ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΥΠΟΛΟΓΙΣΤΩΝ
ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΣΤΗΝ ΡΟΜΠΟΤΙΚΗ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Έλεγχος του Parrot Bebop 2 Drone μέσω του ROS»

Του φοιτητή
Ηλία Γουσιόπουλου
Αρ. Μητρώου: 61

Επιβλέπων
Σπυρίδων Καζαρλής

Ημερομηνία ……

Πρόλογος

Η επιλογή της συγκεκριμένης πτυχιακής εργασίας έγινε με σκοπό την περεταίρω ενασχόληση και γνώση του ραγδαία αναπτυσσόμενου τομέα των UAV και της αυτόνομης πλοήγησης μέσω ανοιχτού λογισμικού.

Στην πορεία της συγγραφής της αφιερώθηκε πολύς χρόνος από την πλευρά μου για την βελτίωση των γνώσεων και των δεξιοτήτων που απαιτούνται για την υλοποίηση ενός τέτοιου συστήματος.

Επίσης μεγάλο κέρδος της όλης διαδικασίας ήταν η χρησιμοποίηση ενός πολύ μοντέρνου και με πολλές δυνατότητες εργαλείου, του ROS, καθώς και η εμβάθυνση στην αρχιτεκτονική αυτόνομης GPS πλοήγησης.

Περίληψη

Ο τομέας των UAV (Unmanned Aerial Vehicle) έχει σημειώσει σημαντική πρόοδο τα τελευταία χρόνια, χάρη στην χρησιμοποίηση ελαφρύτερων υλικών, πιο αποδοτικών μπαταριών και προηγμένων αλγορίθμων ελέγχου.

Η παρούσα εργασία πραγματεύεται τη υλοποίηση διασύνδεσης του Parrot Bebop 2 μέσω του ROS (Robot Operating System) και τη δημιουργία Γραφικής Διεπαφής Χρήστη (GUI) για τον έλεγχο και την αυτόνομη GPS πλοήγησή του μέσω waypoints. Η επικοινωνία μεταξύ ROS και drone πραγματοποιείται με τον bebop_autonomy driver.

Το GUI (bebop_mission_gui.py) ενσωματώνει:

- χειροκίνητο έλεγχο με κουμπιά και πληκτρολόγιο,
- ζωντανή ροή βίντεο σε πραγματικό χρόνο,
- έλεγχο κατεύθυνσης κάμερας,
- ένδειξη μπαταρίας,
- live GPS συντεταγμένες,
- σχεδιασμό αποστολής με waypoints,
- ανάλυση βιωσιμότητας Go/No-Go,
- και καταγραφή τηλεμετρίας πτήσης.

Η αυτόνομη GPS πλοήγηση υλοποιείται μέσω του ενσωματωμένου autopilot του Bebop 2 χρησιμοποιώντας αρχεία MAVLink format QGC WPL 120, τα οποία ανεβαίνουν μέσω FTP και ενεργοποιούνται μέσω ROS topic.

Η έκδοση Ubuntu που χρησιμοποιήθηκε είναι η 18.04 Bionic Beaver και η έκδοση ROS η Melodic.

«Controlling Parrot Bebop 2 drone using ROS »

«Ilias Gousiopoulos»

Abstract

The field of UAVs (Unmanned Aerial Vehicles) has made considerable progress in recent years, thanks to the use of lighter materials, more efficient batteries, and advanced control algorithms.

This work focuses on implementing an interface for the Parrot Bebop 2 using ROS (Robot Operating System) and developing a graphical user interface (GUI) for drone control and autonomous GPS waypoint navigation.

Communication between ROS and the drone is conducted using the bebop_autonomy driver.

The GUI (bebop_mission_gui.py) provides:

- manual control via buttons and keyboard,
- live video feed,
- camera control,
- battery indicator,
- GPS coordinates,
- mission planning with waypoints,
- Go/No-Go feasibility analysis,
- and flight telemetry logging.

Autonomous GPS navigation is implemented via the Bebop 2 onboard autopilot using MAVLink files (QGC WPL 120 format), uploaded via FTP and triggered via ROS topic. This ensures reliable autonomous flight without continuous WiFi connectivity.

The ROS distribution used is Melodic on Ubuntu 18.04 Bionic Beaver.

Περιεχόμενα

Πρόλογος ii
Περίληψη iii
Abstract iv
Περιεχόμενα v
Κεφάλαιο 1ο: Εισαγωγή στα UAV (Μη Επανδρωμένα Αεροσκάφη) 10
1.1 Βασικά μέρη ενός drone 10
1.1.1 Αεροδυναμική δομή (Airframe). 10
1.1.2 Σύστημα πρόωσης (Propulsion). 10
1.1.3 Ηλεκτρονικά & πλοήγηση (Avionics). 11
1.1.4 Σύστημα επικοινωνίας (Communication). 11
1.1.5 Αισθητήρες & ωφέλιμο φορτίο (Payload). 11
1.2 Υπάρχουσες Λύσεις και Κενό Έρευνας 12
1.2.1 Επίσημα εργαλεία Parrot. 12
1.2.2 Σταθμοί ελέγχου εδάφους ανοιχτού κώδικα. 12
1.2.3 Εργαλεία Python και ROS. 12
1.3 Σύγκριση υπαρχουσών λύσεων 13
1.4 Κενό που καλύπτει η παρούσα εργασία 13
Κεφάλαιο 2ο: Λογισμικό 14
2.1 Εισαγωγή στην αναφορά των διαφορετικών λογισμικών και προγραμμάτων που χρειάστηκαν 14
2.1.1 Εγκατάσταση Ubuntu 18.04 Bionic 15
2.1.2 Εγκατάσταση ROS Melodic Morenia 20
2.1.3 Εγκατάσταση του bebop autonomy 20
Κεφάλαιο 3ο:Σύνδεση του Drone Parrot Bebop 2 με το PC 22
3.1 Περιβάλλον και Μεθοδολογία Δοκιμών 22
3.1.1 Σύνδεση μέσω Wi-Fi 22
3.1.2 Σύνδεση με ROS - bebop_autonomy 23
3.1.3 Επιβεβαίωση Σωστής Σύνδεσης 24
3.1.4 Δοκιμές Επικοινωνίας 24
3.1.5 Πρόβλημα cv_bridge 25

Κεφάλαιο 4ο: Αυτόνομη GPS Πλοήγηση — Onboard Autopilot και MAVLink 25
4.1.1 Η Αποτυχημένη Πρώτη Προσέγγιση — cmd_vel GPS Loop 25
4.1.2 Onboard Autopilot 26
4.2 Τι είναι το MAVLink 26
4.2.1 Δομή Αρχείου MAVLink — QGC WPL 120 26
4.3 Κατεύθυνση Μύτης 27
Κεφάλαιο 5ο: Γραφικό Περιβάλλον Χρήστη bebop_mission_gui.py 29
5.1 Αρχιτεκτονική Συστήματος 29
5.2 Δομή GUI — Panel Builder Methods 29
5.3 Ανάλυση Αποστολής — Go/No-Go 30
5.4 Σύστημα Καταγραφής — MissionLogger 30
Κεφάλαιο 6ο: Αναλυτική Επεξήγηση Κώδικα bebop_mission_gui.py 32
6.1 Εισαγωγές Βιβλιοθηκών (Γραμμές 1–39) 32
6.2 Σταθερές Συστήματος (Γραμμές 41–57) 33
6.3 Βοηθητικές Συναρτήσεις (Γραμμές 60–114) 35
6.3.1 haversine(la1, lo1, la2, lo2) — Γραμμές 63-68 35
6.3.2 parse_coords(text) — Γραμμές 70-74 35
6.3.3 parse_mavlink(content) — Γραμμές 76-85 35
6.3.4 analyse(waypoints, speed=3.0, battery=None) — Γραμμές 87-101 36
6.3.5 gen_mavlink(wps, speed=3.0) — Γραμμές 103-114 37
6.4 Κλάση RosWorker (Γραμμές 117–201) 39
6.4.1 Signals (Γραμμές 121-127) 39
6.4.2 Μέθοδος init() (Γραμμές 133-171) 39
6.4.3 Callbacks και Εντολές (Γραμμές 173-201) 40
6.5 Κλάση VideoWorker (Γραμμές 204–252) 41
6.6 Κλάση FtpWorker (Γραμμές 255–285) 43
6.7 Κλάση MissionLogger (Γραμμές 288–389) 44
6.8 Μέθοδοι Κατασκευής UI (Γραμμές 425–917) 47
6.8.1 _build_ui() 47
6.8.2 _build_camera_card() 47
6.8.3 _build_status_card() 47
6.8.4 _build_flight_card() 48

6.8.5 _build_direction_pad() 48
6.8.6 _build_camera_controls() 48
6.8.7 _build_connection_row() 49
6.8.8 _build_waypoint_panel() 49
6.8.9 _build_analysis_panel() 50
6.8.10 _build_mission_control() 50
6.8.11 _build_log_panel() 50
6.8.12 _wire_signals() 51
6.8.13 _card() και _db() 51
6.9 Workers και ROS Slots (Γραμμές 919–1007) 52
6.9.1 _start_workers() 52
6.9.2 _on_conn() 52
6.9.3 _on_batt() 52
6.9.4 _on_state() — Ανίχνευση Ολοκλήρωσης Αποστολής 53
6.9.5 _on_gps() και _on_odom() 53
6.10 Χειροκίνητος Έλεγχος (Γραμμές 1008–1065) 54
6.11 Waypoints, Ανάλυση, FTP (Γραμμές 1066–1234) 55
6.11.1 Waypoint Management 55
6.11.2 _analyse() — Ανάλυση Βιωσιμότητας 55
6.11.3 FTP Λειτουργίες 56
6.12 Έλεγχος Αποστολής (Γραμμές 1235–1350) 57
6.13 _reset_state() — Επαναφορά Μετά Προσγείωση (Γραμμές 1293–1341) 58
6.14 _confirm_close() και _ms() (Γραμμές 1343–1353) 59
6.15 _tick(), _log(), closeEvent(), main() (Γραμμές 1355–1603) 60
6.15.1 _tick() — 1 Hz Loop 60
6.15.2 _log() 60
6.15.3 _css() — Θέμα Ανοιχτού Γκρι 60
6.15.4 closeEvent() 61
6.15.5 main() 61
Κεφάλαιο 7ο: Συμπεράσματα και Μελλοντικές Επεκτάσεις 62
7.1 Συμπεράσματα 62
7.2 Περιορισμοί Συστήματος 62

7.3 Μελλοντικές Επεκτάσεις 63
7.4 YawTracker — Παράλληλος Βρόχος Ελέγχου Κατεύθυνσης Μύτης 63
7.4.1 RTK GPS — Ακρίβεια Εκατοστών αντί Μέτρων 63
7.4.2 ROS Navigation Stack — Τοπική Αποφυγή Εμποδίων 64
7.4.3 Google Maps Interface — Οπτική Τοποθέτηση Waypoints 64
7.4.4 ROS 2 Migration — Μετάβαση σε Σύγχρονη Αρχιτεκτονική 65
7.4.5 Multi-Drone — Συντονισμός Πολλαπλών Bebop 2 65
7.4.6 Return-to-Home — Αυτόματη Επιστροφή σε Ασφαλείς Συνθήκες 66
7.5 Συγκρητική Αξιολόγηση Μελλοντικών Επεκτάσεων 66
Παράρτημα Κώδικα 68
8.1 Πίνακας Ενοτήτων Κώδικα 68
8.1.1 Ενότητα 1 — Imports και Σταθερές 69
8.1.2 Βοηθητικές Συναρτήσεις 70
8.1.3 Κλάση RosWorker 71
8.1.4 Κλάση VideoWorker 72
8.1.5 Κλάση FtpWorker 73
8.1.6 Κλάση MissionLogger 74
8.1.7 Κλάση App — Constructor και State 76
8.1.8 build_camera_card() (Γραμμές 482–494) 78
8.1.9 build_status_card() (Γραμμές 496–520) 78
8.1.10 build_flight_card() (Γραμμές 522–537) 78
8.1.11 _build_direction_pad() (Γραμμές 539–588) 79
8.1.12 _build_camera_controls() (Γραμμές 590–608) 79
8.1.13 _build_connection_row() (Γραμμές 610–626) 80
8.1.14 _build_waypoint_panel() (Γραμμές 628–697) 80
8.1.15 _build_analysis_panel() (Γραμμές 699–754) 81
8.1.16 _build_mission_control() (Γραμμές 756–803) 82
8.1.17 _build_log_panel() (Γραμμές 805–836) 83
8.1.18 _wire_signals, _card, _db 84
8.1.19 _start_workers 86
8.1.20 ROS Slots 87
8.1.21 Χειροκίνητος Έλεγχος 88

8.1.22 Διαχείριση Waypoints....89
8.1.23 Ανάλυση Αποστολής (_analyse)....90
8.1.24 FTP Λειτουργίες....92
8.1.25 Έλεγχος Αποστολής....93
8.1.26 Reset, Close, _ms, _tick, _log....94
8.1.27 CSS Stylesheet (_css)....96
8.1.28 closeEvent....99
8.1.29 main()....99
Πηγές και σύνδεσμοι:....100

Κεφάλαιο 1ο: Εισαγωγή στα UAV (Μη Επανδρωμένα Αεροσκάφη)

Τα Μη Επανδρωμένα Εναέρια Οχήματα (Unmanned Aerial Vehicles - UAVs), γνωστά και ως drones, συνιστούν αεροπορικά συστήματα τα οποία λειτουργούν χωρίς την παρουσία ανθρώπου επί του αεροσκάφους. Η καθοδήγησή τους πραγματοποιείται είτε απομακρυσμένα από χειριστές είτε μέσω αυτόνομων συστημάτων πλοήγησης, που αξιοποιούν τεχνολογίες όπως το GPS και η Τεχνητή Νοημοσύνη. Η εξέλιξή τους έχει οδηγήσει σε εκτεταμένη χρήση σε διάφορους τομείς, από την αμυντική βιομηχανία και την παρακολούθηση έως τη γεωργία και τις υπηρεσίες εναέριας παράδοσης. Η παρούσα εργασία εστιάζει σε μία συγκεκριμένη πλατφόρμα αυτής της κατηγορίας, το Parrot Bebop 2, και στον έλεγχό του μέσω ROS.

1.1 Βασικά μέρη ενός drone

1.1.1 Αεροδυναμική δομή (Airframe).

Η βασική κατασκευή του UAV σχεδιάζεται με γνώμονα την αντοχή, το βάρος και την αεροδυναμική απόδοση. Κατασκευάζεται από υλικά όπως ανθρακονήματα ή κράματα αλουμινίου και διαφοροποιείται σε μοντέλα με σταθερές πτέρυγες, περιστρεφόμενους ρότορες ή υβριδικά συστήματα.

Στο Bebop 2: πρόκειται για quadcopter περιστρεφόμενων ρότορων με ελαφρύ σκελετό ~500 g. Η επιλογή αυτή προσφέρει ευελιξία σε περιορισμένο χώρο, αλλά όπως φάνηκε στις δοκιμές η αντοχή σε άνεμο περιορίζεται περίπου στα 40 km/h.

1.1.2 Σύστημα πρόωσης (Propulsion).

Το σύστημα πρόωσης παρέχει την απαιτούμενη ισχύ για την πτήση. Διακρίνονται οι εξής κύριοι τύποι:

- **Ηλεκτρικοί κινητήρες**, που χρησιμοποιούν μπαταρίες λιθίου (Li-Po).
- **Εσωτερικής καύσης κινητήρες**, οι οποίοι προσφέρουν αυξημένη αυτονομία.
- **Στροβιλοκινητήρες**, που αξιοποιούνται κυρίως σε στρατιωτικά UAVs για υψηλές ταχύτητες και αντοχή σε μεγάλες αποστάσεις.

Στο Bebop 2: χρησιμοποιούνται τέσσερις brushless κινητήρες με ηλεκτρική πρόωση LiPo. Σε αντίθεση με κινητήρες εσωτερικής καύσης ή στροβιλοκινητήρες, η ηλεκτρική πρόωση περιορίζει την πτήση στα ~25 λεπτά — όριο που ενσωματώθηκε ρητά στον έλεγχο Go/No-Go μέσω της σταθεράς MAX_MIN = 25.

1.1.3 Ηλεκτρονικά & πλοήγηση (Avionics).

Τα UAVs είναι εξοπλισμένα με αισθητήρες, όπως γυροσκόπια, επιταχυνσιόμετρα, βαρόμετρα και GPS, τα οποία επιτρέπουν τη σταθεροποίηση και την πλοήγηση με ακρίβεια. Σε αυτόνομα μοντέλα, χρησιμοποιούνται προηγμένοι αλγόριθμοι τεχνητής νοημοσύνης για τη λήψη αποφάσεων σε πραγματικό χρόνο.

Στο Bebop 2: συνδυάζονται επιταχυνσιόμετρο, γυροσκόπιο, μαγνητόμετρο, βαρόμετρο, υπερηχητικός αισθητήρας εδάφους και κάθετη κάμερα optical flow, μαζί με GPS/GLONASS. Ο διαχωρισμός των δύο ομάδων αισθητήρων αποδείχθηκε καθοριστικός: η αδρανειακή σταθεροποίηση και το optical flow λειτουργούν αξιόπιστα ακόμη και σε εσωτερικό χώρο, ενώ το GPS υπέφερε από multipath που έδωσε σφάλμα θέσης 34 km (Κεφάλαιο 3). Αυτή ακριβώς η ασυμμετρία ήταν ο λόγος που ο κλειστός βρόχος cmd_vel + GPS απορρίφθηκε υπέρ του ενσωματωμένου autopilot (Κεφάλαιο 4).

1.1.4 Σύστημα επικοινωνίας (Communication).

Η επικοινωνία μεταξύ UAV και σταθμού ελέγχου επιτυγχάνεται μέσω ραδιοκυμάτων (RF), δορυφορικής σύνδεσης ή κινητών δικτύων (5G), επιτρέποντας τη ζωντανή μετάδοση δεδομένων και εντολών.

Στο Bebop 2: η επικοινωνία βασίζεται αποκλειστικά σε WiFi MIMO διπλής ζώνης, με το ίδιο το drone να λειτουργεί ως access point (σταθερή IP 192.168.42.1). Ο περιορισμός εμβέλειας (~300 m) και η μεταβλητή καθυστέρηση (50–200 ms) του WiFi είναι ακριβώς ο λόγος που η αυτόνομη πλοήγηση σχεδιάστηκε ώστε να εκτελείται ανεξάρτητα από τη σύνδεση μετά την εκκίνηση της αποστολής.

1.1.5 Αισθητήρες & ωφέλιμο φορτίο (Payload).

Τα UAVs φέρουν ειδικό εξοπλισμό ανάλογα με την αποστολή τους. Οι συνηθέστεροι αισθητήρες περιλαμβάνουν:

- **LiDAR** για χαρτογράφηση και ανάλυση επιφανειών.
- **Θερμικές και υπέρυθρες κάμερες** για επιτήρηση και έρευνα-διάσωση.
- **Πολυφασματικούς αισθητήρες** για αγροτική και περιβαλλοντική ανάλυση.

Στο Bebop 2: το ωφέλιμο φορτίο είναι σταθερό και όχι εναλλάξιμο: μία κάμερα 14 MP με fisheye φακό και ψηφιακή σταθεροποίηση. Η ροή της αξιοποιείται στο GUI μέσω του topic /bebop/image_raw/compressed, με άμεση αποκωδικοποίηση JPEG (Κεφάλαιο 6).

1.2 Υπάρχουσες Λύσεις και Κενό Έρευνας

Πριν από την ανάπτυξη του παρόντος συστήματος, διερευνήθηκαν οι διαθέσιμες λύσεις για τον έλεγχο και την παρακολούθηση του Parrot Bebop 2. Οι λύσεις αυτές κατηγοριοποιούνται σε τρεις ομάδες: εμπορικές εφαρμογές της Parrot, γενικοί σταθμοί ελέγχου εδάφους (GCS) ανοιχτού κώδικα, και εργαλεία βασισμένα σε Python/ROS.

1.2.1 Επίσημα εργαλεία Parrot.

Η εταιρεία Parrot παρέχει την εφαρμογή FreeFlight 6 ως το βασικό μέσο ελέγχου του Bebop 2. Η εφαρμογή αυτή λειτουργεί αποκλειστικά σε κινητές πλατφόρμες (iOS/Android), είναι κλειστού κώδικα, και δεν προσφέρει ενσωμάτωση με το οικοσύστημα ROS. Η αυτόνομη πλοήγηση σε waypoints είναι διαθέσιμη μόνο σε εμπορικές εκδόσεις και η πρόσβαση στα τηλεμετρικά δεδομένα του drone σε πραγματικό χρόνο είναι εξαιρετικά περιορισμένη. Παράλληλα, η Parrot έχει αναπτύξει ένα νέο οικοσύστημα ανοιχτού κώδικα (Olympe SDK, AirSDK, OpenFlight) το οποίο επιτρέπει εκτεταμένη προγραμματιστική πρόσβαση — ωστόσο, αυτό στοχεύει αποκλειστικά στη σειρά ANAFI και δεν είναι συμβατό με το Bebop 2 [Parrot, 2022].

1.2.2 Σταθμοί ελέγχου εδάφους ανοιχτού κώδικα.

Τα δύο πιο διαδεδομένα εργαλεία της κατηγορίας αυτής, το QGroundControl (Dronecode Foundation) και το Mission Planner (ArduPilot), βασίζονται στο πρωτόκολλο MAVLink και συνεργάζονται με autopilots που το υποστηρίζουν εγγενώς (PX4, ArduPilot). Το Bebop 2, ωστόσο, χρησιμοποιεί το ιδιόκτητο πρωτόκολλο ARSDK (βασισμένο σε libpomp), το οποίο δεν είναι συμβατό με MAVLink. Η χρήση των παραπάνω εργαλείων με το Bebop 2 απαιτεί αντικατάσταση του εργοστασιακού firmware με PX4 — επέμβαση που καθιστά αδύνατη τη χρήση του bebop_autonomy driver και επαναφέρει το σύστημα σε επίπεδο ασύμβατο με το υπόλοιπο ROS οικοσύστημα [PX4 Autopilot, 2016].

1.2.3 Εργαλεία Python και ROS.

Η βιβλιοθήκη pyparrot (McGovern, 2018) παρέχει Python API για τον έλεγχο του Bebop 2 χωρίς ROS, κυρίως για εκπαιδευτικούς σκοπούς. Δεν προσφέρει γραφικό περιβάλλον, προγραμματισμό αποστολής ή τηλεμετρία σε πραγματικό χρόνο. Στον χώρο του ROS, το πακέτο bebop_autonomy (Monajjemi κ.ά., Autonomy Lab, Simon Fraser University) αποτελεί τον αποκλειστικό driver για την επικοινωνία με το Bebop 2 μέσω ROS, εκθέτοντας τον έλεγχο μέσω topics (/bebop/cmd_vel, /bebop/takeoff κ.ά.) [Monajjemi κ.ά., 2014]. Ωστόσο, το bebop_autonomy παρέχει μόνο τη στρώση επικοινωνίας· ο χειρισμός γίνεται αποκλειστικά μέσω τερματικού ή χειριστηρίου. Υπάρχουν επίσης πειραματικές υλοποιήσεις ελέγχου θέσης μέσω ROS, όπως το bebop2_controller (Newaz, 2019), το οποίο εστιάζει σε indoor πλοήγηση με AprilTags και οπτικοποίηση μέσω rviz — χωρίς γραφικό GCS, χωρίς προγραμματισμό αποστολής waypoints και χωρίς ολοκληρωμένο σύστημα ασφάλειας πτήσης.

1.3 Σύγκριση υπαρχουσών λύσεων

Ο παρακάτω πίνακας συνοψίζει τα χαρακτηριστικά των κυριότερων διαθέσιμων λύσεων σε σχέση με τις απαιτήσεις μιας ολοκληρωμένης ROS-based GCS για το Bebop 2.

Εργαλείο	ROS	Γραφικό περιβάλλον	Waypoints	Live βίντεο	Τηλεμετρία	Bebop 2 native
FreeFlight	✗	✓ (mobile)	✓	✓	Βασική	✓
QGroundControl	✗	✓	✓	✓	✓	✗*
pyparrot	✗	✗	✗	Μερικώς	Βασική	✓
bebop_autonomy	✓	✗	✗	✓ (topic)	✓ (topics)	✓
bebop2_controller	✓	rviz μόνο	✗	✗	Μερικώς	✓
Παρούσα εργασία	**✓**	**✓ (desktop)**	**✓**	**✓**	**✓**	**✓**

* Απαιτεί αντικατάσταση του εργοστασιακού firmware με PX4, καθιστώντας το bebop_autonomy driver μη λειτουργικό.

1.4 Κενό που καλύπτει η παρούσα εργασία

Από την παραπάνω ανάλυση προκύπτει ότι δεν υπάρχει διαθέσιμη, ανοιχτού κώδικα, ROS-ενσωματωμένη γραφική εφαρμογή GCS για το Bebop 2 που να συνδυάζει προγραμματισμό αποστολής waypoints, ζωντανή μεταφορά βίντεο, πλήρη τηλεμετρία και μηχανισμούς ασφάλειας πτήσης σε ένα ενιαίο περιβάλλον. Η παρούσα εργασία καλύπτει αυτό το κενό αναπτύσσοντας ένα GCS βασισμένο σε Python/PyQt5 και ROS Melodic, ειδικά σχεδιασμένο για τη λειτουργία με το bebop_autonomy driver.

Κεφάλαιο 2ο: Λογισμικό

2.1 Εισαγωγή στην αναφορά των διαφορετικών λογισμικών και προγραμμάτων που χρειάστηκαν

Αρχικά είχα επιλέξει να κάνω μία καθαρή εγκατάσταση του λογισμικού Ubuntu σε κάποιο laptop παλιότερο που είχα στην κατοχή μου. Τελικώς επέλεξα να κάνω την εγκατάσταση σε windows 11 laptop με την χρησιμοποίηση του Oracle Virtual Box 7.

Το Virtual Box αποτελεί ένα πρόγραμμα που σου επιτρέπει να δημιουργήσεις ένα εικονικό περιβάλλον μέσα στο οποίο μπορείς να κάνεις εγκατάσταση οποιοδήποτε λειτουργικό σύστημα επιθυμείς. Μέσα στο περιβάλλον αυτό έγινε η εγκατάσταση των Ubuntu 18.04 Bionic Beaver καθώς επίσης και του ROS Melodic. Για την δημιουργία του GUI χρησιμοποιήθηκε το πλαίσιο PyQt5.

