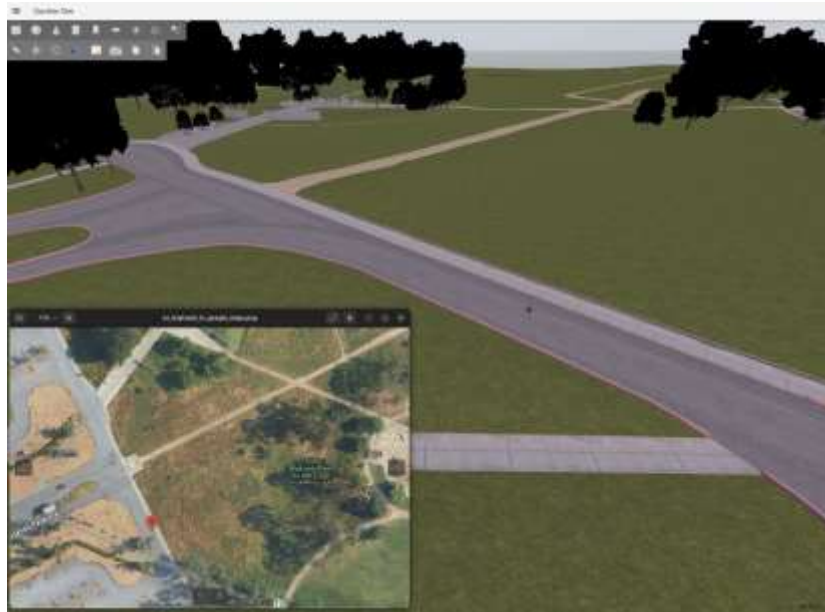
λανθασμένης φοράς κάποιου κινητήρα, παρέχει το πρόγραμμα τη δυνατότητα επιδιόρθωσης.



Εικόνα 65: Έλεγχο φοράς κινητήρων στο QGC

Η σωστή παραμετροποίηση εξασφαλίζει την απαραίτητη ισορροπία ροπών στρέψης και εγγυάται μια σταθερή πτητική συμπεριφορά. Η τήρηση αυτής της διάταξης είναι υποχρεωτική για τον σωστό έλεγχο των αξόνων Roll, Pitch και Yaw, διασφαλίζοντας ότι το MEA θα ανταποκρίνεται με ακρίβεια στις εντολές του χειριστή, ανεξάρτητα από την οπτική απεικόνιση στο περιβάλλον του Gazebo.

Σε αυτό το σημείο είναι όλα έτοιμα για να ξεκινήσει η απογείωση και η εκτέλεση της αποστολής με την αλληλεπίδραση μέσου του χειριστή.



Εικόνα 66: Παρουσίαση του ΜΕΑ μέσα στο ρεαλιστικό εικονικό περιβάλλον Baylands

6.3 Εκκίνηση Κόμβου Εξωτερικού Ελέγχου

Η διαχείριση και απογείωση του ΜΕΑ υλοποιείται είτε μέσω του λογισμικού QGroundControl είτε αυτόνομα μέσω εξωτερικού ελέγχου (offboard control), που αποτελεί και την επιλογή της παρούσας εργασίας. Κεντρικό ρόλο παίζει το αρχείο `offboard_ctrl.launch.py`, το οποίο ενεργοποιεί μια Μηχανή Πεπερασμένων Καταστάσεων (FSM) για τη διαχείριση της πτητικής λογικής όπως έγινε αναφορά στο κεφάλαιο 5.1.3. Ο κόμβος μεταφράζει τις εντολές σε οδηγίες θέσης και προσανατολισμού, συντονίζοντας μεταβάσεις όπως η απογείωση και η αιώρηση. Η χρήση της παραμέτρου `use_sim_time: True` διασφαλίζει τον απόλυτο χρονικό συγχρονισμό με το Gazebo, επιτρέποντας την εκτέλεση σύνθετων αποστολών με υψηλή ακρίβεια.

Στον πίνακα που ακολουθεί παρατίθενται οι μεταβάσεις των καταστάσεων μετά την εκτέλεση του κώδικα, ενώ στην αντίστοιχη εικόνα αποτυπώνεται η έξοδος (output) του τερματικού, επιβεβαιώνοντας την ορθή αλληλουχία των ενεργειών του συστήματος. Τέλος, παρουσιάζεται η κατάσταση αιώρησης (HOVER_MODE) του οχήματος στο περιβάλλον του Gazebo, η οποία σηματοδοτεί τη σταθεροποίηση του ΜΕΑ στο επιθυμητό υψόμετρο μετά την ολοκλήρωση της διαδικασίας απογείωσης.

No.	Κατάσταση	Ενέργεια
1	INIT_MODE	Το όχημα περιμένει εντολές
2	OFFBOARD_MODE	Ενεργοποίηση του εξωτερικού ελέγχου
3	ARM_MODE	Όπλιση του οχήματος
4	TAKEOFF_MODE	Το όχημα απογειώνετε μέχρι να φτάσει τα 5 μ ύψος
5	HOVER_MODE	Το όχημα αιωρείται σε ύψος 5 μ

Πίνακας 36: Μεταβάσεις κατά την εκτέλεση της FSM

```

uav_project_ws5 ros2 launch uav_offboard_ctrl offboard_ctrl.launch.py
[INFO] [launch]: All log files can be found below /home/nikelap/.ros/log/2026-01-03-03-58-47-4
07984-nikelap-alienware-273387
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [px4_ctrl-1]: process started with pid [273390]
[px4_ctrl-1] [INFO] [1767405527.596564069] [px4_ctrl]: Starting Offboard Control with PX4 serv
ices
[px4_ctrl-1] [WARN] [1767405532.843847746] [px4_ctrl]: New publisher discovered on topic '/aca
n', offering incompatible QoS. No messages will be sent to it. Last incompatible policy: DURAB
ILITY_QOS_POLICY
[px4_ctrl-1] [INFO] [1767405531.635163017] [px4_ctrl]: Requesting switch to Offboard mode
[px4_ctrl-1] [INFO] [1767405531.635256823] [px4_ctrl]: Command send
[px4_ctrl-1] [INFO] [1767405531.651153540] [px4_ctrl]: Service replied successfully with 0
[px4_ctrl-1] [INFO] [1767405531.651182072] [px4_ctrl]: Command accepted
[px4_ctrl-1] [INFO] [1767405532.735166545] [px4_ctrl]: Entered offboard mode
[px4_ctrl-1] [INFO] [1767405532.735202571] [px4_ctrl]: Arming mode engaged
[px4_ctrl-1] [INFO] [1767405532.735244992] [px4_ctrl]: Command send
[px4_ctrl-1] [INFO] [1767405532.754436551] [px4_ctrl]: Service replied successfully with 0
[px4_ctrl-1] [INFO] [1767405532.754471901] [px4_ctrl]: Command accepted
[px4_ctrl-1] [INFO] [1767405532.838193714] [px4_ctrl]: Vehicle armed
[px4_ctrl-1] [INFO] [1767405532.835225930] [px4_ctrl]: Take Off engaged
[px4_ctrl-1] [INFO] [1767405542.635162016] [px4_ctrl]: Takeoff altitude of 5.00m reached
[px4_ctrl-1] [INFO] [1767405542.635191341] [px4_ctrl]: Hover mode engaged
[px4_ctrl-1] [INFO] [1767405542.735208953] [px4_ctrl]: Hover at global position:37.412185 -121
.998896 43.064098

```

Εικόνα 67: Εκτέλεση εξωτερικού ελέγχου



Εικόνα 68: Κατάσταση HOVER_MODE του MEA

6.4 Εκκίνηση Κόμβου για τον Τηλεχειρισμό του MEA

Ευρισκόμενο στην κατάσταση HOVER_MODE, το όχημα παραμένει σε αναμονή μέχρι τη λήψη εντολής για μετάβαση στη λειτουργία TELEOP_MODE. Η μετάβαση αυτή επιτυγχάνεται μέσω της εκτέλεσης του κώδικα keyboard_teleop.py, ο οποίος ενεργοποιεί τον αντίστοιχο κόμβο τηλεχειρισμού. Σε αυτή τη φάση, ο έλεγχος του MEA μεταφέρεται εξ ολοκλήρου στον χειριστή, διατηρώντας ωστόσο τη λειτουργία Offboard ενεργή. Ο κόμβος μεταφράζει τις εισόδους από το πληκτρολόγιο σε μηνύματα τύπου Twist, τα οποία δημοσιεύονται στο κανάλι /offboard_velocity_cmd για τον έλεγχο της ταχύτητας του οχήματος.

Συγκεκριμένα, μέσω των πλήκτρων WASD και των βελών κατεύθυνσης, ο χρήστης ελέγχει τις γωνίες Roll, Pitch, Yaw και το υψόμετρο (Z). Παράλληλα, ειδικά πλήκτρα όπως το SPACE και το R έχουν αντιστοιχιστεί σε κρίσιμες λειτουργίες, όπως η ενεργοποίηση της αυτόματης επιστροφής στη βάση (Return to Launch). Το λογισμικό μετατρέπει τις εισόδους σε γραμμικές και γωνιακές ταχύτητες (π.χ. σταθερή γραμμική ταχύτητα 5 m/s), εξασφαλίζοντας την άμεση και ακριβή απόκριση του MEA στις οδηγίες του χειριστή καθ' όλη τη διάρκεια της αποστολής.

```
s/uav_project_ws$ ros2 run uav_navigation keyboard_teleop.py

This node takes keypresses from the keyboard and publishes
them as Twist messages.

Using the arrow keys and WASD you have Mode 2 RC controls.

##### MOVEMENT #####
#
# (w): Up
# (s): Down
# (a): Yaw Left
# (d): Yaw Right
# (Up Arrow): Pitch Forward
# (Down Arrow): Pitch Backward
# (Left Arrow): Roll Left
# (Right Arrow): Roll Right
#
# Press (SPACE) to arm/disarm the drone
# Press (R) to return to base
# Press (ESC) to quit the application
#
#####
```

Εικόνα 69: Μενού πλοήγησης για τον τηλεχειρισμό

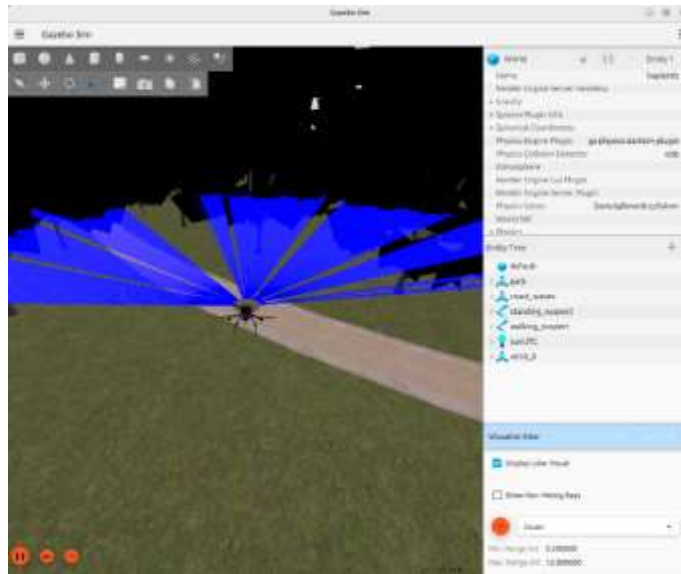


Εικόνα 70: Πλοήγηση στο περιβάλλον Gazebo

6.5 Online Χαρτογράφηση μέσω RTAB-Map

Η online χαρτογράφηση αφορά τη δημιουργία χάρτη σε πραγματικό χρόνο, σε αντίθεση με τη χρήση προϋπάρχουσας χωρικής πληροφορίας. Σε εσωτερικούς χώρους (indoor), η στατική φύση του περιβάλλοντος διευκολύνει την παροχή αξιόπιστων δεδομένων. Αντιθέτως, σε δυναμικά εξωτερικά περιβάλλοντα (outdoor), ο αλγόριθμος RTAB-Map αποδεικνύεται καθοριστικός, καθώς διαχειρίζεται το περιορισμένο εύρος των αισθητήρων μέσω της ανίχνευσης κλεισίματος βρόχου (loop closure). Η δυνατότητα αυτή επιτρέπει τη διόρθωση σφαλμάτων συσσωρευμένης οδομετρίας και την ενσωμάτωση αποσπασματικών δεδομένων σε έναν ενιαίο χάρτη. Έτσι, εξασφαλίζεται η ακριβής αποφυγή εμποδίων και η ασφαλής αυτόνομη πλοήγηση του ΜΕΑ, ακόμη και όταν το περιβάλλον μεταβάλλεται διαρκώς.

Στην παρακάτω εικόνα αποτυπώνεται ο τρόπος με τον οποίο το ΜΕΑ αντιλαμβάνεται το περιβάλλον γύρω του, μέσω της οπτικοποίησης των δεδομένων που συλλέγουν οι αισθητήρες του σε πραγματικό χρόνο.

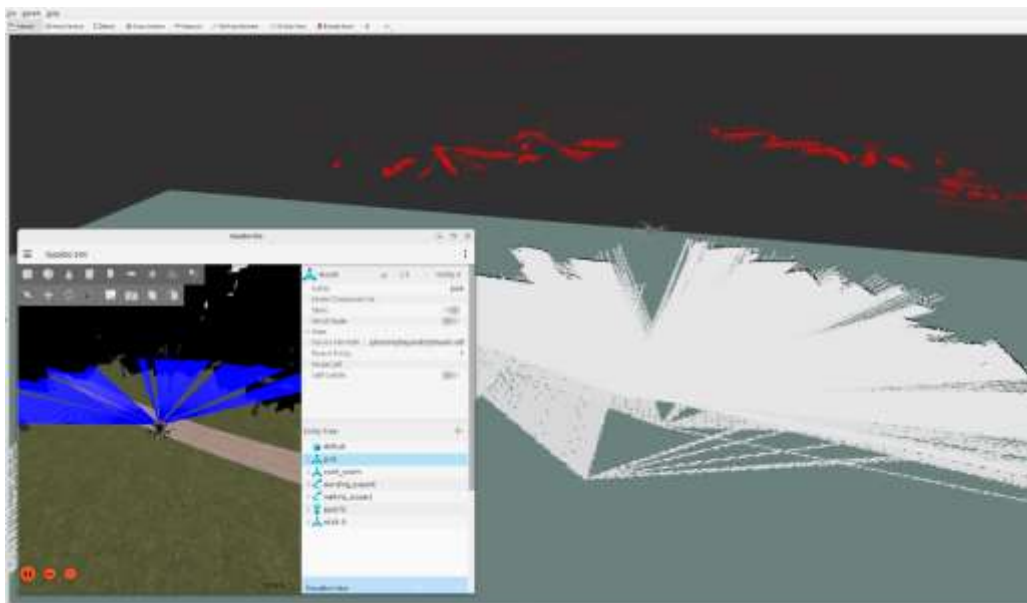


Εικόνα 71: Οπτικοποίηση δεδομένων σάρωσης του αισθητήρα LiDAR στο εικονικό περιβάλλον

Το σύστημα προβάλλει δεδομένα σε πραγματικό χρόνο στα εργαλεία RViz και rtabmap_viz, επιτρέποντας την εποπτεία του τρισδιάστατου νέφους σημείων (point cloud) και της τροχιάς του MEA. Για την ορθή απεικόνιση στο RViz, απαιτούνται οι εξής ρυθμίσεις:

- Fixed Frame: Ορισμός στο map για σταθερή αναφορά κατά την κίνηση.
- TF Display: Ενεργοποίηση μετασχηματισμών (map, odom, base_link) για την οπτικοποίηση της ιεραρχίας του συστήματος.
- Map Cloud και Scan: Προσθήκη plugins για την προβολή του νέφους σημείων και των δεδομένων LiDAR.
- Robot Model: Φόρτωση του τρισδιάστατου μοντέλου για την παρακολούθηση θέσης και προσανατολισμού.

