

Ατομική Διπλωματική Εργασία

**ΑΥΤΟΝΟΜΗ ΕΞΕΡΕΥΝΗΣΗ ΕΣΩΤΕΡΙΚΩΝ ΧΩΡΩΝ ΜΕ
ΣΜΗΝΗ ΑΠΟ ΜΗ ΕΠΑΝΔΡΩΜΕΝΑ ΑΕΡΟΧΗΜΑΤΑ**

Πελοπίδα Αντρέας

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΥΤΟΝΟΜΗ ΕΞΕΡΕΥΝΗΣΗ ΕΣΩΤΕΡΙΚΩΝ ΧΩΡΩΝ ΜΕ ΣΜΗΝΗ ΑΠΟ ΜΗ ΕΠΑΝΔΡΩΜΕΝΑ ΑΕΡΟΧΗΜΑΤΑ

Πελοπίδα Αντρέας

Επιβλέπων Καθηγητής

Δρ Γεωργίου Χρύσης

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των
απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του
Πανεπιστημίου Κύπρου

Μάιος 2025

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέπων καθηγητή μου Δρ. Χρύση Γεωργίου, ο οποίος μου έδωσε την ευκαιρία να ασχοληθώ με ένα πολύ ενδιαφέρον θέμα στα πλαίσια της διπλωματικής εργασίας.

Ιδιαίτερες ευχαριστίες και στον Δρ. Χάρη Ζαχαράτο για την παροχή του εξοπλισμού αλλά και το ενδιαφέρον που έδειξε για αυτή την διπλωματική εργασία.

Επίσης θα ήθελα να ευχαριστήσω την οικογένειά μου για όλη την οικονομική, ψυχολογική και συναισθηματική υποστήριξη καθ' όλη τη διάρκεια των σπουδών μου.

Περίληψη

Σε αυτή την Διπλωματική Εργασία μελετήθηκε η χρήση και ο προγραμματισμός ενός μικρού Μη Επανδρωμένου Αεροχήματος (MEA), του Crazyflie, με στόχο την αυτόματη πλοήγηση και τη χαρτογράφηση εσωτερικών χώρων. Διερευνήθηκαν οι βασικές λειτουργίες του Crazyflie σε κώδικα Python και C, όπως επίσης και οι δυνατότητες επικοινωνίας μεταξύ πολλαπλών Crazyflie, με σκοπό τη συνεργατική χαρτογράφηση και βελτιστοποίηση της κάλυψης ενός χώρου.

Μέσω κατάλληλου λογισμικού, το MEA καταφέρνει να ακολουθήσει μια τροχιά, ακολουθώντας τοίχους και χρησιμοποιώντας αισθητήρες απόστασης, προσφέροντας μια πρώτη και απλή χαρτογράφηση του εσωτερικού χώρου.

Έπειτα, η έρευνα επεκτάθηκε στη χρήση πολλαπλών Crazyflies, με σκοπό τη συνεργατική χαρτογράφηση. Τα MEA επικοινωνούν μεταξύ τους χρησιμοποιώντας πρωτόκολλο P2P, έτσι ώστε να αποφεύγονται οι συγκρούσεις.

Οι αλγόριθμοι πλοήγησης και ανταλλαγής δεδομένων δοκιμάστηκαν τόσο σε *κεντροποιημένα* (centralized) όσο και σε *αποκεντροποιημένα* (de-centralized) συστήματα. Στην πρώτη περίπτωση, ένα κεντρικό σύστημα ελέγχου (ο υπολογιστής) αναλαμβάνει τη λήψη αποφάσεων και τη διανομή εντολών στα Crazyflies, ενώ στη δεύτερη, κάθε MEA βασίζεται σε τοπικές πληροφορίες και στην επικοινωνία με τα υπόλοιπα MEA έτσι ώστε να επιτυγχάνεται η αυτόνομη πλοήγηση.