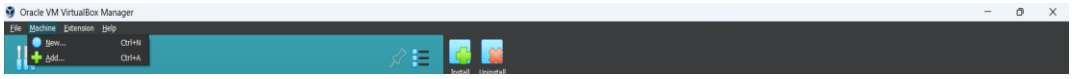
2.1.1 Εγκατάσταση Ubuntu 18.04 Bionic

Η διαδικασία εγκατάστασης του Ubuntu 18.04 Bionic Beaver, έχει ως εξής:

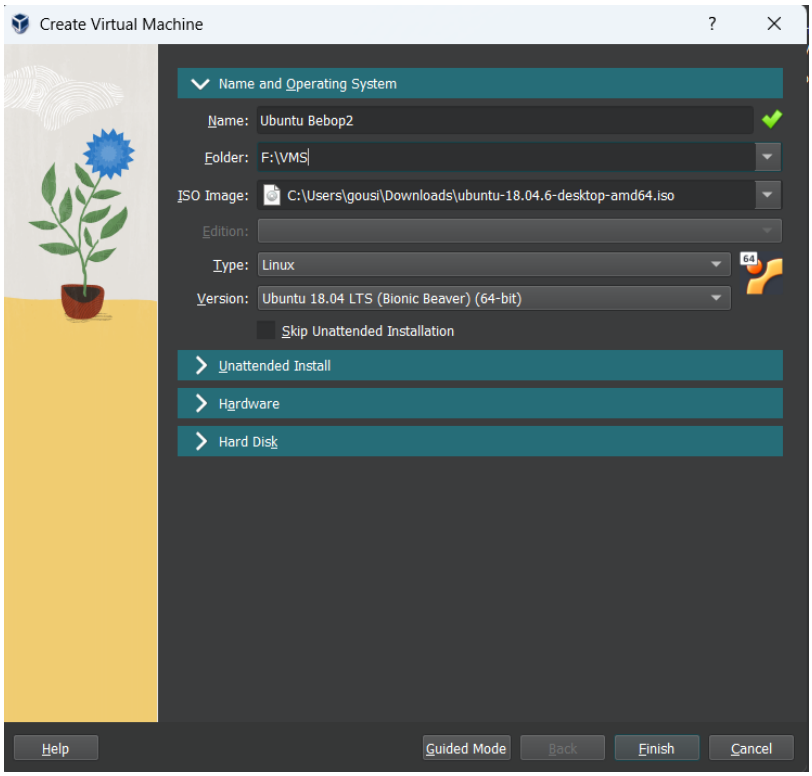
Κατέβασμα της συγκεκριμένης έκδοσης από το https://releases.ubuntu.com/18.04/

Αφού ολοκληρώθηκε το κατέβασμα του αρχείου τα βήματα που ακολουθήθηκαν για την εγκατάσταση του Ubuntu στο Virtual Box 7 ήταν τα ακόλουθα:

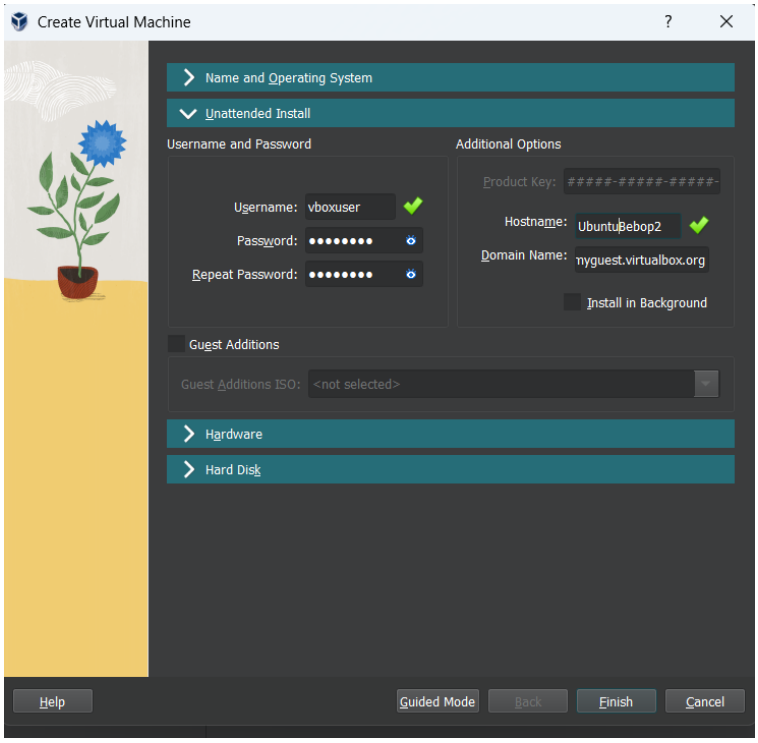
1) Έπρεπε να δημιουργήσουμε καινούρια εικονική μηχανή



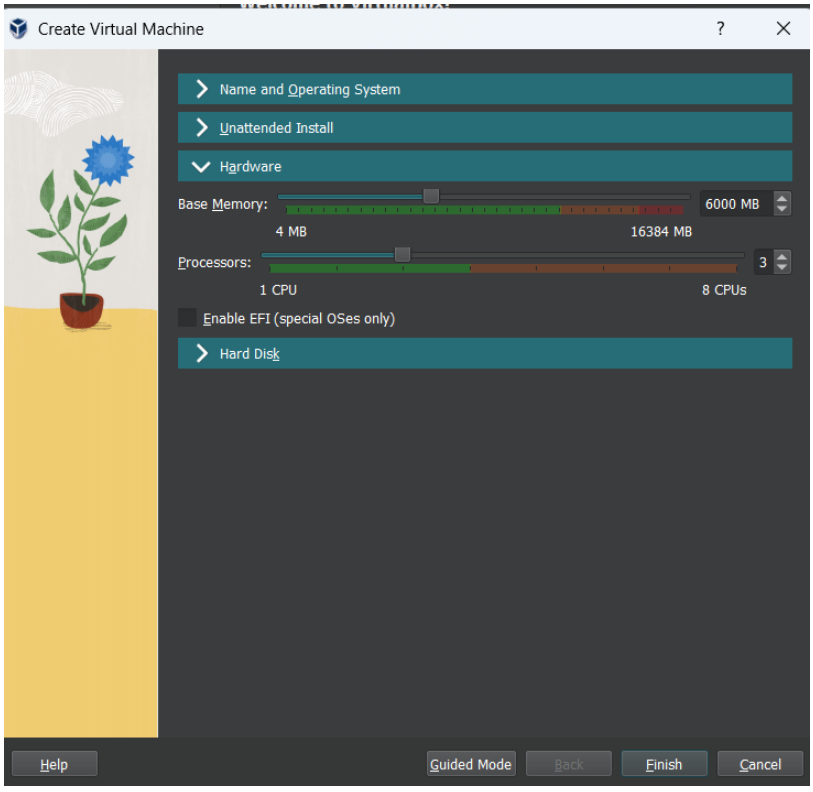
2) Φορτώσαμε το αρχείο image της έκδοσης Ubuntu που κατεβάσαμε δίνοντας το όνομα που θέλουμε να έχει το VM μας



3) Αλλάζουμε τον κωδικό και βάζουμε ένα όνομα στο hostname

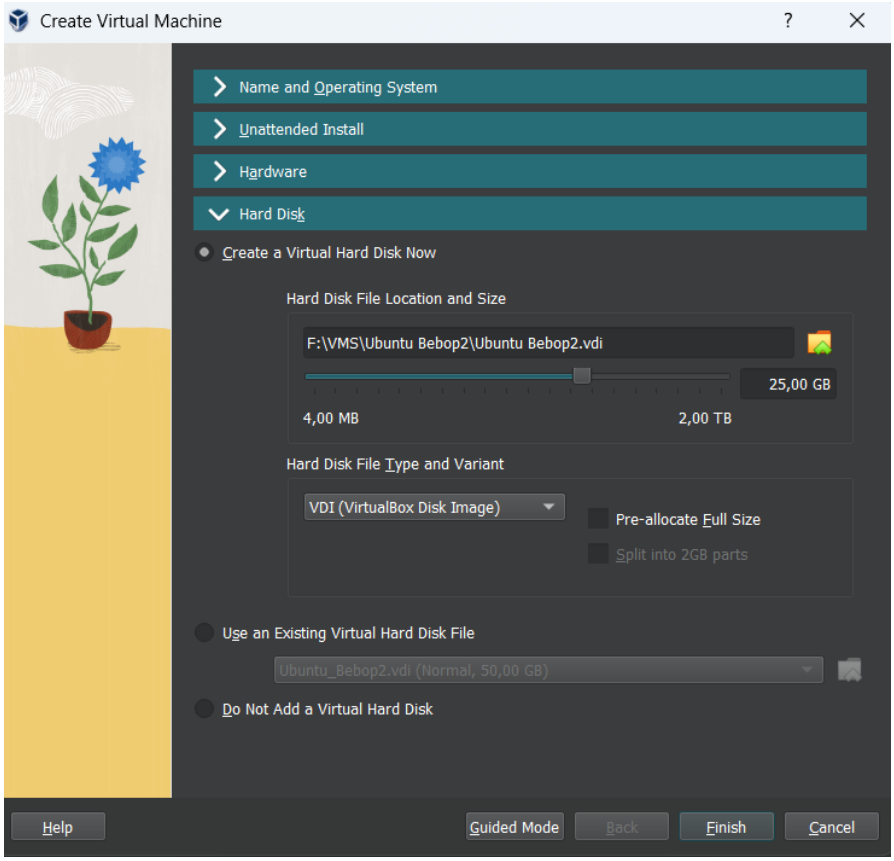


4) Στο tab Hardware επιλέγουμε το μέγεθος της μνήμης RAM που θέλουμε να χρησιμοποιήσουμε (6000MB) καθώς επίσης και τον αριθμό των πυρήνων (3) που θέλουμε να δεσμεύσουμε για το συγκεκριμένο VM.

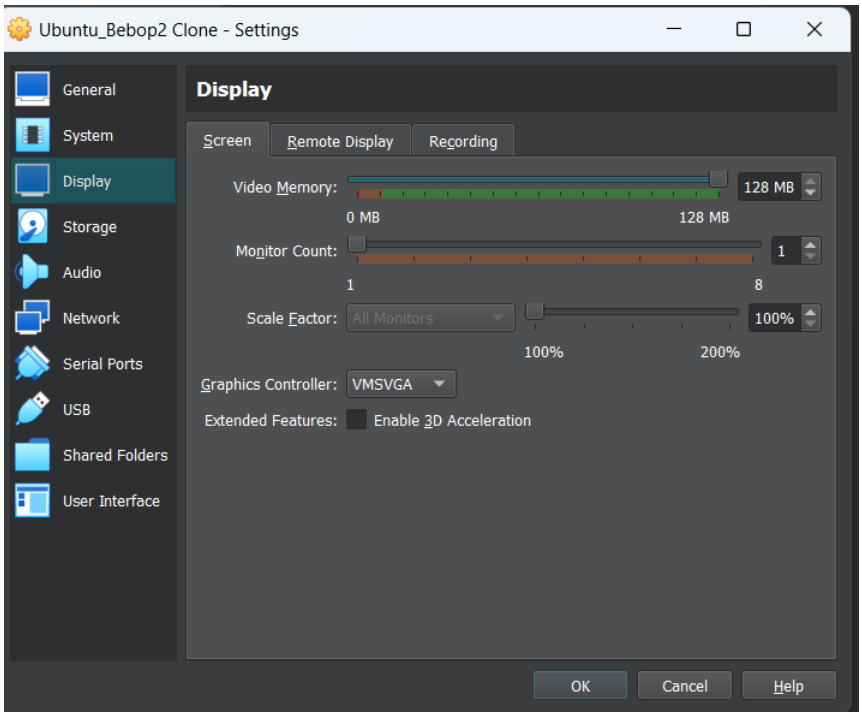


5) Στο μενού Hard Disk επιλέξαμε το μέγεθος του σκληρού δίσκου που θέλουμε να δεσμεύσουμε(25GB), καθώς επίσης και το path στο οποίο θα γίνει η

εγκατάσταση.

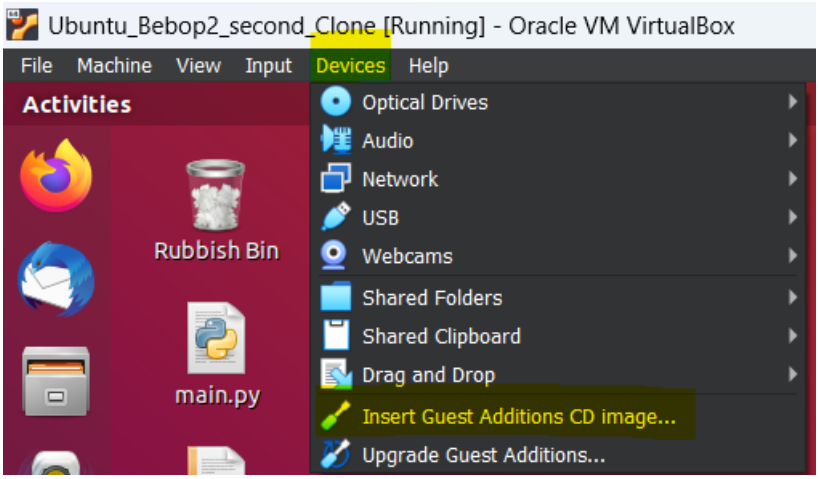


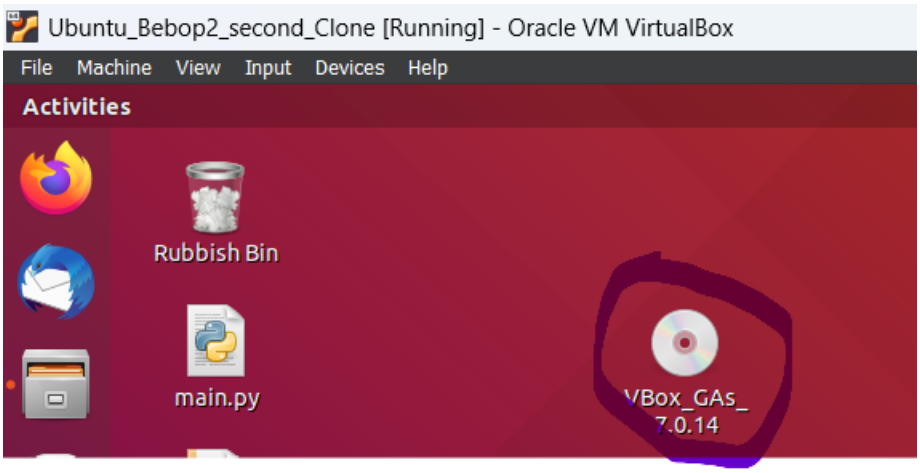
6) Αφού ολοκληρωθεί η δημιουργία της μηχανής με το λογισμικό Ubuntu που επιλέξαμε, θα χρειαστεί να αυξήσουμε τη video memory στο μέγιστο αριθμό που μας δίνει την δυνατότητα (128MB) για να μην έχουμε κάποιο πρόβλημα με το εντολές που περιέχουν εμφάνιση ή video capture.

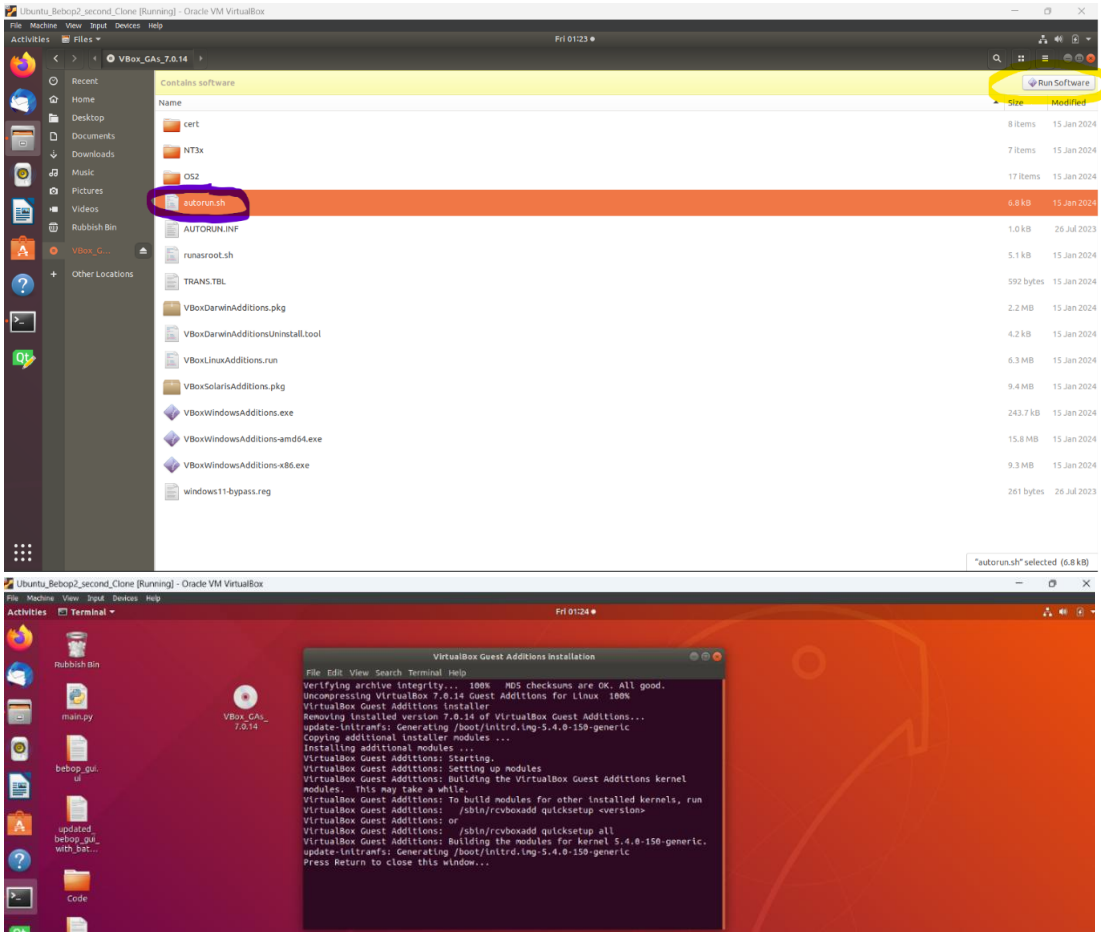


7) Επόμενο βήμα ήταν να μπορέσουμε να ξεκλειδώσουμε την επιλογή να χρησιμοποιήσουμε εξωτερικό usb Wi-Fi adapter καθώς μ' αυτόν τον τρόπο

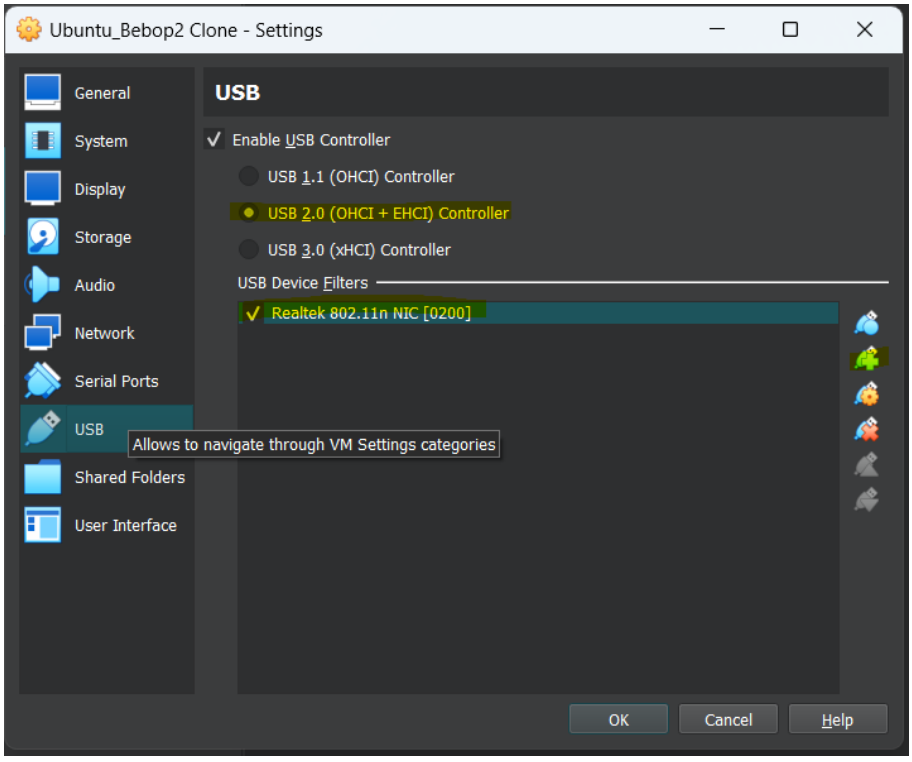
μόνο μπορούμε να συνδεθούμε μέσω Wi-Fi με το drone. Το πρόβλημα έγκειται στο γεγονός ότι το VirtualBox (συμπεριλαμβανομένης της έκδοσης 7) **δεν παρέχει άμεσο passthrough Wi-Fi** στα guest μηχανήματα. Αυτό συχνά μπερδεύει τους χρήστες, επειδή μπορεί να βλέπουν ότι ο υπολογιστής-host έχει Wi-Fi, αλλά μέσα στη VM εμφανίζεται μόνο **ενσύρματη (Ethernet)** κάρτα δικτύου. Οπότε για να ξεκλειδώσουμε αυτήν την επιλογή ακολουθούμε τα παρακάτω βήματα.



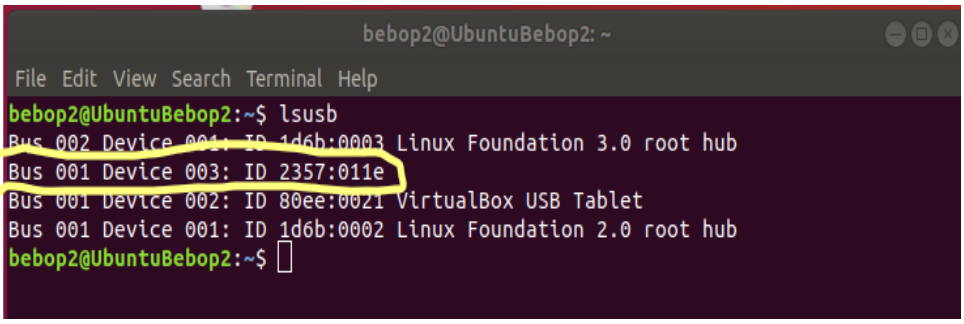




8) Όταν ολοκληρωθεί η εγκατάσταση τότε πηγαίνουμε στο tab USB επιλέγουμε USB 2.0 ή USB 3.0 ανάλογα τι είναι ο Wi-Fi adapter μας και τον προσθέτουμε στη λίστα.



9) Πληκτρολογώντας lsusb σ' ένα terminal στα Ubuntu μας εμφανίζει κάτι της μορφής



αυτό μας δείχνει ότι βλέπει την συσκευή αλλά χρειάζεται drivers για να λειτουργήσει σωστά. Η εγκατάσταση των drivers πραγματοποιείται πληκτρολογώντας μία σειρά εντολών στο terminal. Θα παραθέσω με τι σειρά τις εντολές που έτρεξα αλλά δεν θα τις εξηγήσω παραπάνω γιατί δεν αποτελούν σημαντικό κομμάτι του θέματος της εργασίας.

sudo apt update && sudo apt upgrade -y
sudo apt install build-essential dkms git linux-headers-$(uname -r)
git clone https://github.com/morrownr/8821au-20210708.git
cd 8821au-20210708
sudo ./install-driver.sh

Έτσι ολοκληρώθηκε η προετοιμασία του λειτουργικού συστήματος πάνω στο οποίο έγινε η εγκατάσταση του ROS.

2.1.2 Εγκατάσταση ROS Melodic Morenia

Η επιλογή της συγκεκριμένης έκδοσης ROS 1 έγινε με βάση το γεγονός ότι ο bebop autonomy driver έχει δοκιμαστεί και μπορεί να λειτουργήσει σε συγκεκριμένες εκδόσεις ROS, με βάση το documentation του.

Επειδή η διαδικασία εγκατάστασης έγινε βήμα- βήμα ακολουθώντας τις οδηγίες που υπάρχουν στο https://wiki.ros.org/kinetic/Installation/Ubuntu δεν χρήζει περαιτέρω επεξήγησης.

2.1.3 Εγκατάσταση του bebop autonomy

Η εγκατάσταση του driver είναι μια straight forward διαδικασία βήμα βήμα ακολουθώντας τις οδηγίες που υπάρχουν στο site https://bebop-autonomy. Παρακάτω γίνεται μια γρήγορη και σύντομη αναφορά στα βήματα και στον τρόπο εγκατάστασης του driver. Αρχικά θα πρέπει να αναφέρουμε ότι ο συγκεκριμένος driver έχει δοκιμαστεί για ROS (Indigo, Jade ή Kinetic με βάση το documentation, αλλά δουλεύει σωστά και με την έκδοση Melodic που χρησιμοποιήσαμε εμείς.

Βήματα εγκατάστασης:

1. Δημιουργία (ή χρήση υπάρχοντος) workspace τύπου `catkin`, π.χ.
   ```
   mkdir -p ~/bebop_ws/src && cd ~/bebop_ws
   catkin init
   ```
2. Κλωνοποίηση του αποθετηρίου `bebop_autonomy` στο φάκελο `src` του workspace:
   ```
   git clone https://github.com/AutonomyLab/bebop_autonomy.git
   src/bebop_autonomy
   ```
3. Ενημέρωση της βάσης δεδομένων `rosdep` και εγκατάσταση των dependencies (συμπεριλαμβανομένου του *parrot_arsdk*):
   ```
   rosdep update
   rosdep install --from-paths src -i
   ```
4. Τέλος, χτίσιμο του workspace με `catkin build`

Κεφάλαιο 3ο:Σύνδεση του Drone Parrot Bebop 2 με το PC

3.1 Περιβάλλον και Μεθοδολογία Δοκιμών

Μετά τη δημιουργία του GUI για τον χειρισμό του drone, έπρεπε να ξεκινήσουν οι δοκιμές που θα επιβεβαίωναν την ορθή λειτουργία του. Οι δοκιμές έπρεπε να γίνουν σε ανοιχτό εξωτερικό χώρο, κατά προτίμηση με κάποια βλάστηση γύρω γύρω, ώστε αν κάτι πήγαινε λάθος κι επειδή δεν ήξερα πόσο καλά θα ανταποκριθεί το drone, να μην φύγει ανεξέλεγκτο. Το GUI χειρισμού του drone δούλεψε σωστά από την πρώτη στιγμή.

Όμως το path planning section χρειάστηκε πάρα πολλές φορές επαναλήψεων και διορθώσεων στον κώδικα καθώς επίσης και πολλές φθορές στο drone αφού χρειάστηκε να εκτυπώσω στον 3d εκτυπωτή μου τουλάχιστον 4 τετράδες ελίκων. Αυτό οφειλόταν στο γεγονός ότι το σήμα και το στίγμα του GPS του drone δεν ήταν σταθερό ώστε να μπορέσω να ορίσω ένα σημείο εκκίνησης και να χρησιμοποιήσω συντεταγμένες γύρω από κάποια ορισμένη ακτίνα της τάξεως των 20-30 μέτρων.

Ο στόχος δεν ήταν η δημιουργία ενός GUI που θα παρέχει ακρίβεια εκατοστού, αλλά να μπορεί να κινηθεί μέσα στα επιτρεπτά όρια 2-3. Για να δοκιμαστεί το ότι μπορεί να ακολουθήσει μία αλληλουχία σωστών συντεταγμένων γινόταν αρχικά δοκιμές στο Freeflight Pro το οποίο δημιουργεί και αποθηκεύει τα waypoints σε MANlink file στη μνήμη του drone. Μετά τις συντεταγμένες αυτές τις έβαζα μία μία στο αντίστοιχο πεδίο στο GUI για να δώ αν θα είχε την ίδια η περίπου συμπεριφορά. Μετά χρησιμοποίησα πινέζες στο Google Map και τις συντεταγμένες τις περνούσα στο GUI. Η τρίτη εναλλακτική που χρησιμοποιήθηκε ήταν η δημιουργία MAVlink file χειροκίνητα στη μνήμη του drone στο path /internal_000/flightplans/flightPlan.mavlink.

Επίσης ενδεικτικά και χάριν επιστημονικής περιέργειας, έγινε δοκιμή ο χειρισμός του drone μέσω του ROS αλλά με χειριστήριο XBOX360 μέσω του πακέτου bebop_teleop αλλά ξανά λόγω του περιορισμού της εμβέλειας του Wi-Fi, κατέληξε σε ατύχημα του drone. Πραγματοποίησα κάποια στιγμή κάποιες δοκιμές σε κλειστό χώρο με ένα GUI που έφτιαξα και που αντι για συντεταγμένες έδινες σχετική απόσταση μεταξύ των σημείων το οποίο λειτούργησε σωστά γιατί ουσιαστικά δε στηριζόταν στο GPS signal αλλά σε τοπικά XY μέτρα.

3.1.1 Σύνδεση μέσω Wi-Fi

Το Bebop 2 δημιουργεί το δικό του WiFi access point μόλις ανάψει. Δεν χρειάζεται router — το drone το ίδιο είναι ο router. Η διαδικασία σύνδεσης είναι ως εξής:

1. Ανάβουμε το drone — ακούγονται δύο beep και το κόκκινο LED αρχίζει να αναβοσβήνει.
2. Μόλις το κόκκινο LED ανάψει σταθερά, στον υπολογιστή βλέπουμε δίκτυο με όνομα της μορφής `BebopDrone-XXXXXX` στα διαθέσιμα δίκτυα W-Fi
3. Συνδεόμαστε σε αυτό το δίκτυο χωρίς κωδικό
4. Το drone παίρνει πάντα τη σταθερή IP `192.168.42.1`
5. Ο υπολογιστής παίρνει αυτόματα IP στο εύρος `192.168.42.x` (π.χ. 192.168.42.5)

Η επιβεβαίωση της σύνδεσης με το drone γίνεται πληκτρολογώντας:

```
bash
ping 192.168.42.1
```

Αν απαντά, το drone είναι online.

Το WiFi λειτουργεί σε δύο ζώνες, 2.4 GHz και 5 GHz. Η 5 GHz έχει μικρότερη εμβέλεια αλλά μεγαλύτερο bandwidth — καλύτερη για βίντεο σε κοντινές αποστάσεις. Η εμβέλεια με smartphone είναι περίπου τα 300 μέτρα, με Skycontroller 2 φτάνει 2 χιλιόμετρα. Μετά τα 300 μέτρα η WiFi σύνδεση χάνεται — γι' αυτό το MAVLink autopilot είναι κρίσιμο για αυτόνομη πτήση: εκτελεί το πλάνο ανεξάρτητα από τη σύνδεση μετά την εκκίνηση.

3.1.2 Σύνδεση με ROS - bebop_autonomy

Γίνεται με την εγκατάσταση και την χρήση του driver bebop_autonomy που είναι ένα ROS package επικοινωνίας μεταξύ του ιδιόκτητου πρωτοκόλλου ARSDKv3 της Parrot και του ROS topic/message συστήματος.

Το ARSDK (ARDrone Software Development Kit) είναι η ιδιόκτητη βιβλιοθήκη της Parrot που χειρίζεται την κρυπτογραφημένη WiFi επικοινωνία με το drone σε UDP port 44444. Ο bebop_autonomy driver κάθεται από πάνω και κάνει public/subscribe σε ROS topics. Η εγκατάστασή του περιεγράφηκε στο κεφάλαιο 2.2.10.

Εκκίνηση:

```
# Terminal 1
roscore
# Terminal 2
roslaunch bebop_driver bebop_node.launch
```

ROS Topic (Εντολές)	Τύπος	Λειτουργία
/bebop/takeoff	std_msgs/Empty	Απογείωση
/bebop/land	std_msgs/Empty	Προσγείωση
/bebop/reset	std_msgs/Empty	Emergency — κόβει κινητήρες αμέσως
/bebop/cmd_vel	geometry_msgs/Twist	Χειροκίνητη κίνηση, τιμές -1.0 έως +1.0
/bebop/camera_control	geometry_msgs/Twist	Pan angular.z ±180°, Tilt angular.y ±83°
/bebop/autoflight/start	std_msgs/String	Εκκίνηση onboard autopilot
/bebop/autoflight/pause	std_msgs/Empty	Παύση autopilot
/bebop/autoflight/stop	std_msgs/Empty	Διακοπή autopilot

ROS Topic (Τηλεμετρία)	Τύπος και Περιεχόμενο
/bebop/image_raw/compressed	CompressedImage — JPEG βίντεο 1080p ~30fps
/bebop/fix	NavSatFix — GPS: latitude, longitude, altitude
/bebop/odom	Odometry — θέση x,y,z από IMU + optical flow
/bebop/states/.../BatteryStateChanged	CommonCommonStateBatteryStateChanged — .percent (uint8, 0-100)
/bebop/states/.../FlyingStateChanged	Ardrone3PilotingStateFlyingStateChanged — state 0-8

Η μπαταρία ΔΕΝ είναι Float32. Χρησιμοποιεί CommonCommonStateBatteryStateChanged με .percent πεδίο (uint8, εύρος 0-100). Η χρήση Float32 προκαλεί σιωπηλή αποτυχία — ο subscriber εγγράφεται αλλά δεν λαμβάνει ποτέ μήνυμα.

3.1.3 Επιβεβαίωση Σωστής Σύνδεσης

Τρία επίπεδα επαλήθευσης:

```
# Επίπεδο 1 — Δίκτυο
ping 192.168.42.1   # latency < 10ms = OK

# Επίπεδο 2 — ROS Driver
rosnode list          # πρέπει: /bebop
rostopic list | grep bebop   # δεκάδες topics

# Επίπεδο 3 — Ζωντανά Δεδομένα
rostopic echo /bebop/states/common/CommonState/BatteryStateChanged
rostopic hz /bebop/image_raw/compressed   # ~30 Hz
# Δοκιμή χωρίς κίνηση — στρέψιμο κάμερας:
rostopic pub /bebop/camera_control geometry_msgs/Twist '{angular: {z: 30.0, y: 0.0}}'
```

3.1.4 Δοκιμές Επικοινωνίας

Δοκιμή	Τι δοκιμάστηκε	Αποτέλεσμα
1 — Σύνδεση	Ping, rosnode, battery/state topics	OK. Ανακαλύφθηκε Float32 bug μπαταρίας.
2 — Βίντεο	image_view για /bebop/image_raw	cv_bridge αποτυχία Python 3. Λύση: compressed+OpenCV.
3 — Χειροκίνητος	cmd_vel Twist από terminal	OK. Ανακαλύφθηκε body frame σύστημα αναφοράς.
4 — GPS	NavSatFix εσωτερικά	Σφάλμα 34χλμ λόγω multipath GPS σε εσωτερικό χώρο.

`5 — GPS cmd_vel`	Κλειστός βρόχος GPS+cmd_vel	Αποτυχία: WiFi latency, frame mismatch, fly-away.
`6 — MAVLink`	FTP upload + af_start	Επιτυχής αυτόνομη πτήση. Ακρίβεια 2-6m.