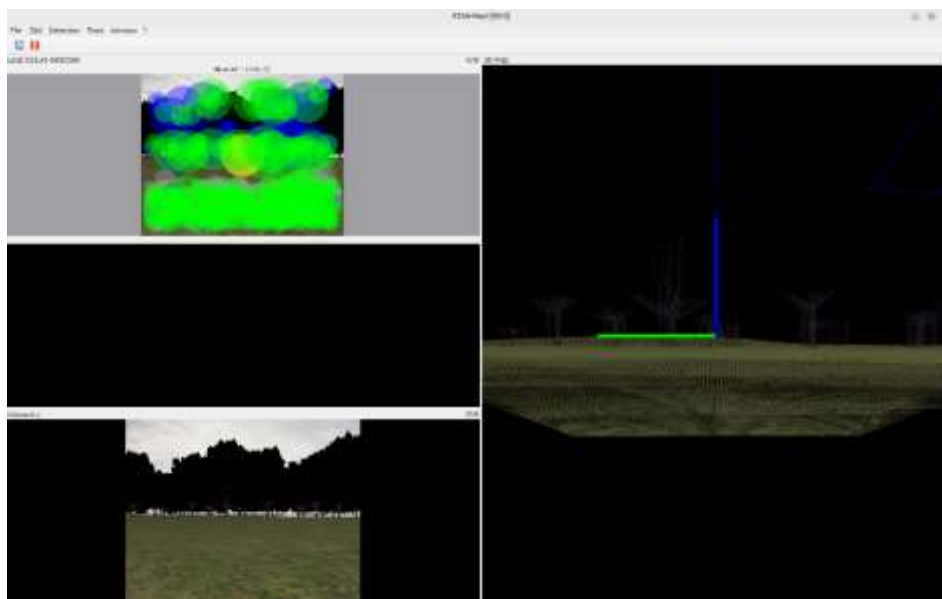
Στην παρακάτω εικόνα αποτυπώνεται η δημιουργία του χάρτη στο RViz, όπου τα εμπόδια αναπαρίστανται με κόκκινο νέφος σημείων (point cloud). Ο κενός χώρος απεικονίζεται με λευκό χρώμα, ενώ τα όρια του εδάφους οριοθετούνται με μαύρο, παρέχοντας μια σαφή γεωμετρική αντίληψη του περιβάλλοντος για την ασφαλή πλοήγηση του MEA.



Εικόνα 72: Προβολή δημιουργίας του χάρτη στο RViz

Η συνδυαστική αυτή παρακολούθηση διασφαλίζει την ποιότητα της χαρτογράφησης και την ακρίβεια της πλοήγησης.

Πέραν του RViz, για την εποπτεία της διαδικασίας SLAM χρησιμοποιείται το εργαλείο `rtabmap_viz` [47], μια εξειδικευμένη διεπαφή που προσφέρει πολύτιμες πληροφορίες για την online χαρτογράφηση σε πραγματικό χρόνο. Στο 3D Map (δεξιά) αναπαρίσταται το παραγόμενο νέφος σημείων με RGB δεδομένα, ενώ η τρέχουσα θέση και ο προσανατολισμός του MEA υποδεικνύονται από τους άξονες του `base_link`. Μια πράσινη γραμμή καταγράφει τη διανυθείσα τροχιά του οχήματος. Στο παράθυρο Loop Closure Detection (πάνω αριστερά), το σύστημα εντοπίζει οπτικά χαρακτηριστικά (πράσινοι κύκλοι), τα οποία όταν ταυτιστούν με προηγούμενες λήψεις (κίτρινοι κύκλοι), διορθώνουν τη συσσωρευμένη ολίσθηση (drift) της οδομετρίας, διασφαλίζοντας τη γεωμετρική ακρίβεια. Παράλληλα, το παράθυρο Odometry (κάτω αριστερά) επιβεβαιώνει την επάρκεια των οπτικών χαρακτηριστικών για την ακριβή παρακολούθηση της κίνησης. Συνολικά, το `rtabmap_viz` αποτελεί κρίσιμο εργαλείο για την επαλήθευση της ορθής αντίληψης του χώρου και των επιτυχών διορθώσεων θέσης κατά τη διάρκεια της αποστολής.



Εικόνα 73: Απεικόνιση δεδομένων στο rtabmap_viz

6.6 Αναγνώριση Ατόμων

Η αναγνώριση αντικειμένων (object detection) συνιστά μια θεμελιώδη διεργασία στον τομέα της τεχνητής νοημοσύνης, επιτρέποντας την αυτόματη ταξινόμηση στοιχείων σε διακριτές κατηγορίες (π.χ. καράβι, άτομο, γάτα, κ.ο.κ). Στο πλαίσιο της παρούσας εργασίας, εκτελείτε ο εντοπισμός ατόμων μέσω του προηγμένου αλγορίθμου YOLOv8, ο οποίος παρουσιάστηκε στο κεφάλαιο 4.1.8. Ο αντίστοιχος κόμβος επεξεργάζεται τη ροή εικόνας που παρέχεται από τη γέφυρα του Gazebo, εφαρμόζοντας το νευρωνικό δίκτυο σε πραγματικό χρόνο για την αναγνώριση ατόμων εντός του οπτικού πεδίου του MEA.

Στις εικόνες που ακολουθούν, αποτυπώνεται η λειτουργία του αλγορίθμου κατά τον εντοπισμό στόχων, όπου παρέχονται πληροφορίες για την κλάση του αντικειμένου και την απόσταση από το MEA σε μέτρα. Κάθε αναγνωρισμένο στοιχείο εσωκλείεται σε ένα πλαίσιο οριοθέτησης (bounding box), το οποίο υποδεικνύει τη θέση του στο οπτικό πεδίο. Ο αλγόριθμος υποστηρίζει την ταυτόχρονη επεξεργασία πολλαπλών αντικειμένων, καθιστώντας εφικτό τον εντοπισμό ομάδων, όπως για παράδειγμα ορειβατών σε αποστολές έρευνας και διάσωσης. Η δυνατότητα αυτή, σε συνδυασμό με την εκτίμηση της απόστασης, επιτρέπει στο σύστημα να

αντιλαμβάνεται τη γεωμετρική θέση των ατόμων στον χώρο, προσδίδοντας σημασιολογική πληροφορία στον παραγόμενο χάρτη.



Εικόνα 74: Αναγνώριση ατόμων (1)



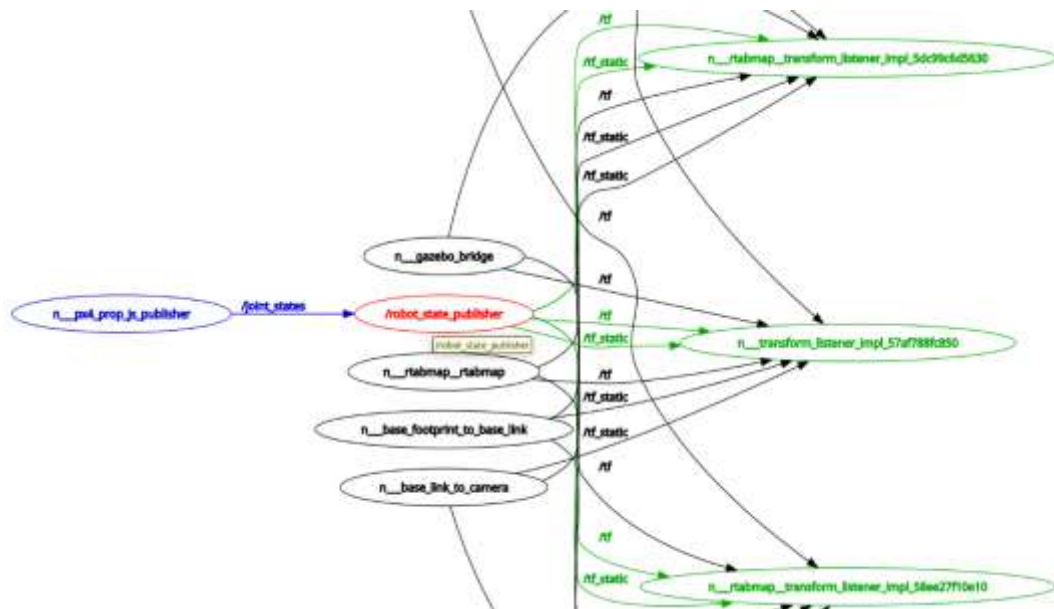
Εικόνα 75: Αναγνώριση ατόμων (2)

6.7 Ανάλυση Επικοινωνίας και Διασύνδεσης Συστατικών

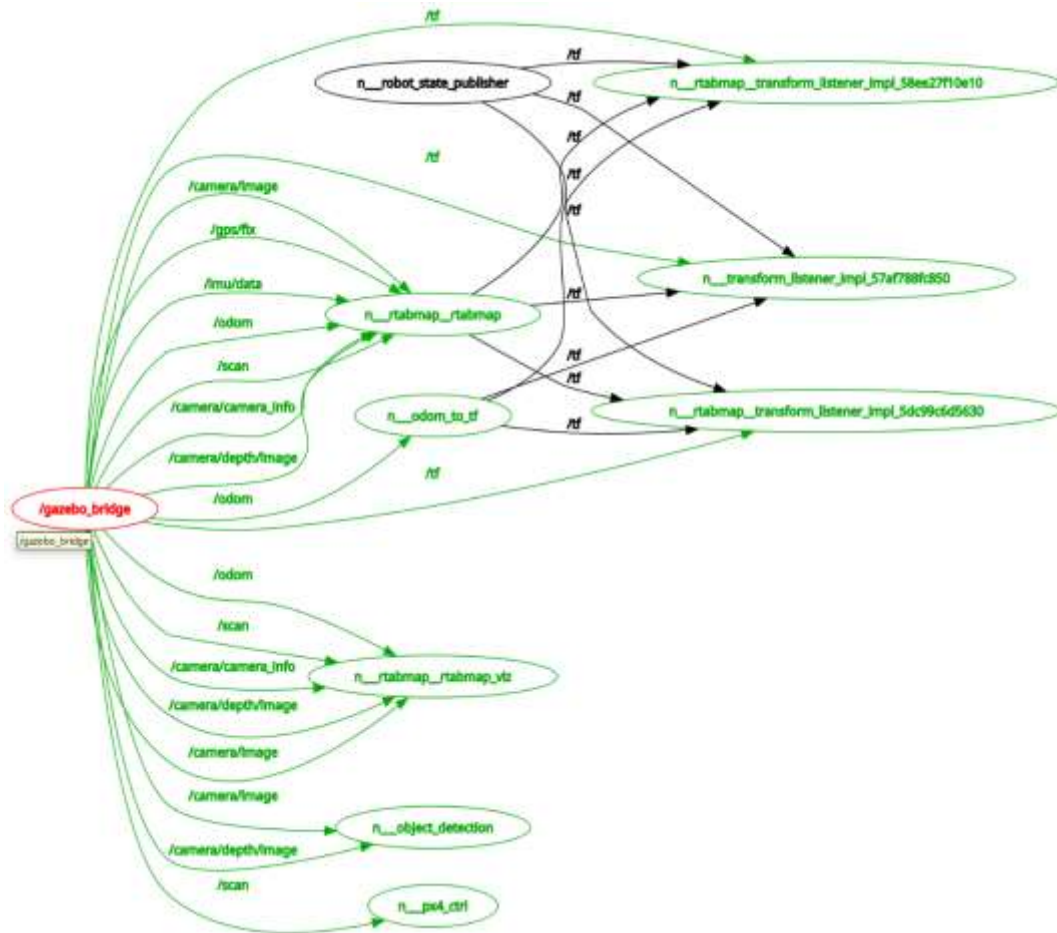
Η παρούσα ενότητα εστιάζει στην ανάλυση της επικοινωνίας μεταξύ των συστατικών του ROS 2 και του PX4 ελεγκτή πτήσης. Η διαδικασία αυτή αποτελεί κρίσιμο στάδιο για την επαλήθευση της ορθής διασύνδεσης των κόμβων και των αλγορίθμων του MEA. Χρησιμοποιώντας τις εντολές του ROS 2, τεκμηριώνεται η

ιεραρχία και η ανταλλαγή μηνυμάτων, διασφαλίζοντας τον απόλυτο έλεγχο και τον συγχρονισμό του οχήματος σε πραγματικό χρόνο κατά την προσομοίωση.

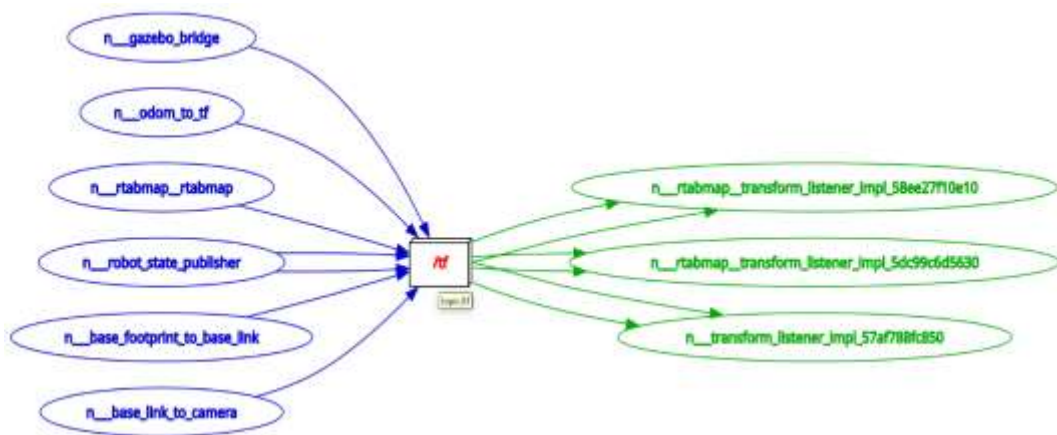
Στα επόμενα σχήματα αποτυπώνεται, μέσω του εργαλείου `rqt_graph`, η αρχιτεκτονική επικοινωνίας μεταξύ των κόμβων (nodes), αναδεικνύοντας τη ροή δεδομένων στα ενεργά κανάλια επικοινωνίας (topics). Σύμφωνα με τη χρωματική κωδικοποίηση του εργαλείου, ο κόμβος υπό εξέταση επισημαίνεται με κόκκινο χρώμα, ενώ οι εισερχόμενες και εξερχόμενες διασυνδέσεις (edges) διακρίνονται με μπλε και πράσινο χρώμα αντίστοιχα. Η οπτικοποίηση αυτή επιτρέπει την πλήρη χαρτογράφηση της αλληλεπίδρασης των υποσυστημάτων και την επαλήθευση της ορθής δρομολόγησης των μηνυμάτων.