Περιεχόμενα

Κεφάλαιο 1	Εισαγωγή.....	6
	1.1 Μη Επανδρωμένα Αεροχήματα	6
	1.2 Σμήνη από ΜΕΑ	7
	1.3 Ορισμός του Προβλήματος	8
	1.4 Στόχος Εργασίας	9
	1.5 Μεθοδολογία	9
	1.5.1 Αρχικά Σταδια	9
	1.5.2 Περιβάλλον Ανάπτυξης	10
	1.5.3 Υλοποίηση Αλγορίθμων και Πειράματα	10
	1.6 Δομή της Εργασίας	11
Κεφάλαιο 2	Το ΜΕΑ Crazyflie	13
	2.1 Επιλογή Crazyflie	13
	2.2 Χαρακτηριστικά Crazyflie 2.1	14
	2.3 Μονάδες Επέκτασης	15
	2.4 Πελάτες	19
	2.5 Crazyradio 2.0 USB	20
	2.6 Αρχική Διάταξη Crazyflie	21
	2.7 Βιβλιοθήκη Crazyflie Python	26
Κεφάλαιο 3	Υλοποίηση Αλγορίθμων.....	29
	3.1 Έλεγχος Πτήσης: Roll, Pitch, Yaw και Thrust	29
	3.2 Εξερεύνηση Δωματίου με ένα Crazyflie	31
	3.3 Επίπεδο Εφαρμογής	37
	3.4 Εγγραφή του Λογισμικού Πάνω στο Crazyflie	41
	3.5 Σμήνος Crazyflie - Η Ιδέα	42
	3.6 Διεπαφή P2P	46
	3.7 Δοκιμή P2P	46
	3.8 Εξερεύνηση Δωματίου με Σμήνος Crazyflie	50

Κεφάλαιο 4	Πειραματική Εφαρμογή	56
	4.1 Ετοιμασία Χώρου	56
	4.2 Προκλήσεις	58
	4.3 Αποτελέσματα Wall Following Ενός Crazyflie	60
	4.4 Αποτελέσματα Σμήνους	63
Κεφάλαιο 5	Συμπεράσματα και Μελλοντικές Εργασίες.....	67
	5.1 Συμπεράσματα	67
	5.2 Βελτιώσεις	68
	5.3 Μελλοντικές Εργασίες	68
Βιβλιογραφία		70

Κεφάλαιο 1

Εισαγωγή

1.1 Μη Επανδρωμένα Αεροχήματα	6
1.2 Σμήνη από ΜΕΑ	7
1.3 Ορισμός του Προβλήματος	8
1.4 Στόχος Εργασίας	9
1.5 Μεθοδολογία	9
1.6 Δομή της Εργασίας	11

1.1 Μη Επανδρωμένα Αεροχήματα

Τα *Μη Επανδρωμένα Αεροχήματα - ΜΕΑ* (Unmanned Aerial Vehicle - UAV), ειδικά τα τετρακόπτερα ελικόπτερα, έχουν σημειώσει αξιοσημείωτη άνοδο τα τελευταία χρόνια. Η ικανότητά τους για κατακόρυφη απογείωση και προσγείωση και ο σχετικά απλός μηχανικός σχεδιασμός τους τα κάνει ελκυστικά για ένα ευρύ φάσμα εφαρμογών στη ρομποτική.

Υπάρχει μεγάλη ζήτηση για οικονομικά και μικρών διαστάσεων τετρακόπτερα ελικόπτερα εσωτερικών χώρων για ερευνητικά έργα και φοιτητικές εργασίες, ιδίως σε εργαστήρια ή χώρους όπου ο διαθέσιμος χώρος είναι περιορισμένος. Επίσης τα ΜΕΑ εσωτερικών χώρων είναι ελκυστικά για τον λόγο ότι σε αρκετές χώρες, οι υπαίθριες πτήσεις ΜΕΑ πολλές φορές απαιτούν ειδικές άδειες και να πληρούν αυστηρά κριτήρια ασφαλείας ώστε να μπορούν να επιχειρούν με ασφάλεια κοντά σε ανθρώπους, ενώ για ΜΕΑ εσωτερικών χώρων δεν απαιτείται κάποια άδεια.

Σε αυτή την εργασία, χρησιμοποιήθηκε το Crazyflie 2.1 από την Bitcraze, το οποίο, χάρη στον ανοικτό πηγαίο κώδικα του και το προσιτό κόστος του, προσφέρει μια

ιδανική βάση για πειράματα αυτόνομης πλοήγησης σε εσωτερικούς χώρους. Οι πολύ μικρές διαστάσεις του, επιτρέπουν τη διείσδυση σε στενούς χώρους, ενώ το χαμηλό τους βάρος τα καθιστά ασφαλή για λειτουργία κοντά σε ανθρώπους ή εύθραυστες εγκαταστάσεις.