3.1.5 Πρόβλημα cv_bridge

Η βιβλιοθήκη cv_bridge δεν λειτουργεί με Python 3 και ROS Melodic. Εμφανίζεται:

```
ImportError: dynamic module does not define module export function (PyInit_cv_bridge_boost)
```

Λύση: χρήση /bebop/image_raw/compressed με άμεση αποκωδικοποίηση JPEG:

```
# ΣΩΣΤΟ — χωρίς cv_bridge:
arr = np.frombuffer(msg.data, dtype=np.uint8).copy()
cv_image = cv2.imdecode(arr, cv2.IMREAD_COLOR)
```

Η νέα μέθοδος χρησιμοποιεί λιγότερο WiFi bandwidth (JPEG vs raw pixels) και επομένως έχει μικρότερη καθυστέρηση βίντεο.

Κεφάλαιο 4ο: Αυτόνομη GPS Πλοήγηση — Onboard Autopilot και MAVLink

4.1.1 Η Αποτυχημένη Πρώτη Προσέγγιση — cmd_vel GPS Loop

Η αρχική προσέγγιση ήταν κλειστός βρόχος GPS ελέγχου από το laptop: ανάγνωση θέσης από /bebop/fix, υπολογισμός απόκλισης από waypoint, αποστολή Twist εντολών μέσω /bebop/cmd_vel. Κατά τις δοκιμές, το drone αιωρείτο ασταθώς γύρω από τα waypoints. Σε πολλές περιπτώσεις παρατηρήθηκε fly-away συμπεριφορά που σταμάτησε μόνο με emergency stop ή με σύγκρουση με κάποιο εμπόδιο και αφού χρειάστηκε να το ακολουθήσω λόγω διακοπής επαφής με το σήμα Wi-Fi. Δοκίμασα με κάποιο άλλο Wi-Fi adapter μεγαλύτερης εμβέλειας αλλά χωρίς αποτέλεσμα.$#Ψ

Απέτυχε για τέσσερις θεμελιώδεις λόγους:

Αιτία Αποτυχίας	Τεχνική Εξήγηση
`WiFi latency 50-200ms`	Καθυστέρηση βρόχου → 15-60cm σφάλμα θέσης ανά κύκλο ελέγχου

GPS ρυθμός 5Hz	Ανεπαρκής για ομαλό κλειστό βρόχο ελέγχου στα 10Hz
Body vs world frame	cmd_vel χρειάζεται yaw γνώση που η οδομετρία δεν παρέχει αξιόπιστα

4.1.2 Onboard Autopilot

Η μελέτη του bebop_autonomy driver αποκάλυψε την ορθή αρχιτεκτονική: χρήση του ενσωματωμένου autopilot του Bebop μέσω αρχείων MAVLink. Ο autopilot εκτελεί βρόγχο ελέγχου στα ~200Hz χρησιμοποιώντας IMU fusion, barometer, optical flow και GPS.

Χαρακτηριστικό	cmd_vel + GPS Loop	MAVLink Onboard Autopilot
Συχνότητα βρόχου	~10Hz (WiFi latency)	~200Hz (onboard ARM CPU)
WiFi εξάρτηση	Συνεχής — κρίσιμη	Μόνο για αρχική εκκίνηση
GPS αξιοποίηση	Ακατέργαστο signal, αστάθεια	Φιλτραρισμένο Parrot SDK
Αξιοπιστία	Fly-away κίνδυνος	Ισοδύναμο FreeFlight Pro

Ροή εκτέλεσης αποστολής:

1. GUI δημιουργεί QGC WPL 120 αρχείο με τα GPS waypoints
2. FTP upload στο /internal_000/flightplans/flightPlan.mavlink του drone
3. Δημοσίευση String μηνύματος στο /bebop/autoflight/start
4. bebop_autonomy driver προωθεί εντολή μέσω ARSDK UDP (port 44444)
5. Drone εκτελεί αποστολή αυτόνομα — χωρίς WiFi σύνδεση

4.2 Τι είναι το MAVLink

Το MAVLink (Micro Air Vehicle Link) είναι ανοιχτό πρωτόκολλο επικοινωνίας για UAVs, σχεδιασμένο το 2009 από τον Lorenz Meier. Αποτελεί de facto standard — χρησιμοποιείται από ArduPilot, PX4, DJI, Parrot, QGroundControl. Για τον Bebop 2 χρησιμοποιείται αποκλειστικά ως format αρχείου σχεδίου πτήσης — όχι ως real-time πρωτόκολλο.

4.2.1 Δομή Αρχείου MAVLink — QGC WPL 120

Το αρχείο είναι ASCII κείμενο με κεφαλίδα και μία εντολή ανά γραμμή, διαχωρισμένη με tabs:

```
QGC WPL 120
IDX  CUR  FRM  CMD   P1  P2   P3  P4  LAT        LON        ALT   AC
0    0    3    178   0   3.0  -1  0   0          0          0     1
1    0    3    2500  0   30   0   0   0          0          0     1
2    0    3    16    0   5    0   0   40.584946  23.339903  10.0  1
3    0    3    16    0   5    0   0   40.586100  23.341200  10.0  1
4    0    3    21    0   0    0   0   40.586100  23.341200  10.0  1
5    0    3    2501  0   0    0   0   0          0          0     1
```

Στήλη	Όνομα	Τιμές και Επεξήγηση
0	INDEX	Αύξων αριθμός 0,1,2,... Ο autopilot εκτελεί σε αυτή τη σειρά.
1	CURRENT	0=κανονική εντολή. 1=τρέχουσα (μόνο για home position).
2	FRAME	3=MAV_FRAME_GLOBAL_RELATIVE_ALT: ύψος AGL (σχετικό με σημείο απογείωσης). 0=AMSL (απόλυτο υψόμετρο).
3	COMMAND	Τύπος εντολής: 16=waypoint, 21=land, 178=speed, 2500=wait, 2501=end.
4-7	PARAM 1-4	Παράμετροι εντολής. Νόημα εξαρτάται από CMD.
8	LATITUDE	Δεκαδικές μοίρες, 7 ψηφία (~1cm ακρίβεια). 0 αν δεν χρησιμοποιείται.
9	LONGITUDE	Δεκαδικές μοίρες, 7 ψηφία. 0 αν δεν χρησιμοποιείται.
10	ALTITUDE	Μέτρα AGL αν FRAME=3. 0 αν δεν χρησιμοποιείται.
11	AUTOCONTINUE	1=αυτόματη συνέχεια. Πάντα 1 για Bebop.

CMD	Όνομα MAVLink	Παράμετροι και Χρήση για Bebop 2
178	MAV_CMD_DO_CHANGE_SPEED	P2=ταχύτητα m/s (εύρος: 0.5-10). P3=-1 (throttle αμετάβλητο). Πάντα πρώτη εντολή.
2500	CONDITION_DELAY (Parrot)	P2=δευτερόλεπτα αναμονής. 30s για GPS stabilisation μετά απογείωση. Κρίσιμο.
16	MAV_CMD_NAV_WAYPOINT	P2=ακτίνα αποδοχής (5m). LAT/LON/ALT=προορισμός. Drone προχωρά όταν εντός 5m.
21	MAV_CMD_NAV_LAND	Προσγείωση. Χωρίς αυτό: drone αιωρείται αόριστα στο τελευταίο WP.
2501	END_OF_MISSION (Parrot)	Σήμα τέλους — Parrot-specific. Χωρίς αυτό ο autopilot δεν αναγνωρίζει τέλος.

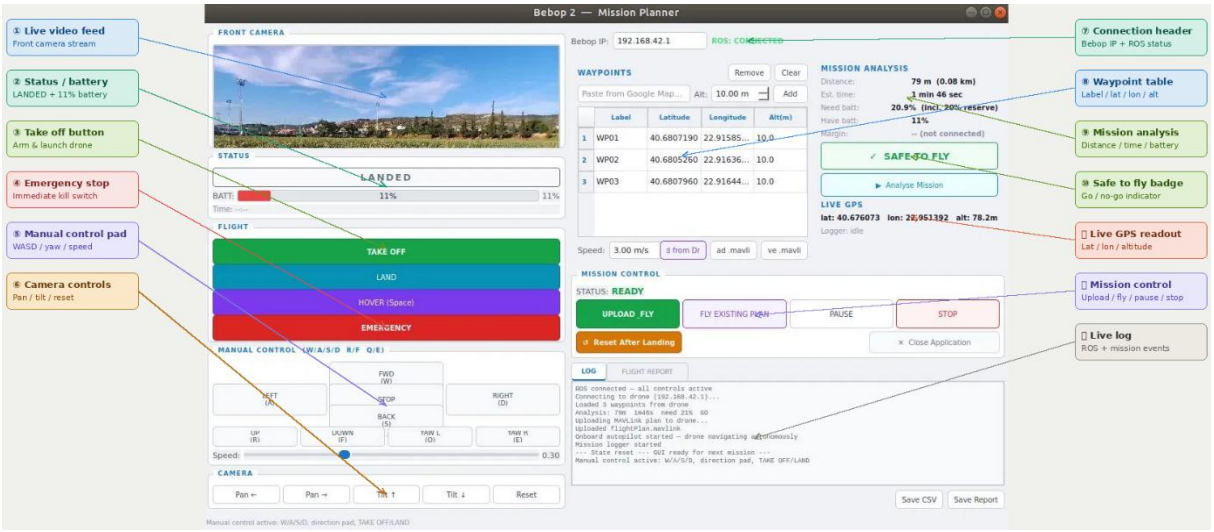
FRAME=3 (MAV_FRAME_GLOBAL_RELATIVE_ALT): ύψος είναι AGL — σχετικό με σημείο απογείωσης. Αν χρησιμοποιηθεί FRAME=0 (AMSL), το drone πετά σε λάθος ύψος αν η τοποθεσία δεν είναι στη στάθμη της θάλασσας.

4.3 Κατεύθυνση Μύτης

Κατά τις δοκιμές παρατηρήθηκε ότι στο FreeFlight Pro η μύτη στρέφεται αυτόματα προς την κατεύθυνση κίνησης, ενώ στη ROS υλοποίηση παραμένει σταθερή. Αυτό αποκαλύπτει υβριδική αρχιτεκτονική FreeFlight: onboard autopilot για θέση + smartphone για συνεχή yaw

εντολές. Στη ROS υλοποίηση επιλέχθηκε σκόπιμα καθαρός onboard autopilot για WiFi-ανεξαρτησία και αρχιτεκτονική καθαρότητα.

Κεφάλαιο 5ο: Γραφικό Περιβάλλον Χρήστη bebop_mission_gui.py



5.1 Αρχιτεκτονική Συστήματος

Το GUI αναπτύχθηκε σε καθαρή Python χωρίς αρχεία .ui από Qt Designer. Χρησιμοποιεί τρία QThread workers για παράλληλη εκτέλεση, εξασφαλίζοντας ότι το κύριο νήμα δεν μπλοκάρει ποτέ:

Thread	Worker	Λειτουργία	Επικοινωνία με GUI
`Main Thread`	App (QMainWindow)	UI events, rendering, timers	Άμεσα
`_rt (ROS Thread)`	RosWorker	rospy.spin() — ROS callbacks	Qt signals (sig_batt κ.λπ.)
`_vt (Video Thread)`	VideoWorker	JPEG decode, resize, cvtColor	sig_frame → setPixmap()
`_ft (FTP Thread)`	FtpWorker	FTP upload/download	sig_done → _ftp_done()

ℹ *Κάθε thread επικοινωνεί με το GUI αποκλειστικά μέσω Qt signals. Αυτό εγγυάται thread safety — μόνο το main thread αγγίζει τα widgets, ποτέ απευθείας από ROS ή FTP thread.*

5.2 Δομή GUI — Panel Builder Methods

Μέθοδος	Περιεχόμενο Πάνελ

`_build_camera_card()`	Οθόνη βίντεο QLabel 480×270px. Σταθερό μέγεθος (16:9 αναλογία Bebop 2).
`_build_status_card()`	Κατάσταση drone, QProgressBar μπαταρίας (0-100%), χρονόμετρο πτήσης MM:SS.
`_build_flight_card()`	TAKE OFF (με confirmation), LAND, HOVER (Space), EMERGENCY. Ελάχ. ύψος 36px.
`_build_direction_pad()`	QGridLayout cross pad + UP/DOWN/YAW row + QSlider ταχύτητας (5-80 → 0.05-0.80).
`_build_camera_controls()`	Pan αριστερά/δεξιά ±5°/κλικ (εύρος ±180°). Tilt πάνω/κάτω ±5°/κλικ (εύρος ±83°). Reset.
`_build_waypoint_panel()`	Paste box, QTableWidget (4 στήλες), QDoubleSpinBox ύψους (1-120m), Load/Save .mavlink.
`_build_analysis_panel()`	5 στατιστικές γραμμές, Go/No-Go label, Analyse κουμπί, live GPS, logger status.
`_build_mission_control()`	Status label + UPLOAD & FLY, FLY EXISTING, PAUSE, STOP. Ελάχ. ύψος 40px.
`_build_log_panel()`	QTabWidget: LOG tab + FLIGHT REPORT tab. Κουμπιά Save CSV, Save Report.
`_wire_signals()`	Όλες οι signal→slot συνδέσεις. Καλείται τελευταία αφού υπάρχουν όλα τα widgets.

5.3 Ανάλυση Αποστολής — Go/No-Go

Πριν από κάθε αποστολή, η _analyse() υπολογίζει και εμφανίζει:

- Απόσταση: άθροισμα Haversine αποστάσεων μεταξύ διαδοχικών waypoints (μέτρα και χιλιόμετρα)
- Χρόνος: 30s GPS αναμονή + χρόνος cruise + κάθετη κίνηση + 50s takeoff/landing (σε min:sec)
- Κατανάλωση: (hover_time×900mA + cruise_time×1400mA) / 2700mAh × 100% + 20% αποθεματικό
- Go/No-Go: ελέγχει χρόνο >25min, απόσταση >300m, μπαταρία ανεπαρκή

5.4 Σύστημα Καταγραφής — MissionLogger

Καταγράφει κάθε δευτερόλεπτο κατά τη διάρκεια αυτόνομης πτήσης:

Πεδίο CSV	Πηγή	Χρήση
`time_s`	Εσωτερικός timer	Δευτερόλεπτα από έναρξη αποστολής

`lat, lon, alt_gps`	/bebop/fix (NavSatFix)	GPS θέση για waypoint accuracy
`odom_x, odom_y, odom_z`	/bebop/odom (Odometry)	Θέση σε μέτρα από σημείο απογείωσης
`battery`	BatteryStateChanged (.percent)	Κατανάλωση κατά την πτήση
`state`	FlyingStateChanged (.state)	Ανάλυση κατάστασης πτήσης

Μετά πτήση, generate_report() υπολογίζει waypoint accuracy ως ελάχιστη Haversine απόσταση μεταξύ κάθε planned WP και οποιασδήποτε καταγεγραμμένης GPS θέσης.

Κεφάλαιο 6ο: Αναλυτική Επεξήγηση Κώδικα bebop_mission_gui.py

6.1 Εισαγωγές Βιβλιοθηκών (Γραμμές 1–39)

Το αρχείο ξεκινά με δύο κρίσιμες γραμμές κεφαλίδας, ακολουθούν οι εισαγωγές βιβλιοθηκών ομαδοποιημένες σε τρία μπλοκ try/except για ανεκτικότητα σφαλμάτων.

Κώδικας / Γραμμή	Επεξήγηση
`#!/usr/bin/env python3`	Shebang: λέει στο Linux να εκτελέσει το αρχείο με Python 3. Απαραίτητο για chmod +x εκτέλεση.
`# -*- coding: utf-8 -*-`	Κωδικοποίηση UTF-8: επιτρέπει ελληνικούς χαρακτήρες σε strings και σχόλια.
`import sys`	sys.argv: ορίσματα γραμμής εντολών για QApplication. sys.exit(): τερματισμός με Qt exit code.
`import os`	os.unlink(): διαγραφή temp αρχείου FTP. os.path.basename(): όνομα αρχείου χωρίς μονοπάτι.
`import math`	math.radians(), math.sin(), math.cos(), math.atan2(), math.sqrt(): χρησιμοποιούνται αποκλειστικά στο haversine().
`import ftplib`	Ενσωματωμένη Python βιβλιοθήκη FTP. ftplib.FTP(): σύνδεση, login, cwd, storbinary, retrlines, quit.
`import tempfile`	tempfile.NamedTemporaryFile(): προσωρινό αρχείο στον δίσκο για FTP upload. Ο δρόμος απαιτεί πραγματικό αρχείο.
`import time`	time.time(): unix timestamps. time.strftime(): μορφοποίηση ημερομηνίας για ονόματα αρχείων.
`import re`	re.findall(): βρίσκει δεκαδικούς αριθμούς σε κείμενο για ανάλυση συντεταγμένων Google Maps.
`import numpy as np`	np.frombuffer(): μετατροπή bytes→array χωρίς αντιγραφή. np.ascontiguousarray(): εγγυάται γραμμική μνήμη για QImage.
`try: import rospy ...`	try/except: αν το ROS δεν είναι εγκατεστημένο, το GUI εκκινεί σε demo mode. ROS_OK=True/False.
`from std_msgs.msg import Empty, Float32, String`	Empty: εντολές χωρίς data (takeoff, land). Float32: αριθμητική τιμή. String: όνομα MAVLink αρχείου.

`from geometry_msgs.msg import Twist`	Twist: linear.x/y/z (γραμμική ταχύτητα) + angular.x/y/z (γωνιακή). Χρησιμοποιείται για cmd_vel ΚΑΙ camera_control.
`from sensor_msgs.msg import CompressedImage, NavSatFix`	CompressedImage: JPEG bytes κάμερας. NavSatFix: GPS (latitude, longitude, altitude AMSL).
`from nav_msgs.msg import Odometry`	Odometry: θέση x,y,z από IMU+optical flow. Χρησιμοποιείται για logging.
`from bebop_msgs.msg import Ardrone3PilotingStateFlyingStateChanged`	Κατάσταση πτήσης: integer 0-8. Μόνο μέσω bebop_autonomy driver.
`from bebop_msgs.msg import CommonCommonStateBatteryStateChanged`	Μπαταρία: πεδίο .percent (uint8, 0-100). ΟΧΙ Float32 — αυτό ήταν το bug.
`try: import cv2`	OpenCV: αποκωδικοποίηση JPEG, resize, cvtColor. Αν λείπει: cv2=None, βίντεο απενεργοποιείται.
`from PyQt5.QtWidgets import (...)`	Όλα τα widget classes. QMainWindow: κύριο παράθυρο. QWidget: βάση widget. QSplitter: συρόμενος διαχωριστής.
`from PyQt5.QtCore import Qt, QTimer, pyqtSignal, QThread, QObject`	QTimer: χρονόμετρα 100ms/1000ms. pyqtSignal: thread-safe σήματα. QThread: παράλληλα νήματα.
`from PyQt5.QtGui import QPixmap, QImage, QColor`	QPixmap: εμφάνιση εικόνας σε QLabel. QImage: μετατροπή numpy array σε Qt. QColor: χρωματισμός.

6.2 Σταθερές Συστήματος (Γραμμές 41–57)

Κώδικας / Γραμμή	Επεξήγηση
`BEBOP_IP = "192.168.42.1"`	Σταθερή IP κάθε Bebop 2. Δεν αλλάζει ποτέ μεταξύ συσκευών. Χρησιμοποιείται από FtpWorker.
`FTP_DIR = "/internal_000/flightplans"`	Διαδρομή αποθήκευσης σχεδίων πτήσης στο εσωτερικό Linux filesystem του drone.
`PLAN_NAME = "flightPlan.mavlink"`	Ακριβές όνομα (κεφαλαίο P): FreeFlight Pro πάντα το ονομάζει έτσι. Linux filesystem: case-sensitive.
`BATT_MAH = 2700`	Χωρητικότητα μπαταρίας LiPo: 2700 mAh. Επίσημο spec Parrot Bebop 2.
`HOVER_MA = 900`	Μέση κατανάλωση ρεύματος σε αιώρηση: ~900 mA. Μετρημένο πειραματικά.

`CRUISE_MA = 1400`	Κατανάλωση σε cruise ~3 m/s: ~1400 mA. Αυξάνεται με την ταχύτητα.
`MAX_MIN = 25`	Μέγιστος χρόνος πτήσης κατά Parrot: 25 λεπτά. Χρησιμοποιείται στο Go/No-Go έλεγχο.
`RESERVE = 20`	Αποθεματικό 20%: δεν χρησιμοποιείται ποτέ για πτήση. Προστέθηκε στο need% στο analyse().
`STATES = {0:('LANDED','#64748b'), ...}`	Λεξικό: κωδικός (int) → (κείμενο, χρώμα hex). 9 καταστάσεις. Χρώμα εμφανίζεται στο lbl_state.
`0:('LANDED','#64748b')`	Προσγειωμένο: σκούρο γκρι. Αρχική και τελική κατάσταση κάθε πτήσης.
`1:('TAKING OFF','#f59e0b')`	Απογείωση: κεχριμπάρι. Μεταβατική κατάσταση ~2-5 δευτερόλεπτα.
`2:('HOVERING','#22d3ee')`	Αιώρηση: κυανό. Σταθερή θέση χωρίς εντολή κίνησης ή αποστολή.
`3:('FLYING','#4ade80')`	Πτήση: πράσινο. Εκτελεί εντολή κίνησης ή autopilot αποστολή.
`4:('LANDING','#f59e0b')`	Προσγείωση: κεχριμπάρι. Μεταβατική κατάσταση ~3-8 δευτερόλεπτα.
`5:('EMERGENCY','#ef4444')`	Έκτακτη ανάγκη: κόκκινο. Κινητήρες κομμένοι — drone πέφτει.
`6:('USR TAKEOFF','#a78bfa')`	Χειροκίνητη απογείωση από χειριστή: μοβ.
`7:('RAMPING','#fb923c')`	Επιτάχυνση κινητήρων: πορτοκαλί. Αμέσως πριν την απογείωση.
`8:('EMG LAND','#ef4444')`	Αναγκαστική προσγείωση: κόκκινο. Χαμηλή μπαταρία ή error.

6.3 Βοηθητικές Συναρτήσεις (Γραμμές 60–114)

6.3.1 haversine(la1, lo1, la2, lo2) — Γραμμές 63-68

Ορθοδρομική απόσταση μεταξύ δύο GPS σημείων. Χρησιμοποιείται από analyse() για συνολική απόσταση και από generate_report() για σφάλμα waypoint.

Κώδικας / Γραμμή	Επεξήγηση
`R = 6_371_000`	Μέση ακτίνα Γης σε μέτρα. Ο _ είναι διαχωριστής χιλιάδων (Python 3.6+) — αναγνωσιμότητα.
`p1,p2 = math.radians(la1), math.radians(la2)`	Μετατροπή πλάτους σε ακτίνια. Η τριγωνομετρία απαιτεί ακτίνια.
`dp = math.radians(la2-la1)`	Διαφορά πλάτους Δφ σε ακτίνια.
`dl = math.radians(lo2-lo1)`	Διαφορά μήκους Δλ σε ακτίνια.
`a = sin(dp/2)**2 + cos(p1)*cos(p2)*sin(dl/2)**2`	Haversine τύπος: a = sin²(Δφ/2) + cos(φ1)·cos(φ2)·sin²(Δλ/2). Τιμή 0-1.
`return R*2*atan2(sqrt(a), sqrt(1-a))`	Αντίστροφος Haversine: 2R·arctan2(√a,√(1-a)) = τόξο σε μέτρα. Εύρος: 0 - ~20.000 χλμ.

6.3.2 parse_coords(text) — Γραμμές 70-74

Εξάγει (lat, lon) από οποιοδήποτε κείμενο. Λειτουργεί με copy-paste από Google Maps.

Κώδικας / Γραμμή	Επεξήγηση
`nums = re.findall(r"[-+]?\d+\.\d+", text.strip())`	Regex: βρίσκει δεκαδικούς με προαιρετικό πρόσημο. "40.584946, 23.339903" → ["40.584946","23.339903"].
`if len(nums) >= 2: return float(nums[0]), float(nums[1])`	Αν ≥2 αριθμοί: πρώτος=lat, δεύτερος=lon. Αγνοεί τυχόν τρίτο (ύψος Google Maps).
`raise ValueError("Could not parse coordinates")`	Εκπέμπει εξαίρεση αν <2 αριθμοί. Ο _add_wp() την πιάνει και δείχνει QMessageBox.

6.3.3 parse_mavlink(content) — Γραμμές 76-85

Αναλύει QGC WPL αρχείο. Επιστρέφει λίστα dicts με cmd, p2, lat, lon, alt. Παραλείπει σιωπηλά κάθε κατεστραμμένη γραμμή.

Κώδικας / Γραμμή	Επεξήγηση
`content.strip().splitlines()[1:]`	Παραλείπει γραμμή 0 (κεφαλίδα 'QGC WPL 120'). [1:] = από δεύτερη γραμμή.

`p = line.strip().split()`	Διαχωρισμός με whitespace. QGC χρησιμοποιεί tabs αλλά το split() δουλεύει και με κενά.
`if len(p) < 12: continue`	MAVLink απαιτεί 12 στήλες. Κενές/ελλιπείς γραμμές παραλείπονται.
`'cmd': int(p[3])`	Στήλη 3 = command. 16=NAV_WAYPOINT, 21=NAV_LAND, 178=DO_CHANGE_SPEED, 2500=CONDITION_DELAY, 2501=END.
`'p2': float(p[5])`	Στήλη 5 = parameter 2. Για CMD 16: ακτίνα αποδοχής (5m). Για CMD 178: ταχύτητα m/s.
`'lat':float(p[8]),'lon':float(p[9]),'alt':float(p[10])`	Στήλες 8,9,10 = GPS. FRAME 3 = alt AGL (πάνω από σημείο απογείωσης, όχι απόλυτο υψόμετρο).
`except: continue`	Silent skip: αγνοεί γραμμές με σφάλμα parsing χωρίς να διακόψει την ανάλυση.

6.3.4 analyse(waypoints, speed=3.0, battery=None) — Γραμμές 87-101

Πλήρης ανάλυση βιωσιμότητας αποστολής. Είσοδος: [(lat,lon,alt),...], speed m/s, battery% ή None.

Κώδικας / Γραμμή	Επεξήγηση
`if not waypoints: return None`	Guard: αν κενή λίστα, επιστρέφει None. Αποτρέπει διαίρεση με μηδέν.
`dist = sum(haversine(...) for i in range(len-1))`	Συνολική οριζόντια απόσταση: άθροισμα Haversine μεταξύ διαδοχικών waypoints. Μονάδα: μέτρα.
`altd = sum(abs(wps[i+1][2]-wps[i][2]) ...)`	Συνολική κάθετη μεταβολή: άθροισμα απόλυτων διαφορών

	ύψους. Κάθε μέτρο ≈ 1 δευτερόλεπτο.
`ct = dist / max(speed, 0.1)`	Χρόνος cruise σε δευτερόλεπτα. max() αποτρέπει ZeroDivisionError αν speed=0.
`ft = 30 + ct + altd + 20 + 30`	Συνολικός χρόνος: 30s GPS αναμονή + cruise + κάθετο + 20s προσγείωση + 30s απογείωση.
`mah = (60+altd)/3600*HOVER_MA + ct/3600*CRUISE_MA`	mAh: (hover_time_hours × 900mA) + (cruise_time_hours × 1400mA). Διαφορετικές καταναλώσεις.
`need = mah/BATT_MAH*100 + RESERVE`	Απαιτούμενη μπαταρία %: (mah/2700)*100 + 20% αποθεματικό. Εύρος: 20-100+%.
`ok = battery >= need; margin = battery - need`	ok=True αν αρκεί. margin>0=περίσσεια, margin<0=έλλειμμα. Και τα δύο None αν battery=None.
`return {'dist':dist,'time':ft,'need':need,'ok':ok,'margin':margin,'speed':speed}`	Dict 6 κλειδιών. Χρησιμοποιείται από _analyse() για ενημέρωση 5 labels + Go/No-Go.

6.3.5 gen_mavlink(wps, speed=3.0) — Γραμμές 103-114

Παράγει QGC WPL 120 αρχείο. Format ταυτόσημο με FreeFlight Pro. Είσοδος: [(lat,lon,alt,label),...].

Κώδικας / Γραμμή	Επεξήγηση
`"QGC WPL 120"`	Κεφαλίδα. Ο onboard autopilot επαληθεύει αυτή τη

	γραμμή πριν φορτώσει το αρχείο.
`0\t0\t3\t178\t0\t{speed:.1f}\t-1\t0\t0\t0\t0\t1`	CMD 178 (DO_CHANGE_SPEED): P2=speed m/s. FRAME 3=AGL. P3=-1: throttle αμετάβλητο.
`1\t0\t3\t2500\t0\t30\t0\t0\t0\t0\t0\t1`	CMD 2500 (CONDITION_DELAY): P2=30s αναμονή μετά απογείωση. Κρίσιμο για GPS σταθεροποίηση.
`{i+2}\t0\t3\t16\t0\t5\t0\t0\t{la:.7f}\t{lo:.7f}\t{al:.1f}\t1`	CMD 16 (NAV_WAYPOINT): P2=5m ακτίνα αποδοχής. 7 δεκαδικά ≈ 1cm ακρίβεια εμφάνισης.
`CMD 21 στο τελευταίο WP`	NAV_LAND: προσγείωση. Χωρίς αυτό το drone αιωρείται στο τελευταίο WP αόριστα.
`CMD 2501 (end marker)`	Parrot-specific: σήμα τέλους αποστολής. Χωρίς αυτό ο autopilot δεν αναγνωρίζει το τέλος.
`return "\n".join(lines)+"\n"`	Ενώνει γραμμές με newline. Τελικό \n απαιτείται από τον FTP parser του drone.

6.4 Κλάση RosWorker (Γραμμές 117–201)

Κληρονομεί QObject για Qt signals. Τρέχει αποκλειστικά σε ξεχωριστό QThread. Επικοινωνεί με GUI μόνο μέσω signals — ποτέ άμεσα. Αυτό εγγυάται thread safety.

6.4.1 Signals (Γραμμές 121-127)

Κώδικας / Γραμμή	Επεξήγηση
`sig_batt = pyqtSignal(float)`	Μπαταρία: 0.0-100.0%. Από CommonCommonStateBatteryStateChanged.percent.
`sig_state = pyqtSignal(int)`	Κατάσταση πτήσης: 0-8. Από Ardrone3PilotingStateFlyingStateChanged.state.
`sig_img = pyqtSignal(object)`	Raw JPEG bytes. Ο τύπος 'object' δέχεται Python bytes χωρίς type checking.
`sig_gps = pyqtSignal(float, float, float)`	GPS: latitude(-90:+90), longitude(-180:+180), altitude(m AMSL).
`sig_odom = pyqtSignal(float, float, float)`	Οδομετρία: x,y,z σε μέτρα από σημείο εκκίνησης.
`sig_conn = pyqtSignal(bool)`	Σύνδεση: True=ROS OK, False=αποτυχία. Εκπέμπεται μία φορά κατά init().
`sig_log = pyqtSignal(str)`	Μηνύματα log. Παραδίδεται στο _log() του GUI.