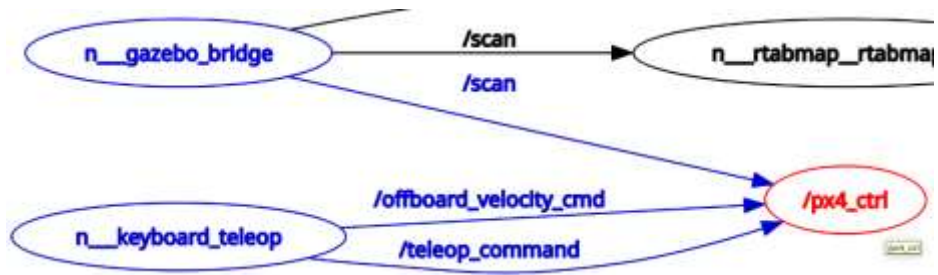
Σχήμα 6: Ανάλυση επικοινωνίας του κόμβου `robot_state_publisher`



Σχήμα 7: Ανάλυση επικοινωνίας του κόμβου gazebo_bridge



Σχήμα 8: Ανάλυση επικοινωνίας του κόμβου TF



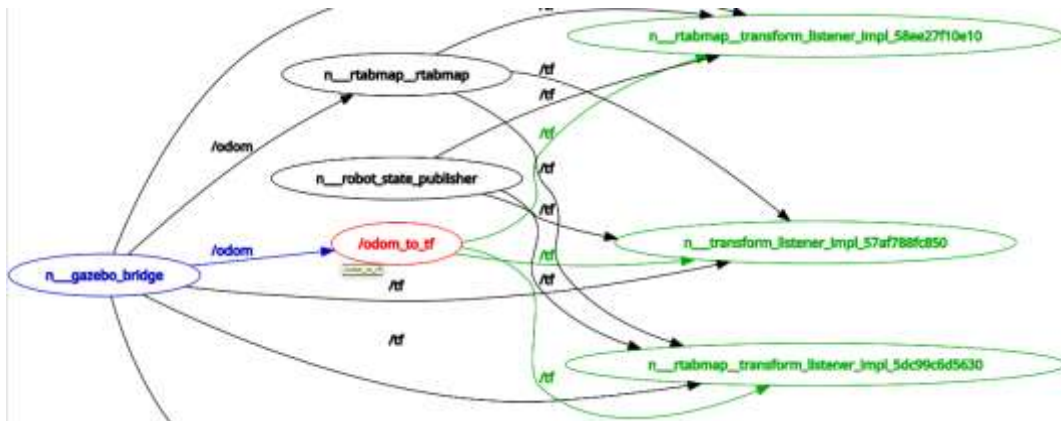
Σχήμα 9: Ανάλυση επικοινωνίας του κόμβου px4_ctrl



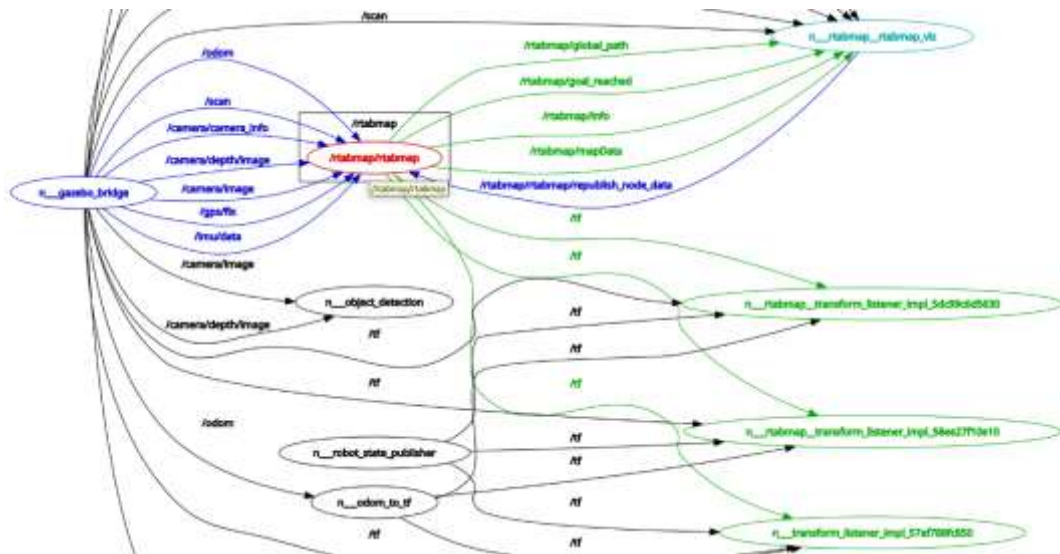
Σχήμα 10: Ανάλυση επικοινωνίας του κόμβου px4_prop_js_publisher



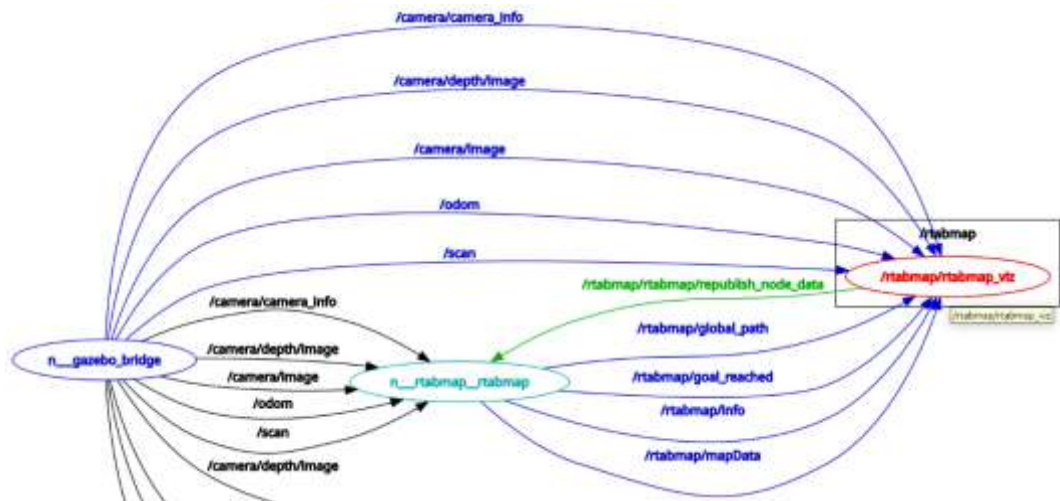
Σχήμα 11: Ανάλυση επικοινωνίας του κόμβου keyboard_teleop



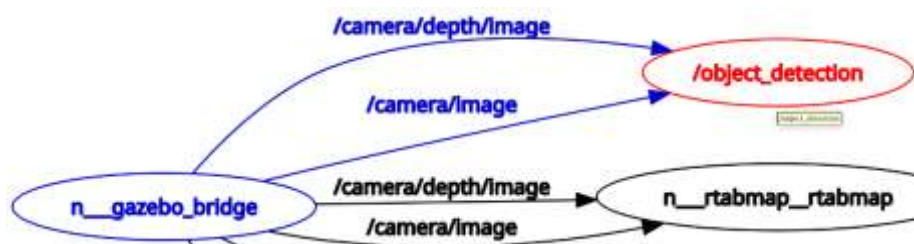
Σχήμα 12: Ανάλυση επικοινωνίας του κόμβου odom_to_tf



Σχήμα 13: Ανάλυση επικοινωνίας του κόμβου rtabmap



Σχήμα 14: Ανάλυση επικοινωνίας του κόμβου rtabmap_viz



Σχήμα 15: Ανάλυση επικοινωνίας του κόμβου object_detection

Τα αποτελέσματα των γραφημάτων αναδεικνύουν την ορθή διασύνδεση και συνεργασία όλων των απαραίτητων κόμβων, επαληθεύοντας τη λειτουργική ακεραιότητα του συστήματος κατά τη διάρκεια της προσομοίωσης.

7. Συμπεράσματα και Μελλοντικές Επεκτάσεις

Στην παρούσα διπλωματικής εργασίας παρουσιάστηκε η επιτυχή ανάπτυξη ενός ολοκληρωμένου και λειτουργικού πλαισίου προσομοίωσης για MEA, το οποίο γεφυρώνει το χάσμα μεταξύ της θεωρητικής σχεδίασης και της πρακτικής εφαρμογής σε σύγχρονα περιβάλλοντα όπως το ROS 2 και τον εξομοιωτή Gazebo. Η αρχιτεκτονική που υλοποιήθηκε μέσω του πακέτου `uav_bringup` απέδειξε ότι η σπονδυλωτή δομή επιτρέπει τον αποτελεσματικό συντονισμό και την παράλληλη εκτέλεση σύνθετων διεργασιών. Η δυνατότητα ταυτόχρονης διαχείρισης του αυτόματου πιλότου PX4, της επικοινωνίας μέσω του Micro-XRCE-DDS Agent και της επεξεργασίας δεδομένων αισθητήρων, επιβεβαίωσε την αξιοπιστία του συστήματος κάτω από απαιτητικές συνθήκες προσομοίωσης.

Η χρήση του εξωτερικού ελέγχου (offboard control) μέσω της μηχανής πεπερασμένων καταστάσεων του κόμβου `px4_ctrl` αποτέλεσε τον πυρήνα της λήψης αποφάσεων. Η προσέγγιση αυτή επέτρεψε στο όχημα να μεταφράζει υψηλού επιπέδου οδηγίες σε συγκεκριμένες καταστάσεις πτήσης, εξασφαλίζοντας ομαλές μεταβάσεις από την απογείωση στην ελεύθερη πτήση και την προσγείωση. Παράλληλα, η ενσωμάτωση του RTAB-Map για τη δημιουργία online χαρτογράφησης και του αλγορίθμου YOLOv8 για την ανίχνευση ατόμων, προσέδωσε στο MEA σημασιολογική αντίληψη του περιβάλλοντος. Η οπτικοποίηση των δεδομένων αυτών μέσω του RViz2 και του `rtabmap_viz` παρείχε στον χειριστή μια ολοκληρωμένη και σαφή εικόνα της αποστολής, αποδεικνύοντας ότι το σύστημα μπορεί να αναγνωρίζει στόχους και να κατασκευάζει χωρικές αναπαραστάσεις με υψηλή ακρίβεια.

Όσον αφορά τις μελλοντικές επεκτάσεις, η παρούσα εργασία θέτει τις βάσεις για την περαιτέρω ενίσχυση της αυτονομίας του συστήματος. Η σημαντικότερη αναβάθμιση αφορά την ενσωμάτωση αλγορίθμων αυτόνομης πλοήγησης σε 2D και 3D χώρους με την δυναμική αποφυγή εμποδίων. Αυτό θα επιτρέψει στο ρομπότ να σχεδιάζει μονοπάτια σε άγνωστα περιβάλλοντα χρησιμοποιώντας απευθείας τον παραγόμενο χάρτη. Στον τομέα της τεχνητής όρασης, η προσθήκη εξειδικευμένων αλγορίθμων για την εκτίμηση της στάσης του σώματος (Pose Estimation) και την αναγνώριση

χειρονομιών θα μπορούσε να αναβαθμίσει την αλληλεπίδραση ανθρώπου-ρομπότ, καθιστώντας το σύστημα ιδανικό για σενάρια έρευνας και διάσωσης όπου η αναγνώριση της κατάστασης ενός θύματος είναι κρίσιμη.

Μια άλλη προοπτική είναι η επέκταση της αρχιτεκτονικής σε λειτουργίες συνεργατικού σμήνους (swarm robotics), όπου πολλαπλά ΜΕΑ θα χαρτογραφούν ταυτόχρονα μεγάλες εκτάσεις και θα μοιράζονται πληροφορίες για τον εντοπισμό στόχων, μειώνοντας δραστικά τον χρόνο ολοκλήρωσης μιας επιχείρησης.

Τέλος, η μετάβαση από το ψηφιακό περιβάλλον στο φυσικό hardware, όπως το Holybro x650, αποτελεί την τελική επιβεβαίωση της εργασίας. Η πλήρης συμβατότητα του αναπτυχθέντος λογισμικού με πραγματικούς ελεγκτές πτήσης διασφαλίζει ότι οι αλγόριθμοι που αναπτύχθηκαν μπορούν να μεταφερθούν στο πεδίο με ελάχιστες τροποποιήσεις. Η υλοποίηση αυτής της διπλωματικής εργασίας δεν αποτελεί απλώς μια μελέτη προσομοίωσης, αλλά ένα ολοκληρωμένο θεμέλιο για την ανάπτυξη επόμενης γενιάς αυτόνομων συστημάτων επιτήρησης και σημασιολογικής χαρτογράφησης.

Βιβλιογραφία

- [1] Beloev, I. H. (2016). A review on current and emerging application possibilities for unmanned aerial vehicles. *Acta Technol. Agric*, 19(3), 70-76.
- [2] Jumaat, N. F. H., Ahmad, B., & Dutsenwai, H. S. (2018, June). Land cover change mapping using high resolution satellites and unmanned aerial vehicle. In *IOP conference series: earth and environmental science* (Vol. 169, No. 1, p. 012076). IOP Publishing.
- [3] Perz, R., & Wronowski, K. (2019). UAV application for precision agriculture. *Aircraft Engineering and Aerospace Technology*, 91(2), 257-263.
- [4] Liao, K. C., & Lu, J. H. (2021). Using UAV to detect solar module fault conditions of a solar power farm with IR and visual image analysis. *Applied Sciences*, 11(4), 1835.
- [5] Drones for every farmer: Making agricultural technology accessible and affordable, Accessed: Jan. 04, 2026. [Online]. Available: <https://etedge-insights.com/industry/agriculture/drones-for-every-farmer-making-agricultural-technology-accessible-and-affordable/>
- [6] Gupta, A., & Fernando, X. (2022). Simultaneous localization and mapping (slam) and data fusion in unmanned aerial vehicles: Recent advances and challenges. *Drones*, 6(4), 85.
- [7] Enrico, R. (2023). The development of ROS-based offboard algorithms for autonomous UAVs intended for Mars exploration (Doctoral dissertation, Politecnico di Torino).
- [8] Hofmann, C., Fichtner, M., Lieret, M., & Franke, J. (2022). Efficient Semantic Mapping in Dynamic Environments. In *VISIGRAPP (4: VISAPP)* (pp. 803-810).
- [9] Wang, T., Yang, S., Wan, M., & Liu, Y. (2025). FFG-YOLO: Improved YOLOv8 for Target Detection of Lightweight Unmanned Aerial Vehicles. *Applied System Innovation*, 8(4), 109.
<https://doi.org/10.3390/asi8040109>
- [10] Fanta-Jende, P., Steininger, D., Kern, A., Widhalm, V., Apud Baca, J. G., Hofstätter, M., ... & Sulzbachner, C. (2023). Semantic real-time mapping with

- UAVs. PFG—Journal of Photogrammetry, Remote Sensing and Geoinformation Science, 91(3), 157-170.
- [11] Labbé, M., & Michaud, F. (2019). RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of field robotics*, 36(2), 416-446.
 - [12] Kozera, C. A. (2018). Military use of unmanned aerial vehicles—a historical study. *Safety & Defense*, 4, 17-21.
 - [13] Colomina, I., & Molina, P. (2014). Unmanned aerial systems for photogrammetry and remote sensing: A review. *ISPRS Journal of photogrammetry and remote sensing*, 92, 79-97.
 - [14] Watanabe, Y., Calise, A., & Johnson, E. (2007, August). Vision-based obstacle avoidance for UAVs. In *AIAA guidance, navigation and control conference and exhibit* (p. 6829).
 - [15] Chung, S. J., Paranjape, A. A., Dames, P., Shen, S., & Kumar, V. (2018). A survey on aerial swarm robotics. *IEEE Transactions on robotics*, 34(4), 837-855.
 - [16] Amaral, L. R. D., Zerbato, C., Freitas, R. G. D., Barbosa Júnior, M. R., & Simões, I. O. P. D. S. (2020). UAV applications in Agriculture 4.0. *Revista Ciência Agronômica*, 51, e20207748.
 - [17] Naidoo, Y., Stopforth, R., & Bright, G. (2011, September). Development of an UAV for search & rescue applications. In *IEEE Africon'11* (pp. 1-6). IEEE.
 - [18] Tzanetos, T., Aung, M., Balaram, J., Grip, H. F., Karras, J. T., Canham, T. K., ... & Williford, K. H. (2022, March). Ingenuity mars helicopter: From technology demonstration to extraterrestrial scout. In *2022 IEEE aerospace conference (AERO)* (pp. 01-19). IEEE.
 - [19] OnyxStar Leading Edge Drones, Accessed: Jan. 04, 2026. [Online]. Available: <https://www.onyxstar.net/search-and-rescue-by-drone/>
 - [20] Remondino, F., Barazzetti, L., Nex, F. C., Scaioni, M., & Sarazzi, D. (2011). UAV photogrammetry for mapping and 3D modeling: Current status and future perspectives. In *Proceedings of the international conference on unmanned aerial vehicle in geomatics (UAV-g): 14-16 September 2011, Zurich*,

Switzerland (pp. 25-31). International Society for Photogrammetry and Remote Sensing (ISPRS).