1.2 Σμήνη από MEA

Ένα σμήνος από MEA, είναι πολλά MEA τα οποία συνεργάζονται μαζί παράλληλα για να λύσουν ένα πρόβλημα [18]. Αντί να βασιζόμαστε σε ένα μεμονωμένο MEA, αξιοποιούμε πολλά MEA που συνεργάζονται, εμπνευσμένα από παραδείγματα της φύσης όπως τα σμήνη πουλιών και τα μυρμήγκια και παρά τους περιορισμούς που έχει κάθε μεμονωμένο MEA σε υπολογιστική ισχύ και αισθητήρες, η συλλογική τους συμπεριφορά μπορεί να επιλύει σύνθετα προβλήματα πιο γρήγορα και με μεγαλύτερη αντοχή σε αστοχίες, ακριβώς όπως συμβαίνει στη φύση με τα μυρμήγκια ή τις μέλισσες.

Τα σμήνη από MEA συνδυάζουν αλγόριθμους με τοπικούς αισθητήρες και δυνατότητες ασύρματης επικοινωνίας, έτσι ώστε πολλαπλά MEA να συνεργάζονται και να επιτυγχάνουν έναν κοινό στόχο και αυτό αποτελεί και τη βασική φιλοσοφία: κάθε MEA συμπεριφέρεται ανάλογα με τις τοπικές μετρήσεις των αισθητήρων του και όταν χρειάζεται, ανάλογα με την (περιορισμένη) επικοινωνία με γειτονικά MEA. Η πιθανή απώλεια κάποιων MEA δεν ακυρώνει τη συνολική λειτουργία, καθώς τα υπόλοιπα MEA αναπροσαρμόζονται και ανακατανέμουν το φόρτο εργασίας. Επιπλέον, η παράλληλη λειτουργία πολλών MEA οδηγεί σε ταχύτερη κάλυψη περιοχής. Αντί να πραγματοποιείται η εξερεύνηση σειριακά από ένα μόνο MEA, το σμήνος διασπείρεται ταυτόχρονα, επιταχύνοντας μια πιο αποτελεσματική κάλυψη του χώρου και συλλογή δεδομένων.

Οι εφαρμογές των σμηνών από MEA είναι πολλές: από αποστολές αναζήτησης-διάσωσης σε κατεστραμμένα κτίρια και αποθήκες [19], εφαρμογή λιπασμάτων και φυτοφαρμάκων στην γεωργία [20], μέχρι και ψυχαγωγικές εναέριες επιδείξεις [21]. Η έρευνα συνεχίζει να εστιάζει σε προκλήσεις όπως η ασφαλής

αποφυγή συγκρούσεων, η αυτόνομη πλοήγηση σε σύνθετα περιβάλλοντα και η αποδοτική κατανομή εργασιών σε πραγματικό χρόνο.

Με άλλα λόγια, τα σμήνη από ΜΕΑ συνιστούν ένα πολλά υποσχόμενο πεδίο. Η συνεργασία μεταξύ των ΜΕΑ προσφέρει λύσεις σε περιπτώσεις όπου είναι δύσκολο ή δεν είναι δυνατό καν να μπορεί να επέμβει ο άνθρωπος αλλά και σε περιπτώσεις όπου απλά επιθυμούμε να λυθεί το πρόβλημα γρηγορότερα. Έτσι, τα σμήνη από ΜΕΑ δεν αποτελούν απλώς μια καινοτομική τεχνολογία, αλλά μια πρακτική λύση που μπορεί να βελτιώσει την αποδοτικότητα και την ασφάλεια σε πολλές περιπτώσεις, καθιστώντας την έρευνα και την υλοποίησή τους επένδυση με μακρόχρονες προοπτικές.

1.3 Ορισμός του Προβλήματος

Το βασικό πρόβλημα που αντιμετωπίζεται σε αυτή την εργασία είναι η αυτοματοποιημένη χαρτογράφηση και κάλυψη ενός κλειστού χώρου (δωματίου) με χρήση του ΜΕΑ, Crazyflie. Αρχικά, διερευνάται η δυνατότητα πτήσης ενός μόνο Crazyflie για ακολουθία των τοίχων (WallFollowing) και συλλογή δεδομένων, με στόχο τη δημιουργία ενός χάρτη του δωματίου. Στη συνέχεια, το σύστημα πρέπει να επεκτείνεται ώστε να συμπεριλαμβάνει κι άλλα Crazyflies που πετούν ταυτόχρονα σε διαφορετικές διαδρομές εντός του ίδιου χώρου.