6.4.2 Μέθοδος init() (Γραμμές 133-171)

Κώδικας / Γραμμή	Επεξήγηση
`if not ROS_OK: sig_conn.emit(False); return`	Αν rospy δεν φορτώθηκε: GUI demo mode. Εκπέμπει False και τερματίζει.
`rospy.init_node("bebop_missiongui", anonymous=True, disable_signals=True)`	anonymous=True: επιτρέπει πολλαπλές εκτελέσεις. disable_signals=True: ΚΡΙΣΙΜΟ — αποτρέπει ROS να υποκλέψει Ctrl+C από Qt.
`def pub(topic,typ,qs=1): return rospy.Publisher(topic,typ,queue_size=qs)`	Εσωτερική βοηθητική: μειώνει επανάληψη. qs=queue_size.
`self._p['cmd'] = pub('/bebop/cmd_vel', Twist, 10)`	queue_size=10: αρκετά μεγάλη ουρά για 10Hz δημοσίευση χωρίς απώλεια εντολών.
`self._p['tkf'] = pub('/bebop/takeoff', Empty)`	Απογείωση. Ένα Empty αρκεί.
`self._p['lnd'] = pub('/bebop/land', Empty)`	Προσγείωση.

`self._p['emg'] = pub('/bebop/reset', Empty)`	Emergency: κόβει κινητήρες αμέσως — drone πέφτει.
`self._p['cam'] = pub('/bebop/camera_control', Twist)`	Κάμερα: angular.z=pan°, angular.y=tilt°.
`self._p['af_s'] = pub('/bebop/autoflight/start', String)`	Autopilot εκκίνηση: data=αρχείο ή '' για existing.
`self._p['af_p'] = pub('/bebop/autoflight/pause', Empty)`	Παύση autopilot.
`self._p['af_x'] = pub('/bebop/autoflight/stop', Empty)`	Διακοπή autopilot.
`CommonCommonStateBatteryStateChanged, lambda m: sig_batt.emit(float(m.percent))`	ΚΡΙΣΙΜΟ: .percent πεδίο (uint8). Float32 subscriber = σιωπηλή αποτυχία (παλαιό bug).
`buff_size=2**26, tcp_nodelay=True`	Βίντεο: 64MB buffer αποτρέπει απόρριψη μεγάλων JPEG. tcp_nodelay: άμεση παράδοση.
`self.sig_conn.emit(True); rospy.spin()`	Ενημερώνει GUI ότι ROS OK. spin() μπλοκάρει νήμα επ' αόριστον.
`except Exception as e: sig_conn.emit(False)`	Οποιαδήποτε αποτυχία: εκπέμπει False, GUI δείχνει FAILED.

6.4.3 Callbacks και Εντολές (Γραμμές 173-201)

Κώδικας / Γραμμή	Επεξήγηση
`def _cb_img(self, m): self.sig_img.emit(bytes(m.data))`	Μόνο bytes() — χωρίς αποκωδικοποίηση. <1ms. Διατηρεί spin loop ελεύθερο.
`def _cb_odom(self, m): self.sig_odom.emit(x,y,z)`	Εξάγει x,y,z από nested: Odometry→pose→pose→position.x/y/z.
`def _pub(self, key, msg=None)`	Βοηθητική: self._p.get(key) → publish(msg or Empty()). Silent αν δεν υπάρχει publisher.
`def takeoff(self): self._pub('tkf')`	Απογείωση. Δεν ελέγχει κατάσταση — ο _takeoff() GUI το κάνει.
`def land(self): self._pub('lnd')`	Προσγείωση.
`def emergency(self): self._pub('emg')`	Motor cut. Χωρίς επιβεβαίωση εδώ — ο _emergency() GUI το κάνει.
`def cmd_vel(self, lx, ly, lz, az)`	Twist: linear.x=lx, linear.y=ly, linear.z=lz, angular.z=az. Τιμές -1.0 έως +1.0.

`def camera(self, pan, tilt)`	Twist: angular.z=pan°, angular.y=tilt°. Clamp γίνεται στο _cam() πριν κλήση.
`def af_start(self, name='')`	String: data='' → existing plan, data='flightPlan.mavlink' → named file.
`def af_pause(self): self._pub('af_p')`	Παύση: drone αιωρείται στην τρέχουσα θέση.
`def af_stop(self): self._pub('af_x')`	Διακοπή: drone αιωρείται, δέχεται cmd_vel ξανά.

6.5 Κλάση VideoWorker (Γραμμές 204–252)

Αποκωδικοποιεί JPEG καρέ στο δικό του QThread με 15fps cap. Single-slot buffer: πάντα εμφανίζει το πιο πρόσφατο καρέ.

Κώδικας / Γραμμή	Επεξήγηση
`sig_frame = pyqtSignal(QPixmap)`	Εκπέμπει έτοιμο QPixmap στο main thread. setPixmap() καλείται εκεί — thread-safe.
`self._lock = threading.Lock()`	Κλείδωμα νήματος: _pending γράφεται από ROS thread, διαβάζεται από Video thread.
`self._pending = None`	Single slot: αποθηκεύει μόνο το πιο πρόσφατο καρέ. Παλαιά αντικαθίστανται.
`self._decoding = False`	Φρουρός: αποτρέπει παράλληλη αποκωδικοποίηση αν η CPU αργεί.
`t.setInterval(67)`	67ms ≈ 15fps. Δίνει 32-47ms περιθώριο μετά αποκωδικοποίηση (20-35ms).
`self._timer = t`	Αποθηκεύει αναφορά: αποτρέπει garbage collection του timer.
`def push(self, raw): with self._lock: self._pending = raw`	Called από ROS thread. Lock + overwrite = lock-free single-slot. Παλαιά καρέ χάνονται.
`if self._decoding: return`	Skip αν η αποκωδικοποίηση τρέχει — δεν δημιουργείται ουρά καρέ.
`with self._lock: d=self._pending; self._pending=None`	Παίρνει καρέ ατομικά και καθαρίζει slot.
`if d is None or cv2 is None: return`	Guard: τίποτα να αποκωδικοποιηθεί ή OpenCV απών.
`self._decoding = True`	Ενεργοποιεί φρουρό πριν αποκωδικοποίηση.

`arr = np.frombuffer(d, dtype=np.uint8).copy()`	frombuffer(): χωρίς αντιγραφή. .copy(): ΚΡΙΣΙΜΟ — αποτρέπει buffer release mid-decode που προκαλεί εικόνα.
`img = cv2.imdecode(arr, cv2.IMREAD_COLOR)`	Αποκωδικοποιεί JPEG → BGR array. Επιστρέφει None για κατεστραμμένα frames.
`if img is None: return`	Αγνοεί κατεστραμμένα JPEG (συνήθης από WiFi packet loss).
`img = cv2.resize(img, (480,270), interpolation=cv2.INTER_NEAREST)`	INTER_NEAREST: επιλέγει κοντινότερο pixel — 3x ταχύτερο από INTER_LINEAR. Αδιόρατη διαφορά.
`rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)`	BGR→RGB: OpenCV χρησιμοποιεί BGR, Qt χρειάζεται RGB. Χωρίς αυτό: ανεστραμμένα χρώματα.
`rgb = np.ascontiguousarray(rgb)`	Εγγυάται γραμμική μνήμη. Χωρίς αυτό QImage μπορεί να διαβάσει λάθος byte offset.
`qi = QImage(rgb.tobytes(), w, h, ch*w, QImage.Format_RGB888)`	ch*w = bytes per row (stride). Format_RGB888 = 3 bytes/pixel, χωρίς alpha channel.
`self.sig_frame.emit(QPixmap.fromImage(qi))`	Εκπέμπει QPixmap. Main thread: self.lbl_cam.setPixmap(px).
`finally: self._decoding = False`	ΠΑΝΤΑ επαναφέρει φρουρό — ακόμα αν υπήρξε exception στη μέση.

6.6 Κλάση FtpWorker (Γραμμές 255–285)

FTP λειτουργίες σε ξεχωριστό QThread. Κάθε μέθοδος μπλοκάρει μέχρι ολοκλήρωσης. Αποτελέσματα μέσω sig_done(ok, operation, data).

Κώδικας / Γραμμή	Επεξήγηση
`sig_done = pyqtSignal(bool, str, str)`	ok=True/False, op='upload'/'download', data=μήνυμα ή περιεχόμενο ή σφάλμα.
`tmp = tempfile.NamedTemporaryFile(mode='w', suffix='.mavlink', delete=False)`	delete=False: δεν διαγράφεται αυτόματα. Το διαγράφουμε χειροκίνητα στο finally.
`tmp.write(content); tmp.close()`	Γράφει MAVLink string και κλείνει (flush στον δίσκο). Απαραίτητο πριν FTP.
`ftp = ftplib.FTP(ip, timeout=8)`	Timeout 8s: αποτρέπει αναμονή αν το drone δεν είναι διαθέσιμο.
`ftp.login()`	Ανώνυμη σύνδεση: Bebop δεν απαιτεί κωδικό.
`ftp.cwd(FTP_DIR)`	Αλλαγή σε /internal_000/flightplans/
`ftp.storbinary(f'STOR {PLAN_NAME}', f)`	STOR: FTP εντολή αποθήκευσης binary αρχείου.
`sig_done.emit(True,'upload',f'Uploaded {PLAN_NAME}')`	Επιτυχία: _ftp_done() εκκινεί autopilot.
`except Exception as e: sig_done.emit(False,'upload',str(e))`	Αποτυχία: _ftp_done() δείχνει error dialog.
`finally: os.unlink(tmp.name)`	ΠΑΝΤΑ διαγράφει temp — ακόμα αν FTP απέτυχε.
`mavs = [f for f in files if f.lower().endswith('.mavlink')]`	Case-insensitive: βρίσκει 'flightPlan.mavlink' ή 'flightplan.mavlink'.
`ftp.retrlines(f'RETR {mavs[0]}', buf.append)`	RETR: λήψη γραμμή-γραμμή. buf.append ως callback.
`sig_done.emit(True,'download','\n'.join(buf))`	Ενώνει γραμμές και εκπέμπει ολόκληρο το MAVLink string.

6.7 Κλάση MissionLogger (Γραμμές 288–389)

Καταγράφει τηλεμετρία 1 Hz κατά αυτόνομη πτήση. Παράγει αναφορά ακρίβειας waypoint για την πτυχιακή εργασία.

Κώδικας / Γραμμή	Επεξήγηση
`def reset(self): self._records=[]; self._t0=None; self._active=False; self._planned=[]; self._name=''`	Πλήρης επαναφορά: κενά records, μηδέν χρόνος, ανενεργός, κενά planned waypoints.
`def start(self, planned_wps, mission_name='mission')`	Καλεί reset(), αποθηκεύει planned για σύγκριση, t0=time.time(), active=True.
`self._planned=[(w[0],w[1],w[2]) for w in planned_wps]`	Μετατρέπει [[lat,lon,alt,label]] → [(lat,lon,alt)]: αποθηκεύει χωρίς label.
`def stop(self): self._active=False`	Απενεργοποιεί. Νέες κλήσεις record() αγνοούνται. Records παραμένουν.
`def has_data(self): return len(self._records) > 0`	Χρησιμοποιείται πριν generate_report() και πριν ενεργοποίηση Save κουμπιών.
`if not self._active: return`	Guard: αγνοεί κλήσεις έξω από αποστολή.
`'time_s': round(time.time()-self._t0, 1)`	Elapsed σε δευτερόλεπτα, 1 δεκαδικό (ακρίβεια 100ms).
`'lat','lon','alt_gps'`	GPS από /bebop/fix. lat=0,lon=0 = χωρίς GPS lock.
`'odom_x': round(odom_x,3)`	Οδομετρία σε μέτρα, 3 δεκαδικά = 1mm precision.
`'battery': battery`	Ποσοστό % ή None αν δεν έχει ληφθεί μήνυμα.
`'state': state_int`	Κατάσταση 0-8 για ανάλυση χρόνου ανά κατάσταση.
`def save_csv(self, path)`	csv.DictWriter: γράφει κεφαλίδα + 9 στήλες + όλα τα records. Excel/pandas/MATLAB.
`gps=[(r['lat'],r['lon']) for r in _records if r['lat']!=0 and r['lon']!=0]`	Φιλτράρει GPS lock=False εγγραφές (lat=0,lon=0).
`dists=[haversine(pla,plo,g[0],g[1]) for g in gps]`	Για κάθε planned WP: απόσταση από ΟΛΕΣ τις καταγεγραμμένες GPS θέσεις.
`mn=min(dists)`	Ελάχιστη = πλησιέστερο σημείο πέρασε το drone κοντά στο WP.
`if mean_err<=6: PASS / elif<=15: acceptable / else: GPS error`	≤6m = εντός hardware spec. 6-15m = αποδεκτό. >15m = GPS lock πρόβλημα.
`from collections import Counter`	Τοπική εισαγωγή. Counter: μέτρηση δευτερολέπτων σε κάθε κατάσταση.

Κλάση App — Κατασκευαστής (Γραμμές 390–424)

Κώδικας / Γραμμή	Επεξήγηση
`class App(QMainWindow)`	Κληρονομεί QMainWindow: title bar, status bar, central widget area.
`self.setWindowTitle("Bebop 2 — Mission Planner")`	Τίτλος παραθύρου στη γραμμή τίτλου και taskbar.
`self.setMinimumSize(1100, 700)`	Ελάχιστο μέγεθος: κάτω από 1100x700 τα layouts αλληλοεπικαλύπτονται.
`self._wps = []`	Κεντρική αποθήκη waypoints: [[lat,lon,alt,label],...]. Μία λίστα, πολλαπλές views.
`self._battery = None`	None μέχρι πρώτο ROS μήνυμα. Ελέγχεται: if self._battery is not None.
`self._state = 0`	Τρέχουσα κατάσταση. 0=LANDED αρχικά (ασφαλής default).
`self._conn = False`	False μέχρι ROS init(). Ελέγχεται πριν κάθε εντολή.
`self._flying = False`	True μόνο κατά autopilot. Ελέγχεται από _on_state() για landing detection.
`self._paused = False`	True αν αποστολή σε παύση. btn_pau.setText() αντανακλά την τιμή.
`self._plan = None`	Τελευταίο MAVLink string. None αν δεν έχει φορτωθεί ή παραχθεί.
`self._t0 = None`	time.time() έναρξης αποστολής. None = δεν τρέχει αποστολή.
`self._lx=self._ly=self._lz=self._az=0.0`	Τρέχουσες ταχύτητες body frame. 0=hover. Ενημερώνονται από pad/keyboard.
`self._spd = 0.3`	Multiplier ταχύτητας: 0.05-0.80. Default 0.30. Αλλάζει από slider.
`self._pan = self._tilt = 0.0`	Γωνίες κάμερας: pan ±180°, tilt ±83°. Αλλάζουν από _cam().
`self._lat=self._lon=self._alt_gps=0.0`	Τελευταία GPS θέση. 0 = χωρίς lock.
`self._ox=self._oy=self._oz=0.0`	Τελευταία οδομετρία σε μέτρα.
`self._logger = MissionLogger()`	Ένας logger: επαναχρησιμοποιείται μέσω reset().

`self._vel_t.setInterval(100)`	cmd_vel κάθε 100ms (10Hz). Αρκετό για ομαλό χειρισμό.
`self._tick_t.setInterval(1000)`	Logger + χρονόμετρο κάθε 1000ms (1Hz).

6.8 Μέθοδοι Κατασκευής UI (Γραμμές 425–917)

6.8.1 _build_ui()

Κώδικας / Γραμμή	Επεξήγηση
`cw = QWidget(); self.setCentralWidget(cw)`	QMainWindow δεν δέχεται layout άμεσα — απαιτεί κεντρικό QWidget.
`root = QHBoxLayout(cw)`	Οριζόντια διάταξη δύο στηλών.
`root.setContentsMargins(8,8,8,8)`	8px padding στις 4 πλευρές.
`root.setSpacing(8)`	8px μεταξύ αριστερής και δεξιάς στήλης.
`left = QVBoxLayout(); left.setSpacing(6)`	Κάθετη αριστερή στήλη. 6px μεταξύ panels.
`left.addStretch()`	Elastic spacer: σπρώχνει panels πάνω, κενό κάτω.
`root.addLayout(left, 5)`	stretch=5: αριστερή = 5/11 πλάτους.
`root.addLayout(right, 6)`	stretch=6: δεξιά = 6/11 — ελαφρώς πλατύτερη.
`spl = QSplitter(Qt.Horizontal)`	Συρόμενος διαχωριστής waypoints/analysis.
`spl.setSizes([520, 260])`	Αρχικά: 520px waypoints, 260px analysis.
`log_tabs, rep_row = self._build_log_panel()`	Επιστρέφει δύο αντικείμενα — QTabWidget και QHBoxLayout.
`self._wire_signals()`	Καλείται ΤΕΛΕΥΤΑΙΟ — όλα τα widgets πρέπει να υπάρχουν πρώτα.

6.8.2 _build_camera_card()

Κώδικας / Γραμμή	Επεξήγηση
`self.lbl_cam.setFixedSize(480, 270)`	16:9 ratio = Bebop 2 native. FIXED: δεν αλλάζει με resize παραθύρου.
`self.lbl_cam.setAlignment(Qt.AlignCenter)`	Κεντράρει 'No video feed' κείμενο.
`self.lbl_cam.setObjectName('cam')`	CSS: QLabel#cam → σκούρο background (#1a1a2e) για βίντεο.

6.8.3 _build_status_card()

Κώδικας / Γραμμή	Επεξήγηση

`self.lbl_state = QLabel('DISCONNECTED')`	Κατάσταση drone. Ενημερώνεται από _on_state() με κείμενο+χρώμα.
`self.batt_bar = QProgressBar(); .setRange(0,100)`	Γραμμή μπαταρίας. setValue(n) θέτει επίπεδο. Χρώμα αλλάζει από _on_batt().
`self.lbl_batt = QLabel('--%')`	Αριθμητικό ποσοστό δίπλα στη γραμμή.
`self.lbl_time = QLabel('Time: --:--')`	Χρονόμετρο πτήσης MM:SS. Ενημερώνεται κάθε δευτερόλεπτο από _tick().

6.8.4 _build_flight_card()

Κώδικας / Γραμμή	Επεξήγηση
`self.btn_tkf: objectName='btn_tkf'`	TAKE OFF: λευκό κείμενο σε πράσινο. Έχει confirmation dialog στο _takeoff().
`self.btn_lnd: objectName='btn_lnd'`	LAND: λευκό σε πετρόλ. Άμεσο — χωρίς confirmation.
`self.btn_hov: objectName='btn_hov'`	HOVER: λευκό σε μοβ. Μηδενίζει ταχύτητες.
`self.btn_emg: objectName='btn_emg'`	EMERGENCY: λευκό σε κόκκινο+bold. Critical dialog στο _emergency().
`for b in [...]: b.setMinimumHeight(36)`	36px: αρκετά μεγάλα για πάτημα κατά πτήση.

6.8.5 _build_direction_pad()

Κώδικας / Γραμμή	Επεξήγηση
`g = QGridLayout(); g.setSpacing(3)`	Πλέγμα 3×3. 3px spacing.
`g.addWidget(self.btn_fwd, 0, 1)`	(row=0,col=1) = κεντρικό επάνω.
`g.addWidget(self.btn_stop, 1, 1)`	(row=1,col=1) = κέντρο = STOP.
`self.spd_sl.setRange(5, 80)`	Slider: 5-80, /100 = 0.05-0.80 multiplier.
`lambda v: setattr(self, '_spd', v/100)`	valueChanged: ενημερώνει _spd. setattr() από λάμδα.

6.8.6 _build_camera_controls()

Κώδικας / Γραμμή	Επεξήγηση
`self.btn_pl: 'Pan ←'`	Καλεί _cam(-5,0). Pan εύρος: -180° έως +180°.

`self.btn_tu: 'Tilt ↑'`	Καλεί _cam(0,5). Tilt εύρος: -83° έως +83° (φυσικό όριο gimbal Bebop 2).
`self.btn_cr: 'Reset'`	Καλεί _cam_abs(0,0): κάμερα ευθεία εμπρός.

6.8.7 _build_connection_row()

Κώδικας / Γραμμή	Επεξήγηση
`self.txt_ip = QLineEdit(BEBOP_IP); .setFixedWidth(130)`	Εκκινεί με '192.168.42.1'. Επεξεργάσιμο για custom IP.
`self.lbl_ros`	Ενημερώνεται από _on_conn(): πράσινο CONNECTED ή κόκκινο FAILED.
`row.addStretch()`	Σπρώχνει widgets αριστερά.

6.8.8 _build_waypoint_panel()

Κώδικας / Γραμμή	Επεξήγηση
`self.txt_coord.setPlaceholderText('Paste from Google Maps: ...')`	Ghost κείμενο όταν το πεδίο είναι κενό.
`self.txt_coord.returnPressed.connect(self._add_wp)`	Enter key = προσθήκη waypoint. Ίδιο με κλικ Add.
`self.spin_alt.setRange(1, 120)`	Ύψος AGL: 1m ελάχιστο (ασφάλεια), 120m μέγιστο (νομικό όριο).
`self.wt = QTableWidget(0, 4)`	0 γραμμές αρχικά, 4 στήλες. Γραμμές δυναμικά.
`QHeaderView.Stretch`	Όλες στήλες μοιράζονται ισόποσα το πλάτος.
`QAbstractItemView.SelectRows`	Κλικ σε κελί = επιλογή ολόκληρης γραμμής.
`self.wt.setMaximumHeight(180)`	Αποτρέπει τον πίνακα να πάρει όλο το ύψος.
`self.wt.itemChanged.connect(self._tbl_edit)`	Ζωντανή επεξεργασία κελιών → ενημέρωση _wps.
`self.spin_spd.setRange(0.5, 10)`	Ταχύτητα cruise: 0.5-10 m/s. Default 3.0.
`self.btn_load_drone.setObjectName('btn_ff')`	CSS: μοβ — διακρίνεται ως FTP λειτουργία.

6.8.9 _build_analysis_panel()

Κώδικας / Γραμμή	Επεξήγηση
`def stat_row(label)`	Τοπική συνάρτηση: δημιουργεί ζεύγος (key,value) labels. Επιστρέφει value label.
`self.v_dist, self.v_time, self.v_need, self.v_have, self.v_marg`	5 value labels, αρχικά '--'. Ενημερώνονται από _analyse().
`self.lbl_gng.setMinimumHeight(38)`	38px: αρκετό για κείμενο SAFE/NOT FLY.
`self.lbl_gps_pos`	Ζωντανές συντεταγμένες. Ενημερώνεται από _on_gps() κάθε NavSatFix.
`self.lbl_log_status`	Κατάσταση logger: idle / RECORDING N / DONE.

6.8.10 _build_mission_control()

Κώδικας / Γραμμή	Επεξήγηση
`self.lbl_ms = QLabel('IDLE')`	Status αποστολής. _ms() αλλάζει κείμενο+χρώμα.
`self.btn_uf: 'UPLOAD & FLY' (btn_go)`	Πράσινο. gen_mavlink → FTP upload → af_start. Τρία βήματα σε ένα.
`self.btn_fe: 'FLY EXISTING PLAN' (btn_fe)`	Μοβ. Παρακάμπτει FTP: af_start('').
`self.btn_pau: 'PAUSE'`	Toggle: κείμενο αλλάζει PAUSE↔RESUME.
`self.btn_stp: 'STOP' (btn_st)`	Κόκκινο. af_stop + reset flags + hover.
`self.btn_reset = QPushButton('↺ Reset After Landing')`	ΝΕΟΣ: κεχριμπάρι. Force-clear όλα τα mission flags.
`self.btn_reset.setToolTip("Force-clears all mission state...")`	Tooltip: εμφανίζεται με hover για οδηγίες χρήσης.
`self.btn_close: '✕ Close Application'`	Γκρι. Γίνεται κόκκινο on hover. Καλεί _confirm_close().
`bottom_row.addStretch()`	Σπρώχνει Reset αριστερά και Close δεξιά.

6.8.11 _build_log_panel()

Κώδικας / Γραμμή	Επεξήγηση
`log_tabs = QTabWidget(); .setMaximumHeight(170)`	Δύο καρτέλες: LOG και FLIGHT REPORT. Max 170px ύψος.

`self.log = QTextEdit(); .setReadOnly(True)`	LOG: read-only. append() για προσθήκη και auto-scroll.
`self.log_report = QTextEdit(); .setReadOnly(True)`	FLIGHT REPORT: γεμίζει από generate_report() μετά πτήση.
`self.btn_save_csv.setEnabled(False)`	Αρχικά ανενεργά. Ενεργοποιούνται μόνο αν υπάρχουν δεδομένα.
`return log_tabs, rep_row`	Επιστρέφει ΔΥΟ αντικείμενα. Ο καλών τα προσθέτει ξεχωριστά.

6.8.12 _wire_signals()

Κώδικας / Γραμμή	Επεξήγηση
`self.btn_tkf.clicked.connect(self._takeoff)`	clicked: εκπέμπεται κατά release — αποτρέπει τυχαίο πάτημα.
`btn.pressed.connect(lambda lx=lx,...: self._setv(...))`	pressed: κατά πάτημα → ορίζει ταχύτητα.
`btn.released.connect(self._hover)`	released: κατά αποδέσμευση → μηδενίζει ταχύτητα.
`lambda lx=lx, ly=ly, lz=lz, az=az`	Default args: ΚΡΙΣΙΜΟ. Χωρίς αυτό όλα τα λ θα έπαιρναν τελευταία τιμή loop.
`self.btn_stop.clicked.connect(self._hover)`	Κεντρικό STOP pad κουμπί = hover.
`self.btn_reset.clicked.connect(self._reset_state)`	ΝΕΟΣ: reset button → _reset_state().
`self.btn_close.clicked.connect(self._confirm_close)`	Close button → _confirm_close().

6.8.13 _card() και _db()

Κώδικας / Γραμμή	Επεξήγηση
`b = QGroupBox(t); b.setObjectName('card')`	QGroupBox: container με τίτλο και border. CSS: QGroupBox#card.
`b.layout().setContentsMargins(6,14,6,6)`	Top 14px: χώρος για τίτλο GroupBox.
`b = QPushButton(t); b.setObjectName('dp'); b.setMinimumSize(58,44)`	Direction pad: 58×44px. Εύκολο πάτημα.

6.9 Workers και ROS Slots (Γραμμές 919–1007)

6.9.1 _start_workers()

Κώδικας / Γραμμή	Επεξήγηση
`self._vid = VideoWorker(); self._vt = QThread()`	Δημιουργεί worker ΚΑΙ thread χωριστά.
`self._vid.moveToThread(self._vt)`	Μεταφέρει VideoWorker. Όλες methods τρέχουν στο _vt.
`sig_frame.connect(lambda px: self.lbl_cam.setPixmap(px))`	QPixmap → main thread → setPixmap(). Thread-safe.
`self._vt.started.connect(self._vid.start_timer)`	Timer δημιουργείται μέσα στο νήμα — σωστή thread affinity.
`self._ros.sig_img.connect(self._vid.push)`	Bytes: ROS thread → Video thread χωρίς main thread.
`self._rt.started.connect(self._ros.init)`	init() καλείται αυτόματα όταν ξεκινά το νήμα.

6.9.2 _on_conn()

Κώδικας / Γραμμή	Επεξήγηση
`self._conn = ok`	Αποθηκεύει state. Ελέγχεται πριν κάθε εντολή ROS.
`'ROS: CONNECTED' (πράσινο) / 'ROS: FAILED' (κόκκινο)`	Inline CSS στο lbl_ros.

6.9.3 _on_batt()

Κώδικας / Γραμμή	Επεξήγηση
`self._battery = v; vi = int(v)`	float για ανάλυση, int για QProgressBar.
`c = '#4ade80' if vi>30 else ('#f59e0b' if vi>15 else '#ef4444')`	Τριαδική: πράσινο>30%, κεχριμπάρι 15-30%, κόκκινο<15%.
`self.batt_bar.setStyleSheet(f'QProgressBar::chunk{{background:{c};...}}')`	Δυναμικό CSS: αλλάζει χρώμα chunk.

`self.v_have.setText(f'{vi}%')`	Ενημερώνει 'Have batt:' label ταυτόχρονα.

6.9.4 _on_state() — Ανίχνευση Ολοκλήρωσης Αποστολής

Κώδικας / Γραμμή	Επεξήγηση
`prev = self._state; self._state = s`	Αποθηκεύει ΠΡΟΗΓΟΥΜΕΝΗ κατάσταση — κρίσιμο για τον έλεγχο μετάβασης.
`name, col = STATES.get(s, ('UNKNOWN','#64748b'))`	Ασφαλής ανάκτηση: default αν s άγνωστο.
`if s==0 and prev!=0 and self._flying:`	Τρεις απαραίτητες συνθήκες: LANDED + μετάβαση από non-LANDED + αποστολή τρέχει.
`self._flying=False; self._paused=False; self._t0=None`	Επαναφορά mission flags.
`self._hover()`	Μηδενίζει ταχύτητες αμέσως.
`self._ros.cmd_vel(0.0,0.0,0.0,0.0)`	ΚΡΙΣΙΜΟ: explicit zero cmd_vel ξυπνά drone από post-autopilot frozen state.
`self._logger.stop(); report = self._logger.generate_report()`	Σταματά καταγραφή και παράγει αναφορά.
`self.log_report.setPlainText(report)`	setPlainText(): αντικαθιστά ολόκληρο περιεχόμενο.
`self.btn_save_csv.setEnabled(True)`	Ενεργοποιεί Save κουμπιά μόνο αφού υπάρχουν δεδομένα.

6.9.5 _on_gps() και _on_odom()

Κώδικας / Γραμμή	Επεξήγηση
`self._lat=lat; self._lon=lon; self._alt_gps=alt`	Αποθηκεύει για logger record().
`self.lbl_gps_pos.setText(f'lat: {lat:.6f} lon: {lon:.6f} alt: {alt:.1f}m')`	6 δεκαδικά ≈ 11cm GPS ακρίβεια εμφάνισης.
`def _on_odom(self, x, y, z): self._ox=x; self._oy=y; self._oz=z`	Αποθηκεύει για logger. Δεν εμφανίζεται στο GUI — μόνο στο CSV.