- [21] Holybro, Accessed: Jan. 04, 2026. [Online]. Available: <https://holybro.com/>
- [22] GrabCad, Jan. 04, 2026. [Online]. Available: <https://grabcad.com/library>
- [23] Holybro X650 Frame Kit, Accessed: Jan. 04, 2026. [Online]. Available: <https://holybro.com/products/x650-development-kit>
- [24] Tattu 6S 9000 mAh 25 C Lipo Battery, Accessed: Jan. 04, 2026. [Online]. Available: <https://www.hobby-shop.gr/tattu-9000mah-22.2v-25c-6s1p-lipo-battery-pack/>
- [25] Tekko32 F4 4in1 50A ESC(AM32), Accessed: Jan. 04, 2026. [Online]. Available: <https://holybro.com/products/tekko32-f4-4in1-50a-esc>
- [26] T-Motor MN4014 330KV Brushless Motor for Holybro X650 CW / CCW, Accessed: Jan. 04, 2026. [Online]. Available: <https://www.flyingtech.co.uk/product/t-motor-mn4014-330kv-brushless-motor-for-holybro-x650-cw-ccw/>
- [27] Pixhawk 6X (ICM-45686), Accessed: Jan. 04, 2026. [Online]. Available: <https://holybro.com/collections/flight-controllers/products/pixhawk-6x>
- [28] SiK Telemetry Radio V3, Accessed: Jan. 04, 2026. [Online]. Available: <https://holybro.com/products/sik-telemetry-radio-v3>
- [29] M8N GPS, Accessed: Jan. 04, 2026. [Online]. Available: <https://holybro.com/collections/standard-gps-module/products/m8n-gps>
- [30] Open Source Computer Vision Library (OpenCV), Accessed: Jan. 04, 2026. [Online]. Available: <https://github.com/opencv/opencv>
- [31] MAVLink Developer Guide, Accessed: Jan. 04, 2026. [Online]. Available: <https://mavlink.io/en/>
- [32] Pixhawk Jetson Baseboard Bundle, Accessed: Jan. 04, 2026. [Online]. Available: <https://holybro.com/products/pixhawk-jetson-baseboard>
- [33] Intel® RealSense™ Depth Camera D455, Accessed: Jan. 04, 2026. [Online]. Available: <https://www.intel.com/content/www/us/en/products/sku/205847/intel-realsense-depth-camera-d455/specifications.html>
- [34] RPLIDAR A2, Accessed: Jan. 04, 2026. [Online]. Available: <https://www.slamtec.com/en/lidar/a2>

- [35] TAROT 1555 Foldable Props, Accessed: Jan. 04, 2026. [Online]. Available: <https://www.ariesrc.gr/gr/metal-folding-prop/12788-tarot-1555-foldable-props-ccw.html>
- [36] Cignoni, P., Ranzuglia, G., Callieri, M., Corsini, M., Ganovelli, F., Pietroni, N., & Tarini, M. (2011). MeshLab.
- [37] Ubuntu 24.04 LTS, Accessed: Jan. 04, 2026. [Online]. Available: <https://ubuntu.com/download/desktop>
- [38] ROS 2 Jazzy (Jazzy Jalisco), Accessed: Jan. 04, 2026. [Online]. Available: <https://docs.ros.org/en/jazzy/index.html>
- [39] Gazebo Sim (Harmonic), Accessed: Jan. 04, 2026. [Online]. Available: <https://gazebosim.org/home>
- [40] PX4 Control Stack doc, Accessed: Jan. 04, 2026. [Online]. Available: <https://docs.px4.io/main/en/development/development>
- [41] uXRCE-DDS (PX4-ROS 2/DDS Bridge), Accessed: Jan. 04, 2026. [Online]. Available: https://docs.px4.io/main/en/middleware/uxrce_dds
- [42] QGroundControl, Accessed: Jan. 04, 2026. [Online]. Available: https://docs.qgroundcontrol.com/Stable_V5.0/en/qgc-user-guide/
- [43] RTAB-Map (Real-Time Appearance-Based Mapping), Accessed: Jan. 04, 2026. [Online]. Available: <https://introlab.github.io/rtabmap/>
- [44] Ultralytics YOLOv8 library, Accessed: Jan. 04, 2026. [Online]. Available: <https://docs.ultralytics.com/>
- [45] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).
- [46] Πλαίσια αναφοράς PX4 και ROS 2, Accessed: Jan. 04, 2026. [Online]. Available: https://docs.px4.io/main/en/ros/external_position_estimation
- [47] Rtabmap_viz, Accessed: Jan. 04, 2026. [Online]. Available: https://wiki.ros.org/rtabmap_viz

Παράρτημα Α: Διαδικασία Εγκατάσταση Λογισμικού

A.1 Εγκατάσταση Λειτουργικού Συστήματος Ubuntu

1. Εγκατάσταση Ubuntu 24.04 LTS (Long Term Support)

Σύνδεσμος: <https://ubuntu.com/download/desktop>

A.2 Εγκατάσταση ROS2 και όλα τα απαραίτητα πακέτα

1. Σύνδεσμος; ROS2 Jazzy: <https://docs.ros.org/en/jazzy/Installation/Ubuntu-Install-Debs.html>

2. `sudo apt update && sudo apt upgrade`
`sudo apt-get update && sudo apt-get install wmctrl && sudo apt-get install xdotool`

3. Εγκατάσταση απαραίτητων προγραμμάτων και πακέτων ROS 2:

```
sudo apt install ros-$ROS_DISTRO-rqt
sudo apt install ros-$ROS_DISTRO-xacro
sudo apt install -y ros-$ROS_DISTRO-urdf-tutorial
sudo apt install ros-jazzy-tf2-tools
sudo apt install ros-jazzy-message-tf-frame-transformer
sudo apt install ros-jazzy-odom-to-tf-ros2
sudo apt install ros-jazzy-joy ros-jazzy-joy ros-jazzy-joy-teleop
Σύνδεσμος: https://pypi.org/project/pynput/
pip install pynput για teleop με πληκτρολ
```

A.3 Εγκατάσταση Gazebo Sim:

1. `sudo apt-get install ros-$ROS_DISTRO-ros-gz`
2. `sudo apt install gz-sim8-cli`
3. Δοκιμή του προσομοιωτή: `gz sim`.

Προσοχή: Αν υπάρξει πρόβλημα στην εκτέλεση του Gazebo, τότε εφαρμόζετε η απόδοση λογισμικού αντί της απόδοσης υλικού με έναν από τους τρόπους:

```
echo 'export LIBGL_ALWAYS_SOFTWARE=1' >> ~/.bashrc
```

```
source ~/.bashrc
```

Αν δεν λυθεί το παραπάνω το πρόβλημα, τότε:

```
echo 'export QT_QPA_PLATFORM=xcb' >> ~/.bashrc
```

```
source ~/.bashrc
```

4. Εγκατάσταση πρόσθετων και απαραίτητων πακέτων για το Gazebo:

```
sudo apt update
```

```
sudo apt install ros-$ROS_DISTRO-ros-gz-bridge \
```

```
ros-$ROS_DISTRO-ros-gz-sim \
```

```
ros-$ROS_DISTRO-ros-gz-image \
```

```
ros-$ROS_DISTRO-ros-gz-sim-demos
```

5. Δοκιμή εκτέλεσης με το όχημα και ένα περιβάλλον:

```
export GZ_SIM_RESOURCE_PATH=$HOME/.gz/models:$HOME/.simulation-  
gazebo/models
```

- Για την προσομοίωση στο Gazebo πρέπει να προσαρμοστούν οι ρυθμίσεις για την κάμερα και του αισθητήρα LiDAR (δηλαδή όχι οι πραγματικές ιδιότητες τους), έτσι ώστε το `real_time_factor` να μη πέφτει κάτω από 0.9.
- Για να εκτελείτε η προσομοίωση στο Gazebo με υψηλές ταχύτητες ορίζετε η χρήση της RTX 4080 κάρτα γραφικών. Έτσι δε θα υπάρχουν προβλήματα με μεγάλα STL αρχεία για την αναπαράσταση.

```
sudo prime-select nvidia
```

 και μετά επανεκκίνηση.

A.4 Εγκατάσταση PX4-Autopilot

1. Δημιουργία φακέλου για την αποθήκευση του πακέτου.

```
git clone https://github.com/PX4/PX4-Autopilot.git --recursive
```

```
bash ./PX4-Autopilot/Tools/setup/ubuntu.sh
```

```
cd PX4-Autopilot/
```

```
make px4_sitl
```

```
make px4_sitl gz_x500 -> αυτόματο άνοιγμα SITL με το όχημα X500.
```

Σημείωση: Για την επαναδημιουργία του σωστού οχήματος, εκτελείτε η εντολή

```
make px4_sitl gz_x650.
```

A.5 Εγκατάσταση του QGroundControl

1. Σύνδεσμος: https://docs.qgroundcontrol.com/Stable_V5.0/en/qgc-user-guide/getting_started/download_and_install.html

2. Λήψη του QGroundControl.AppImage

```
cd ~/Downloads
```

```
chmod +x QGroundControl-x86_64.AppImage
```

```
./QGroundControl.AppImage
```

Δοκιμή διαφόρων λειτουργιών όπως arm, take off ...

Αν το όχημα ανταποκρίνεται σωστά, συνέχεια με το

- i. Κάνουμε το μοντέλο x500 από τον φάκελο PX4-Autopilot/Tools/simulation/gz/models/x500 αντιγραφή και το μετονομάζουμε σε x650.
- ii. Επίσης αντιγράφουμε και το ROMF του x500, δίνουμε όνομα και έναν νέο κωδικό: PX4-Autopilot/ROMFS/px4fmu_common/init.d-posix/airframes/**4229_gz_x650**.
- iii. `make px4_sitl gz_x500` -> Η εντολή ξεκινάει τη μεταγλώττιση (compile) και κάνει αντιγραφή όλα τα όχηματα από το ROMFS/px4fmu_common/init.d-posix/airframes/ σε build/px4_sitl_default/rootfs/etc/init.d-posix/airframes/
- iv. Έλεγχος αν έχει εκτελεστεί σωστά:
- v. `ls build/px4_sitl_default/rootfs/etc/init.d-posix/airframes | grep 4229`
- vi. Αν δεν έχει γίνει η αντιγραφή τότε την κάνουμε χειροκίνητη:

Από το root PX4-Autopilot root:

```
cp ROMFS/px4fmu_common/init.d-posix/airframes/4229_gz_x650 \
build/px4_sitl_default/rootfs/etc/init.d-posix/airframes/
```

Στο αρχείο ~/.bashrc πρέπει να ορίσουμε που να βρει το Gazebo τα μοντέλα, γιατί αν το βάλουμε στο launch αρχείο, τότε θα τρέχει μόνο μέσα στην διεργασία (thread) του και όχι στο σύστημα (global) :

```
export
```

```
GZ_SIM_RESOURCE_PATH=$HOME/.gz/models:$HOME/.simulation-gazebo/models
```

Εκτελούμε: \$ PX4_SYS_AUTOSTART=4229

PX4_GZ_MODEL=x650 \

build/px4_sitl_default/bin/px4

για να δούμε αν κάναμε όλα σωστά. Στο Gazebo θα πρέπει να εμφανίζετε πλέον το δικό μας όχημα x650 και όχι το x550.

A.6 Εγκαθίδρυση Επικοινωνίας μεταξύ ROS2 και PX4

1. Σύνδεσμος: <https://github.com/eProsima/Micro-XRCE-DDS-Agent/tree/ros2>
2. Αλλαγή στον φάκελο external του έργου uav_project_ws:

```
git clone https://github.com/PX4/px4_msgs.git
```

```
git clone https://github.com/PX4/px4_ros_com.git
```

```
git clone https://github.com/eProsima/Micro-XRCE-DDS-Agent.git
```

```
cd Micro-XRCE-DDS-Agent
```

```
mkdir build && cd build && cmake ..
```

```
make
```

```
sudo make install
```

```
sudo ldconfig /usr/local/lib/
```

A.7 Εγκατάσταση Computer Vision package

1. `sudo apt-get install ros-jazzy-vision-opencv`
`sudo apt install ros-jazzy-cv-bridge -> Internal Library`
`sudo apt install ros-jazzy-ros-gz-image -> External Bridge`
`sudo apt install python3-pip`
2. Αν προκύψει το εξής πρόβλημα: externally-managed-environment (with caution)
οι λύσεις παρουσιάζονται στα εξής βίντεο:
<https://www.youtube.com/watch?v=g2TDfWDgwKE>
<https://stackoverflow.com/questions/38869231/python-cant-find-the-file-pip-conf>
<https://stackoverflow.com/questions/75608323/how-do-i-solve-error-externally-managed-environment-every-time-i-use-pip-3>
3. `pip3 install ultralytics -> YOLO8`
4. Επιδιόρθωση προβλήματος συμβατότητας:

```
pip3 install "numpy<2" "opencv-python<=4.10.0.84" "ultralytics<=8.2.70" --  
break-system-packages --force-reinstall  
pip3 install "setuptools<80" --break-system-packages
```

Μετά την εγκατάσταση ενεργοποιείτε ξανά το break-system-packages (setting again to false)

A.8 Εγκατάσταση RTAB Map

1.

```
sudo apt install ros-$ROS_DISTRO-rtabmap-ros* \  
ros-$ROS_DISTRO-librealsense2 \  
ros-$ROS_DISTRO-realsense2-* \  
ros-$ROS_DISTRO-stereo-image-proc  
sudo apt-get install libpcl-dev  
sudo apt-get install ros-$ROS_DISTRO-pcl-*  
sudo apt-get install ros-$ROS_DISTRO-rqt-image-view  
sudo apt install ros-$ROS_DISTRO-rtabmap-rviz-*  
sudo apt install ros-${ROS_DISTRO}-rtabmap-ros  
sudo apt install ros-${ROS_DISTRO}-rtabmap-viz  
git clone -b ros2 https://github.com/introlab/rtabmap_ros.git -> Για το πακέτο  
rtabmap_costmap_plugins  
sudo apt update sudo apt install -y \  
ros-jazzy-grid-map-ros \  
ros-jazzy-ros2-aruco-interfaces
```