Ένα άλλο ζήτημα είναι πώς μπορούν τα ΜΕΑ να επικοινωνούν μεταξύ τους με σε αυτόν τον περιορισμένο χώρο έτσι ώστε να αποφεύγονται οι συγκρούσεις μεταξύ των ΜΕΑ (κάθε ΜΕΑ πρέπει να αντιλαμβάνεται και να απομακρύνεται από κοντινά ΜΕΑ) και να βελτιστοποιείται η κάλυψη.

Με αυτόν τον τρόπο επιδιώκεται η δημιουργία ενός αυτοματοποιημένου συστήματος χαρτογράφησης εσωτερικών χώρων, το οποίο θα δουλεύει όπως πρέπει ακόμη και στην περίπτωση απώλειας κάποιου από τα ΜΕΑ, θα εξοικονομεί χρόνο και ενέργεια, βελτιστοποιώντας την κάλυψη του χώρου και θα επιτρέπει επεκτασιμότητα, δηλαδή προσθήκη ή αφαίρεση ΜΕΑ χωρίς να πρέπει να αλλάξει κάτι στο σχεδιασμό.

1.4 Στόχος Εργασίας

Σκοπός αυτής της διπλωματικής εργασίας είναι η μελέτη, υλοποίηση και αξιολόγηση ενός συστήματος σμήνους από ΜΕΑ χαρτογράφησης και αποφυγής εμποδίων με χρήση των ΜΕΑ Crazyflie.

Πιο συγκεκριμένα, επιδιώκουμε να πειραματιστούμε με τη χρήση αλγορίθμων τύπου Wall Following οι οποίοι θα βασίζονται στα δεδομένα από αισθητήρες. Ο αλγόριθμος θα επιτρέπει στα ΜΕΑ να εντοπίζουν τοίχους και να τους ακολουθούν. Ακόμη, θα πρέπει να υπάρχει κάποιου είδους επικοινωνία μεταξύ των ΜΕΑ έτσι ώστε να υπάρχει αποκεντροποιημένη λήψη αποφάσεων αποφυγής συγκρούσεων. Τέλος, θα αξιολογήσουμε την απόδοση του σμήνους σε πραγματικές πτήσεις.

1.5 Μεθοδολογία

1.5.1 Αρχικά Στάδια

Η πρώτη φάση της εργασίας μας ξεκίνησε με την διερεύνηση των διαθέσιμων μικρών ιπτάμενων ΜΕΑ εσωτερικού χώρου. Μετά από εκτενή έρευνα αγοράς και σύγκριση χαρακτηριστικών, αποφασίσαμε να επενδύσουμε στα Crazyflie της Bitcraze. Η επιλογή βασίστηκε στο σχετικά χαμηλό κόστος τους, την ενεργή υποστήριξη της κοινότητας και το πλήρες ανοιχτό λογισμικό (firmware και Python API) [2] που διευκολύνει την προσαρμογή.

Αφού παραλάβαμε το Crazyflie 2.1, αφιερώσαμε αρκετές ώρες στην εγκατάσταση και στους πρώτους πειραματισμούς. Μελετήθηκαν διαδικτυακά σεμινάρια και παρουσιάσεις στο YouTube, τεχνικά άρθρα και οδηγοί του κατασκευαστή, αλλά και εργασίες άλλων ομάδων που αξιοποίησαν το Crazyflie σε έρευνες εσωτερικής πλοήγησης. Παράλληλα, παραγγείλαμε διάφορες *Μονάδες Επέκτασης* (Expansion Decks), τα οποία δίνουν διάφορες νέες λειτουργικότητες στο Crazyflie και εξηγούνται στο Κεφάλαιο 2.3, ώστε να πειραματιστούμε με τη σταθερότητα πτήσης, την οπτική οδομετρία, την αποφυγή εμποδίων, κτλ.