6.10 Χειροκίνητος Έλεγχος (Γραμμές 1008–1065)

Κώδικας / Γραμμή	Επεξήγηση
`def _takeoff(self)`	Ελέγχει _conn. QMessageBox.question() επιβεβαίωση. Μετά: self._ros.takeoff().
`def _land(self)`	if self._conn: self._ros.land(). Χωρίς confirmation — επείγον.
`def _hover(self): self._lx=self._ly=self._lz=self._az=0.0`	Μηδενίζει ΟΛΑ τα velocity components. Drone hovering.
`def _emergency(self)`	QMessageBox.critical() κόκκινο dialog. Μετά: self._ros.emergency() (motor cut).
`def _setv(self, lx,ly,lz,az): s=self._spd; self._lx=lx*s...`	Ορίζει ταχύτητες × _spd. lx,ly,lz,az ∈ {-1,0,+1}.
`def _pub_vel(self): if self._conn: self._ros.cmd_vel(...)`	Δημοσιεύει κάθε 100ms. Autopilot αγνοεί cmd_vel κατά αποστολή — χωρίς παρεμβολή.
`def _cam(self, dp, dt): self._pan=max(-180,min(180,self._pan+dp))...`	Clamp pan [-180,+180] και tilt [-83,+83]. Φυσικά όρια gimbal.
`def _cam_abs(self, p, t): self._pan=p; self._tilt=t; self._ros.camera(p,t)`	Θέτει ακριβείς γωνίες. Reset: (0,0) = εμπρός οριζόντια.
`W/S: lx=±1 (εμπρός/πίσω)`	linear.x body frame.
`A/D: ly=±1 (αριστερά/δεξιά)`	linear.y body frame.
`R/F: lz=±1 (πάνω/κάτω)`	linear.z body frame.
`Q/E: az=±1 (yaw αριστερά/δεξιά)`	angular.z body frame.
`Space: _hover()`	Άμεσο μηδένισμα όλων ταχυτήτων.
`keyReleaseEvent: self._hover()`	Κατά αποδέσμευση πλήκτρου: αμέσως hover.

6.11 Waypoints, Ανάλυση, FTP (Γραμμές 1066–1234)

6.11.1 Waypoint Management

Κώδικας / Γραμμή	Επεξήγηση
`def _add_wp(self)`	parse_coords() + spin_alt.value() → _wps.append([lat,lon,alt,'WP01']) → _refresh().
`n = len(self._wps)+1; f'WP{n:02d}'`	WP01, WP02 κ.λπ. :02d = 2 ψηφία με leading zero.
`except ValueError as e: QMessageBox.warning(self,'Error',str(e))`	Εμφανίζει μήνυμα αν δεν βρεθούν συντεταγμένες.
`def _del_wp(self): rows=sorted({i.row() for i...}, reverse=True)`	Set: μοναδικά rows. Reverse: διαγραφή από κάτω προς πάνω — αποτρέπει index shift.
`self._wps.pop(r)`	pop(index): αφαιρεί και επιστρέφει στοιχείο.
`def _clr_wp(self)`	QMessageBox.question() → _wps.clear() → _refresh().
`def _refresh(self): self.wt.blockSignals(True)`	Αποτρέπει itemChanged loop κατά setItem(). blockSignals False μετά.
`self.wt.setRowCount(0)`	Διαγράφει ΟΛΕΣ γραμμές σε μία κλήση.
`QTableWidgetItem(f'{wp[0]:.7f}')`	7 δεκαδικά ≈ 1cm GPS ακρίβεια.
`def _tbl_edit(self, item): c=item.column()`	c=0: label, c=1: lat, c=2: lon, c=3: alt. float() → ValueError αγνοείται.

6.11.2 _analyse() — Ανάλυση Βιωσιμότητας

Κώδικας / Γραμμή	Επεξήγηση
`try: ... except Exception as ex: self._log(f'Analyse error: {ex}')`	try/except: αν κάτι αποτύχει, λογαρίζει αντί να κρασάρει σιωπηλά.
`wps = [(w[0],w[1],w[2]) for w in self._wps]`	List comprehension: [[...]] → [(lat,lon,alt)].
`if not wps and self._plan: parse_mavlink(self._plan)`	Fallback: αν _wps κενό, χρησιμοποιεί φορτωμένο plan.
`a = analyse(wps, speed=self.spin_spd.value(), battery=self._battery)`	Καλεί analyse(). battery=None αν δεν έχει ληφθεί ROS μήνυμα.
`t = a['time']; m,s = int(t//60), int(t%60)`	Δευτερόλεπτα → λεπτά,δευτερόλεπτα. floor division.

`time_col = '#dc2626' if t/60 > MAX_MIN else '#1a1a2e'`	Κόκκινο αν υπερβαίνει 25 λεπτά.
`if t/60>MAX_MIN: issues.append(...)`	Έλεγχος χρόνου: >25 min.
`if a['dist']>300: issues.append(...)`	Έλεγχος εμβέλειας: >300m WiFi range.
`if self._battery is not None and not a['ok']: issues.append(...)`	Έλεγχος μπαταρίας (μόνο αν έχει τιμή).
`txt = 'X  DO NOT FLY\n' + '\n'.join(f'• {x}' for x in issues)`	Μορφοποίηση No-Go: X + bullet list λόγων.
`'background:#fef2f2;color:#dc2626;...'` / `'background:#f0fdf4;color:#16a34a;...'`	No-Go: ανοιχτό κόκκινο. Go: ανοιχτό πράσινο — ορατό στο light theme.

6.11.3 FTP Λειτουργίες

Κώδικας / Γραμμή	Επεξήγηση
`def _load_drone(self): self.btn_load_drone.setEnabled(False)`	Απενεργοποιεί κατά λήψη — αποτρέπει double click.
`self._ftp.download(self.txt_ip.text().strip())`	strip() αφαιρεί κενά χαρακτήρες από IP.
`def _load_file(self): QFileDialog.getOpenFileName(...)`	Φίλτρο: 'MAVLink (*.mavlink);;All (*.*)'
`def _save_file(self)`	Αν _wps: gen_mavlink(). Αλλιώς: αποθηκεύει _plan.
`def _ftp_done(self, ok, op, data): if op=='download': ... elif op=='upload':...`	Ένας handler για δύο λειτουργίες. op string διακρίνει.
`self._ros.af_start(PLAN_NAME)`	Triggers autopilot αμέσως μετά επιτυχές upload.
`self._flying=True; self._t0=time.time()`	Mission flags + χρόνος έναρξης.
`ts=time.strftime('%Y%m%d_%H%M%S')`	'20250527_143022': μοναδικό timestamp για όνομα αποστολής.
`self._logger.start(self._wps, mission_name=f'mission_{ts}')`	Εκκινεί logger με planned waypoints για σύγκριση.
`def _load_plan(self, content, src)`	Αποθηκεύει _plan, parse, φιλτράρει CMD 16, κτίζει _wps, _refresh(), _analyse().
`navs=[r for r in rows if r['cmd']==16 and not(r['lat']==0 and r['lon']==0)]`	Φιλτράρει: μόνο navigation WPs με έγκυρες συντεταγμένες.

6.12 Έλεγχος Αποστολής (Γραμμές 1235–1350)

Κώδικας / Γραμμή	Επεξήγηση
`def _upload_fly(self)`	Ελέγχει _wps και _conn. gen_mavlink() → _ms('UPLOADING') → _ftp.upload().
`self._plan = content; self.btn_uf.setEnabled(False)`	Αποθηκεύει plan, απενεργοποιεί κουμπί κατά upload.
`def _fly_existing(self)`	Ελέγχει _conn. Confirmation dialog. af_start('') → _flying=True → logger.start().
`self._ros.af_start('')`	'' = χρησιμοποιεί αρχείο ήδη στο drone. Χωρίς FTP.
`def _pause_resume(self): if not self._flying: return`	Guard: αδρανεί αν δεν τρέχει αποστολή.
`af_pause() / af_start(PLAN_NAME)`	Toggle: παύση→αιώρηση, συνέχιση→af_start με filename.
`def _stop(self): self._ros.af_stop()`	af_stop() + _flying=False + _paused=False + _t0=None + _hover().
`self._ms('STOPPED','#64748b'); self.btn_uf.setEnabled(True); btn_pau.setText('PAUSE')`	Επαναφορά UI.
`def _save_csv(self): QFileDialog → self._logger.save_csv(path)`	Αποθηκεύει CSV για ανάλυση.
`def _save_report(self): f.write(self.log_report.toPlainText())`	Αποθηκεύει αναφορά ως .txt.

6.13 _reset_state() — Επαναφορά Μετά Προσγείωση (Γραμμές 1293–1341)

Καλείται από το ↺ Reset After Landing κουμπί όταν το drone δεν αποκρίνεται μετά ολοκλήρωση αποστολής.

Τεχνικό υπόβαθρο: Μετά ολοκλήρωση MAVLink αποστολής, το firmware του Bebop 2 εισέρχεται σε brief 'post-autopilot cooldown' state. Κατά αυτό το διάστημα αγνοεί cmd_vel εντολές εκτός αν λάβει explicit zero velocity packet. Ταυτόχρονα, εσωτερικά flags του GUI μπορεί να έχουν κολλήσει, αποτρέποντας την ομαλή λειτουργία.

Κώδικας / Γραμμή	Επεξήγηση
`self._flying = False`	Καθαρίζει το κύριο flag που εμποδίζει manual έλεγχο.
`self._paused = False`	Καθαρίζει flag παύσης.
`self._t0 = None`	Καθαρίζει mission start time.
`self._hover()`	Μηδενίζει _lx,_ly,_lz,_az = 0.0. Αποτρέπει stale velocity.
`if self._conn: self._ros.af_stop()`	Explicit autopilot stop: αν τρέχει ακόμα σε background.
`if self._conn: self._ros.cmd_vel(0.0,0.0,0.0,0.0)`	ΚΡΙΣΙΜΟ: zero cmd_vel ξυπνά drone από frozen state αμέσως.
`self._ms('READY','#16a34a')`	Πράσινο READY: σηματοδοτεί έτοιμο για νέα αποστολή.
`self.btn_uf.setEnabled(True)`	Ενεργοποιεί UPLOAD & FLY ξανά.
`self.btn_pau.setText('PAUSE')`	Επαναφέρει κείμενο κουμπιού.
`self.lbl_time.setText('Time: --:--')`	Καθαρίζει χρονόμετρο.
`self._logger.reset()`	Καθαρίζει records για νέα αποστολή.
`has_report = bool(self.log_report.toPlainText().strip())`	Ελέγχει αν υπάρχει ήδη αναφορά στο tab.
`self.btn_save_csv.setEnabled(has_report)`	Κρατά ενεργά τα Save κουμπιά μόνο αν υπάρχει αναφορά.
`self._log('--- State reset --- GUI ready for next mission ---')`	Εκτυπώνει στο LOG tab και status bar.

6.14 _confirm_close() και _ms() (Γραμμές 1343–1353)

Κώδικας / Γραμμή	Επεξήγηση
`def _confirm_close(self)`	Δείχνει confirmation dialog πριν κλείσιμο.
`msg = 'A mission is running...' if self._flying else 'Close the application?'`	Διαφορετικό μήνυμα αν αποστολή τρέχει.
`QMessageBox.Yes|QMessageBox.No, QMessageBox.No`	Default=No: ασφαλέστερο — αποτρέπει τυχαίο κλείσιμο.
`if reply==QMessageBox.Yes: self.close()`	Καλεί close() → closeEvent() για καθαρό τερματισμό.
`def _ms(self, t, c): self.lbl_ms.setText(t); setStyleSheet(...)`	Helper: ενημερώνει κείμενο + χρώμα mission status σε μία κλήση.
`IDLE: -- UPLOADING: κεχριμπάρι RUNNING: πράσινο PAUSED: κεχριμπάρι COMPLETE: πετρόλ STOPPED: γκρι FAILED: κόκκινο READY: πράσινο`	Οκτώ δυνατές καταστάσεις.

6.15 _tick(), _log(), closeEvent(), main() (Γραμμές 1355–1603)

6.15.1 _tick() — 1 Hz Loop

Κώδικας / Γραμμή	Επεξήγηση
`if self._t0 and self._flying:`	Ενεργό μόνο κατά αποστολή. _t0=None → skip.
`e = int(time.time()-self._t0)`	Elapsed seconds ως integer.
`self.lbl_time.setText(f'Time: {e//60:02d}:{e%60:02d}')`	MM:SS. e//60=λεπτά, e%60=δευτερόλεπτα. :02d = leading zero.
`if self._battery and self._battery < 15: self._log('WARNING...')`	Προειδοποίηση κάθε δευτερόλεπτο αν κρίσιμη μπαταρία.
`self._logger.record(lat=self._lat, lon=self._lon, ...)`	Καταγράφει τρέχουσα τηλεμετρία: GPS, οδομετρία, μπαταρία, κατάσταση.
`if n > 0 and n % 10 == 0: lbl_log_status.setText(f'RECORDING {n} samples')`	Ενημερώνει status κάθε 10 δευτερόλεπτα.

6.15.2 _log()

Κώδικας / Γραμμή	Επεξήγηση
`self.log.append(m)`	QTextEdit.append(): προσθέτει στο τέλος + auto-scroll.
`self.statusBar().showMessage(m)`	Εμφανίζεται στο κάτω μέρος παραθύρου — ορατό χωρίς LOG tab ανοιχτό.

6.15.3 _css() — Θέμα Ανοιχτού Γκρι

Κώδικας / Γραμμή	Επεξήγηση
`#f0f2f5 — window background`	Πολύ ανοιχτό γκρι. Ουδέτερο, επαγγελματικό.
`#ffffff — card background`	Λευκό για panels — αντίθεση με φόντο.
`#c8cdd4 — borders`	Μεσαίο γκρι.
`#1a1a2e — primary text`	Σχεδόν μαύρο.
`#1976d2 — accent/headers`	Μπλε για section headers.
`#16a34a — safe/takeoff`	Πράσινο: TAKE OFF, UPLOAD & FLY, SAFE TO FLY.
`#dc2626 — danger`	Κόκκινο: EMERGENCY, DO NOT FLY, STOP.
`#7c3aed — secondary`	Μοβ: HOVER, Load from Drone.

`#d97706 — warning/reset`	Κεχριμπάρι: UPLOADING, PAUSED, RESET button.
`#0891b2 — land/teal`	Πετρόλ: LAND, COMPLETE.
`QGroupBox#card`	Λευκό background, μπλε τίτλος, γκρι border.
`QPushButton:hover`	Μπλε background + border.
`QPushButton#btn_reset`	Λευκό κείμενο σε κεχριμπάρι background — ξεχωρίζει.
`QPushButton#btn_close:hover`	Κόκκινο on hover — signal επικινδυνότητας.
`QProgressBar::chunk`	Χρώμα αλλάζει dynamically από _on_batt().

6.15.4 closeEvent()

Κώδικας / Γραμμή	Επεξήγηση
`e.accept()`	ΠΡΩΤΗ γραμμή: παράθυρο εξαφανίζεται αμέσως. Cleanup τρέχει μετά.
`self._vel_t.stop(); self._tick_t.stop()`	Σταματούν timers — αποτρέπουν εντολές κατά τερματισμό.
`if self._flying: self._ros.af_stop()`	Ασφαλής τερματισμός: drone δεν μένει σε autopilot.
`rospy.signal_shutdown('GUI closed')`	Σπάει το rospy.spin() στο RosWorker νήμα.
`self._rt.quit(); self._ft.quit(); self._vt.quit()`	Σηματοδοτεί τερματισμό. Δεν μπλοκάρει.
`self._rt.wait(300); if self._rt.isRunning(): self._rt.terminate()`	300ms (όχι 1000ms): 3x ταχύτερο. Force-kill αν δεν σταματήσει.

6.15.5 main()

Κώδικας / Γραμμή	Επεξήγηση
`app = QApplication(sys.argv)`	Ακριβώς μία ανά process.
`app.setStyle('Fusion')`	Consistent look Linux/Windows/Mac.
`win = App()`	Δημιουργεί παράθυρο: UI build + workers + timers.
`win.show()`	Widgets κρυμμένα by default — show() τα εμφανίζει.
`sys.exit(app.exec_())`	exec_(): Qt event loop. Μπλοκάρει μέχρι close. sys.exit: exit code στο OS.

Κεφάλαιο 7ο: Συμπεράσματα και Μελλοντικές Επεκτάσεις

7.1 Συμπεράσματα

Η παρούσα εργασία απέδειξε ότι ο έλεγχος και η αυτόνομη GPS πλοήγηση του Parrot Bebop 2 μέσω ROS είναι εφικτή και αξιόπιστη, υπό την προϋπόθεση χρήσης της σωστής αρχιτεκτονικής.

Το κύριο τεχνικό εύρημα είναι η αναγνώριση της θεμελιώδους διαφοράς μεταξύ δύο προσεγγίσεων αυτόνομης πλοήγησης: ο κλειστός βρόχος GPS ελέγχου μέσω cmd_vel από laptop αποδείχθηκε αναξιόπιστος λόγω WiFi latency, ασυνέπειας πλαισίων αναφοράς και GPS αστάθειας. Ο ενσωματωμένος autopilot μέσω MAVLink αποδείχθηκε αξιόπιστος εκτελώντας τον βρόχο ελέγχου στα ~200Hz απευθείας στον επεξεργαστή του drone.

Δευτερεύοντα αλλά αξιόλογα ευρήματα: (1) το πρόβλημα multipath GPS σε εσωτερικό χώρο που έδινε σφάλμα 34 χιλιομέτρων, (2) η ασυμβατότητα cv_bridge με Python 3 και η λύση μέσω compressed images, (3) η ανακάλυψη ότι το FreeFlight Pro χρησιμοποιεί υβριδική αρχιτεκτονική με παράλληλο yaw control που δεν τεκμηριώνεται επίσημα.

Το σύστημα bebop_mission_gui.py παρέχει ολοκληρωμένο περιβάλλον: χειροκίνητος έλεγχος με πληκτρολόγιο και κουμπιά, ζωντανό βίντεο, σχεδιασμός αποστολής GPS με waypoints, ανάλυση Go/No-Go, αυτόνομη εκτέλεση μέσω MAVLink, καταγραφή τηλεμετρίας 1Hz, και αξιολόγηση ακρίβειας waypoint μετά πτήση.

7.2 Περιορισμοί Συστήματος

Η παρούσα υλοποίηση επιτυγχάνει τους στόχους που τέθηκαν, ωστόσο ορισμένοι τεχνικοί και αρχιτεκτονικοί περιορισμοί καθορίζουν τα όρια εφαρμογής του συστήματος.

GPS ακρίβεια και εσωτερικοί χώροι. Η ακρίβεια θέσης υπαίθρια ανέρχεται σε ±2–6 m, γεγονός που καθιστά αδύνατη τη χρήση του συστήματος σε εφαρμογές που απαιτούν ακρίβεια κάτω από 1 m, όπως precision landing ή φωτογραμμετρία. Σε εσωτερικό χώρο το πρόβλημα είναι ακόμη σοβαρότερο: το GPS multipath οδηγεί σε σφάλμα θέσης της τάξης δεκάδων χιλιομέτρων (34 km καταγράφηκαν στις δοκιμές), καθιστώντας πρακτικά αδύνατη την αυτόνομη GPS πλοήγηση σε κλειστό περιβάλλον.

Εξάρτηση από WiFi και εμβέλεια. Το Bebop 2 επικοινωνεί αποκλειστικά μέσω WiFi (εμβέλεια ~300 m, latency 50–200 ms), χωρίς εναλλακτική δίοδο επικοινωνίας. Απώλεια σήματος κατά τη διάρκεια αποστολής οδηγεί σε άμεση αναστολή λειτουργίας. Παρότι δοκιμάστηκε εξωτερική κεραία WiFi μεγαλύτερης εμβέλειας, δεν παρατηρήθηκε ουσιαστική βελτίωση στη συγκεκριμένη διάταξη.

Απουσία αποφυγής εμποδίων. Το σύστημα εκτελεί πλοήγηση waypoint υποθέτοντας ελεύθερο εναέριο διάδρομο. Δεν υπάρχει μηχανισμός ανίχνευσης ή αποφυγής εμποδίων σε πραγματικό χρόνο, γεγονός που περιορίζει τη χρήση σε ελεγχόμενα περιβάλλοντα με οπτική επαφή με τον χειριστή.

Αυτονομία και μέγεθος αποστολής. Η LiPo μπαταρία περιορίζει τον συνολικό χρόνο πτήσης στα ~25 λεπτά (σταθερά MAX_MIN = 25), θέτοντας άνω όριο στη συνολική απόσταση και τον αριθμό waypoints κάθε αποστολής. Το σύστημα επαληθεύει αυτό το όριο κατά τον Pre-flight έλεγχο Go/No-Go, αλλά δεν μπορεί να αντισταθμίσει τον εγγενή περιορισμό της μπαταρίας.

Αρχιτεκτονικοί περιορισμοί. Το σύστημα είναι σχεδιασμένο για έλεγχο ενός μόνο drone· δεν υποστηρίζει συντονισμό πολλαπλών UAV. Η κάμερα είναι σταθερής θέσης χωρίς gimbal, περιορίζοντας την ποιότητα εναέριων λήψεων. Τέλος, η ασυμβατότητα του cv_bridge με Python 3 σε ROS Melodic ανάγκασε τη χρήση εναλλακτικής μεθόδου αποκωδικοποίησης εικόνας, περιορίζοντας την άμεση επεκτασιμότητα με τυπικά ROS imaging εργαλεία.

7.3 Μελλοντικές Επεκτάσεις

Βασιζόμενοι στα αποτελέσματα και τους περιορισμούς που εντοπίστηκαν, προτείνονται οι ακόλουθες επτά κατευθύνσεις για μελλοντική έρευνα και ανάπτυξη:

7.4 YawTracker — Παράλληλος Βρόχος Ελέγχου Κατεύθυνσης Μύτης

Κατά τις δοκιμές παρατηρήθηκε ότι το FreeFlight Pro διατηρεί τη μύτη του drone στραμμένη προς την κατεύθυνση κίνησης, ενώ η ROS υλοποίηση αφήνει τη μύτη σταθερή στην αρχική κατεύθυνση. Η έρευνα αποκάλυψε ότι το FreeFlight χρησιμοποιεί υβριδική αρχιτεκτονική: onboard autopilot για θέση + smartphone για συνεχή yaw εντολές μέσω ARSDK control stream.

Ο YawTracker είναι ένα background thread που τρέχει παράλληλα με τον autopilot κατά τη διάρκεια αποστολής. Υπολογίζει σε πραγματικό χρόνο τη γωνία bearing (πυξίδα) προς το επόμενο waypoint και στέλνει αναλογικές yaw εντολές μέσω cmd_vel.angular.z. Ο onboard autopilot αγνοεί τις linear εντολές cmd_vel αλλά δέχεται τις angular.z — αυτό είναι το κλειδί που κάνει την προσέγγιση εφικτή.

Το bearing υπολογίζεται με τον τύπο:

```
def bearing(lat1, lon1, lat2, lon2):
    import math
    la1,la2 = math.radians(lat1), math.radians(lat2)
    dl = math.radians(lon2 - lon1)
    x = math.sin(dl) * math.cos(la2)
    y = math.cos(la1)*math.sin(la2) - math.sin(la1)*math.cos(la2)*math.cos(dl)
    return (math.degrees(math.atan2(x, y)) + 360) % 360   # 0=Βορράς, 90=Ανατολή
```

Ο έλεγχος yaw γίνεται αναλογικά (P-controller): angular error × gain = yaw command. Το gain 0.008 δίνει ομαλή στροφή χωρίς ταλάντωση. Το thread τρέχει σε 10Hz — αρκετό για ομαλή παρακολούθηση κατεύθυνσης.

Αυτή η επέκταση επιλέχθηκε σκόπιμα να μην συμπεριληφθεί στην παρούσα υλοποίηση για να διατηρηθεί η αρχιτεκτονική καθαρότητα του pure onboard autopilot. Η υλοποίησή της παρουσιάζεται εδώ ως τεκμηρίωση της ερευνητικής διαδικασίας.

7.4.1 RTK GPS — Ακρίβεια Εκατοστών αντί Μέτρων

Το κύριο τεχνικό εμπόδιο της παρούσας υλοποίησης είναι η ακρίβεια του GPS του Bebop 2: ±2-6 μέτρα υπό ιδανικές συνθήκες. Αυτό καθιστά αδύνατη τη χρήση σε εφαρμογές που απαιτούν ακρίβεια κάτω από 1 μέτρο, όπως precision landing, survey φωτογραμμετρίας, ή αυτόματη επιθεώρηση υποδομών.

Η τεχνολογία RTK (Real-Time Kinematic) GPS επιλύει αυτό το πρόβλημα χρησιμοποιώντας δύο δέκτες: έναν rover (στο drone) και έναν base station (σταθερός στο έδαφος). Ο base station, γνωρίζοντας ακριβώς τη δική του θέση, υπολογίζει και μεταδίδει διορθώσεις στον rover σε πραγματικό χρόνο. Αυτές οι διορθώσεις εξαλείφουν τα σφάλματα ιονόσφαιρας και τροπόσφαιρας, επιτυγχάνοντας ακρίβεια 1-2 εκατοστά.

Για το Bebop 2 συγκεκριμένα, η υλοποίηση RTK απαιτεί αντικατάσταση του ενσωματωμένου GPS module με εξωτερικό RTK-capable module (π.χ. u-blox ZED-F9P) και ανάπτυξη custom ARSDK

plugin για τη μετάδοση RTK corrections μέσω WiFi. Εναλλακτικά, μπορεί να χρησιμοποιηθεί NTRIP (Networked Transport of RTCM via Internet Protocol) μέσω του υπάρχοντος WiFi link του drone.

Αυτή η επέκταση θα μετέτρεπε το σύστημα από ερευνητικό εργαλείο σε εμπορικά ανταγωνιστικό GCS για επαγγελματικές drone εφαρμογές.

Διαφορά ακρίβειας: Consumer GPS ±2-6m → SBAS (WAAS/EGNOS) ±1-3m → DGPS ±0.3-1m → RTK ±0.01-0.02m. Κάθε βήμα απαιτεί αύξουσα πολυπλοκότητα υλικού.

Σε σχέση με την παρούσα εργασία, η μετρημένη ακρίβεια waypoint των 2–6 m (Κεφάλαιο 3) αντιστοιχεί ακριβώς στο επίπεδο «Consumer GPS» του παραπάνω πίνακα· η αναβάθμιση σε RTK θα ήταν η μόνη επέκταση που μετακινεί το σύστημα σε εφαρμογές ακριβείας, με κόστος όμως σημαντικής πολυπλοκότητας υλικού.

7.4.2 ROS Navigation Stack — Τοπική Αποφυγή Εμποδίων

Στην παρούσα υλοποίηση το drone πλοηγείται αποκλειστικά με βάση GPS waypoints, χωρίς καμία επίγνωση του περιβάλλοντος. Αν υπάρχει εμπόδιο μεταξύ δύο waypoints (π.χ. δέντρο, κτίριο, άλλο drone), ο autopilot θα πετάξει ευθεία πάνω από ή μέσα σε αυτό.

Το ROS Navigation Stack (κυρίως move_base) παρέχει τοπικό σχεδιασμό τροχιάς που αποφεύγει εμπόδια σε πραγματικό χρόνο. Η ενσωμάτωσή του με τον Bebop 2 απαιτεί:

- Αισθητήρα απόστασης: κάμερα βάθους (depth camera) ή LiDAR. Το Bebop 2 δεν έχει ενσωματωμένο, οπότε χρειάζεται εξωτερικό (π.χ. Intel RealSense D435i, βάρος ~73g).
- Χάρτη κόστους (costmap): το move_base δημιουργεί 2D χάρτη με ζώνες κόστους γύρω από εμπόδια. Υψηλό κόστος = αποφεύγεται.
- Τοπικό planner (local planner): υπολογίζει cmd_vel εντολές για αποφυγή εμποδίων σε ορίζοντα ~5 μέτρων ενώ παράλληλα κατευθύνεται προς το waypoint.
- Παγκόσμιο planner (global planner): χρησιμοποιεί A* ή Dijkstra για εύρεση βέλτιστης διαδρομής από τρέχουσα θέση στο waypoint αποφεύγοντας γνωστά εμπόδια.

Η πρόκληση εδώ είναι ότι το move_base σχεδιάστηκε για τροχοφόρα ρομπότ 2D. Η προσαρμογή του για 3D πτήση απαιτεί προσαρμογή των costmaps σε τρεις διαστάσεις, κάτι που έχει υλοποιηθεί σε ερευνητικά πλαίσια (π.χ. με το MAVROS + move_base bridge) αλλά δεν υπάρχει out-of-the-box για Bebop 2.

Εναλλακτική προσέγγιση: χρήση του ROS OctoMap για 3D χαρτογράφηση εμποδίων και custom local planner που δημοσιεύει cmd_vel παρακάμπτοντας το move_base.

7.4.3 Google Maps Interface — Οπτική Τοποθέτηση Waypoints

Η τρέχουσα διεπαφή waypoints απαιτεί χειροκίνητη εισαγωγή GPS συντεταγμένων (επικόλληση από Google Maps). Αυτό είναι λειτουργικό αλλά όχι εργονομικό: ο χρήστης πρέπει να ανοίξει Google Maps σε ξεχωριστό παράθυρο, να κάνει δεξί κλικ, να αντιγράψει τις συντεταγμένες, να τις επικολλήσει στο GUI.

Η επέκταση αυτή ενσωματώνει διαδραστικό χάρτη απευθείας στο GUI. Ο χρήστης κάνει κλικ σε σημείο του χάρτη και το waypoint προστίθεται αυτόματα με τις σωστές συντεταγμένες. Η τρέχουσα διαδρομή GPS (από το MissionLogger) σχεδιάζεται πάνω στον χάρτη σε πραγματικό χρόνο κατά τη διάρκεια πτήσης.

Υλοποίηση με τρεις εναλλακτικές:

- Folium + QWebEngineView Πλεονέκτημα: απλό. Μειονέκτημα: περιορισμένη αμφίδρομη επικοινωνία HTML↔Python.
- PyQtWebEngine + Leaflet.js: πλήρης WebEngine με JavaScript bridge. Κλικ σε χάρτη → JavaScript event → Python callback → προσθήκη waypoint. Πλεονέκτημα: ισχυρό. Μειονέκτημα: πολύπλοκο setup.
- OpenStreetMap tiles + PyQt5 QPainter: εμφάνιση tiles χάρτη χωρίς browser. Πλεονέκτημα: ελαφρύ, offline. Μειονέκτημα: περισσότερος κώδικας για tile management.

Η πλέον πρακτική επιλογή για το συγκεκριμένο σύστημα είναι η δεύτερη (PyQtWebEngine + Leaflet), καθώς επιτρέπει ζωντανή απεικόνιση της διαδρομής GPS πάνω στον χάρτη χρησιμοποιώντας Leaflet polyline που ενημερώνεται από Python μέσω runJavaScript().