Παράρτημα Β: Οδηγίες Αναπαραγωγή Προσομοίωσης

1. Άνοιγμα Τερματικού (Terminal): Μετάβαση σε φάκελο της επιλογής για την κλωνοποίηση.
2. Κλωνοποίηση του αποθετηρίου:
`git clone https://github.com/michailtam/uav-mapping-and-ai-human-detection.git`
3. `cd uav-mapping-and-ai-human-detection -> git submodule update --init --recursive`
4. Εγκατάσταση εξαρτήσεων (dependencies) πακέτων
`rosdep install -i --from-path src --rosdistro jazzy -y`
5. Τερματικό 1:
 - i. Δημιουργία χώρου εργασίας (workspace): `colcon build`.
 - ii. Ενεργοποίηση χώρου εργασίας: `source install/setup.bash`
 1. Εκτέλεση αρχείου shell script: `./src/uav_bringup/sh/open_ros_terminals.sh`
6. Τερματικό 2:
 - i. Ενεργοποίηση χώρου εργασίας: `source install/setup.bash`
 - ii. Εκτέλεση αρχείου εκκίνησης: `ros2 launch uav_offboard_ctrl offboard_ctrl.launch.py`
7. Τερματικό 3:
 - i. Ενεργοποίηση χώρου εργασίας: `source install/setup.bash`
 - ii. Εκτέλεση κόμβου: `ros2 run uav_navigation keyboard_teleop.py`

Παράρτημα Γ: Πηγαίος Κώδικας και Αρχεία Διαμόρφωσης

Στο παρόν παράρτημα παρατίθεται ο πηγαίος κώδικας των σεναρίων εκκίνησης (launch files) καθώς και των κόμβων που αναπτύχθηκαν σε γλώσσες C++ και Python. Λόγω του μεγάλου όγκου των δεδομένων, παρουσιάζονται τα σημαντικότερα τμήματα του κώδικα που καθορίζουν τη λογική του συστήματος. Ο πλήρης και λειτουργικός πηγαίος κώδικας του έργου, συμπεριλαμβανομένων των οδηγιών εγκατάστασης και των αρχείων παραμετροποίησης, είναι διαθέσιμος στην πλατφόρμα GitHub στην εξής διεύθυνση:

<https://github.com/michailtam/uav-mapping-and-ai-human-detection>

Γ.1 Πηγαίος Κώδικας Πακέτου (uav_description)

Γ.1.1 Αρχείο Εκκίνησης (display.launch.py)

```
import os
import xacro
from launch_ros.actions import Node
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch.conditions import IfCondition
from ament_index_python.packages import get_package_share_directory

def generate_launch_description():

    # Package and file paths
    pkg_uav_description_share_dir = get_package_share_directory('uav_description')
    rviz_config_file = os.path.join(pkg_uav_description_share_dir, 'rviz',
    'default_config.rviz')

    # Create the robot_description from xacro file
    xacro_file = os.path.join(pkg_uav_description_share_dir, 'urdf',
    'uav.urdf.xacro')
    robot_description_config = xacro.process_file(xacro_file)
    robot_urdf = robot_description_config.toxml()

    # Declare launch arguments and default parameters
    use_rviz = LaunchConfiguration('use_rviz'),
    use_sim_time = LaunchConfiguration('use_sim_time')

    use_rviz_decl = DeclareLaunchArgument(
        name='use_rviz',
        default_value='false'
    )

    use_sim_time_arg = DeclareLaunchArgument(
        name="use_sim_time",
        default_value='true'
```

```

)

# Launch the robot-state-publisher node
robot_state_publisher_node = Node(
    package='robot_state_publisher',
    executable='robot_state_publisher',
    name='robot_state_publisher',
    parameters=[{
        'use_sim_time': use_sim_time,
        'robot_description': robot_urdf
    }])

# Launch RViz node
rviz_node = Node(
    condition=IfCondition(use_rviz),
    package='rviz2',
    executable='rviz2',
    output='screen',
    arguments=['-d', rviz_config_file],
    parameters=[{"use_sim_time": use_sim_time}]
)

ld = LaunchDescription()
ld.add_action(use_rviz_decl)
ld.add_action(use_sim_time_arg)
ld.add_action(robot_state_publisher_node)
ld.add_action(rviz_node)

return ld

```

Γ.2 Πηγαίος Κώδικας Πακέτου (uav_simulation)

Γ.2.1 Αρχείο Εκκίνησης (ros_gz_bridge.launch.py)

```

from launch import LaunchDescription
from launch.substitutions import PathJoinSubstitution
from launch_ros.actions import Node
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from ament_index_python.packages import get_package_share_directory

def generate_launch_description():

    # Package and file paths
    pkg_share_uav_simulation = get_package_share_directory('uav_simulation')

    # Set default launch arguments
    ros_gz_bridge_config = PathJoinSubstitution([
        pkg_share_uav_simulation,
        'config',
        'ros_gz_bridge.yaml'])

    # Bridge ROS topics and Gazebo messages for establishing communication
    gazebo_ros_bridge_cmd = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        name='gazebo_bridge',
        parameters=[{
            'config_file': ros_gz_bridge_config,

```

```

        'use_sim_time': True
    }],
    output='screen'
)

# Convert odometry message to TF transform
# Publishes dynamic TF: /odom -> /base_link
odom_to_tf_node = Node(
    package='odom_to_tf_ros2',
    executable='odom_to_tf',
    name='odom_to_tf',
    output='screen',
    parameters=[{'use_sim_time': True}],
    remappings=[('odom/perfect', '/odom')])

# Propeller joint states publisher
offboard_ctrl_node = Node(
    package='uav_offboard_ctrl',
    executable='px4_prop_js_publisher',
    name='px4_prop_js_publisher',
    output='screen',
    parameters=[{'use_sim_time': True}])

ld = LaunchDescription()
ld.add_action(gazebo_ros_bridge_cmd)
ld.add_action(odom_to_tf_node)
ld.add_action(offboard_ctrl_node)

return ld

```

Γ.3 Πηγαίος Κώδικας Πακέτου (uav_offboard_control)

Γ.3.1 Αρχείο Εκκίνησης (px4_autopilot.launch.py)

```

from launch import LaunchDescription
from launch.actions import ExecuteProcess

def generate_launch_description():

    # Execute the simulation with px4 flight controller
    px4_autopilot_proc = ExecuteProcess(
        cmd=['./external/PX4-Autopilot/build/px4_sitl_default/bin/px4'],
        output='screen',
        shell=True
    )

    ld = LaunchDescription()
    ld.add_action(px4_autopilot_proc)
    return ld

```

Γ.3.2 Αρχείο Εκκίνησης (xrce_dds_agent.launch.py)

```

from launch import LaunchDescription
from launch.actions import ExecuteProcess, RegisterEventHandler
from launch.event_handlers import OnShutdown

def generate_launch_description():

```

```

# Start the XRCE-DDS agent to establish communication between px4 and ros2
agent_proc = ExecuteProcess(
    cmd=['MicroXRCEAgent', 'udp4', '-p', '8888'],
    output='screen',
    shell=False
)

# Stop the XRCE-DDS agent before shutdown
kill_agent_on_shutdown_proc = RegisterEventHandler(
    event_handler=OnShutdown(
        on_shutdown=[ExecuteProcess(
            cmd=['pkill', '-f', 'MicroXRCEAgent'],
            output='screen',
            shell=False)]
    )
)

ld = LaunchDescription()
ld.add_action(agent_proc)
ld.add_action(kill_agent_on_shutdown_proc)
return ld

```

Γ.3.3 Αρχείο Εκκίνησης (start_px4_js_publisher.launch.py)

```

import rclpy
from launch import LaunchDescription
from launch.actions import OpaqueFunction
from launch_ros.actions import Node
from rclpy.node import Node as RclNode

def launch_px4_prop_js_pub(context, *args, **kwargs):
    # Use a tiny rclpy node inside the launch to wait for the PX4 topic
    # /fmu/out/actuator_motors.
    rclpy.init(args=None)
    tmp_node = RclNode('wait_for_px4_actuator_topic')
    target_topic = '/fmu/out/actuator_motors'
    tmp_node.get_logger().info(f'Waiting for topic: {target_topic}')

    found = False
    while not found:
        topics = tmp_node.get_topic_names_and_types()
        found = any(name == target_topic for name, _ in topics)
        if not found:
            rclpy.spin_once(tmp_node, timeout_sec=0.5)

    tmp_node.get_logger().info(
        f'PX4 actuator topic detected -> launching px4_prop_js_publisher and
px4_ctrl1'
    )
    tmp_node.destroy_node()
    rclpy.shutdown()

    # Start publishing joint states delayed from PX4 (not from Gazebo)
    # Note: This is required for RViz to display propellers and rotating on place.
    px4_prop_joint_states_pub_node = Node(
        package='uav_offboard_ctrl',
        executable='px4_prop_js_publisher', # Be sure this matches CMakeLists
setting
        name='px4_prop_js_publisher',
        parameters=[{
            'use_sim_time': True
        }],

```

```

        output='screen'
    )
    # IMPORTANT: OpaqueFunction must return the actions, not a LaunchDescription
    return [
        px4_prop_joint_states_pub_node,
    ]

def generate_launch_description():
    ld = LaunchDescription()

    # Add the OpaqueFunction that waits for PX4 and launches the two nodes
    ld.add_action(OpaqueFunction(function=launch_px4_prop_js_pub))

    return ld

```

Γ.3.4 Αρχείο Εκκίνησης (offboard_ctrl.launch.py)

```

from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():

    # Start the FCU control
    px4_ctrl_node = Node(
        package='uav_offboard_ctrl',
        executable='px4_ctrl',
        name='px4_ctrl',
        parameters=[{
            'use_sim_time': True
        }],
        output='screen'
    )

    ld = LaunchDescription()
    ld.add_action(px4_ctrl_node)

    return ld

```

Γ.3.5 Αρχείο Κεφαλίδας (px4_ctrl.h)

```

#ifndef FCU_CONTROL_H
#define FCU_CONTROL_H

#include <rclcpp/rclcpp.hpp>
#include <px4_msgs/msg/offboard_control_mode.hpp>
#include <px4_msgs/msg/trajectory_setpoint.hpp>
#include <px4_msgs/msg/vehicle_control_mode.hpp>
#include <px4_msgs/msg/vehicle_local_position.hpp>
#include <px4_msgs/msg/vehicle_global_position.hpp>
#include <px4_msgs/msg/position_setpoint_triplet.hpp>
#include <px4_msgs/msg/vehicle_status.hpp>
#include <px4_msgs/msg/home_position.hpp>
#include <px4_msgs/msg/vehicle_attitude.hpp>
#include <px4_msgs/srv/vehicle_command.hpp>
#include <px4_msgs/msg/battery_status.hpp>
#include <GeographicLib/LocalCartesian.hpp>
#include <geometry_msgs/msg/pose_stamped.hpp>
#include <tf2_geometry_msgs/tf2_geometry_msgs.hpp>
#include <tf2/LinearMath/Quaternion.hpp>
#include <geometry_msgs/msg/pose_stamped.hpp>

```

```

#include <geometry_msgs/msg/twist.hpp>
#include <std_msgs/msg/int32.hpp>
#include <sensor_msgs/msg/laser_scan.hpp>
#include <cmath>

using namespace std::chrono;
using namespace std::chrono_literals;
using namespace px4_msgs::msg;
using namespace std;

struct PosInENU {
    /**
     * @brief Stores the position ENU coordinates (for Gazebo).
     */
    double x, y, z, yaw;
};

struct PosInNED {
    /**
     * @brief Stores the position NED coordinates (for PX4).
     */
    double x, y, z, yaw;
};

struct PosInWGS84 {
    /**
     * @brief Stores position WGS84 coordinates.
     */
    double lat, lon, alt;
};

struct Velocity {
    /**
     * @brief Stores the linear and angular velocity
     */
    float linear_x, linear_y, linear_z, angular_z;
};

struct ScanDistance {
    /**
     * @brief Stores the front, right, back and left distances (if obstacle
    detected)
     */
    float front, right, back, left;
};

class OffboardControl : public rclcpp::Node
{
public:
    OffboardControl();
    void engageOffboardMode();
    void arm();
    void disarm();
    void takeOff();
    void hover();
    void land();
    void RTL();
    void teleop();
    void run();
    void localPoseCallback(const px4_msgs::msg::VehicleLocalPosition::SharedPtr
msg);

```

```

    void globalPoseCallback(const px4_msgs::msg::VehicleGlobalPosition::SharedPtr
msg);
    void statusCallback(const px4_msgs::msg::VehicleStatus::SharedPtr msg);
    void vehicleAttitudeCallback(const px4_msgs::msg::VehicleAttitude::SharedPtr
msg);
    void velocityCallback(const geometry_msgs::msg::Twist::SharedPtr msg);
    void teleopArmedCallback(const std_msgs::msg::Int32::SharedPtr msg);
    void batteryStatusCallback(const px4_msgs::msg::BatteryStatus::SharedPtr msg);

private:
    void publishOffboardControlMode();
    void publishTrajectorySetpoint(float pos_x, float pos_y, float pos_z, float
pos_yaw);
    void sendVehicleCommand(uint16_t command, float param1, float param2);
    void srvCallback(rclcpp::Client<px4_msgs::srv::VehicleCommand>::SharedFuture
future);
    void homePoseCallback(px4_msgs::msg::HomePosition::SharedPtr msg);
    void goalPoseCallback(px4_msgs::msg::PositionSetpointTriplet::SharedPtr msg);
    void timerUpdateStateMachine(void);

    enum class State{
        INIT_MODE,           // Initialization Mode
        OFFBOARD_MODE,       // Flight Mode
        ARM_MODE,            // Flight Mode
        DISARM_MODE,         // Flight Mode
        TAKEOFF_MODE,        // Flight Mode
        HOVER_MODE,          // Flight Mode
        POSITION_MODE,        // Flight Mode
        LAND_MODE,           // Flight Mode
        RTL_MODE,            // Flight Mode (Return To Land)
        TELEOP_MODE          // Teleoperation Mode
    } state_;

    uint8_t service_result_;
    bool service_done_;
    float take_off_height_;
    rclcpp::TimerBase::SharedPtr timer_;
    std::vector<PosInENU> trajectory_;
    size_t trajectory_index_ = 0;
    bool msg_logged_ = false;
    bool teleop_armed_ = false;
    float true_yaw_ = 0.0; // Current yaw value of drone
    float cmd_yaw_ = 0.0; // The yaw value send as command
    float battery_status_perc_; // Saves the battery status in percent
    bool key_pressed_ = false; // Determines if a key for velocity was pressed by
the teleop node

    px4_msgs::msg::VehicleStatus::SharedPtr vehicle_status_msg_;

    rclcpp::Publisher<OffboardControlMode>::SharedPtr offboard_ctrl_mode_pub_;
    rclcpp::Publisher<TrajectorySetpoint>::SharedPtr traj_setpoint_pub_;
    rclcpp::Subscription<px4_msgs::msg::HomePosition>::SharedPtr uav_home_pos_sub_;
    rclcpp::Subscription<px4_msgs::msg::VehicleLocalPosition>::SharedPtr
uav_local_pos_sub_;
    rclcpp::Subscription<px4_msgs::msg::VehicleGlobalPosition>::SharedPtr
uav_global_pos_sub_;
    rclcpp::Subscription<px4_msgs::msg::PositionSetpointTriplet>::SharedPtr
uav_target_pos_sub_;
    rclcpp::Subscription<px4_msgs::msg::VehicleStatus>::SharedPtr uav_status_sub_;
    rclcpp::Subscription<px4_msgs::msg::VehicleAttitude>::SharedPtr
uav_attitude_sub_;