1.5.2 Περιβάλλον Ανάπτυξης

Αρχικά, το περιβάλλον ανάπτυξης ήταν σε MacOS. Σύντομα όμως διαπιστώσαμε ότι η συμβατότητα των βιβλιοθηκών και οι απαιτήσεις ρύθμισης (εικονικά περιβάλλοντα, βιβλιοθήκες C++) δημιουργούσαν σημαντική καθυστέρηση. Το MacOS διευκόλυνε την εγκατάσταση των εξαρτήσεων και βιβλιοθηκών για τη διεπαφή Python και τα γρήγορα πειράματα, αλλά αντιμετώπισε δυσκολίες όταν ήρθε η ώρα να μεταγλωττίσουμε τον κώδικα C/C++ του ενσωματωμένου λογισμικού. Μετά αλλάξαμε το περιβάλλον ανάπτυξης στην *εικονική μηχανή* (VM) [8] της Bitcraze που τρέχει σε Oracle VirtualBox πάνω σε Windows οικοδεσπότη. Το VM περιείχε ήδη όλες τις απαραίτητες βιβλιοθήκες, μεταγλωττιστή για το ενσωματωμένο λογισμικό και τη διεπαφή Python, παρέχοντας πολύ πιο εύκολη λειτουργικότητα σε σύγκριση με το MacOS χωρίς επιπλέον ρυθμίσεις και κόπο.

1.5.3 Υλοποίηση Αλγορίθμων και Πειράματα

Στη συνέχεια έγινε η δοκιμή του αλγορίθμου WallFollowing της Bitcraze σε Python. Σκοπός ήταν το Crazyflie να πετάξει σε ένα χώρο (διαμορφωμένο με χαρτοκιβώτια και τοίχους) και να διατηρεί σταθερή απόσταση από τον τοίχο. Για να το πετύχουμε αυτό κατανοήσαμε σε βάθος τη διεπαφή Python (crazyflie-python-lib) [9], μελετώντας παραδείγματα και τον κώδικα στο GitHub και στη συνέχεια γραψαμε δικά μας scripts που διάβαζαν κομμάτια τηλεμετρίας και καθοδηγούσαν τις κινήσεις του Crazyflie από εντολές του υπολογιστή. Επίσης καταγράφηκαν οι πορείες του Crazyflie έτσι ώστε να γίνει χαρτογράφηση του χώρου σε 3D περιβάλλον, επεξεργαστήκαμε τα δεδομένα και επιβεβαιώσαμε σε βίντεο τη σωστή συμπεριφορά.

Αυτή η φάση φανέρωσε πολλούς περιορισμούς του υλικού: το Crazyflie σε χαμηλό φωτισμό δυσκολευόταν να πετάξει, ο αισθητήρας για αποφυγή εμποδίων δεν έβλεπε λεπτά εμπόδια ή πολλές φορές δεν έβλεπε τίποτα, το MEA πολλές φορές έπεφτε από μόνο του και πολλά άλλα. Αναλυτικά προβλήματα παρουσιάζονται στο Κεφάλαιο 4.

Μετά την επιτυχή δοκιμή ενός Crazyflie, αποφασίσαμε να προχωρήσουμε σε πλήρως αυτόνομη πτήση χωρίς υπολογιστή, δηλαδή κάθε Crazyflie θα υλοποιεί τον αλγόριθμο ενσωματωμένα, χρησιμοποιώντας P2P επικοινωνία μεταξύ των Crazyflie για αποφυγή

συγκρούσεων. Ως βάση για το σμήνος, χρησιμοποιήσαμε την εργασία “Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment” [14]. Παρά το ότι ο πηγαίος κώδικας ήταν γραμμένος για το πρώιμο Crazyflie 2.0 και ένα παλαιό ενσωματωμένο λογισμικό χωρίς P2P, μας προσέφερε μια εικόνα του SGBA (Swarm Gradient Bug Algorithm), επιτρέποντάς μας να σχεδιάσουμε και να πραγματοποιήσουμε περαιτέρω πειράματα. Περισσότερες λεπτομέρειες στο Κεφάλαιο 3.5.