7.4.4 ROS 2 Migration — Μετάβαση σε Σύγχρονη Αρχιτεκτονική

Το ROS 1 Melodic έφτασε στο τέλος ζωής του (End of Life) τον Μάιο 2023. Αυτό σημαίνει ότι δεν λαμβάνει πλέον security patches ή bug fixes. Επίσης, το ROS 1 χρησιμοποιεί Python 2.7 εσωτερικά (παρόλο που η εφαρμογή μας τρέχει με Python 3 μέσω του bebop_autonomy), και δεν υποστηρίζει real-time εκτέλεση.

Το ROS 2 (Humble Hawksbill ή Iron Irwini) παρέχει σημαντικές βελτιώσεις:

- Python 3 native support: όλα τα ROS 2 packages γράφονται σε Python 3. Εξαλείφεται το cv_bridge πρόβλημα.
- DDS (Data Distribution Service): αντικαθιστά το rosmaster. Δεν υπάρχει single point of failure — αν ο master πέσει σε ROS 1 χάνεται η επικοινωνία. Σε ROS 2 οι nodes επικοινωνούν peer-to-peer.
- Real-time δυνατότητες: υποστήριξη RTOS και deterministic execution για εφαρμογές που απαιτούν εγγυημένους χρόνους απόκρισης.
- Quality of Service (QoS) policies: έλεγχος αξιοπιστίας, latency, και throughput για κάθε topic ξεχωριστά.
- Lifecycle nodes: nodes με ελεγχόμενο κύκλο ζωής (configure → activate → deactivate → cleanup).

Η πρόκληση είναι ότι δεν υπάρχει επίσημος ROS 2 driver για Bebop 2. Η μετάβαση απαιτεί είτε (α) port του bebop_autonomy σε ROS 2 (σημαντική εργασία), είτε (β) χρήση ros1_bridge που επιτρέπει ROS 1 και ROS 2 nodes να συνυπάρχουν, είτε (γ) χρήση νεότερου Parrot drone (ANAFI) που έχει Olympe SDK με Python 3 native support και μπορεί να ενσωματωθεί με ROS 2 πιο εύκολα.

7.4.5 Multi-Drone — Συντονισμός Πολλαπλών Bebop 2

Η τρέχουσα υλοποίηση ελέγχει ένα μόνο drone. Πολλές εφαρμογές απαιτούν στόλο drones: αποτύπωση μεγάλων εκτάσεων, έρευνα και διάσωση, εξασφαλισμένη κάλυψη περιοχής.

Στο ROS, ο συντονισμός πολλαπλών robots γίνεται μέσω namespaces. Κάθε drone παίρνει το δικό του namespace που προτάσσεται σε όλα τα topics:

```
# Drone 1:
roslaunch bebop_driver bebop_node.launch namespace:=bebop1 ip:=192.168.42.1
# Drone 2 (διαφορετικό WiFi channel):
roslaunch bebop_driver bebop_node.launch namespace:=bebop2 ip:=192.168.43.1
```

```
# Topics γίνονται:
/bebop1/takeoff,  /bebop1/cmd_vel,  /bebop1/fix  ...
/bebop2/takeoff,  /bebop2/cmd_vel,  /bebop2/fix  ...
```

Το GUI θα χρειαστεί επέκταση για multi-drone dashboard: dropdown επιλογής drone ή ταυτόχρονη εμφάνιση πολλαπλών video feeds, ξεχωριστοί χάρτες waypoints, και κεντρικός συντονισμός αποστολών για να αποφεύγονται conflicts (δύο drones στο ίδιο waypoint ταυτόχρονα).

Η μεγαλύτερη πρόκληση είναι το WiFi: κάθε Bebop 2 δημιουργεί το δικό του access point. Το laptop πρέπει να είναι συνδεδεμένο σε πολλαπλά WiFi δίκτυα ταυτόχρονα, κάτι που απαιτεί πολλαπλές WiFi κάρτες ή ένα WiFi router που κάνει bridge τα δίκτυα των drones.

Η νομική χρήση πολλαπλών UAV ταυτόχρονα στον ίδιο εναέριο χώρο απαιτεί ειδικές άδειες από την ΥΠΑ (Υπηρεσία Πολιτικής Αεροπορίας) στην Ελλάδα και αντίστοιχες αρχές σε άλλες χώρες.

7.4.6 Return-to-Home — Αυτόματη Επιστροφή σε Ασφαλείς Συνθήκες

Η τρέχουσα υλοποίηση δεν έχει μηχανισμό αυτόματης ανάκτησης σε κρίσιμες καταστάσεις. Αν η μπαταρία πέσει κάτω από ασφαλές επίπεδο κατά τη διάρκεια αποστολής, το drone συνεχίζει να εκτελεί το plan μέχρι να πέσει η μπαταρία — υπάρχει κίνδυνος αναγκαστικής προσγείωσης μακριά από το σημείο εκκίνησης.

Η επέκταση Return-to-Home (RTH) λειτουργεί ως εξής:

- Κατά την απογείωση αποθηκεύεται η θέση GPS του σημείου εκκίνησης (home position) από το /bebop/fix.
- Παράλληλος thread παρακολουθεί συνεχώς: (α) ποσοστό μπαταρίας, (β) ποιότητα GPS signal, (γ) WiFi signal strength.
- Αν μπαταρία < 15% ή WiFi χαθεί για > 5 δευτερόλεπτα: διακόπτεται αυτόματα η τρέχουσα αποστολή και δημιουργείται νέο MAVLink plan με ένα waypoint — το home position.
- Το νέο plan ανεβαίνει μέσω FTP και εκτελείται αμέσως με af_start().

Υπάρχει ένα σημαντικό τεχνικό ζήτημα: αν το WiFi χαθεί, δεν μπορεί να γίνει FTP upload του RTH plan. Γι' αυτό η υλοποίηση πρέπει να ανεβάζει ΗΔΗ ένα RTH emergency plan στο drone κατά την αρχική σύνδεση, ώστε το drone να μπορεί να το εκτελέσει αυτόνομα αν χαθεί η επικοινωνία. Αυτό αξιοποιεί ακριβώς την ανεξαρτησία WiFi που παρέχει η MAVLink αρχιτεκτονική.

Σημείωση: το Bebop 2 διαθέτει ενσωματωμένο Return-to-Home που ενεργοποιείται αν χαθεί το σήμα FreeFlight Pro. Ωστόσο αυτό δεν ενεργοποιείται αυτόματα κατά τη χρήση ROS — χρειάζεται ρητή εντολή af_start() με RTH plan.

7.5 Συγκρητική Αξιολόγηση Μελλοντικών Επεκτάσεων

Για να βοηθήσουμε στην ιεράρχηση των επεκτάσεων, παρακάτω αξιολογείται κάθε μία ως προς πολυπλοκότητα υλοποίησης, εκτιμώμενο χρόνο ανάπτυξης, και αναμενόμενο όφελος:

Επέκταση	Πολυπλοκότητα	Χρόνος	Κύριο Όφελος

YawTracker	Χαμηλή	1-2 εβδομάδες	Βελτίωση UX — μύτη ακολουθεί κατεύθυνση
Google Maps Interface	Μέτρια	2-3 εβδομάδες	Πολύ ευκολότερη τοποθέτηση waypoints
Return-to-Home	Μέτρια	2-3 εβδομάδες	Ασφάλεια — αποτρέπει απώλεια drone
Multi-drone	Υψηλή	1-2 μήνες	Κάλυψη μεγάλων εκτάσεων
ROS 2 Migration	Πολύ Υψηλή	3-6 μήνες	Μελλοντική συντηρησιμότητα
ROS Navigation Stack	Πολύ Υψηλή	3-6 μήνες	Αυτόνομη αποφυγή εμποδίων
RTK GPS	Υψηλή (hardware)	2-4 μήνες	Ακρίβεια εκατοστών — επαγγελματικές εφαρμογές

Με βάση αυτή την αξιολόγηση, η προτεινόμενη σειρά υλοποίησης είναι: (1) YawTracker και Return-to-Home ως άμεσες βελτιώσεις που προσθέτουν σημαντικό όφελος με χαμηλό κόστος, (2) Google Maps Interface για βελτίωση χρηστικότητας, (3) RTK GPS για αναβάθμιση ακρίβειας, (4) Multi-drone ως πιο φιλόδοξη επέκταση, (5) ROS 2 Migration ως μακροπρόθεσμη στρατηγική απόφαση.

Παράρτημα Κώδικα

8.1 Πίνακας Ενοτήτων Κώδικα

Γραμμές	Περιεχόμενο
1-59	Imports, Constants, STATES dict
60-116	Helper Functions: haversine, parse_coords, parse_mavlink, analyse, gen_mavlink
117-203	Class RosWorker — Signals, init(), subscribers, publishers, callbacks
204-254	Class VideoWorker — JPEG decode pipeline, threading, QPixmap emission
255-287	Class FtpWorker — upload(), download()
288-389	Class MissionLogger — record(), generate_report(), save_csv()
390-479	Class App — Constructor, state variables, _build_ui()
480-837	Class App — Panel Builders (camera, status, flight, direction pad, camera controls, connection, waypoints, analysis, mission control, log)
838-918	_wire_signals(), _card(), _db(), _start_workers()
919-945	_start_workers() — thread and worker setup
946-1007	ROS Slots — _on_conn, _on_batt, _on_state, _on_gps, _on_odom
1008-1065	Manual Control — _takeoff, _land, _hover, _emergency, _setv, _pub_vel, _cam, keyPress/Release
1066-1109	Waypoints — _add_wp, _del_wp, _clr_wp, _refresh, _tbl_edit
1110-1179	Mission Analysis — _analyse()
1180-1234	FTP — _load_drone, _load_file, _save_file, _ftp_done, _load_plan
1235-1292	Mission Control — _upload_fly, _fly_existing, _pause_resume, _stop, _save_csv, _save_report
1293-1371	_reset_state, _confirm_close, _ms, _tick, _log
1372-1563	CSS Stylesheet — _css() — complete QSS theme
1564-1596	closeEvent() — thread cleanup and shutdown
1597-1602	main() — application entry point

8.1.1 Ενότητα 1 — Imports και Σταθερές

Lines 1 – 59 | Imports, Constants, STATES

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Bebop 2 Mission Planner  --  Simple GUI
Run:  python3 bebop_mission_gui.py
Requires: roscore + bebop_driver already running
"""
import sys, os, math, ftplib, tempfile, time, re
import numpy as np

# ── ROS ──────────────────────────────────────────────────────────────
try:
    import rospy
    from std_msgs.msg import Empty, Float32, String
    from geometry_msgs.msg import Twist
    from sensor_msgs.msg import CompressedImage, NavSatFix
    from nav_msgs.msg import Odometry
    from bebop_msgs.msg import (
        Ardrone3PilotingStateFlyingStateChanged,
        CommonCommonStateBatteryStateChanged
    )
    ROS_OK = True
except ImportError:
    ROS_OK = False

try:
    import cv2
except ImportError:
    cv2 = None

from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QWidget, QLabel, QPushButton,
    QVBoxLayout, QHBoxLayout, QGridLayout, QGroupBox,
    QTableWidget, QTableWidgetItem, QHeaderView, QAbstractItemView,
    QDoubleSpinBox, QTextEdit, QProgressBar, QLineEdit,
    QFileDialog, QMessageBox, QSplitter, QSlider
)
from PyQt5.QtCore import Qt, QTimer, pyqtSignal, QThread, QObject
from PyQt5.QtGui import QPixmap, QImage, QColor

# ── Constants ────────────────────────────────────────────────────────
BEBOP_IP   = "192.168.42.1"
FTP_DIR    = "/internal_000/flightplans"
PLAN_NAME  = "flightPlan.mavlink"
BATT_MAH   = 2700
HOVER_MA   = 900
CRUISE_MA  = 1400
MAX_MIN    = 25
RESERVE    = 20

STATES = {
    0:("LANDED","#64748b"),   1:("TAKING OFF","#f59e0b"),
    2:("HOVERING","#22d3ee"), 3:("FLYING","#4ade80"),
    4:("LANDING","#f59e0b"),  5:("EMERGENCY","#ef4444"),
    6:("USR TAKEOFF","#a78bfa"), 7:("RAMPING","#fb923c"),
    8:("EMG LAND","#ef4444"),
}


```

8.1.2 Βοηθητικές Συναρτήσεις

Lines 60 – 116 | haversine | parse_coords | parse_mavlink | analyse | gen_mavlink

```
# ═══════════════════════════════════════════════════════════════
#  HELPERS
# ═══════════════════════════════════════════════════════════════
def haversine(la1, lo1, la2, lo2):
    R = 6_371_000
    p1,p2 = math.radians(la1), math.radians(la2)
    dp = math.radians(la2-la1); dl = math.radians(lo2-lo1)
    a = math.sin(dp/2)**2 + math.cos(p1)*math.cos(p2)*math.sin(dl/2)**2
    return R*2*math.atan2(math.sqrt(a), math.sqrt(1-a))

def parse_coords(text):
    nums = re.findall(r"[-+]?\d+\.\d+", text.strip())
    if len(nums) >= 2:
        return float(nums[0]), float(nums[1])
    raise ValueError("Could not parse coordinates")

def parse_mavlink(content):
    rows = []
    for line in content.strip().splitlines()[1:]:
        p = line.strip().split()
        if len(p) < 12: continue
        try:
            rows.append({'cmd':int(p[3]),'p2':float(p[5]),
                         'lat':float(p[8]),'lon':float(p[9]),'alt':float(p[10])})
        except: continue
    return rows

def analyse(waypoints, speed=3.0, battery=None):
    """waypoints: [(lat,lon,alt), ...]"""
    if not waypoints: return None
    dist = sum(haversine(waypoints[i][0],waypoints[i][1],
                         waypoints[i+1][0],waypoints[i+1][1])
               for i in range(len(waypoints)-1))
    altd = sum(abs(waypoints[i+1][2]-waypoints[i][2]) for i in range(len(waypoints)-1))
    ct = dist / max(speed, 0.1)
    ft = 30 + ct + altd + 20 + 30          # GPS wait + cruise + alt + land + takeoff
    mah = (60+altd)/3600*HOVER_MA + ct/3600*CRUISE_MA
    need = mah/BATT_MAH*100 + RESERVE
    ok = margin = None
    if battery is not None:
        ok = battery >= need; margin = battery - need
    return {'dist':dist,'time':ft,'need':need,'ok':ok,'margin':margin,'speed':speed}

def gen_mavlink(wps, speed=3.0):
    """wps: [(lat,lon,alt,label),...]"""
    lines = ["QGC WPL 120"]
    lines.append(f"0\t0\t3\t178\t0\t{speed:.1f}\t-1\t0\t0\t0\t0\t1")
    lines.append("1\t0\t3\t2500\t0\t30\t0\t0\t0\t0\t0\t1")
    for i,(la,lo,al,_) in enumerate(wps):
        lines.append(f"{i+2}\t0\t3\t16\t0\t5\t0\t0\t{la:.7f}\t{lo:.7f}\t{al:.1f}\t1")
    if wps:
        la,lo,al,_ = wps[-1]

lines.append(f"{len(wps)+2}\t0\t3\t21\t0\t0\t0\t0\t{la:.7f}\t{lo:.7f}\t{al:.1f}\t1")
    lines.append(f"{len(wps)+3}\t0\t3\t2501\t0\t0\t0\t0\t0\t0\t0\t1")
    return "\n".join(lines)+"\n"
```

8.1.3 Κλάση RosWorker

Lines 117 – 203 | Signals | init() | Subscribers | Publishers | Callbacks

```
117  # ═══════════════════════════════════════════════════════════════════════
118  #  ROS WORKER  (runs on its own QThread, communicates only via signals)
119  # ═══════════════════════════════════════════════════════════════════════
120  class RosWorker(QObject):
121      sig_batt    = pyqtSignal(float)
122      sig_state   = pyqtSignal(int)
123      sig_img     = pyqtSignal(object)
124      sig_gps     = pyqtSignal(float, float, float)
125      sig_odom    = pyqtSignal(float, float, float)
126      sig_conn    = pyqtSignal(bool)
127      sig_log     = pyqtSignal(str)
128
129      def __init__(self):
130          super().__init__()
131          self._p = {}   # publishers dict
132
133      def init(self):
134          if not ROS_OK:
135              self.sig_log.emit("rospy not found — ROS unavailable")
136              self.sig_conn.emit(False); return
137          try:
138              rospy.init_node("bebop_missiongui", anonymous=True, disable_signals=True)
139              def pub(topic, typ, qs=1):
140                  return rospy.Publisher(topic, typ, queue_size=qs)
141              self._p['cmd']   = pub("/bebop/cmd_vel",           Twist, 10)
142              self._p['tkf']   = pub("/bebop/takeoff",           Empty)
143              self._p['lnd']   = pub("/bebop/land",              Empty)
144              self._p['emg']   = pub("/bebop/reset",             Empty)
145              self._p['cam']   = pub("/bebop/camera_control",    Twist)
146              self._p['af_s']  = pub("/bebop/autoflight/start",  String)
147              self._p['af_p']  = pub("/bebop/autoflight/pause",  Empty)
148              self._p['af_x']  = pub("/bebop/autoflight/stop",   Empty)
149              # Battery uses bebop_msgs type with .percent field (uint8 0-100)
150              # NOT Float32 -- that was the bug causing no battery display
151              rospy.Subscriber(
152                  "/bebop/states/common/CommonState/BatteryStateChanged",
153                  CommonCommonStateBatteryStateChanged,
154                  lambda m: self.sig_batt.emit(float(m.percent))
155              )
156              rospy.Subscriber("/bebop/states/ardrone3/PilotingState/FlyingStateChanged",
157                               Ardrone3PilotingStateFlyingStateChanged,
158                               lambda m: self.sig_state.emit(m.state))
159              rospy.Subscriber("/bebop/fix", NavSatFix,
160                               lambda m: self.sig_gps.emit(m.latitude, m.longitude,
m.altitude), queue_size=1)
161              rospy.Subscriber("/bebop/odom", Odometry, self._cb_odom, queue_size=1)
162              if cv2:
163                  rospy.Subscriber("/bebop/image_raw/compressed", CompressedImage,
164                                   self._cb_img, queue_size=1, buff_size=2**26,
165                                   tcp_nodelay=True)
166              self.sig_conn.emit(True)
167              self.sig_log.emit("ROS connected — all controls active")
168              rospy.spin()
169          except Exception as e:
170              self.sig_log.emit(f"ROS init failed: {e}")
171              self.sig_conn.emit(False)
172
173      def _cb_img(self, m):
174          self.sig_img.emit(bytes(m.data))
175
176      def _cb_odom(self, m):
177          self.sig_odom.emit(
178              m.pose.pose.position.x,
179              m.pose.pose.position.y,
180              m.pose.pose.position.z)
181
182      def _pub(self, key, msg=None):
183          p = self._p.get(key)
184          if p: p.publish(msg if msg is not None else Empty())
```

```
185
186     def takeoff(self):   self._pub('tkf')
187     def land(self):      self._pub('lnd')
188     def emergency(self): self._pub('emg')
189
190     def cmd_vel(self, lx, ly, lz, az):
191         m = Twist(); m.linear.x=lx; m.linear.y=ly; m.linear.z=lz; m.angular.z=az
192         self._pub('cmd', m)
193
194     def camera(self, pan, tilt):
195         m = Twist(); m.angular.z=pan; m.angular.y=tilt; self._pub('cam', m)
196
197     def af_start(self, name=""):
198         m = String(); m.data=name; self._pub('af_s', m)
199
200     def af_pause(self): self._pub('af_p')
201     def af_stop(self):  self._pub('af_x')
202
203
```

8.1.4 Κλάση VideoWorker

Lines 204 – 254 | push() | _decode() | JPEG pipeline

```
204 # ═══════════════════════════════════════════════════════════════════════
205 #  VIDEO WORKER  (decodes frames on its own thread, emits QPixmap)
206 # ═══════════════════════════════════════════════════════════════════════
207 class VideoWorker(QObject):
208     sig_frame = pyqtSignal(QPixmap)
209
210     def __init__(self):
211         super().__init__()
212         import threading
213         self._lock = threading.Lock()
214         self._pending = None     # single slot -- always newest frame only
215         self._decoding = False   # guard: never decode two frames at once
216
217     def start_timer(self):
218         # 15fps display cap -- gives decoder enough time to finish
219         # before next frame is requested, preventing backlog
220         t = QTimer(); t.setInterval(67); t.timeout.connect(self. decode); t.start()
221         self._timer = t
222
223     def push(self, raw):
224         # Called from ROS thread via Qt signal -- just store latest, drop old
225         with self._lock:
226             self. pending = raw   # overwrite -- only newest frame matters
227
228     def _decode(self):
229         # Skip if previous decode is still running (CPU too slow for current rate)
230         if self._decoding: return
231         with self._lock:
232             d = self. pending
233             self._pending = None
234         if d is None or cv2 is None: return
235         self._decoding = True
236         try:
237             # Copy bytes into a new array -- prevents buffer release mid-decode
238             # which causes distorted/corrupt frames
239             arr = np.frombuffer(d, dtype=np.uint8).copy()
240             img = cv2.imdecode(arr, cv2.IMREAD_COLOR)
241             if img is None: return
242             img = cv2.resize(img, (480, 270), interpolation=cv2.INTER_NEAREST)
243             rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
244             h, w, ch = rgb.shape
245             # Ensure contiguous memory before passing to QImage
246             rgb = np.ascontiguousarray(rgb)
247             qi = QImage(rgb.tobytes(), w, h, ch * w, QImage.Format_RGB888)
248             self.sig_frame.emit(QPixmap.fromImage(qi))
249         except Exception:
250             pass
251         finally:
252             self._decoding = False
```

8.1.5 Κλάση FtpWorker

Lines 255 – 287 | upload() | download()

```
255  # ═══════════════════════════════════════════════════════════════════════
256  #  FTP WORKER  (FTP on its own thread, signals results back)
257  # ═══════════════════════════════════════════════════════════════════════
258  class FtpWorker(QObject):
259      sig_done = pyqtSignal(bool, str, str)   # ok, operation, data/error
260
261      def upload(self, content, ip):
262          tmp = tempfile.NamedTemporaryFile(mode='w', suffix='.mavlink', delete=False)
263          tmp.write(content); tmp.close()
264          try:
265              ftp = ftplib.FTP(ip, timeout=8); ftp.login(); ftp.cwd(FTP_DIR)
266              with open(tmp.name,'rb') as f: ftp.storbinary(f"STOR {PLAN_NAME}", f)
267              ftp.quit()
268              self.sig_done.emit(True, "upload", f"Uploaded {PLAN_NAME}")
269          except Exception as e:
270              self.sig_done.emit(False, "upload", str(e))
271          finally:
272              os.unlink(tmp.name)
273
274      def download(self, ip):
275          try:
276              ftp = ftplib.FTP(ip, timeout=8); ftp.login(); ftp.cwd(FTP_DIR)
277              files = [f for f in ftp.nlst() if f not in ('.','..')]
278              mavs  = [f for f in files if f.lower().endswith('.mavlink')]
279              if not mavs:
280                  ftp.quit(); self.sig_done.emit(False,"download",f"No .mavlink on drone.
Files: {files}"); return
281              buf=[]; ftp.retrlines(f"RETR {mavs[0]}", buf.append); ftp.quit()
282              self.sig_done.emit(True, "download", "\n".join(buf))
283          except Exception as e:
284              self.sig_done.emit(False, "download", str(e))
285
286
287
```

8.1.6 Κλάση MissionLogger

Lines 288 – 389 | reset | start | stop | record | save_csv | generate_report

```
288  # ═══════════════════════════════════════════════════════════════════════
289  #  MISSION LOGGER  --  records GPS/battery/state at 1 Hz, generates report
290  # ═══════════════════════════════════════════════════════════════════════
291  class MissionLogger:
292      def __init__(self): self.reset()
293
294      def reset(self):
295          self._records=[]; self._t0=None; self._active=False
296          self. planned=[]; self. name=""
297
298      def start(self, planned_wps, mission_name="mission"):
299          self.reset()
300          self._planned=[(w[0],w[1],w[2]) for w in planned_wps]
301          self._name=mission_name; self._t0=time.time(); self._active=True
302
303      def stop(self): self._active=False
304
305      def has_data(self): return len(self._records) > 0
306
307      def record(self, lat, lon, alt_gps, battery, state_int,
308                 odom_x=0, odom_y=0, odom_z=0):
309          if not self._active: return
310          self._records.append({
311              'time_s':  round(time.time()-self._t0, 1) if self._t0 else 0,
312              'lat':     lat,    'lon':    lon,   'alt_gps': alt_gps,
313              'odom_x':  round(odom_x,3), 'odom_y': round(odom_y,3), 'odom_z':
round(odom_z,3),
314              'battery': battery, 'state': state_int,
315          })
316
317      def save_csv(self, path):
318          import csv
319
fields=['time_s','lat','lon','alt_gps','odom_x','odom_y','odom_z','battery','state']
320          with open(path,'w',newline='') as f:
321              w=csv.DictWriter(f,fieldnames=fields); w.writeheader();
w.writerows(self._records)
322
323      def generate_report(self):
324          if not self._records: return "No data recorded."
325          R=[]; add=R.append
326          add("="*62)
327          add("  BEBOP 2 AUTONOMOUS GPS MISSION  --  FLIGHT REPORT")
328          add("="*62)
329          add(f"  Mission:   {self._name}")
330          add(f"  Date/Time: {time.strftime('%Y-%m-%d %H:%M:%S',
time.localtime(self._t0))}")
331          add(f"  Samples:   {len(self._records)}  (1 Hz logging)")
332          dur=self._records[-1]['time_s']
333          add(f"  Duration:  {int(dur//60)} min {int(dur%60):02d} sec")
334          add("")
335          # Battery
336          batts=[r['battery'] for r in self._records if r['battery'] is not None]
337          if batts:
338              add(f"  Battery:   start={batts[0]:.0f}%  end={batts[-1]:.0f}%  "
339                  f"consumed={batts[0]-batts[-1]:.0f}%")
340          add("")
341          # GPS track
342          gps=[(r['lat'],r['lon']) for r in self._records if r['lat']!=0 and r['lon']!=0]
343          add(f"  GPS TRACK:")
344          add(f"    Valid GPS samples: {len(gps)} / {len(self._records)}")
345          if len(gps)>1:
346              dist=sum(haversine(gps[i][0],gps[i][1],gps[i+1][0],gps[i+1][1])
347                       for i in range(len(gps)-1))
348              add(f"    Total distance flown: {dist:.1f} m  ({dist/1000:.3f} km)")
349          add("")
350          # Waypoint accuracy
351          if self._planned and len(gps)>0:
352              add("  WAYPOINT ACCURACY  (planned vs closest actual GPS point):")
```

```
            add(f"    {'WP':<5} {'Planned lat':>12} {'Planned lon':>12} {'Alt':>7}
{'Error':>9}")
            add("    "+"-"*50)
            errors=[]
            for i,(pla,plo,pal) in enumerate(self._planned):
                dists=[haversine(pla,plo,g[0],g[1]) for g in gps]
                mn=min(dists); errors.append(mn)
                add(f"    WP{i+1:<4} {pla:>12.6f} {plo:>12.6f} {pal:>6.1f}m
{mn:>8.1f}m")
            add("")
            add(f"    Mean error: {sum(errors)/len(errors):.1f} m")
            add(f"    Max error:  {max(errors):.1f} m")
            add(f"    Min error:  {min(errors):.1f} m")
            add("")
            add("  INTERPRETATION:")
            add("    Bebop 2 GPS hardware accuracy: ~2-6 metres.")
            add("    Errors within this range mean the onboard autopilot")
            add("    is performing correctly within hardware limits.")
            mean_err=sum(errors)/len(errors)
            if mean_err <= 6:
                add("    RESULT: Mean error within GPS hardware spec. PASS.")
            elif mean_err <= 15:
                add("    RESULT: Mean error above GPS spec but acceptable.")
            else:
                add("    RESULT: Large error -- check GPS lock quality.")
        add("")
        # State breakdown
        from collections import Counter
        snames={0:"LANDED",1:"TAKING OFF",2:"HOVERING",3:"FLYING",
                4:"LANDING",5:"EMERGENCY",6:"USR TAKEOFF",7:"RAMPING",8:"EMG LAND"}
        counts=Counter(r['state'] for r in self._records)
        add("  FLIGHT STATE BREAKDOWN:")
        for s,n in sorted(counts.items()):
            pct=n/len(self._records)*100
            add(f"    {snames.get(s,str(s)):<15} {n:>4} sec  ({pct:.0f}%)")
        add("")
        add("="*62)
        return "\n".join(R)

```

8.1.7 Κλάση App — Constructor και State

Lines 390 – 479 | __init__ | State variables | _build_ui

```
390  # ═══════════════════════════════════════════════════════════════════════
391  #  MAIN WINDOW
392  # ═══════════════════════════════════════════════════════════════════════
393  class App(QMainWindow):
394      def __init__(self):
395          super().__init__()
396          self.setWindowTitle("Bebop 2  —  Mission Planner")
397          self.setMinimumSize(1100, 700)
398
399          # State
400          self._wps      = []      # [[lat,lon,alt,label], ...]
401          self._battery  = None
402          self._state    = 0
403          self._conn     = False
404          self._flying   = False
405          self._paused   = False
406          self._plan     = None    # current mavlink string
407          self._t0       = None    # mission start time
408          self._lx = self._ly = self._lz = self._az = 0.0
409          self._spd = 0.3
410          self._pan = self._tilt = 0.0
411          self._lat = self._lon = self._alt_gps = 0.0
412          self._ox  = self._oy  = self._oz = 0.0
413          self._logger = MissionLogger()
414
415          self._build_ui()
416          self._css()
417          self._start_workers()
418
419          # Timers
420          self._vel_t = QTimer(); self._vel_t.setInterval(100)
421          self._vel_t.timeout.connect(self._pub_vel); self._vel_t.start()
422          self._tick_t = QTimer(); self._tick_t.setInterval(1000)
423          self._tick_t.timeout.connect(self._tick); self._tick_t.start()
424
425      # ═══════════════════════════════════════════════════════════════════
426      #  UI BUILD  --  each panel is its own method for clarity
427      #  Called once from __init__ in this order:
428      #    _build_ui  ->  _build_camera_card
429      #                ->  _build_status_card
430      #                ->  _build_flight_card
431      #                ->  _build_direction_pad
432      #                ->   build camera controls
433      #                ->  _build_connection_row
434      #                ->  _build_waypoint_panel
435      #                ->  _build_analysis_panel
436      #                ->  _build_mission_control
437      #                ->  _build_log_panel
438      #                ->   wire signals
439      # ═══════════════════════════════════════════════════════════════════
440
441      def _build_ui(self):
442          """
443          Top-level UI builder.
444          Creates the root two-column layout and calls each panel builder.
445          Left column (stretch 5): camera, status, flight commands, manual control.
446          Right column (stretch 6): connection row, waypoints, analysis, mission, log.
447          """
448          cw = QWidget(); self.setCentralWidget(cw)
449          root = QHBoxLayout(cw)
450          root.setContentsMargins(8, 8, 8, 8)
451          root.setSpacing(8)
452
453          # Left column
454          left = QVBoxLayout(); left.setSpacing(6)
455          left.addWidget(self._build_camera_card())
456          left.addWidget(self._build_status_card())
457          left.addWidget(self._build_flight_card())
458          left.addWidget(self._build_direction_pad())
459          left.addWidget(self._build_camera_controls())
460          left.addStretch()
```