```

```

    rclcpp::Subscription<px4_msgs::msg::BatteryStatus>::SharedPtr
    battery_status_sub_;

    // Twist message subscribers for teleop
    rclcpp::Subscription<geometry_msgs::msg::Twist>::SharedPtr uav_velocity_sub_;
    rclcpp::Subscription<std_msgs::msg::Int32>::SharedPtr teleop_armed_sub_;

    // Service for vehicle commands
    rclcpp::Client<px4_msgs::srv::VehicleCommand>::SharedPtr vehicle_cmd_client_;

    PosInENU curr_loc_;           // Local position coordinates in NED
    PosInWGS84 curr_glob_;        // Global position coordinates in WGS84
    PosInENU target_loc_;         // Local target position coordinates in NED
    PosInWGS84 target_glob_;      // Global target position coordinates in NED
    PosInENU home_loc_;           // Local home position coordinates in NED
    PosInWGS84 home_glob_;        // Global home position coordinates in NED
    PosInENU hover_pos_loc_;      // Fixed hover position
    Velocity velocity_;           // The linear and angular velocities to publish to
PX4
};

#endif

```

Γ.3.6 Εκτέλεση Αρχείο (px4_ctrl.cpp)

```

#include "uav_offboard_ctrl/px4_ctrl.h"

#define DEG_TO_RAD 0.01745329251

OffboardControl::OffboardControl() : Node("offboard_ctrl"),
    /**
     * @brief
     */
    state_{State::INIT_MODE},    // Initial state at startup
    service_result_{0},          // State of the service request
    service_done_{false},        // Flag that indicates if the service request has
succeeded or not
    take_off_height_{-5.0},      // NED Coord system i.e. z is pointing downwards
but in ROS is facing upwards
    battery_status_perc_{1.0}    // Batter status 100%
    {

    // Create the service for changing the control mode
    RCLCPP_INFO(this->get_logger(), "Starting Offboard Control with PX4 services");
    vehicle_cmd_client_ = this->
>create_client<px4_msgs::srv::VehicleCommand>("/fmu/vehicle_command");

    // Before proceeding wait until the service is available
    while (!vehicle_cmd_client_->wait_for_service(1s)) {
        if (!rclcpp::ok()) {
            RCLCPP_ERROR(this->get_logger(), "Interrupted while waiting for the
service. Exiting.");
            return;
        }
        RCLCPP_INFO(this->get_logger(), "Service not available, waiting again...");
    }

    timer_ = this->create_wall_timer(100ms,
std::bind(&OffboardControl::timerUpdateStateMachine, this)); // 10 Hz

    // To correctly receive data, set the quality of service

```

```

    rclcpp::QoS
    qos_profile(rclcpp::QoSInitialization::from_rmw(rmw_qos_profile_default));

    // Setting the QoS parameters
    qos_profile.reliability(RMW_QOS_POLICY_RELIABILITY_BEST_EFFORT);
    qos_profile.durability(RMW_QOS_POLICY_DURABILITY_TRANSIENT_LOCAL);
    qos_profile.liveliness(RMW_QOS_POLICY_LIVELINESS_AUTOMATIC);
    qos_profile.history(RMW_QOS_POLICY_HISTORY_KEEP_LAST);
    qos_profile.keep_last(10);

    // ** PX4 Publishers **
    // Create the offboard control mode and trajectory publisher
    offboard_ctrl_mode_pub_ = this-
>create_publisher<px4_msgs::msg::OffboardControlMode>("/fmu/in/offboard_control_mode", 10);
    traj_setpoint_pub_ = this-
>create_publisher<px4_msgs::msg::TrajectorySetpoint>("/fmu/in/trajectory_setpoint", 10);

    // ** PX4 Subscribers **
    uav_home_pos_sub_ = this-
>create_subscription<px4_msgs::msg::HomePosition>("/fmu/out/home_position_v1",
    qos_profile,
        std::bind(&OffboardControl::homePoseCallback, this,
std::placeholders::_1));
    uav_local_pos_sub_ = this-
>create_subscription<px4_msgs::msg::VehicleLocalPosition>("/fmu/out/vehicle_local_position_v1", qos_profile, // QoS of 10
        std::bind(&OffboardControl::localPoseCallback, this,
std::placeholders::_1));
    uav_global_pos_sub_ = this-
>create_subscription<px4_msgs::msg::VehicleGlobalPosition>("/fmu/out/vehicle_global_position", qos_profile, // QoS of 10
        std::bind(&OffboardControl::globalPoseCallback, this,
std::placeholders::_1));
    uav_target_pos_sub_ = this-
>create_subscription<px4_msgs::msg::PositionSetpointTriplet>("/fmu/out/position_setpoint_triplet", qos_profile,
        std::bind(&OffboardControl::goalPoseCallback, this,
std::placeholders::_1));
    uav_status_sub_ = this-
>create_subscription<px4_msgs::msg::VehicleStatus>("/fmu/out/vehicle_status_v1",
    qos_profile,
        std::bind(&OffboardControl::statusCallback, this,
std::placeholders::_1));
    uav_attitude_sub_ = this-
>create_subscription<px4_msgs::msg::VehicleAttitude>("/fmu/out/vehicle_attitude",
    qos_profile,
        std::bind(&OffboardControl::vehicleAttitudeCallback, this,
std::placeholders::_1));

    // Subscribers and Publishers for vehicle management
    uav_velocity_sub_ = this-
>create_subscription<geometry_msgs::msg::Twist>("/offboard_velocity_cmd",
    qos_profile,
        std::bind(&OffboardControl::velocityCallback, this,
std::placeholders::_1));
    teleop_armed_sub_ = this-
>create_subscription<std_msgs::msg::Int32>("/teleop_command", qos_profile,
        std::bind(&OffboardControl::teleopArmedCallback, this,
std::placeholders::_1));

```

```

        battery_status_sub_ = this->
>create_subscription<px4_msgs::msg::BatteryStatus>("/fmu/out/battery_status_v1",
qos_profile,
        std::bind(&OffboardControl::batteryStatusCallback, this,
std::placeholders::_1));
    }

void
OffboardControl::srvCallback(rclcpp::Client<px4_msgs::srv::VehicleCommand>::SharedF
uture future) {
    /**
     * @brief Callback function which waits until the service has completed with
"Command accepted",
     * this means that the requested command was executed.
     */
    auto status = future.wait_for(1s);
    if (status == std::future_status::ready)
    {
        auto reply = future.get()->reply;
        uint8_t service_result_ = reply.result;
        RCLCPP_INFO(this->get_logger(), "Service replied successfully with %d",
service_result_);

        switch (service_result_)
        {
            case reply.VEHICLE_CMD_RESULT_ACCEPTED:
                RCLCPP_INFO(this->get_logger(), "Command accepted");
                break;
            case reply.VEHICLE_CMD_RESULT_TEMPORARILY_REJECTED:
                RCLCPP_WARN(this->get_logger(), "Command temporarily rejected");
                break;
            case reply.VEHICLE_CMD_RESULT_DENIED:
                RCLCPP_WARN(this->get_logger(), "Command denied");
                break;
            case reply.VEHICLE_CMD_RESULT_UNSUPPORTED:
                RCLCPP_WARN(this->get_logger(), "Command unsupported");
                break;
            case reply.VEHICLE_CMD_RESULT_FAILED:
                RCLCPP_WARN(this->get_logger(), "Command failed");
                break;
            case reply.VEHICLE_CMD_RESULT_IN_PROGRESS:
                RCLCPP_WARN(this->get_logger(), "Command in progress");
                break;
            case reply.VEHICLE_CMD_RESULT_CANCELLED:
                RCLCPP_WARN(this->get_logger(), "Command cancelled");
                break;
            default:
                RCLCPP_WARN(this->get_logger(), "Command reply unknown");
                break;
        }
        service_done_ = true;
    }
    else {
        RCLCPP_INFO(this->get_logger(), "Service In-Progress...");
    }
}

void OffboardControl::engageOffboardMode() {
    /**
     * @brief Send the command to switch to offboard mode.
     * This mode is required to handle the drone by code.
     */

```

```

RCLCPP_INFO(this->get_logger(), "Request switching to Offboard mode");
sendVehicleCommand(VehicleCommand::VEHICLE_CMD_DO_SET_MODE, 1, 6);
}

void OffboardControl::publishOffboardControlMode() {
    /**
     * @brief Change to offboard mode with the desired parameters.
     */
    OffboardControlMode mode{};
    if(state_ == State::TAKEOFF_MODE || state_ == State::LAND_MODE) {
        // Pure position control for stability
        mode.position = true; // Means: Take my data and try to reach the
        // position with the help of your sensors
        mode.velocity = false; // Means: Don't hear anything about velocity,
        // ignore it
    } else if (key_pressed_) { // If a key for velocity sending was pressed on
        // teleop node
        // Direct velocity control
        mode.position = false;
        mode.velocity = true;
    } else { // Direct position control
        mode.position = true;
        mode.velocity = false;
    }
    mode.acceleration = false;
    mode.attitude = false;
    mode.body_rate = false;
    mode.timestamp = this->get_clock()->now().nanoseconds() / 1000;
    offboard_ctrl_mode_pub_->publish(mode);
}

void OffboardControl::publishTrajectorySetpoint(float pos_x, float pos_y, float
pos_z, float pos_yaw) {
    /**
     * @brief Publish a trajectory setpoint (i.e. the position to fly to).
     */

    // Compute velocity from Body Frame (FLU) to World Frame (ENU)
    float cos_yaw = std::cos(true_yaw_);
    float sin_yaw = std::sin(true_yaw_);
    float vel_global_x = (velocity_.linear_x * cos_yaw - velocity_.linear_y *
sin_yaw);
    float vel_global_y = (velocity_.linear_x * sin_yaw + velocity_.linear_y *
cos_yaw);

    TrajectorySetpoint traj_msg{};

    // Check if position or velocity mode is desired
    if (std::isnan(pos_x) && key_pressed_) { // VELOCITY MODE
        traj_msg.position = {std::nanf(""), std::nanf(""), std::nanf("")};
        traj_msg.velocity = {vel_global_x, vel_global_y, velocity_.linear_z};
        traj_msg.yaw = std::nanf(""); // Must be NaN to use yaw speed
        traj_msg.yawspeed = cmd_yaw_; // Controlling turn rate
    } else if (!key_pressed_) { // POSITION MODE
        traj_msg.position = {pos_x, pos_y, pos_z};
        traj_msg.velocity = {std::nanf(""), std::nanf(""), std::nanf("")};
        traj_msg.yaw = pos_yaw;
        traj_msg.yawspeed = std::nanf("");
    }

    traj_msg.timestamp = this->get_clock()->now().nanoseconds() / 1000;
    traj_setpoint_pub_->publish(traj_msg);
}

```

```

        key_pressed_ = false;
    }

void OffboardControl::sendVehicleCommand(uint16_t command, float param1, float
param2) {
    /**
     * @brief Publish vehicle commands
     * @param command    Command code (matches VehicleCommand and MAVLink MAV_CMD
codes)
     * @param param1     Command parameter 1
     * @param param2     Command parameter 2
     */
    auto req = std::make_shared<px4_msgs::srv::VehicleCommand::Request>();

    // Populate the command parameters to send
    VehicleCommand cmd{};
    cmd.param1 = param1;
    cmd.param2 = param2;
    cmd.command = command;      // Command ID
    cmd.target_system = 1;      // System which executes the command
    cmd.target_component = 1;    // Component which should execute the command, 0
for all components
    cmd.source_system = 1;      // System which sends the command
    cmd.source_component = 1;    // Component which sends the command
    cmd.from_external = true;
    cmd.timestamp = this->get_clock()->now().nanoseconds() / 1000;
    req->request = cmd;

    // Callback function, which is triggered when the service response is ready
    service_done_ = false;
    auto result = vehicle_cmd_client->async_send_request(req,
std::bind(&OffboardControl::srvCallback,
        this, std::placeholders::_1));
    RCLCPP_INFO(this->get_logger(), "Command send");
}

void OffboardControl::velocityCallback(const geometry_msgs::msg::Twist::SharedPtr
msg) {
    /**
     * @brief Receives the twist commands from Teleop and converts NED -> FLU.
     */
    if (teleop_armed_) {
        // NED -> FLU Transformation. FLU is the body frame coordinate system of
the drone (Front-Left-Up)
        velocity_.linear_x = msg->linear.x;    // Forward/Backward - X (FLU) is -Y
(NED)
        velocity_.linear_y = msg->linear.y;    // Left/Right - Y (FLU) is X (NED)
        velocity_.linear_z = -msg->linear.z;    // Convert ROS "Up" (+) to PX4
"Down" (-) - Z (FLU) is -Z (NED)
        // A conversion for angular z is done in the vehicleAttitudeCallback method
(it's the '-' in front of true_yaw)
        cmd_yaw_ = msg->angular.z;
        key_pressed_ = true;
    }
}

void OffboardControl::timerUpdateStateMachine(void) {
    /**
     * @brief Finite-State-Machine (FSM) executed in specific time intervals.
     */
    static uint8_t num_of_steps = 0; // Counter to engage the offboard mode

```

```

// Note: Always publish OffboardControlMode
publishOffboardControlMode();

switch (state_)
{
case State::INIT_MODE:
    // Check for teleop mode to start arming the vehicle.
    if(teleop_armed_) {
        engageOffboardMode();
        state_ = State::OFFBOARD_MODE;
    }
    break;
case State::OFFBOARD_MODE:
    if(service_done_) {
        // Wait 10 steps (warming up offboard mode) before engaging the arm
mode
        if (++num_of_steps > 10) {
            RCLCPP_INFO(this->get_logger(), "Entered offboard mode");
            RCLCPP_INFO(this->get_logger(), "Arming mode engaged");
            state_ = State::ARM_MODE;
            arm();
        }
    } else {
        RCLCPP_ERROR(this->get_logger(), "Failed to enter offboard mode,
exiting");
        rclcpp::shutdown();
    }
    break;
case State::ARM_MODE:
    if(service_done_) {
        if (service_result_==0) {
            RCLCPP_INFO(this->get_logger(), "Vehicle armed");
            RCLCPP_INFO(this->get_logger(), "Take Off engaged");
            state_ = State::TAKEOFF_MODE;
        }
    } else {
        RCLCPP_ERROR(this->get_logger(), "Arming failed, exiting!");
        rclcpp::shutdown();
    }
    break;
case State::TAKEOFF_MODE:
    takeOff();
    if(!service_done_) {
        RCLCPP_ERROR(this->get_logger(), "Take Off failed, exiting!");
        rclcpp::shutdown();
    }
    break;
case State::HOVER_MODE:
    if(service_done_) {
        if(service_result_==0) {
            if (!msg_logged_) {
                RCLCPP_INFO(this->get_logger(), "Hover at global position:%f %f
%f",
curr_glob_.lat, curr_glob_.lon, curr_glob_.alt);
                msg_logged_ = true;
            }
            hover();
        }
    } else {
        RCLCPP_ERROR(this->get_logger(), "Hover mode failed, exiting!");
        rclcpp::shutdown();
    }
}