Για να γίνει αυτό μετέφεραμε τον πηγαίο κώδικα σε *εξωτερικό* (out-of-tree) περιβάλλον με τη βοήθεια του Επιπέδου Εφαρμογής (Κεφάλαιο 3.3). Αφού μελετήθηκε η διεπαφή P2P, η οποία είναι ακόμα σε δοκιμαστικό στάδιο, έπρεπε να προσαρμοστεί η επικοινωνία, αντικαθιστώντας τις χειροκίνητες *αναμεταδόσεις* (broadcasts) σε επικοινωνία με τη νέα διεπαφή P2P.

Για τη συγγραφή του κώδικα αξιοποιήσαμε τη δομή του πηγαίου κώδικα, τη ρύθμιση των παραμέτρων και των ορίων, το φιλτράρισμά τους με φίλτρα διάμεσης τιμής, καθώς και την ενσωμάτωση του μηχανισμού ανάκλησης P2P (P2P callback), διαδικασίες που είχαν έντονα δημιουργικό χαρακτήρα. Ωστόσο, οι πειραματικές δοκιμές στο πεδίο απαιτούσαν επαναλαμβανόμενες προσπάθειες: πολλές καταρρεύσεις, εξαντλημένες μπαταρίες, προβλήματα με το πρόγραμμα οδήγησης, απρόβλεπτες συμπεριφορές και άλλα. Κάθε αποτυχία όμως οδήγησε σε βελτιστοποίηση του κώδικα και σε βαθύτερη κατανόηση του υλικού, μέχρι να επιτύχουμε σταθερό και αξιόπιστο αποτέλεσμα.

Με αυτήν την διαδικασία επιλογής πλατφόρμας, εξοικείωσης, ανάπτυξης κώδικα, επαναλήψεων hardware και λογισμικού, αναπτύξαμε μια μεθοδολογία που συνδυάζει: Python scripts, πλήρη αυτόνομη υλοποίηση, και P2P επικοινωνία.

1.6 Δομή της Εργασίας

Στην παρούσα διπλωματική εργασία η διάρθρωση των κεφαλαίων έχει ως εξής:

Κεφάλαιο 2: Το MEA Crazyflie:

Αναλύεται η επιλογή του Crazyflie: ιστορικό, τεχνικά χαρακτηριστικά, διαθέσιμες μονάδες επέκτασης και αισθητήρες, καθώς και η διαδικασία κατασκευής/σύνθεσης της πλατφόρμας για τις ανάγκες της εργασίας.

Κεφάλαιο 3: Υλοποίηση Αλγορίθμων:

Περιγράφονται οι αλγόριθμοι WallFollowing και οι μέθοδοι επικοινωνίας και αποφυγής συγκρούσεων.

Κεφάλαιο 4: Πειραματική Εφαρμογή - Αποτελέσματα και Ανάλυση:

Αναλύονται τα πειραματικά σενάρια σε πραγματικό περιβάλλον.

Κεφάλαιο 5: Συμπεράσματα και Μελλοντικές Εργασίες:

Συνοψίζονται τα κύρια ευρήματα, αξιολογείται η επίτευξη των στόχων της εργασίας και προτείνονται επεκτάσεις για πιο σύνθετα σενάρια.

Κεφάλαιο 2

Το MEA Crazyflie

2.1 Επιλογή Crazyflie	13
2.2 Χαρακτηριστικά Crazyflie 2.1	14
2.3 Μονάδες Επέκτασης	15
2.4 Πελάτες	19
2.5 Crazyradio 2.0 USB	20
2.6 Αρχική Διάταξη Crazyflie	21
2.7 Βιβλιοθήκη Crazyflie Python	26

2.1 Επιλογή Crazyflie

Το αρχικά δίλημμα ήταν το εξής: ποιο MEA να επιλέξουμε για να χαρτογραφήσουμε εσωτερικούς χώρους;

Έπειτα από αρκετή αναζήτηση στο διαδίκτυο ανακαλύψαμε το Crazyflie της Bitcraze. Παρατηρήσαμε ότι πως πανεπιστήμια σε ολόκληρο τον κόσμο δημοσιεύουν μελέτες με το ίδιο MEA, ενώ στο φόρουμ της Bitcraze και στο GitHub υπάρχει πάντα κάποιος να απαντήσει σε ερωτήσεις ή να μοιραστεί νέο παράδειγμα κώδικα [16]. Όμως το πραγματικό του πλεονέκτημα είναι το ότι ολόκληρο το ενσωματωμένο λογισμικό είναι ανοικτού κώδικα, με ενεργή κοινότητα υποστήριξης και συνεχείς ενημερώσεις στο GitHub [2].