```
        root.addLayout(left, 5)

        # Right column
        right = QVBoxLayout(); right.setSpacing(6)
        right.addLayout(self._build_connection_row())
        spl = QSplitter(Qt.Horizontal)
        spl.addWidget(self._build_waypoint_panel())
        spl.addWidget(self._build_analysis_panel())
        spl.setSizes([520, 260])
        right.addWidget(spl)
        right.addWidget(self._build_mission_control())
        log_tabs, rep_row = self._build_log_panel()
        right.addWidget(log_tabs)
        right.addLayout(rep_row)
        root.addLayout(right, 6)

        self._wire_signals()
        self.statusBar().showMessage("Initialising...")

```

8.1.8 build_camera_card() (Γραμμές 482–494)

Lines 482 – 494 | 480×270 video QLabel

```
482      def _build_camera_card(self):
483          """
484          480x270 video display label inside a group box.
485          Fixed size matches the Bebop 2 camera 16:9 aspect ratio.
486          Video frames are set via lbl_cam.setPixmap() from the VideoWorker thread.
487          """
488          card = self._card("FRONT CAMERA")
489          self.lbl_cam = QLabel("No video feed")
490          self.lbl_cam.setFixedSize(480, 270)
491          self.lbl_cam.setAlignment(Qt.AlignCenter)
492          self.lbl_cam.setObjectName("cam")
493          card.layout().addWidget(self.lbl_cam)
494          return card
```

8.1.9 build_status_card() (Γραμμές 496–520)

Lines 496 – 520 | State label | Battery bar | Flight timer

```
496      def _build_status_card(self):
497          """
498          Three rows:
499            - State label: text and colour update from _on_state() via ROS
500            - Battery row: QProgressBar (0-100) + percentage label
501            - Flight timer: updated every second by _tick()
502          """
503          card = self._card("STATUS")
504          self.lbl_state = QLabel("DISCONNECTED")
505          self.lbl_state.setObjectName("state_lbl")
506          self.lbl_state.setAlignment(Qt.AlignCenter)
507          card.layout().addWidget(self.lbl_state)
508
509          batt_row = QHBoxLayout()
510          batt_row.addWidget(QLabel("BATT:"))
511          self.batt_bar = QProgressBar(); self.batt_bar.setRange(0, 100)
512          self.lbl_batt = QLabel("--%")
513          batt_row.addWidget(self.batt_bar)
514          batt_row.addWidget(self.lbl_batt)
515          card.layout().addLayout(batt_row)
516
517          self.lbl_time = QLabel("Time: --:--")
518          self.lbl_time.setObjectName("small")
519          card.layout().addWidget(self.lbl_time)
520          return card
```

8.1.10 build_flight_card() (Γραμμές 522–537)

Lines 522 – 537 | TAKE OFF | LAND | HOVER | EMERGENCY

```
522      def _build_flight_card(self):
523          """
524          Four command buttons stacked vertically.
525          TAKE OFF and EMERGENCY have confirmation dialogs (see _takeoff, _emergency).
526          LAND and HOVER send commands immediately with no confirmation.
527          Min height 36px ensures buttons are easy to press.
528          """
529          card = self._card("FLIGHT")
530          self.btn_tkf = QPushButton("TAKE OFF");
self.btn_tkf.setObjectName("btn_tkf")
531          self.btn_lnd = QPushButton("LAND");
self.btn_lnd.setObjectName("btn_lnd")
532          self.btn_hov = QPushButton("HOVER (Space)");
self.btn_hov.setObjectName("btn_hov")
533          self.btn_emg = QPushButton("EMERGENCY");
self.btn_emg.setObjectName("btn_emg")
534          for b in [self.btn_tkf, self.btn_lnd, self.btn_hov, self.btn_emg]:
535              b.setMinimumHeight(36)
536              card.layout().addWidget(b)
537          return card
```

8.1.11 _build_direction_pad() (Γραμμές 539–588)

Lines 539 – 588 | Cross grid | Altitude+Yaw row | Speed slider

```
539      def _build_direction_pad(self):
540          """
541          Manual velocity control pad.
542
543          Cross grid (QGridLayout 3x3) for horizontal movement:
544            row 0 col 1 = Forward  (W)
545            row 1 col 0 = Left     (A)
546            row 1 col 1 = Stop
547            row 1 col 2 = Right    (D)
548            row 2 col 1 = Back     (S)
549
550          Second row (QHBoxLayout) for altitude and yaw:
551            UP (R)  DOWN (F)  YAW L (Q)  YAW R (E)
552
553          Speed slider: range 5-80, divides by 100 to give 0.05-0.80 multiplier.
554          All buttons use pressed/released signals for continuous movement.
555          Keyboard equivalents are handled in keyPressEvent/keyReleaseEvent.
556          """
557          card = self._card("MANUAL CONTROL  (W/A/S/D  R/F  Q/E)")
558
559          # Cross grid
560          g = QGridLayout(); g.setSpacing(3)
561          self.btn_fwd   = self._db("FWD\n(W)");    g.addWidget(self.btn_fwd,   0, 1)
562          self.btn_left  = self._db("LEFT\n(A)");    g.addWidget(self.btn_left,  1, 0)
563          self.btn_stop  = self._db("STOP");         g.addWidget(self.btn_stop,  1, 1)
564          self.btn_right = self._db("RIGHT\n(D)");   g.addWidget(self.btn_right, 1, 2)
565          self.btn_back  = self._db("BACK\n(S)");    g.addWidget(self.btn_back,  2, 1)
566          card.layout().addLayout(g)
567
568          # Altitude + yaw row
569          g2 = QHBoxLayout()
570          self.btn_up   = self._db("UP\n(R)");     self.btn_up.setMinimumHeight(28)
571          self.btn_dn   = self._db("DOWN\n(F)");   self.btn_dn.setMinimumHeight(28)
572          self.btn_yawl = self._db("YAW L\n(Q)"); self.btn_yawl.setMinimumHeight(28)
573          self.btn_yawr = self._db("YAW R\n(E)"); self.btn_yawr.setMinimumHeight(28)
574          for b in [self.btn_up, self.btn_dn, self.btn_yawl, self.btn_yawr]:
575              g2.addWidget(b)
576          card.layout().addLayout(g2)
577
578          # Speed slider
579          spd_row = QHBoxLayout()
580          spd_row.addWidget(QLabel("Speed:"))
581          self.spd_sl = QSlider(Qt.Horizontal)
582          self.spd_sl.setRange(5, 80); self.spd_sl.setValue(30)
583          self.spd_sl.valueChanged.connect(lambda v: setattr(self, '_spd', v/100))
584          self.lbl_spd = QLabel("0.30")
585          self.spd_sl.valueChanged.connect(lambda v: self.lbl_spd.setText(f"{v/100:.2f}"))
586          spd_row.addWidget(self.spd_sl); spd_row.addWidget(self.lbl_spd)
587          card.layout().addLayout(spd_row)
588          return card
```

8.1.12 _build_camera_controls() (Γραμμές 590–608)

Lines 590 – 608 | Pan ±180° | Tilt ±83° | Reset

```
590      def _build_camera_controls(self):
591          """
592          Five buttons in a horizontal row.
593          Pan Left / Pan Right: adjust angular.z in steps of 5 degrees (-180 to +180)
594          Tilt Up  / Tilt Down: adjust angular.y in steps of 5 degrees (-83  to +83)
595          Reset: sets both pan and tilt to 0 (forward-facing centre position)
596          Commands publish to /bebop/camera_control as a Twist message.
597          """
598          card = self._card("CAMERA")
599          row = QHBoxLayout()
600          self.btn_pl = QPushButton("Pan ←"); self.btn_pl.setFixedHeight(26)
601          self.btn_pr = QPushButton("Pan →"); self.btn_pr.setFixedHeight(26)
602          self.btn_tu = QPushButton("Tilt ↑"); self.btn_tu.setFixedHeight(26)
```

```
603          self.btn_td = QPushButton("Tilt ↓"); self.btn_td.setFixedHeight(26)
604          self.btn_cr = QPushButton("Reset");   self.btn_cr.setFixedHeight(26)
605          for b in [self.btn_pl, self.btn_pr, self.btn_tu, self.btn_td, self.btn_cr]:
606              row.addWidget(b)
607          card.layout().addLayout(row)
608          return card
```

8.1.13 _build_connection_row() (Γραμμές 610–626)

Lines 610 – 626 | IP input | ROS status label

```
610      def _build_connection_row(self):
611          """
612          Single horizontal row at the top of the right column.
613          IP text box (QLineEdit, fixed 130px): editable drone IP address,
614            default 192.168.42.1 which is fixed on every Bebop 2.
615          ROS status label: updated by _on_conn() when ROS connects or fails.
616          addStretch() pushes both widgets to the left.
617          """
618          row = QHBoxLayout()
619          row.addWidget(QLabel("Bebop IP:"))
620          self.txt_ip = QLineEdit(BEBOP_IP); self.txt_ip.setFixedWidth(130)
621          self.lbl_ros = QLabel("ROS: connecting...")
622          self.lbl_ros.setObjectName("ros status")
623          row.addWidget(self.txt_ip)
624          row.addWidget(self.lbl_ros)
625          row.addStretch()
626          return row
```

8.1.14 _build_waypoint_panel() (Γραμμές 628–697)

Lines 628 – 697 | Paste box | Table | Speed spinbox | File buttons

```
628      def _build_waypoint_panel(self):
629          """
630          Left half of the QSplitter. Contains:
631
632          Header row: "WAYPOINTS" label + Remove + Clear buttons.
633
634          Coordinate paste row:
635            QLineEdit with placeholder text -- user pastes Google Maps coordinates.
636            returnPressed signal triggers _add_wp() so pressing Enter adds the waypoint.
637            Altitude spinbox: range 1-120m, default 10m, step 1m.
638            Add button: same action as pressing Enter.
639
640          QTableWidget (0 rows, 4 columns):
641            Columns: Label | Latitude (7dp) | Longitude (7dp) | Alt (1dp)
642            Stretch mode: all columns share width equally.
643            SelectRows: clicking any cell selects the full row.
644            MaxHeight 180px: prevents table growing too large.
645            itemChanged signal: calls _tbl_edit() for live editing.
646            blockSignals(True/False) used in _refresh() to prevent signal loop.
647
648          File row: speed spinbox + Load from Drone + Load .mavlink + Save .mavlink
649          """
650          w = QWidget(); lay = QVBoxLayout(w); lay.setSpacing(5)
651
652          # Header
653          hdr = QHBoxLayout()
654          hdr.addWidget(QLabel("WAYPOINTS", objectName="sec")); hdr.addStretch()
655          self.btn_del = QPushButton("Remove"); self.btn_del.setFixedHeight(24)
656          self.btn_clr = QPushButton("Clear");  self.btn_clr.setFixedHeight(24)
657          hdr.addWidget(self.btn_del); hdr.addWidget(self.btn_clr)
658          lay.addLayout(hdr)
659
660          # Paste row
661          paste = QHBoxLayout()
662          self.txt_coord = QLineEdit()
663          self.txt_coord.setPlaceholderText("Paste from Google Maps: 40.584946,
23.339903")
664          self.txt_coord.returnPressed.connect(self._add_wp)
665          paste.addWidget(self.txt_coord)
666          self.spin_alt = QDoubleSpinBox()
```

```
667          self.spin_alt.setRange(1, 120); self.spin_alt.setValue(10)
668          self.spin_alt.setSuffix(" m"); self.spin_alt.setFixedWidth(80)
669          paste.addWidget(QLabel("Alt:")); paste.addWidget(self.spin_alt)
670          self.btn_add = QPushButton("Add"); self.btn_add.setFixedWidth(48)
671          paste.addWidget(self.btn_add)
672          lay.addLayout(paste)
673
674          # Table
675          self.wt = QTableWidget(0, 4); self.wt.setObjectName("wt")
676          self.wt.setHorizontalHeaderLabels(["Label","Latitude","Longitude","Alt(m)"])
677          self.wt.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
678          self.wt.setSelectionBehavior(QAbstractItemView.SelectRows)
679          self.wt.setMaximumHeight(180)
680          self.wt.itemChanged.connect(self._tbl_edit)
681          lay.addWidget(self.wt)
682
683          # File row
684          fr = QHBoxLayout()
685          fr.addWidget(QLabel("Speed:"))
686          self.spin_spd = QDoubleSpinBox()
687          self.spin_spd.setRange(0.5, 10); self.spin_spd.setValue(3.0)
688          self.spin_spd.setSuffix(" m/s"); self.spin_spd.setFixedWidth(85)
689          fr.addWidget(self.spin_spd); fr.addStretch()
690          self.btn_load_drone = QPushButton("Load from Drone")
691          self.btn_load_drone.setFixedHeight(26);
self.btn_load_drone.setObjectName("btn_ff")
692          self.btn_load_file = QPushButton("Load .mavlink");
self.btn_load_file.setFixedHeight(26)
693          self.btn_save_file = QPushButton("Save .mavlink");
self.btn_save_file.setFixedHeight(26)
694          for b in [self.btn_load_drone, self.btn_load_file, self.btn_save_file]:
695              fr.addWidget(b)
696          lay.addLayout(fr)
697          return w
```

8.1.15 _build_analysis_panel() (Γραμμές 699–754)

Lines 699 – 754 | Stat rows | Go/No-Go | Live GPS | Logger status

```
699      def _build_analysis_panel(self):
700          """
701          Right half of the QSplitter. Contains:
702
703          Five stat rows (key label + value label pairs):
704            Distance    -- total haversine distance in metres and km
705            Est. time   -- estimated flight time in min:sec
706            Need batt   -- battery % required including 20% reserve
707            Have batt   -- current battery % from ROS (or -- if not connected)
708            Margin      -- difference: green if positive, red if negative
709
710          Go/No-Go indicator (QLabel, min height 38px):
711            Green background + SAFE TO FLY, or red + DO NOT FLY + reason list.
712            Updated by _analyse() when the Analyse button is pressed.
713
714          Analyse button: triggers _analyse() to recalculate all values.
715
716          Live GPS section: shows current lat/lon from /bebop/fix at full 6dp precision.
717          Logger status: shows recording state and sample count during missions.
718          """
719          w = QWidget(); lay = QVBoxLayout(w); lay.setSpacing(5)
720          lay.addWidget(QLabel("MISSION ANALYSIS", objectName="sec"))
721
722          def stat_row(label):
723              """Creates one key-value row, adds it to lay, returns the value label."""
724              row = QHBoxLayout()
725              k = QLabel(label); k.setObjectName("sk")
726              v = QLabel("--");  v.setObjectName("sv")
727              row.addWidget(k); row.addWidget(v)
728              lay.addLayout(row)
729              return v
730
731          self.v_dist = stat_row("Distance:")
732          self.v_time = stat_row("Est. time:")
733          self.v_need = stat_row("Need batt:")
734          self.v_have = stat_row("Have batt:")
```

```
735          self.v_marg = stat_row("Margin:")
736
737          self.lbl_gng = QLabel("--")
738          self.lbl_gng.setObjectName("gng")
739          self.lbl_gng.setAlignment(Qt.AlignCenter)
740          self.lbl_gng.setMinimumHeight(38)
741          lay.addWidget(self.lbl_gng)
742
743          self.btn_analyse = QPushButton("▶  Analyse Mission")
744          self.btn_analyse.setObjectName("btn_an"); self.btn_analyse.setMinimumHeight(32)
745          lay.addWidget(self.btn_analyse)
746
747          lay.addWidget(QLabel("LIVE GPS", objectName="sec"))
748          self.lbl_gps_pos = QLabel("lat: --   lon: --");
self.lbl_gps_pos.setObjectName("sv")
749          lay.addWidget(self.lbl_gps_pos)
750          self.lbl_log_status = QLabel("Logger: idle");
self.lbl_log_status.setObjectName("sk")
751          lay.addWidget(self.lbl_log_status)
752
753          lay.addStretch()
754          return w
```

8.1.16 _build_mission_control() (Γραμμές 756–803)

Lines 756 – 803 | UPLOAD & FLY | PAUSE | STOP | Reset | Close

```
756      def _build_mission_control(self):
757          """
758          Mission status label + four mission control buttons.
759
760          Status label (lbl_ms): text and colour updated by _ms() throughout the mission.
761            Colours: amber=uploading/paused, green=running, cyan=complete,
red=failed/stopped.
762
763          UPLOAD & FLY: generates MAVLink from _wps, uploads via FTP, triggers autopilot.
764          FLY EXISTING PLAN: triggers whatever plan is already stored on the drone.
765          PAUSE: calls /bebop/autoflight/pause, button text toggles to RESUME.
766          STOP: calls /bebop/autoflight/stop, resets all mission state, restores manual
control.
767
768          All four buttons min height 40px for easy pressing during flight.
769          """
770          card = self._card("MISSION CONTROL")
771
772          status_row = QHBoxLayout()
773          status_row.addWidget(QLabel("STATUS:"))
774          self.lbl_ms = QLabel("IDLE"); self.lbl_ms.setObjectName("ms")
775          status row.addWidget(self.lbl ms); status row.addStretch()
776          card.layout().addLayout(status_row)
777
778          btn_row = QHBoxLayout()
779          self.btn_uf  = QPushButton("UPLOAD & FLY");
self.btn_uf.setObjectName("btn_go")
780          self.btn fe  = QPushButton("FLY EXISTING PLAN");
self.btn_fe.setObjectName("btn_fe")
781          self.btn_pau = QPushButton("PAUSE")
782          self.btn_stp = QPushButton("STOP");
self.btn_stp.setObjectName("btn_st")
783          for b in [self.btn_uf, self.btn_fe, self.btn_pau, self.btn_stp]:
784              b.setMinimumHeight(40); btn_row.addWidget(b)
785          card.layout().addLayout(btn_row)
786
787          # Reset + Close row
788          bottom_row = QHBoxLayout()
789          self.btn_reset = QPushButton("\u21ba  Reset After Landing")
790          self.btn_reset.setObjectName("btn_reset")
791          self.btn_reset.setFixedHeight(30)
792          self.btn_reset.setToolTip(
793              "Force-clears all mission state so the drone responds normally.\n"
794              "Press this if the drone is not responding after landing.")
795          bottom_row.addWidget(self.btn_reset)
796          bottom_row.addStretch()
797          self.btn_close = QPushButton("\u2715  Close Application")
798          self.btn_close.setObjectName("btn_close")
```

```
799          self.btn_close.setFixedHeight(30)
800          self.btn_close.setFixedWidth(180)
801          bottom_row.addWidget(self.btn_close)
802          card.layout().addLayout(bottom row)
803          return card
```

8.1.17 _build_log_panel() (Γραμμές 805–836)

Lines 805 – 836 | LOG tab | FLIGHT REPORT tab | Save buttons

```
805      def _build_log_panel(self):
806          """
807          Returns (log_tabs, rep_row) -- two separate items added to right column.
808
809          QTabWidget with two tabs:
810            LOG tab: scrollable read-only QTextEdit. Text added via _log(msg).
811            FLIGHT REPORT tab: scrollable read-only QTextEdit. Populated by
812              MissionLogger.generate_report() after mission complete or stop.
813
814          Save buttons row (disabled until a mission generates data):
815            Save CSV: writes all logged records to a CSV file via MissionLogger.save_csv()
816            Save Report: writes the report text to a plain text file.
817          Both buttons are enabled by _on_state() or _stop() after flight data exists.
818          """
819          from PyQt5.QtWidgets import QTabWidget
820          log_tabs = QTabWidget(); log_tabs.setObjectName("tabs");
log_tabs.setMaximumHeight(170)
821
822          self.log = QTextEdit(); self.log.setReadOnly(True);
self.log.setObjectName("log")
823          log_tabs.addTab(self.log, "LOG")
824
825          self.log_report = QTextEdit(); self.log_report.setReadOnly(True)
826          self.log_report.setObjectName("log")
827          self.log report.setPlaceholderText("Flight report appears here after
mission...")
828          log_tabs.addTab(self.log_report, "FLIGHT REPORT")
829
830          rep_row = QHBoxLayout()
831          self.btn_save_csv    = QPushButton("Save CSV");
self.btn_save_csv.setFixedHeight(26)
832          self.btn_save_report = QPushButton("Save Report");
self.btn_save_report.setFixedHeight(26)
833          self.btn_save_csv.setEnabled(False); self.btn_save_report.setEnabled(False)
834          rep_row.addStretch()
835          rep_row.addWidget(self.btn_save_csv); rep_row.addWidget(self.btn_save_report)
836          return log_tabs, rep_row
```

8.1.18 _wire_signals, _card, _db

Lines 838 – 918 | All signal→slot connections | Widget helpers

```
838      def _wire_signals(self):
839          """
840          Connects all button signals to their handler methods.
841          Called last by _build_ui() once all widgets exist.
842          Separated from the widget creation code so the connection
843          map is easy to read and modify.
844
845          pressed/released pattern on direction pad buttons:
846            pressed  -> _setv(lx,ly,lz,az) : sets velocity while held
847            released -> _hover()           : zeroes velocity on release
848            Default argument capture (lx=lx) is essential -- without it
849            all lambdas would share the last loop iteration's values.
850          """
851          # Flight commands
852          self.btn_tkf.clicked.connect(self._takeoff)
853          self.btn_lnd.clicked.connect(self._land)
854          self.btn_hov.clicked.connect(self._hover)
855          self.btn_emg.clicked.connect(self._emergency)
856
857          # Direction pad -- held movement
858          for btn, lx, ly, lz, az in [
859              (self.btn_fwd,   1, 0, 0, 0), (self.btn_back, -1, 0, 0, 0),
860              (self.btn_left,  0, 1, 0, 0), (self.btn_right,  0,-1, 0, 0),
861              (self.btn_up,    0, 0, 1, 0), (self.btn_dn,     0, 0,-1, 0),
862              (self.btn_yawl,  0, 0, 0, 1), (self.btn_yawr,   0, 0, 0,-1),
863          ]:
864              btn.pressed.connect(lambda lx=lx,ly=ly,lz=lz,az=az: self._setv(lx,ly,lz,az))
865              btn.released.connect(self._hover)
866          self.btn_stop.clicked.connect(self._hover)
867
868          # Camera controls
869          self.btn_pl.clicked.connect(lambda: self._cam(-5, 0))
870          self.btn_pr.clicked.connect(lambda: self._cam( 5, 0))
871          self.btn_tu.clicked.connect(lambda: self._cam( 0, 5))
872          self.btn_td.clicked.connect(lambda: self._cam( 0,-5))
873          self.btn_cr.clicked.connect(lambda: self._cam_abs(0, 0))
874
875          # Waypoint management
876          self.btn_add.clicked.connect(self._add_wp)
877          self.btn_del.clicked.connect(self._del_wp)
878          self.btn_clr.clicked.connect(self._clr_wp)
879          self.btn_load_drone.clicked.connect(self._load_drone)
880          self.btn_load_file.clicked.connect(self._load_file)
881          self.btn_save_file.clicked.connect(self._save_file)
882
883          # Mission analysis and control
884          self.btn_analyse.clicked.connect(self._analyse)
885          self.btn_uf.clicked.connect(self._upload_fly)
886          self.btn_fe.clicked.connect(self._fly_existing)
887          self.btn_pau.clicked.connect(self._pause_resume)
888          self.btn_stp.clicked.connect(self._stop)
889
890          # Report saving
891          self.btn_save_csv.clicked.connect(self._save_csv)
892          self.btn_save_report.clicked.connect(self._save_report)
893
894          # Reset and Close
895          self.btn_reset.clicked.connect(self._reset_state)
896          self.btn_close.clicked.connect(self._confirm_close)
897
898      # ── Widget helpers ──────────────────────────────────────────────
899
900      def _card(self, t):
901          """
902          Creates a QGroupBox with title t, dark card styling, and a QVBoxLayout.
903          ContentsMargins (6,14,6,6): top 14px leaves room for the group box title text.
904          All panels in the GUI use this helper for visual consistency.
905          """
906          b = QGroupBox(t); b.setObjectName("card"); b.setLayout(QVBoxLayout())
907          b.layout().setContentsMargins(6, 14, 6, 6); b.layout().setSpacing(4)
908          return b
```

```
909
910     def _db(self, t):
911         """
912         Creates a direction pad button with minimum size 58x44px.
913         Larger than standard buttons so they are easy to press during flight.
914         Object name 'dp' applies the direction pad CSS style.
915         """
916         b = QPushButton(t); b.setObjectName("dp"); b.setMinimumSize(58, 44)
917         return b
918
```

8.1.19 _start_workers

Lines 919 – 945 | VideoWorker | FtpWorker | RosWorker thread setup

```
919      # ── START WORKERS ──────────────────────────────────────────────────────
920      def _start_workers(self):
921          # Video
922          self._vid = VideoWorker(); self._vt = QThread()
923          self._vid.moveToThread(self._vt)
924          self._vid.sig_frame.connect(lambda px: self.lbl_cam.setPixmap(px))
925          self._vt.started.connect(self._vid.start_timer); self._vt.start()
926
927          # FTP
928          self._ftp = FtpWorker(); self._ft = QThread()
929          self._ftp.moveToThread(self._ft)
930          self._ftp.sig_done.connect(self._ftp_done)
931          self._ft.start()
932
933          # ROS — start immediately
934          self._ros = RosWorker(); self._rt = QThread()
935          self._ros.moveToThread(self._rt)
936          self._ros.sig_batt.connect(self._on_batt)
937          self._ros.sig_state.connect(self._on_state)
938          self._ros.sig_img.connect(self._vid.push)
939          self._ros.sig_gps.connect(self._on_gps)
940          self._ros.sig_odom.connect(self._on_odom)
941          self._ros.sig_conn.connect(self._on_conn)
942          self._ros.sig_log.connect(self._log)
943          self._rt.started.connect(self._ros.init)
944          self._rt.start()
945
```

8.1.20 ROS Slots

Lines 946 – 1007 | _on_conn | _on_batt | _on_state | _on_gps | _on_odom

```
946      # ── ROS SLOTS ──────────────────────────────────────────────────────
947      def _on_conn(self, ok):
948          self._conn = ok
949          if ok:
950              self.lbl_ros.setText("ROS: CONNECTED")
951              self.lbl_ros.setStyleSheet("color:#4ade80;font-weight:bold;")
952          else:
953              self.lbl_ros.setText("ROS: FAILED — check roscore + bebop_driver")
954              self.lbl_ros.setStyleSheet("color:#ef4444;")
955
956      def _on_batt(self, v):
957          self._battery = v; vi = int(v)
958          self.batt_bar.setValue(vi); self.lbl_batt.setText(f"{vi}%")
959          c = "#4ade80" if vi>30 else ("#f59e0b" if vi>15 else "#ef4444")
960          self.batt_bar.setStyleSheet(f"QProgressBar::chunk{{background:{c};border-
radius:2px;}}")
961          self.v_have.setText(f"{vi}%")
962
963      def _on_state(self, s):
964          prev = self._state
965          self._state = s
966          name, col = STATES.get(s, ("UNKNOWN","#64748b"))
967          self.lbl_state.setText(f"  {name}  ")
968          self.lbl_state.setStyleSheet(
969              f"color:{col};border:1px solid {col};border-radius:4px;"
970              f"padding:4px;font-size:13px;font-weight:bold;letter-spacing:2px;")
971
972          # Mission complete detection:
973          # Conditions: (1) now LANDED  (2) was NOT landed before  (3) mission was running
974          # No elapsed time guard -- it was causing the "stops responding" bug when
975          # _t0 was None or the mission was short. The prev!=0 guard is sufficient.
976          if s == 0 and prev != 0 and self._flying:
977              self._flying = False
978              self._paused = False
979              self._t0 = None
980              self._hover()                          # zero cmd_vel immediately
981              self._ms("COMPLETE", "#0891b2")
982              self.btn_uf.setEnabled(True)
983              self.btn_pau.setText("PAUSE")
984              self._log("Mission complete — manual control fully restored")
985              self._log("Keyboard (W/A/S/D/R/F/Q/E) and direction pad are active")
986              # Also publish a zero cmd_vel immediately to wake drone from autopilot state
987              if self._conn:
988                  try: self._ros.cmd_vel(0.0, 0.0, 0.0, 0.0)
989                  except: pass
990              # Generate flight report
991              self._logger.stop()
992              if self._logger.has_data():
993                  report = self._logger.generate_report()
994                  self.log_report.setPlainText(report)
995                  self.btn_save_csv.setEnabled(True)
996                  self.btn_save_report.setEnabled(True)
997                  self.lbl_log_status.setText("Logger: DONE — report ready")
998                  self.lbl_log_status.setStyleSheet("color:#0891b2;font-size:10px;")
999                  self._log("Flight report ready — see FLIGHT REPORT tab")
1000
1001     def _on_gps(self, lat, lon, alt):
1002         self._lat=lat; self._lon=lon; self._alt_gps=alt
1003         self.lbl_gps_pos.setText(f"lat: {lat:.6f}   lon: {lon:.6f}   alt: {alt:.1f}m")
1004
1005     def _on_odom(self, x, y, z):
1006         self._ox=x; self._oy=y; self._oz=z
1007
```

8.1.21 Χειροκίνητος Έλεγχος

```
  Lines 1008 – 1065  |  takeoff  |  land  |  hover  |  emergency  |  setv  |
pub_vel  |  cam  |  key events
```