```

```

        break;
    case State::TELEOP_MODE:
        teleop();
        break;
    case State::LAND_MODE:
        if(service_done_) {
            if(service_result_==0 and curr_loc_.z >= -0.0) {
                if (!msg_logged_) {
                    RCLCPP_INFO(this->get_logger(), "Landing completed");
                    RCLCPP_INFO(this->get_logger(), "Disarming mode engaged");
                    state_ = State::DISARM_MODE;
                    msg_logged_ = true;
                    disarm();
                } else {
                    RCLCPP_ERROR(this->get_logger(), "Landing mode failed,
exiting!");
                    rclcpp::shutdown();
                }
            }
        }
        break;
    case State::DISARM_MODE:
        if(service_done_) {
            if(service_result_==0) {
                if (!msg_logged_) {
                    RCLCPP_INFO(this->get_logger(), "Vehicle disarmed!");
                    msg_logged_ = true;
                }
            } else {
                RCLCPP_ERROR(this->get_logger(), "Disarming mode failed,
exiting!");
                rclcpp::shutdown();
            }
        }
        break;
    case State::RTL_MODE:
        if(service_done_) {
            if(service_result_==0) {
                if (!msg_logged_) {
                    RCLCPP_INFO(this->get_logger(), "Vehicle returned to base and
landed!");
                    msg_logged_ = true;
                }
            } else {
                RCLCPP_ERROR(this->get_logger(), "RTL mode failed, exiting!");
                rclcpp::shutdown();
            }
        }
        break;
    default:
        if (!msg_logged_) {
            RCLCPP_INFO(this->get_logger(), "Unknown state mode!");
            msg_logged_ = true;
        }
        break;
    }
}

void OffboardControl::arm() {
    /**
     * @brief Publish the command to arm the vehicle.
     */

```

```

        sendVehicleCommand(VehicleCommand::VEHICLE_CMD_COMPONENT_ARM_DISARM,
                           VehicleCommand::ARMING_ACTION_ARM, 0.0); // param 2 = 0.0
    }

    void OffboardControl::disarm() {
        /**
         * @brief Publish the command to disarm the vehicle.
         */
        sendVehicleCommand(VehicleCommand::VEHICLE_CMD_COMPONENT_ARM_DISARM,
                           VehicleCommand::ARMING_ACTION_DISARM, 0.0); // param 2 = 0.0
        msg_logged_ = false;
    }

    void OffboardControl::takeOff() {
        /**
         * @brief Publish the command to take off.
         */

        // Takeoff until approx. 5 m altitude has been reached
        if (curr_loc_.z >= (take_off_height_+0.1)) {
            publishTrajectorySetpoint(curr_loc_.x, curr_loc_.y, take_off_height_,
                                     curr_loc_.yaw);
            return;
        }
        // When the desired altitude has been reached, hover at current position
        // (i.e. switch to HOVER_MODE).
        RCLCPP_INFO(this->get_logger(), "Takeoff altitude of %.2fm reached", (-
1)*take_off_height_);
        RCLCPP_INFO(this->get_logger(), "Hover mode engaged");
        state_ = State::HOVER_MODE;

        // Set the current position as the new hover position
        hover_pos_loc_.x = curr_loc_.x;
        hover_pos_loc_.y = curr_loc_.y;
        hover_pos_loc_.z = curr_loc_.z;
        hover_pos_loc_.yaw = curr_loc_.yaw;
        publishTrajectorySetpoint(curr_loc_.x, curr_loc_.y, take_off_height_,
                                 curr_loc_.yaw);
        msg_logged_ = false;
    }

    void OffboardControl::land() {
        /**
         * @brief Land the vehicle, process gets fully taken charged by PX4 and not by
         the program.
         */
        sendVehicleCommand(VehicleCommand::VEHICLE_CMD_NAV_LAND, 0.0, 0.0);
    }

    void OffboardControl::RTL() {
        /**
         * @brief Return to base, process gets fully taken charged by PX4 and not by
         the program.
         */
        sendVehicleCommand(VehicleCommand::VEHICLE_CMD_NAV_RETURN_TO_LAUNCH, 0.0, 0.0);
    }

    void OffboardControl::hover() {
        /**
         * @brief Hover the drone at current height (only in teleop mode).
         */
        if (teleop_armed_) {

```

```

        publishTrajectorySetpoint(hover_pos_loc_.x, hover_pos_loc_.y,
hover_pos_loc_.z, hover_pos_loc_.yaw);
        state_ = State::TELEOP_MODE;
        RCLCPP_INFO(this->get_logger(), "Teleop mode engaged.");
    } else {
        publishTrajectorySetpoint(hover_pos_loc_.x, hover_pos_loc_.y,
hover_pos_loc_.z, hover_pos_loc_.yaw);
    }
}

void OffboardControl::teleop() {
    /**
     * @brief Operations that get executed in Teleop mode (i.e. the user handles
the drone).
     */
    // Passing NaNs tells the function to use velocity control
    if (teleop_armed_ && key_pressed_) {
        publishTrajectorySetpoint(std::nanf(""), std::nanf(""), std::nanf(""),
std::nanf(""));
        hover_pos_loc_.x = curr_loc_.x;
        hover_pos_loc_.y = curr_loc_.y;
        hover_pos_loc_.z = curr_loc_.z;
        hover_pos_loc_.yaw = curr_loc_.yaw;
    } else if (!key_pressed_) { // Position control
        publishTrajectorySetpoint(hover_pos_loc_.x, hover_pos_loc_.y,
hover_pos_loc_.z, hover_pos_loc_.yaw);
    }
}

void OffboardControl::localPoseCallback(const
px4_msgs::msg::VehicleLocalPosition::SharedPtr msg) {
    /**
     * @brief Save the current local position of the UAV (i.e. NED).
     */
    curr_loc_.x = msg->x;
    curr_loc_.y = msg->y;
    curr_loc_.z = msg->z;
    curr_loc_.yaw = msg->heading;
}

void OffboardControl::globalPoseCallback(const
px4_msgs::msg::VehicleGlobalPosition::SharedPtr msg) {
    /**
     * @brief Save the current global position of the UAV (i.e. WGS84).
     */
    curr_glob_.lat = msg->lat;
    curr_glob_.lon = msg->lon;
    curr_glob_.alt = msg->alt;
}

void OffboardControl::homePoseCallback(const px4_msgs::msg::HomePosition::SharedPtr
msg) {
    /**
     * @brief Save the local and global home position.
     */
    home_loc_.x = msg->x;
    home_loc_.y = msg->y;
    home_loc_.z = msg->z;
    home_glob_.lat = msg->lat;
    home_glob_.lon = msg->lon;
    home_glob_.alt = msg->alt;
}

```

```

void OffboardControl::goalPoseCallback(const
px4_msgs::msg::PositionSetpointTriplet::SharedPtr msg) {
    /**
     * @brief Save the global goal position.
     */

    if (state_ == State::HOVER_MODE)
    {
        target_glob_.lat = msg->current.lat;
        target_glob_.lon = msg->current.lon;
        target_glob_.alt = msg->current.alt;

        RCLCPP_INFO(this->get_logger(), "Global goal position lat:%f lon:%f alt:%f
set and send to Nav2",
                    home_glob_.lat, home_glob_.lon, home_glob_.alt);
    }
}

void OffboardControl::vehicleAttitudeCallback(const
px4_msgs::msg::VehicleAttitude::SharedPtr msg) {
    /**
     * @brief Listen for the current trajectory values from the drone calculate the
yaw value
     * based on the Quaternion values.
     */

    // PX4 Quaternion: q[0]=W, q[1]=X, q[2]=Y, q[3]=Z
    float q0 = msg->q[0]; // w
    float q1 = msg->q[1]; // x
    float q2 = msg->q[2]; // y
    float q3 = msg->q[3]; // z

    // true_yaw is the drones current yaw value
    true_yaw_ = (std::atan2(2.0 * (q0 * q3 + q1 * q2), 1.0 - 2.0 * (q2 * q2 + q3 *
q3)));
}

void OffboardControl::teleopArmedCallback(const std_msgs::msg::Int32::SharedPtr
msg) {
    /**
     * @brief Listen for TELEOP_MODE to start user interaction. Also, listen for
special commands.
     */
    int cmd = msg->data;

    if (cmd == 1 or cmd == 0) {
        teleop_armed_ = cmd;
    } else if (cmd == 2) {
        state_ = State::LAND_MODE;
        RCLCPP_INFO(this->get_logger(), "Landing mode engaged. Disarming after
landed.");
        land();
    } else if (cmd == 3) {
        state_ = State::RTL_MODE;
        RCLCPP_INFO(this->get_logger(), "RTL mode engaged. Disarming after
landed.");
        RTL();
    }
}

```

```

void OffboardControl::statusCallback(const px4_msgs::msg::VehicleStatus::SharedPtr
msg) {
    /**
     * @brief Saves the vehicle status.
     */
    vehicle_status_msg_ = msg;
}

void OffboardControl::batteryStatusCallback(const
px4_msgs::msg::BatteryStatus::SharedPtr msg) {
    /**
     * Saves the current battery status in percentage.
     */
    battery_status_perc_ = msg->remaining;
}

void OffboardControl::run() {
    /**
     * @brief Start the program execution loop.
     */
    rclcpp::spin(shared_from_this());
}

// Program execution
int main(int argc, char *argv[]) {
    rclcpp::init(argc, argv);
    auto node = std::make_shared<OffboardControl>();
    node->run();

    rclcpp::shutdown();
    return 0;
}

```

Γ.3.7 Αρχείο Κεφαλίδας (px4_propeller_jointstate_pub.h)

```

#ifndef PX4_PROPELLER_JOINT_STATE_PUB_H
#define PX4_PROPELLER_JOINT_STATE_PUB_H

#include <array>
#include <cmath>
#include <string>
#include <vector>
#include <limits>
#include "rclcpp/rclcpp.hpp"
#include "sensor_msgs/msg/joint_state.hpp"
#include "px4_msgs/msg/actuator_motors.hpp"
#include "px4_msgs/msg/vehicle_status.hpp" // Type for vehicle_status_v1

/**
 * @brief This node publishes /joint_states for the 4 propeller joints, using REAL
PX4 motor outputs
 * from /fmu/out/actuator_motors. Propellers spin in RViz only when PX4 is ARMED.
When disarmed to
 * no motion (angles freeze).
 */
class Px4PropellerJointStatePub : public rclcpp::Node
{
public:
    Px4PropellerJointStatePub();

private:

```

```

void statusCallback(const px4_msgs::msg::VehicleStatus::SharedPtr msg);
void motorCallback(const px4_msgs::msg::ActuatorMotors::SharedPtr msg);

rclcpp::Publisher<sensor_msgs::msg::JointState>::SharedPtr joint_pub_;
rclcpp::Subscription<px4_msgs::msg::ActuatorMotors>::SharedPtr motor_sub_;
rclcpp::Subscription<px4_msgs::msg::VehicleStatus>::SharedPtr status_sub_;

std::array<std::string, 4> joint_names_;
std::array<double, 4> angles_;
bool armed_;
};

#endif

```

Γ.3.8 Εκτελέσιμο Αρχείο (px4_propeller_jointstate_pub.cpp)

```

#include "uav_offboard_ctrl/px4_propeller_jointstate_pub.h"

Px4PropellerJointStatePub::Px4PropellerJointStatePub() :
Node("px4_propeller_jointstate_publisher"),
    joint_names_{
        "front_prop_ccw_joint",
        "back_prop_ccw_joint",
        "front_prop_cw_joint",
        "back_prop_cw_joint"},
    angles_{0.0, 0.0, 0.0, 0.0}, // Initial joint angles
    armed_(false)
{
    joint_pub_ = this-
>create_publisher<sensor_msgs::msg::JointState>("joint_states", 10);

    // Use SensorData QoS (best effort) because PX4 publishes actuator outputs
    // with a similar QoS profile. Using default Reliable QoS can cause missed
    messages.
    auto qos = rclcpp::SensorDataQoS();

    // Publisher for /joint_states (RViz uses this via robot_state_publisher)
    motor_sub_ = this-
>create_subscription<px4_msgs::msg::ActuatorMotors>("/fmu/out/actuator_motors",
    qos,
        std::bind(&Px4PropellerJointStatePub::motorCallback, this,
    std::placeholders::_1));

    // Subscribe to PX4 motor outputs (normalized values 0..1)
    status_sub_ = this-
>create_subscription<px4_msgs::msg::VehicleStatus>("/fmu/out/vehicle_status_v1",
    qos,
        std::bind(&Px4PropellerJointStatePub::statusCallback, this,
    std::placeholders::_1));

    RCLCPP_INFO(this->get_logger(), "Px4PropellerJointStatePub started (real
motors, armed gating).");
}

void Px4PropellerJointStatePub::statusCallback(const
px4_msgs::msg::VehicleStatus::SharedPtr msg)
{
    using px4_msgs::msg::VehicleStatus;
    armed_ = (msg->arming_state == VehicleStatus::ARMING_STATE_ARMED);
    RCLCPP_INFO_THROTTLE(this->get_logger(), *this->get_clock(), 2000, "arming_state
= %u (armed=%s)",

```

```

        msg->arming_state, armed_ ? "true" : "false");
    }

    void Px4PropellerJointStatePub::motorCallback(const
px4_msgs::msg::ActuatorMotors::SharedPtr msg)
    {
        /**
         * @brief Read PX4 arming_state and determine whether the props are allowed to
spin.
         */
        // dt per message (approximate, purely visual)
        const double dt = 0.02;
        const double scale = 300.0; // Normalized 1.0 -> 300 rad/s

        std::array<double, 4> speeds{0.0, 0.0, 0.0, 0.0};
        for (size_t i = 0; i < speeds.size() && i < msg->control.size(); ++i) {
            double v = static_cast<double>(msg->control[i]);
            if (std::isfinite(v)) {
                speeds[i] = v * scale;
            } else {
                speeds[i] = 0.0;
            }
        }

        // If not armed, force speeds to zero -> no spin, even if PX4 publishes junk
        if (!armed_) {
            speeds = {0.0, 0.0, 0.0, 0.0};
        }

        // Apply rotation direction (CW vs CCW)
        // Assuming: index 0,1 are CCW, index 2,3 are CW
        speeds[2] = -speeds[2]; // CW rotates opposite
        speeds[3] = -speeds[3]; // CW rotates opposite

        RCLCPP_INFO_THROTTLE(
            this->get_logger(), *this->get_clock(), 500,
            "armed=%s motors(norm): [%.3f %.3f %.3f %.3f]",
            armed_ ? "true" : "false",
            (double)msg->control[0],
            (double)msg->control[1],
            (double)msg->control[2],
            (double)msg->control[3]);

        // Integrate only armed speeds; when disarmed, speeds=0 so angles freeze
        for (size_t i = 0; i < angles_.size(); ++i) {
            angles_[i] += speeds[i] * dt;
            angles_[i] = std::fmod(angles_[i], 2.0 * M_PI);
        }

        sensor_msgs::msg::JointState js;
        js.header.stamp = this->now();
        js.name.assign(joint_names_.begin(), joint_names_.end());
        js.position.assign(angles_.begin(), angles_.end());
        js.velocity.assign(speeds.begin(), speeds.end());

        joint_pub_->publish(js);
    }

    // Program execution
    int main(int argc, char * argv[])
    {
        rclcpp::init(argc, argv);

```

```

    auto node = std::make_shared<Px4PropellerJointStatePub>();
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}

```

Γ.4 Πηγαίος Κώδικας Πακέτου (uav_navigation)