Κανένα άλλο MEA δεν έδειχνε να ταιριάζει τόσο καλά με τις ανάγκες μας: μικρό μέγεθος, ευελιξία στον κώδικα, υποστήριξη από μια ενεργή κοινότητα και τον κατασκευαστή και χαμηλό κόστος (σχετικά).

2.2 Χαρακτηριστικά Crazyflie 2.1



Εικόνα 2.1 Crazyflie

Το Crazyflie 2.1, όπως παρουσιάζεται στην Εικόνα 2.1 αποτελεί μια ευέλικτη και ανοικτού κώδικα πλατφόρμα ανάπτυξης. βάρους μόλις 27g, που χωρά στην παλάμη του χεριού.

Διαθέτει εξαιρετικά συμπαγή και ανθεκτική κατασκευή, με βάρος απογείωσης μόλις 27 g και διαστάσεις 92×92×29 mm. Στο εσωτερικό του φιλοξενεί δύο ισχυρούς μικροεπεξεργαστές: τον **STM32F405** (Cortex-M4, 168 MHz, 192 kB SRAM, 1 MB flash) για την κύρια εφαρμογή πτήσης και τον **nRF51822** (Cortex-M0, 32 MHz, 16 kB SRAM, 128 kB flash) για τη διαχείριση του ραδιοφώνου και της ισχύος. Διαθέτει micro-USB και μπαταρία LiPo με 250mAh, που προσφέρει πλήρη φόρτιση μέσα σε περίπου 40 λεπτά. Με την εργοστασιακή μπαταρία, το Crazyflie πετά για περίπου 7 λεπτά, ενώ μπορεί να σηκώσει φορτίο μέχρι 15 g [17].

Επιπρόσθετα, διαθέτει τέσσερις κινητήρες διαστάσεων 7×16 mm ($K_v \approx 14\,000$ rpm/V), οι οποίοι λειτουργούν στα 4,2 V, έχουν βάρος μόλις 2,7 g και αξονική διάμετρο 0,8 mm (μήκος άξονα 3,5 mm). Αυτοί οι υψηλών στροφών και χαμηλού βάρους κινητήρες στρέφουν πλαστικές έλικες μήκους 45 mm.

Επιπλέον, οι ενσωματωμένοι αισθητήρες του Crazyflie βασίζονται σε 10-DOF (degrees of freedom) Inertial Measurement Unit (IMU), δηλαδή MPU-9250 με τρεις άξονες: γυροσκόπιο, επιταχυνσιόμετρο και μαγνητόμετρο, καθώς και έναν επιπλέον αισθητήρα πίεσης υψηλής ακρίβειας (LPS25H). Όλοι οι αισθητήρες παρέχουν μετρήσεις για τη σταθεροποίηση της πτήσης. Μέσω της διεπαφής επέκτασης, οι χρήστες έχουν άμεση πρόσβαση σε διαύλους όπως UART, I2C και SPI.

Η επικοινωνία γίνεται είτε μέσω Bluetooth από iOS/Android είτε από Windows/Mac OS X/Linux με το Crazyradio, έχοντας αποδεδειγμένα εμβέλεια πέρα από τα 200 m σε ανοιχτό χώρο. Περισσότερες πληροφορίες μπορείτε να βρείτε στα [1, 2].

Η συναρμολόγηση είναι απλή, χωρίς ανάγκη κόλλησης, και οι Μονάδες Επέκτασης εντοπίζονται αυτόματα. Η διαδικασία για το Crazyflie 2.1 ολοκληρώνεται σε λιγότερο από δέκα λεπτά: τοποθετούμε τους κινητήρες στις ειδικές βάσεις και μετά τοποθετούμε τους έλικες, φροντίζοντας να ταιριάζουμε τις δεξιόστροφες (CW) και αριστερόστροφες (CCW) λεπίδες με την κοίλη πλευρά προς τα πάνω. Στη συνέχεια προσθέτουμε το ελαστικό στήριγμα για τη μπαταρία, εγκαθιστούμε τις ειδικές υποδοχές (headers) και τοποθετούμε τη μπαταρία στη θέση της. Τέλος, πατάμε το κουμπί λειτουργίας· κατά την εκκίνηση οι έλικες περιστρέφονται διαδοχικά ως έλεγχος λειτουργίας.