```
1008      # ── MANUAL CONTROL ──────────────────────────────────────────────
1009      def _takeoff(self):
1010          if not self._conn: self._log("Not connected to ROS"); return
1011          if QMessageBox.question(self,"Takeoff","Send TAKEOFF?",
1012                  QMessageBox.Yes|QMessageBox.No)==QMessageBox.Yes:
1013              self._ros.takeoff()
1014
1015      def _land(self):
1016          if self._conn: self._ros.land()
1017
1018      def _hover(self):
1019          self._lx=self._ly=self._lz=self._az=0.0
1020
1021      def _emergency(self):
1022          if QMessageBox.critical(self,"EMERGENCY","Cut motors NOW?",
1023                  QMessageBox.Yes|QMessageBox.No)==QMessageBox.Yes:
1024              if self._conn: self._ros.emergency()
1025
1026      def _setv(self, lx, ly, lz, az):
1027          s = self._spd
1028          self._lx=lx*s; self._ly=ly*s; self._lz=lz*s; self._az=az*s
1029
1030      def _pub_vel(self):
1031          # Always publish -- the drone's onboard autopilot ignores cmd_vel
1032          # while a flight plan is running, so this never interferes with missions.
1033          # After landing, this immediately restores manual control with zero delay.
1034          if self._conn:
1035              self._ros.cmd_vel(self._lx, self._ly, self._lz, self._az)
1036
1037      def _cam(self, dp, dt):
1038          self._pan  = max(-180, min(180, self._pan +dp))
1039          self._tilt = max(-83,  min(83,  self._tilt+dt))
1040          if self._conn: self._ros.camera(self._pan, self._tilt)
1041
1042      def _cam_abs(self, p, t):
1043          self._pan=p; self._tilt=t
1044          if self._conn: self._ros.camera(p, t)
1045
1046      def keyPressEvent(self, e):
1047          # Allow manual keys even after mission -- autopilot has already stopped
1048          k = e.key()
1049          if   k==Qt.Key_W:     self._setv(1,0,0,0)
1050          elif k==Qt.Key_S:     self._setv(-1,0,0,0)
1051          elif k==Qt.Key_A:     self._setv(0,1,0,0)
1052          elif k==Qt.Key_D:     self._setv(0,-1,0,0)
1053          elif k==Qt.Key_R:     self._setv(0,0,1,0)
1054          elif k==Qt.Key_F:     self._setv(0,0,-1,0)
1055          elif k==Qt.Key_Q:     self._setv(0,0,0,1)
1056          elif k==Qt.Key_E:     self._setv(0,0,0,-1)
1057          elif k==Qt.Key_Space: self._hover()
1058          elif k==Qt.Key_T:     self._takeoff()
1059          elif k==Qt.Key_L:     self._land()
1060
1061      def keyReleaseEvent(self, e):
1062          if e.key() in (Qt.Key_W,Qt.Key_S,Qt.Key_A,Qt.Key_D,
1063                         Qt.Key_R,Qt.Key_F,Qt.Key_Q,Qt.Key_E):
1064              self._hover()
1065
```

8.1.22 Διαχείριση Waypoints

Lines 1066 – 1109 | _add_wp | _del_wp | _clr_wp | _refresh | _tbl_edit

```
1066      # ── WAYPOINTS ──────────────────────────────────────────────────────
1067      def _add_wp(self):
1068          try:
1069              lat, lon = parse_coords(self.txt_coord.text())
1070              alt = self.spin_alt.value()
1071              n = len(self._wps)+1
1072              self._wps.append([lat, lon, alt, f"WP{n:02d}"])
1073              self._refresh(); self.txt_coord.clear()
1074              self._log(f"Added WP{n:02d}: {lat:.6f}, {lon:.6f}  alt={alt:.0f}m")
1075          except ValueError as e:
1076              QMessageBox.warning(self,"Error",str(e))
1077
1078      def _del_wp(self):
1079          rows = sorted({i.row() for i in self.wt.selectedItems()}, reverse=True)
1080          for r in rows:
1081              if r < len(self._wps): self._wps.pop(r)
1082          self._refresh()
1083
1084      def _clr_wp(self):
1085          if self._wps and QMessageBox.question(
1086                  self,"Clear","Remove
all?",QMessageBox.Yes|QMessageBox.No)==QMessageBox.Yes:
1087              self._wps.clear(); self._refresh()
1088
1089      def _refresh(self):
1090          self.wt.blockSignals(True); self.wt.setRowCount(0)
1091          for i, wp in enumerate(self._wps):
1092              self.wt.insertRow(i)
1093              self.wt.setItem(i,0, QTableWidgetItem(wp[3]))
1094              self.wt.setItem(i,1, QTableWidgetItem(f"{wp[0]:.7f}"))
1095              self.wt.setItem(i,2, QTableWidgetItem(f"{wp[1]:.7f}"))
1096              self.wt.setItem(i,3, QTableWidgetItem(f"{wp[2]:.1f}"))
1097          self.wt.blockSignals(False)
1098
1099      def _tbl_edit(self, item):
1100          r = item.row()
1101          if r >= len(self._wps): return
1102          try:
1103              c = item.column(); v = item.text()
1104              if c==0: self._wps[r][3]=v
1105              elif c==1: self._wps[r][0]=float(v)
1106              elif c==2: self._wps[r][1]=float(v)
1107              elif c==3: self._wps[r][2]=float(v)
1108          except: pass
1109
```

8.1.23 Ανάλυση Αποστολής (_analyse)

Lines 1110 – 1179 | Distance | Time | Battery | Go/No-Go

```
1110      # ── ANALYSIS ──────────────────────────────────────────────────────
1111      def _analyse(self):
1112          try:
1113              wps = [(w[0], w[1], w[2]) for w in self._wps]
1114              if not wps and self._plan:
1115                  rows = parse_mavlink(self._plan)
1116                  wps = [(r['lat'], r['lon'], r['alt']) for r in rows
1117                         if r['cmd'] == 16 and not (r['lat'] == 0 and r['lon'] == 0)]
1118              if not wps:
1119                  QMessageBox.information(self, "No Waypoints",
1120                      "Add waypoints first or load a plan."); return
1121
1122              a = analyse(wps, speed=self.spin_spd.value(), battery=self._battery)
1123              if not a: return
1124
1125              t = a['time']; m, s = int(t // 60), int(t % 60)
1126
1127              # Distance
1128              self.v_dist.setText(f"{a['dist']:,.0f} m  ({a['dist']/1000:.2f} km)")
1129
1130              # Time -- warn if over limit
1131              time_col = "#dc2626" if t/60 > MAX_MIN else "#1a1a2e"
1132              self.v_time.setText(f"{m} min {s:02d} sec")
1133              self.v_time.setStyleSheet(f"color:{time_col};font-weight:bold;")
1134
1135              # Battery need
1136              self.v_need.setText(f"{a['need']:.1f}%  (incl. {RESERVE}% reserve)")
1137
1138              # Battery have
1139              if self._battery is not None:
1140                  self.v_have.setText(f"{self._battery:.0f}%")
1141              else:
1142                  self.v_have.setText("-- (not connected)")
1143
1144              # Margin
1145              if a['margin'] is not None:
1146                  col  = "#16a34a" if a['ok'] else "#dc2626"
1147                  sign = "+" if a['margin'] >= 0 else ""
1148                  self.v_marg.setText(f"{sign}{a['margin']:.1f}%")
1149                  self.v_marg.setStyleSheet(f"color:{col};font-weight:bold;font-
size:10px;")
1150              else:
1151                  self.v marg.setText("-- (not connected)")
1152                  self.v_marg.setStyleSheet("color:#888ea8;font-size:10px;")
1153
1154              # Go / No-Go
1155              issues = []
1156              if t / 60 > MAX_MIN:
1157                  issues.append(f"Flight time {m}m{s:02d}s exceeds {MAX MIN} min battery")
1158              if a['dist'] > 300:
1159                  issues.append(f"Distance {a['dist']:.0f}m may exceed WiFi range (300m)")
1160              if self._battery is not None and not a['ok']:
1161                  issues.append(f"Battery {self._battery:.0f}% insufficient (need
{a['need']:.0f}%)")
1162
1163
1164              if issues:
1165                  txt = "\u2717  DO NOT FLY\n" + "\n".join(f"  \u2022 {x}" for x in
issues)
1166                  self.lbl_gng.setText(txt)
1167                  self.lbl_gng.setStyleSheet("background:#fef2f2;color:#dc2626;font-
weight:bold;font-size:10px;border:1px solid #dc2626;border-radius:4px;padding:4px;")
1168              else:
1169                  self.lbl_gng.setText("\u2713  SAFE TO FLY")
1170                  self.lbl_gng.setStyleSheet("background:#f0fdf4;color:#16a34a;font-
weight:bold;font-size:13px;border:1px solid #16a34a;border-radius:4px;padding:4px;")
1171
1172              self._log(f"Analysis: {a['dist']:.0f}m  {m}m{s:02d}s  "
1173                        f"need {a['need']:.0f}%  "
1174                        f"{'GO' if not issues else 'NO-GO: '+issues[0]}")
1175
```

```
1176          except Exception as ex:
1177              self._log(f"Analyse error: {ex}")
1178              import traceback; traceback.print_exc()
1179
```

8.1.24 FTP Λειτουργίες

Lines 1180 – 1234 | _load_drone | _load_file | _save_file | _ftp_done | _load_plan

```
1180      # ── FTP ──────────────────────────────────────────────────────────────
1181      def _load_drone(self):
1182          self.btn_load_drone.setEnabled(False); self.btn_load_drone.setText("Loading...")
1183          self._log(f"Connecting to drone ({self.txt_ip.text()})...")
1184          self._ftp.download(self.txt_ip.text().strip())
1185
1186      def _load_file(self):
1187          path,_ = QFileDialog.getOpenFileName(self,"Open","","MAVLink (*.mavlink);;All
(*.*)")
1188          if path:
1189              with open(path) as f: content = f.read()
1190              self._load_plan(content, os.path.basename(path))
1191
1192      def _save_file(self):
1193          if not self._wps and not self._plan:
1194              QMessageBox.information(self,"Nothing","Add waypoints first."); return
1195          content = gen_mavlink([(w[0],w[1],w[2],w[3]) for w in self._wps],
1196                                speed=self.spin_spd.value()) if self._wps else self._plan
1197          path,_ = QFileDialog.getSaveFileName(self,"Save","flightplan.mavlink","MAVLink
(*.mavlink)")
1198          if path:
1199              with open(path,'w') as f: f.write(content)
1200              self._log(f"Saved: {path}")
1201
1202      def _ftp_done(self, ok, op, data):
1203          if op == "download":
1204              self.btn_load_drone.setEnabled(True); self.btn_load_drone.setText("Load from
Drone")
1205              if ok: self._load_plan(data, "drone")
1206              else:
1207                  self._log(f"Load failed: {data}")
1208                  QMessageBox.critical(self,"Failed",data)
1209          elif op == "upload":
1210              if ok:
1211                  self._log(data)
1212                  self._ros.af_start(PLAN_NAME)
1213                  self._flying=True; self._paused=False; self._t0=time.time()
1214                  self._ms("AUTOPILOT RUNNING","#4ade80"); self.btn_uf.setEnabled(False)
1215                  self._log("Onboard autopilot started — drone navigating autonomously")
1216                  ts=time.strftime('%Y%m%d_%H%M%S')
1217                  self._logger.start(self._wps, mission_name=f"mission_{ts}")
1218                  self.lbl_log_status.setText(f"Logger: RECORDING [{ts}]")
1219                  self.lbl_log_status.setStyleSheet("color:#4ade80;font-size:10px;")
1220                  self._log("Mission logger started")
1221              else:
1222                  self._ms("UPLOAD FAILED","#ef4444"); self.btn_uf.setEnabled(True)
1223                  QMessageBox.critical(self,"Upload Failed",
1224                      f"{data}\n\nMake sure you are on Bebop WiFi.")
1225
1226      def _load_plan(self, content, src):
1227          self._plan = content
1228          rows = parse_mavlink(content)
1229          navs = [r for r in rows if r['cmd']==16 and not(r['lat']==0 and r['lon']==0)]
1230          self._wps = [[r['lat'],r['lon'],r['alt'],f"WP{i+1:02d}"] for i,r in
enumerate(navs)]
1231          self._refresh()
1232          self._log(f"Loaded {len(self._wps)} waypoints from {src}")
1233          self._analyse()
1234
```

8.1.25 Έλεγχος Αποστολής

Lines 1235 – 1292 | _upload_fly | _fly_existing | _pause_resume | _stop | _save_csv | _save_report

```
1235      # ── MISSION CONTROL ──────────────────────────────────────────────────
1236      def _upload_fly(self):
1237          if not self._wps:
1238              QMessageBox.warning(self,"No Waypoints","Add waypoints first."); return
1239          if not self._conn:
1240              QMessageBox.critical(self,"Not Connected","ROS not connected."); return
1241          content = gen_mavlink([(w[0],w[1],w[2],w[3]) for w in self._wps],
1242                                speed=self.spin_spd.value())
1243          self._plan = content
1244          self._ms("UPLOADING...","#f59e0b"); self.btn_uf.setEnabled(False)
1245          self._log("Uploading MAVLink plan to drone...")
1246          self._ftp.upload(content, self.txt_ip.text().strip())
1247
1248      def _fly_existing(self):
1249          if not self._conn:
1250              QMessageBox.critical(self,"Not Connected","ROS not connected."); return
1251          if QMessageBox.question(self,"Fly Existing Plan",
1252                  "Trigger the plan already on the drone?\nMake sure area is clear.",
1253                  QMessageBox.Yes|QMessageBox.No) != QMessageBox.Yes: return
1254          self._ros.af_start("")
1255          self._flying=True; self._paused=False; self._t0=time.time()
1256          self._ms("AUTOPILOT RUNNING","#4ade80"); self.btn_uf.setEnabled(False)
1257          self._log("Triggered existing plan on drone")
1258          ts=time.strftime('%Y%m%d_%H%M%S')
1259          self._logger.start(self._wps, mission_name=f"existing_{ts}")
1260          self.lbl_log_status.setText(f"Logger: RECORDING [{ts}]")
1261          self.lbl_log_status.setStyleSheet("color:#4ade80;font-size:10px;")
1262
1263      def _pause_resume(self):
1264          if not self._flying: return
1265          if not self._paused:
1266              self._ros.af_pause(); self._paused=True
1267              self.btn_pau.setText("RESUME"); self._ms("PAUSED","#f59e0b")
1268          else:
1269              self._ros.af_start(PLAN_NAME); self._paused=False
1270              self.btn_pau.setText("PAUSE"); self._ms("AUTOPILOT RUNNING","#4ade80")
1271
1272      def _stop(self):
1273          self._ros.af_stop()
1274          self._flying=False; self._paused=False; self._t0=None
1275          self._hover()  # zero velocity so drone hovers
1276          self._ms("STOPPED","#64748b")
1277          self._log("Mission stopped — manual control restored")
1278          self.btn_uf.setEnabled(True)
1279          self._ms("STOPPED","#64748b"); self.btn_uf.setEnabled(True);
self.btn_pau.setText("PAUSE")
1280
1281      def _save_csv(self):
1282          path, _ = QFileDialog.getSaveFileName(self, "Save CSV",
1283              f"flight_{time.strftime('%Y%m%d_%H%M%S')}.csv", "CSV (*.csv)")
1284          if path: self._logger.save_csv(path); self._log(f"CSV saved: {path}")
1285
1286      def _save_report(self):
1287          path, _ = QFileDialog.getSaveFileName(self, "Save Report",
1288              f"report_{time.strftime('%Y%m%d_%H%M%S')}.txt", "Text (*.txt)")
1289          if path:
1290              with open(path,'w') as f: f.write(self.log_report.toPlainText())
1291              self._log(f"Report saved: {path}")
1292
```

8.1.26 Reset, Close, _ms, _tick, _log

Lines 1293 – 1371 | _reset_state | _confirm_close | _ms | _tick | _log

```
1293      def _reset_state(self):
1294          """
1295          Force-reset all mission and drone state flags.
1296
1297          Called when the drone is not responding after landing.
1298          This does NOT restart ROS or reconnect -- it just clears the
1299          internal GUI state flags that may have got stuck, and sends
1300          a fresh stop + hover command so the drone is ready for new commands.
1301          """
1302          # Clear all mission flags
1303          self._flying  = False
1304          self._paused  = False
1305          self._t0      = None
1306
1307          # Zero all velocity so drone hovers if airborne
1308          self._hover()
1309
1310          # Send explicit stop to autopilot in case it is still running
1311          if self._conn:
1312              try:
1313                  self._ros.af_stop()
1314              except Exception:
1315                  pass
1316
1317          # Send a zero cmd_vel immediately -- wakes drone up from frozen state
1318          if self._conn:
1319              try:
1320                  self._ros.cmd_vel(0.0, 0.0, 0.0, 0.0)
1321              except Exception:
1322                  pass
1323
1324          # Reset UI
1325          self._ms("READY", "#16a34a")
1326          self.btn_uf.setEnabled(True)
1327          self.btn_pau.setText("PAUSE")
1328          self.lbl_time.setText("Time: --:--")
1329
1330          # Reset logger for next mission
1331          self._logger.reset()
1332          self.lbl_log_status.setText("Logger: idle")
1333          self.lbl_log_status.setStyleSheet("color:#888ea8;font-size:10px;")
1334
1335          # Re-enable save buttons only if report already generated
1336          has_report = bool(self.log_report.toPlainText().strip())
1337          self.btn_save_csv.setEnabled(has_report)
1338          self.btn_save_report.setEnabled(has_report)
1339
1340          self._log("--- State reset --- GUI ready for next mission ---")
1341          self._log("Manual control active: W/A/S/D, direction pad, TAKE OFF/LAND")
1342
1343      def _confirm_close(self):
1344          msg = ('A mission is running.\nStop mission and close?'
1345                 if self._flying else 'Close the application?')
1346          if QMessageBox.question(self, 'Close', msg,
1347                  QMessageBox.Yes | QMessageBox.No,
1348                  QMessageBox.No) == QMessageBox.Yes:
1349              self.close()
1350
1351      def _ms(self, t, c):
1352          self.lbl_ms.setText(t)
1353          self.lbl_ms.setStyleSheet(f"color:{c};font-weight:bold;font-size:12px;")
1354
1355      # ── TICK ──────────────────────────────────────────────────────────────
1356      def _tick(self):
1357          if self._t0 and self._flying:
1358              e = int(time.time()-self._t0); self.lbl_time.setText(f"Time:
{e//60:02d}:{e%60:02d}")
1359              if self._battery and self._battery < 15:
1360                  self._log("WARNING: Battery critical (<15%)!")
1361              self._logger.record(
1362                  lat=self._lat, lon=self._lon, alt_gps=self._alt_gps,
```

```
                battery=self._battery, state_int=self._state,
                odom_x=self._ox, odom_y=self._oy, odom_z=self._oz)
            n = len(self._logger._records)
            if n > 0 and n % 10 == 0:
                self.lbl_log_status.setText(f"Logger: RECORDING  {n} samples")

    def _log(self, m):
        self.log.append(m); self.statusBar().showMessage(m)

```

8.1.27 CSS Stylesheet (_css)

Lines 1372 – 1563 | Complete QSS light grey theme

```
    # ── CSS ──────────────────────────────────────────────────────────────
    def _css(self):
        """
        Light grey professional theme.

        Colour palette:
          #f0f2f5  window / widget background   (very light grey)
          #ffffff  card / panel background       (white)
          #e2e6ea  input / table background      (soft grey)
          #c8cdd4  borders                       (medium grey)
          #1a1a2e  primary text                  (near black)
          #555e6b  secondary text / labels       (dark grey)
          #1976d2  accent / section headers      (blue)
          #16a34a  safe / takeoff                (green)
          #dc2626  danger / emergency / stop     (red)
          #7c3aed  secondary actions             (purple)
          #d97706  warning / paused              (amber)
          #0891b2  hover / land                  (teal)
        """
        self.setStyleSheet("""
        QMainWindow,QWidget{
            background:#f0f2f5;
            color:#1a1a2e;
            font-family:'Segoe UI',Arial,sans-serif;
            font-size:11px;}

        QGroupBox#card{
            border:1px solid #c8cdd4;
            border-radius:6px;
            background:#ffffff;
            color:#1976d2;
            font-weight:bold;
            font-size:9px;
            letter-spacing:1px;
            margin-top:6px;}
        QGroupBox#card::title{
            subcontrol-origin:margin;
            left:8px;
            top:-1px;
            padding:0 4px;
            color:#1976d2;
            background:#ffffff;}

        QLabel{color:#555e6b;}
        QLabel#state_lbl{
            font-size:13px;font-weight:bold;letter-spacing:1px;
            padding:4px;border:1px solid #c8cdd4;
            border-radius:4px;background:#f8f9fa;}
        QLabel#cam{
            background:#1a1a2e;border:2px solid #c8cdd4;
            color:#888ea8;font-size:12px;border-radius:4px;}
        QLabel#ros_status{color:#d97706;font-size:10px;font-weight:bold;}
        QLabel#small{color:#888ea8;font-size:10px;}
        QLabel#sec{
            color:#1976d2;font-size:10px;
            font-weight:bold;letter-spacing:1px;}
        QLabel#sk{color:#888ea8;font-size:10px;}
        QLabel#sv{color:#1a1a2e;font-size:10px;font-weight:bold;}
        QLabel#ms{color:#d97706;font-weight:bold;font-size:12px;}
        QLabel#gng{
            color:#888ea8;border:1px solid #c8cdd4;
            border-radius:4px;padding:4px;background:#f8f9fa;}

        QPushButton{
            background:#ffffff;
            color:#374151;
            border:1px solid #c8cdd4;
            border-radius:5px;
            padding:5px 8px;
            font-family:'Segoe UI',Arial,sans-serif;
            font-size:10px;}
```

```
        QPushButton:hover{
            background:#e8f0fe;
            color:#1976d2;
            border-color:#1976d2;}
        QPushButton:pressed{background:#d0e4fc;}
        QPushButton#dp{
            font-size:9px;
            color:#374151;
            background:#f8f9fa;}
        QPushButton#dp:hover{background:#e8f0fe;color:#1976d2;}

        QPushButton#btn_tkf{
            color:#ffffff;background:#16a34a;
            border-color:#15803d;font-weight:bold;}
        QPushButton#btn_tkf:hover{background:#15803d;}
        QPushButton#btn_tkf:pressed{background:#166534;}

        QPushButton#btn_lnd{
            color:#ffffff;background:#0891b2;
            border-color:#0e7490;}
        QPushButton#btn_lnd:hover{background:#0e7490;}

        QPushButton#btn_hov{
            color:#ffffff;background:#7c3aed;
            border-color:#6d28d9;}
        QPushButton#btn_hov:hover{background:#6d28d9;}

        QPushButton#btn_emg{
            color:#ffffff;background:#dc2626;
            border-color:#b91c1c;font-weight:bold;}
        QPushButton#btn_emg:hover{background:#b91c1c;}
        QPushButton#btn_emg:pressed{background:#991b1b;}

        QPushButton#btn_ff{
            color:#7c3aed;border-color:#7c3aed;background:#f5f3ff;}
        QPushButton#btn_ff:hover{background:#ede9fe;}

        QPushButton#btn_an{
            color:#0891b2;border-color:#0891b2;background:#ecfeff;}
        QPushButton#btn_an:hover{background:#cffafe;}

        QPushButton#btn_go{
            color:#ffffff;background:#16a34a;
            border-color:#15803d;font-weight:bold;}
        QPushButton#btn_go:hover{background:#15803d;}

        QPushButton#btn_fe{
            color:#7c3aed;border-color:#7c3aed;background:#f5f3ff;}
        QPushButton#btn_fe:hover{background:#ede9fe;}

        QPushButton#btn_st{
            color:#dc2626;border-color:#dc2626;background:#fef2f2;}
        QPushButton#btn_st:hover{background:#fee2e2;}

        QProgressBar{
            background:#e2e6ea;
            border:1px solid #c8cdd4;
            border-radius:3px;height:14px;
            text-align:center;}
        QProgressBar::chunk{background:#16a34a;border-radius:2px;}

        QSlider::groove:horizontal{
            background:#d1d5db;height:4px;border-radius:2px;}
        QSlider::handle:horizontal{
            background:#1976d2;width:16px;height:16px;
            margin:-6px 0;border-radius:8px;border:2px solid #ffffff;}

        QDoubleSpinBox,QLineEdit{
            background:#ffffff;color:#1a1a2e;
            border:1px solid #c8cdd4;
            border-radius:4px;padding:3px;}
        QDoubleSpinBox:focus,QLineEdit:focus{border-color:#1976d2;}

        QTextEdit#log{
            background:#f8f9fa;color:#555e6b;
            border:1px solid #c8cdd4;font-size:9px;
            font-family:'Courier New',monospace;}
```

```
1520
1521            QTableWidget#wt{
1522                background:#ffffff;color:#1a1a2e;
1523                border:1px solid #c8cdd4;
1524                gridline-color:#e2e6ea;
1525                selection-background-color:#dbeafe;}
1526            QHeaderView::section{
1527                background:#f0f2f5;color:#1976d2;
1528                border:1px solid #c8cdd4;
1529                padding:4px;font-size:9px;font-weight:bold;}
1530
1531            QStatusBar{background:#f0f2f5;color:#888ea8;font-size:9px;}
1532            QSplitter::handle{background:#c8cdd4;width:2px;}
1533
1534            QTabWidget#tabs::pane{
1535                border:1px solid #c8cdd4;background:#ffffff;}
1536            QTabBar::tab{
1537                background:#f0f2f5;color:#888ea8;
1538                border:1px solid #c8cdd4;
1539                padding:4px 12px;font-size:9px;
1540                border-bottom:none;border-radius:4px 4px 0 0;}
1541            QTabBar::tab:selected{
1542                background:#ffffff;color:#1976d2;
1543                border-bottom:2px solid #1976d2;font-weight:bold;}
1544            QTabBar::tab:hover{background:#e8f0fe;color:#1976d2;}
1545
1546            QScrollBar:vertical{
1547                background:#f0f2f5;width:10px;border-radius:5px;}
1548            QScrollBar::handle:vertical{
1549                background:#c8cdd4;border-radius:5px;min-height:20px;}
1550            QScrollBar::handle:vertical:hover{background:#1976d2;}
1551            QPushButton#btn_reset{
1552                color:#ffffff;background:#d97706;
1553                border:1px solid #b45309;font-size:10px;font-weight:bold;}
1554            QPushButton#btn_reset:hover{background:#b45309;}
1555            QPushButton#btn_reset:pressed{background:#92400e;}
1556            QPushButton#btn_close{
1557                color:#6b7280;border:1px solid #d1d5db;
1558                background:#f9fafb;font-size:10px;}
1559            QPushButton#btn_close:hover{
1560                color:#dc2626;border-color:#dc2626;background:#fef2f2;}
1561            QPushButton#btn_close:pressed{background:#fee2e2;}
1562            """)
1563
```

8.1.28 closeEvent

Lines 1564 – 1596 | Instant close | Thread cleanup | ROS shutdown

```
1564      def closeEvent(self, e):
1565          # Accept immediately so the window disappears at once
1566          e.accept()
1567
1568          # Stop timers -- no more cmd_vel or tick after this
1569          self._vel_t.stop()
1570          self._tick_t.stop()
1571
1572          # Send stop to drone if mid-mission (fire-and-forget)
1573          if self._flying:
1574              try: self._ros.af_stop()
1575              except: pass
1576
1577          # Shut down ROS -- this also breaks rospy.spin() in RosWorker
1578          if ROS_OK:
1579              try:
1580                  if rospy.core.is_initialized():
1581                      rospy.signal_shutdown("GUI closed")
1582              except: pass
1583
1584          # Tell all threads to stop -- don't wait, just signal
1585          self._rt.quit()
1586          self._ft.quit()
1587          self._vt.quit()
1588
1589          # Terminate threads that haven't finished within 300ms
1590          # (300ms total, not 1000ms per thread = 3x faster)
1591          self._rt.wait(300); self._ft.wait(300); self._vt.wait(300)
1592          if self._rt.isRunning(): self._rt.terminate()
1593          if self._ft.isRunning(): self._ft.terminate()
1594          if self._vt.isRunning(): self._vt.terminate()
1595
1596
```

8.1.29 main()

Lines 1597 – 1602 | Application entry point

```
1597  def main():
1598      app = QApplication(sys.argv); app.setStyle("Fusion")
1599      win = App(); win.show(); sys.exit(app.exec_())
1600
1601  if   name   == "  main  ":
1602      main()
```

Πηγές και σύνδεσμοι:

1) Shuttleworth, Mark. "Ubuntu Philosophy." Canonical Ltd., https://ubuntu.com/community/philosophy.
2) Hill, Benjamin Mako. The Official Ubuntu Book. 9th ed., Prentice Hall, 2016.
3) Helmke, Matthew. Ubuntu Linux Unleashed 2021 Edition. Sams Publishing, 2020.
4) Ubuntu Official Documentation: https://help.ubuntu.com/
5) Ubuntu Community Support: https://ubuntu.com/support/community-support
6) Linux Foundation. "About Linux." https://www.linuxfoundation.org/about-linux/
7) Torvalds, Linus. "Just for Fun: The Story of an Accidental Revolutionary." HarperBusiness, 2001.
8) Quigley, Amber Graner, Kyle Rankin, and Benjamin Mako Hill. Ubuntu Server Administration. 3rd ed., Prentice Hall, 2013.
9) Sobell, Mark G. A Practical Guide to Ubuntu Linux. 4th ed., Prentice Hall, 2015.
10) Kerrisk, Michael. The Linux Programming Interface: A Linux and UNIX System Programming Handbook. No Starch Press, 2010.
11) Ubuntu Wiki: https://wiki.ubuntu.com/
12) Ubuntu Forums: https://ubuntuforums.org/
13) Canonical Ltd.: https://canonical.com/
14) Debian Project (basis for Ubuntu): https://www.debian.org/
15) ROS (Robot Operating System): https://www.ros.org/
16) Parrot Bebop 2 Official Page: https://www.parrot.com/us/drones/parrot-bebop-2
17) Drone Industry Insights. "UAV Market Overview and Trends": https://droneii.com/
18) Valavanis, Kimon P., and George J. Vachtsevanos. Handbook of Unmanned Aerial Vehicles. Springer, 2015.
19) FAA – Federal Aviation Administration: "Unmanned Aircraft Systems (UAS)": https://www.faa.gov/uas
20) Bebop-Autonomy GitHub Repository: https://github.com/AutonomyLab/bebop_autonomy
21) UAV Coach – Guide and resources for drone pilots: https://uavcoach.com/
22) European Union Aviation Safety Agency (EASA) – Drone Regulations: https://www.easa.europa.eu/domains/civil-drones
23) Bebop autonomy driver walkthrough [Online]. Available: https://bebop-autonomy.readthedocs.io/en/latest/
24) Ros melodic installation guide [Online]. Available: https://wiki.ros.org/melodic/Installation/Ubuntu
25) Κανάλι στο You Tube που διεξήγαγε πειράματα με το Bebop Parrot 2 https://www.youtube.com/@roboticaposgrado
26) Μάθημα στο Udemy από τον Antonio Brandi των PAL Robotics: Self Driving and ROS - Learn by Doing! Odometry & Control Create a Self-Driving robot and learn about Robot Localization and Sensor Fusion using Kalman Filters
27) Build a Drone: A Step-by-Step Guide to Designing, Constructing, and Flying Your Very Own Drone. by Barry Davies

28) J. L. Spudich and B. H. Satir, *Sensory Receptors and Signal Transduction*. New York: Wiley-Liss, 2001
29) McGovern, A. pyparrot: Python interface for Parrot Drones. University of Oklahoma. https://pyparrot.readthedocs.io (2018).
30) Newaz, R. bebop2_controller: ROS package for controlling Parrot Bebop 2 in indoor environments https://github.com/RedwanNewaz/bebop2_controller . (2019).
31) Dronecode Foundation. QGroundControl Ground Control Station.https://qgroundcontrol.com (2014–2024).
32) Parrot SA. Open Source Development Tools.https://developer.parrot.com (2022).