Γ.4.1 Εκτελέσιμο Αρχείο (keyboard_teleop.py)

```

#!/usr/bin/env python3

"""
This file contains functions on how to teleop a drone. When a key gets
released the value gets set to zero.

Example:
    When the key for forward pitch was pressed the drone flies forward,
    and when it get's released the drone stops.
"""

import sys
import geometry_msgs.msg
import std_msgs.msg
import rclpy
from pynput.keyboard import Key, Listener
from rclpy.qos import QoSProfile, QoSReliabilityPolicy, QoSHistoryPolicy,
QoSDurabilityPolicy

if sys.platform == 'win32':
    import msvcrt
else:
    import termios
    import tty

instruction_msg = """
This node reads keys pressed by the keyboard and publishes
special commands and Twist messages to the PX4 control node.

##### MOVEMENT #####
#
#      (w): Up
#      (s): Down
#      (a): Yaw Left
#      (d): Yaw Right
#      (Up Arrow): Pitch Forward
#      (Down Arrow): Pitch Backward
#      (Left Arrow): Roll Left
#      (Right Arrow): Roll Right
#
#
#      Press (SPACE) to arm/disarm the drone
#
#      Press (R) to return to base
#      Press (ESC) to quit the application
#
#####
"""

```

```

def set_echo(enable):
    # Get terminal attributes
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    new_settings = termios.tcgetattr(fd)

    if enable:
        # Turn echo ON
        new_settings[3] = new_settings[3] | termios.ECHO
    else:
        # Turn echo OFF
        new_settings[3] = new_settings[3] & ~termios.ECHO

    termios.tcsetattr(fd, termios.TCSADRAIN, new_settings)

def main():
    # ----- Embedded Functions -----
    ---
    def on_press(key):
        """
        Handles when a key has been pressed
        :param key: The key that was pressed
        """
        nonlocal x, y, z, th, status, x_val, y_val, z_val, yaw_val, arm_toggle,
        has_takeoff

        # Transform the key code to char
        key_ch = getattr(key, 'char', key)

        if has_takeoff:
            if key in moveBindings.keys() or getattr(key, 'char', key) in
moveBindings.keys():
                key_as_char = getattr(key, 'char', key)
                x = moveBindings[key_as_char][0]
                y = moveBindings[key_as_char][1]
                z = moveBindings[key_as_char][2]
                th = moveBindings[key_as_char][3]

                x_val = (x * speed) + x_val
                y_val = (y * speed) + y_val
                z_val = (z * speed) + z_val

                # Create the Twist message to send to PX4
                twist = geometry_msgs.msg.Twist()
                twist.linear.x = float(x * speed)
                twist.linear.y = float(y * speed)
                twist.linear.z = float(z * speed)
                twist.angular.x = 0.0
                twist.angular.y = 0.0
                # 1*(-1) or 1*(1) -> velocity multiplication factor
                twist.angular.z = float(th * turn)
                pub.publish(twist)

            elif key_ch == Key.space and not has_takeoff: # Arm
                arm_toggle = not arm_toggle # Flip the value to arm or disarm
                arm_msg = std_msgs.msg.Int32()
                arm_msg.data = arm_toggle
                teleop_cmd_pub.publish(arm_msg) # 0 or 1
                print(f"Arm/Disarm toggle: {arm_msg.data}")
                has_takeoff = True

```

```

    # Check if landing mode is desired
    if isinstance(key_ch, str) and key_ch.lower() == 'l': # Land
        arm_msg = std_msgs.msg.Int32()
        arm_msg.data = 2
        teleop_cmd_pub.publish(arm_msg)
        print(f"Landing engaged")
        has_takeoff = False

    # Check if RTL mode is desired
    if isinstance(key_ch, str) and key_ch.lower() == 'r': # RTL
        arm_msg = std_msgs.msg.Int32()
        arm_msg.data = 3
        teleop_cmd_pub.publish(arm_msg)
        print(f"RTL engaged")

    # Check for ending the program
    elif key == Key.esc: # End/Quit
        # Stop listener
        print("Teleop operation ended!")
        return False

def on_release(key):
    """
    Handle when a key has been released
    :param key: The key that was released
    """
    nonlocal x, y, z, th, status, x_val, y_val, z_val, yaw_val, arm_toggle

    status = 0.0
    x_val = 0.0
    y_val = 0.0
    z_val = 0.0
    yaw_val = 0.0
    x = 0.0
    y = 0.0
    z = 0.0
    th = 0.0

    twist = geometry_msgs.msg.Twist()
    twist.linear.x = x
    twist.linear.y = y
    twist.linear.z = z
    twist.angular.x = 0.0
    twist.angular.y = 0.0
    twist.angular.z = 0.0

    pub.publish(twist)

# -----
--

rclpy.init()

node = rclpy.create_node('keyboard_teleop')

# Display the movement instructions and the cursor in terminal
print(instruction_msg)
set_echo(False)

qos_profile = QoSProfile(
    reliability=QoSReliabilityPolicy.BEST_EFFORT,
    durability=QoSDurabilityPolicy.TRANSIENT_LOCAL,

```

```

        history=QoSHistoryPolicy.KEEP_LAST,
        depth=10
    )

    pub = node.create_publisher(geometry_msgs.msg.Twist, '/offboard_velocity_cmd',
qos_profile)
    teleop_cmd_pub = node.create_publisher(std_msgs.msg.Int32, '/teleop_command',
qos_profile)
    arm_toggle = False

    speed = 5.0      # Moves 5 meters per second
    turn = 1.0       # Rotates at 1 radians per second
    x = 0.0
    y = 0.0
    z = 0.0
    th = 0.0
    status = 0.0
    x_val = 0.0
    y_val = 0.0
    z_val = 0.0
    yaw_val = 0.0

    moveBindings = {
        'w': (0, 0, 1, 0),      # Z+
        's': (0, 0, -1, 0),     # Z-
        'a': (0, 0, 0, -1),     # Yaw+
        'd': (0, 0, 0, 1),      # Yaw-
        Key.up : (1, 0, 0, 0),   # Pitch forward -> Up Arrow
        Key.down : (-1, 0, 0, 0), # Pitch backwards -> Down Arrow
        Key.right : (0, 1, 0, 0), # Right Arrow
        Key.left : (0, -1, 0, 0), # Left Arrow
    }

    has_takeoff = False

    try:
        # Collect all events until released
        with Listener(on_press = on_press, on_release=on_release) as listener:
            listener.join()

    except Exception as e:
        print(e)

    finally:
        set_echo(True)

        twist = geometry_msgs.msg.Twist()
        twist.linear.x = 0.0
        twist.linear.y = 0.0
        twist.linear.z = 0.0
        twist.angular.x = 0.0
        twist.angular.y = 0.0
        twist.angular.z = 0.0
        pub.publish(twist)

        node.get_logger().info("Keyboard teleop shutdown")
        rclpy.shutdown()

if __name__ == '__main__':
    main()

```

Γ.5 Πηγαίος Κώδικας Πακέτου (uav_mapping)

Γ.5.1 Αρχείο Εκκίνησης (rtabmap_online.launch.py)

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.actions import IncludeLaunchDescription, DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch.conditions import IfCondition
from launch.launch_description_sources import PythonLaunchDescriptionSource

def generate_launch_description():

    # Package and file paths
    pkg_uav_mapping_share_dir = get_package_share_directory('uav_mapping')
    rviz_config_file = os.path.join(pkg_uav_mapping_share_dir, 'rviz',
    'mapping_config.rviz')

    # Declare launch arguments and default parameters
    use_sim_time = LaunchConfiguration('use_sim_time', default='true')
    use_rviz = LaunchConfiguration('use_rviz')

    use_sim_time_decl = DeclareLaunchArgument(
        'use_sim_time',
        default_value='true')

    use_rviz_decl = DeclareLaunchArgument(
        name='use_rviz',
        default_value='true')

    # RTAB-Map Launch
    rtabmap_launch_path = os.path.join(
        get_package_share_directory('rtabmap_launch'), 'launch',
    'rtabmap.launch.py')

    rtabmap_stack = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(rtabmap_launch_path),
        launch_arguments={
            'rgb_topic': '/camera/image',
            'depth_topic': '/camera/depth/image',
            'camera_info_topic': '/camera/camera_info',
            'subscribe_scan': 'true',
            'scan_topic': '/scan',
            'approx_sync': 'true',
            'frame_id': 'base_link',

            # --- ODOMETRY SETTINGS ---
            'visual_odometry': 'false', # Rely on Gazebo Odom
            'odom_topic': '/odom',

            'use_sim_time': use_sim_time,
            'rtabmap_args': '--RGBD/NeighborLinkRefining true --Reg/Strategy 1 -d',
            'qos': '1'
        }.items()
    )

    # Static Transform: Footprint -> Base Link
    tf_base_footprint_to_base = Node(
        package='tf2_ros',
```

```

        executable='static_transform_publisher',
        name='base_footprint_to_base_link',
        arguments=['0', '0', '0', '0', '0', '0', 'x650_0/base_footprint',
'base_link'],
        parameters=[{'use_sim_time': use_sim_time}],
        output='screen'
    )

    # Static Transform: Base Link -> Camera Frame
    tf_base_to_cam = Node(
        package='tf2_ros',
        executable='static_transform_publisher',
        name='base_link_to_camera',
        # x y z yaw pitch roll parent child
        arguments=['0.1', '0', '0', '-1.57', '0', '-1.57', 'base_link',
'x650_0/base_link/rgbd_cam'],
        parameters=[{'use_sim_time': use_sim_time}],
        output='screen'
    )

    rviz_node = Node(
        condition=IfCondition(use_rviz),
        package='rviz2',
        executable='rviz2',
        output='screen',
        arguments=['-d', rviz_config_file],
        parameters=[{"use_sim_time": use_sim_time}]
    )

    return LaunchDescription([
        use_sim_time_decl,
        use_rviz_decl,
        rtabmap_stack,
        tf_base_footprint_to_base,
        tf_base_to_cam,
        rviz_node
    ])

```

Γ.6 Πηγαίος Κώδικας Πακέτου (uav_vision)

Γ.6.1 Εκτελέσιμο Αρχείο (object_detection.py)

```

#!/usr/bin/env python3
import os
import rclpy
import cv2
import message_filters
import numpy as np
from rclpy.node import Node
from sensor_msgs.msg import Image
from rclpy.qos import QoSProfile, QoSReliabilityPolicy, QoSHistoryPolicy
from cv_bridge import CvBridge
from ament_index_python.packages import get_package_share_directory

from ultralytics import YOLO

class ObjectDetection(Node):
    def __init__(self):
        super().__init__("object_detection")

```

```

# Model Setup
package_share_directory = get_package_share_directory('uav_vision')
model_path = os.path.join(package_share_directory, 'models', 'yolov8n.pt')
self._model = YOLO(model_path)
self.bridge = CvBridge()

# QoS Profile (this has to match Gazebo/PX4 standard)
sensor_qos = QoSProfile(
    reliability=QoSReliabilityPolicy.BEST_EFFORT,
    history=QoSHistoryPolicy.KEEP_LAST,
    depth=10)

# Use message_filters.Subscriber to fuse both subscribers (required for the
distance calculation)
self.rgb_sub = message_filters.Subscriber(self, Image, "/camera/image",
qos_profile=sensor_qos)
self.depth_sub = message_filters.Subscriber(self, Image,
"/camera/depth/image", qos_profile=sensor_qos)

# Variables in the list must match the names defined above (self.rgb_sub,
self.depth_sub)
self.ts = message_filters.ApproximateTimeSynchronizer(
    [self.rgb_sub, self.depth_sub],
    queue_size=10,
    slop=0.1
)
self.ts.registerCallback(self.synchronized_clbk)

# Publisher
self.rgb_pub_ = self.create_publisher(Image, "/image/detection",
sensor_qos)

# Persistent Window setup
cv2.namedWindow("Suspect Detection", cv2.WINDOW_NORMAL)
cv2.resizeWindow("Suspect Detection", 800, 600)
self.get_logger().info("Node started. Waiting for synchronized RGB and
Depth frames...")

def synchronized_clbk(self, rgb_msg, depth_msg):
    try:
        # Convert images
        color_image = self.bridge.imgmsg_to_cv2(rgb_msg, "bgr8")
        depth_image = self.bridge.imgmsg_to_cv2(depth_msg, "32FC1") # Float32
meters

        # AI Detection (NMS applied via max_det=1)
        results = self._model.predict(color_image, classes=[0], conf=0.6,
max_det=1, verbose=False)

        # Define default annotated frame (incase no detection)
        annotated_frame = color_image.copy()

        for r in results:
            # Plot YOLO boxes first
            annotated_frame = r.plot()

            for box in r.bboxes:
                # Calculate center coordinates
                x1, y1, x2, y2 = box.xyxy[0]
                cx, cy = int((x1 + x2) / 2), int((y1 + y2) / 2)

```

```

        # Safeguard against out-of-bounds
        cy = min(max(cy, 2), depth_image.shape[0] - 3)
        cx = min(max(cx, 2), depth_image.shape[1] - 3)

        # Extract average depth from 5x5 ROI
        depth_roi = depth_image[cy-2:cy+3, cx-2:cx+3]
        dist = np.nanmean(depth_roi)

        if not np.isnan(dist) and not np.isinf(dist):
            dist_label = f"distance: {dist:.2f}m" # Format label to
look like the YOLO label

        # Use the same font settings as YOLO for matching
dimensions
        font = cv2.FONT_HERSHEY_SIMPLEX
        font_scale = 0.6
        thickness = 2

        # Calculate size of the distance label
        (w, h), _ = cv2.getTextSize(dist_label, font, font_scale,
thickness)

        # Position it directly above the person label
        label_x = int(x1)
        label_y = int(y1) - 22 # Offset to sit above the blue
person label

        # Draw background rectangle (matching the person label
style)
        cv2.rectangle(annotated_frame, (label_x, label_y - h - 5),
(label_x + w, label_y + 5),
                        (75,75,75), -1)

        # Draw the text
        cv2.putText(annotated_frame, dist_label, (label_x,
label_y),
                    font, font_scale, (0, 255, 0), thickness)

        # Display and Publish the annotated image
        cv2.imshow("Suspect Detection", annotated_frame)
        cv2.waitKey(1)

        out_msg = self.bridge.cv2_to_imgmsg(annotated_frame, encoding="bgr8")
        out_msg.header = rgb_msg.header
        self.rgb_pub_.publish(out_msg)

    except Exception as e:
        self.get_logger().error(f"Sync Callback failed: {e}")

def main(args=None):
    rclpy.init(args=args)
    node = ObjectDetection()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info('Shutting down...')
    finally:
        # Final cleanup
        cv2.destroyAllWindows()
        node.destroy_node()
        rclpy.shutdown()

```

```
if __name__ == "__main__":
    main()
```

Γ.7 Πηγαίος Κώδικας Πακέτου (uav_bringup)

Γ.7.1 Αρχείο Εκκίνησης (start_sitl.launch.py)

```
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.substitutions import PathJoinSubstitution
from launch.launch_description_sources import PythonLaunchDescriptionSource
from ament_index_python.packages import get_package_share_directory

def generate_launch_description():

    # Package directories
    pkg_share_uav_description = get_package_share_directory("uav_description")
    pkg_share_uav_offboard_ctrl = get_package_share_directory("uav_offboard_ctrl")

    # Include the launch of PX4-Autopilot
    px4_autopilot_incl = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            PathJoinSubstitution([pkg_share_uav_offboard_ctrl, 'launch',
'px4_autopilot.launch.py'])
        )
    )

    # Include the launch of robot_description and /joint_states BEFORE gazebo and
px4 starts
    robotdescription_incl = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            PathJoinSubstitution([pkg_share_uav_description, 'launch',
'display.launch.py'])
        )
    )

    ld = LaunchDescription()
    ld.add_action(robotdescription_incl)
    ld.add_action(px4_autopilot_incl)
    return ld
```