2.3 Μονάδες Επέκτασης

Το Crazyflie 2.1 υποστηρίζει επεκτάσεις μέσω των Μονάδων Επέκτασης, οι οποίες τοποθετούνται εύκολα πάνω στις ειδικές υποδοχές και ανιχνεύονται αυτόματα από το ενσωματωμένο λογισμικό. Κάθε μονάδα επέκτασης προσθέτει μια καινούργια δυνατότητα στο Crazyflie.

Για παράδειγμα κάποιες από τις μονάδες:

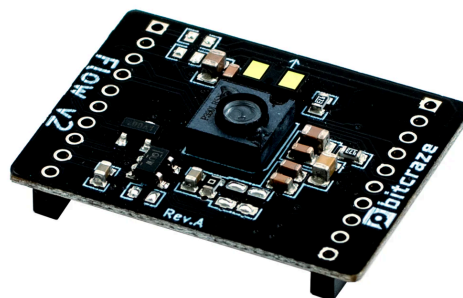
- Flow Deck v2: Για ακριβής διατήρηση θέσης και σταθερό ύψος σε εσωτερικούς χώρους χωρίς GPS.

- Multiranger Deck: Αισθητήρας απόστασης για αποφυγή εμποδίων, wall following και ασφαλής πτήσεις κοντά σε τοίχους.
- SD card Deck: Δίνει τη δυνατότητα ανάγνωσης και εγγραφής αρχείων σε μια κάρτα micro SD. Για παράδειγμα, θα μπορούσε να χρησιμοποιηθεί για αποθήκευση ρυθμίσεων.
- AI Deck: Επεκτείνει τις υπολογιστικές δυνατότητες και επιτρέπει την εκτέλεση πιο σύνθετων αλγορίθμων που βασίζονται στην τεχνητή νοημοσύνη.

Για περισσότερες πληροφορίες στο [3]

Για τους σκοπούς αυτής της εργασίας χρησιμοποιήθηκαν τα Flow Deck v2 και Multi-Ranger Deck.

Flow Deck v2 [4]:



Εικόνα 2.2 Flow Deck v2

Το Flow Deck, όπως παρουσιάζεται στην Εικόνα 2.2, προσθέτει στο Crazyflie 2.1 ένα συμπαγές σύστημα ανίχνευσης κίνησης υψηλής ακρίβειας, μετατρέποντάς το σε έναν πραγματικό 3D ιπτάμενο ρομπότ. Συνδυάζει δύο συμπληρωματικούς αισθητήρες:

- **Αισθητήρας VL53L1x ToF:**
Μετρά με χιλιοστομετρική ακρίβεια την απόσταση από το έδαφος (έως 4 m), χρησιμοποιώντας ακτίνες υπερύθρου ασφαλείς για τα μάτια.

- **Αισθητήρας οπτικής ροής PMW3901:**

Παρακολουθεί την οριζόντια κίνηση αναλύοντας μοτίβα στην επιφάνεια κάτω από το MEA.

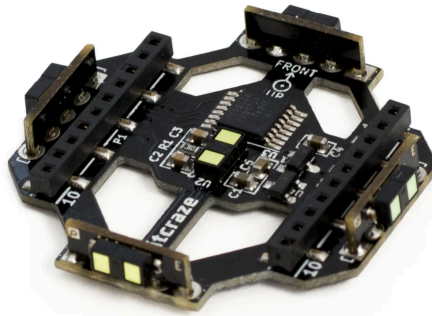
Με το Flow Deck, το Crazyflie μπορεί να:

- Προγραμματίζεται για ακριβείς 3D διαδρομές σε οποιαδήποτε διεύθυνση, διατηρώντας σταθερό ύψος και θέση.
- Παρέχει υποβοηθούμενη πτήση για αρχάριους, χάρη στην αυτοματοποιημένη σταθεροποίηση.

Βασικά Χαρακτηριστικά

- Γρήγορη και ακριβής μέτρηση απόστασης (0,05–4 m).
- Ασφαλής για τα μάτια μέτρηση χρόνου πτήσης.
- Αξιόπιστη ανίχνευση οριζόντιας κίνησης.
- Δυνατότητα πλήρως προγραμματισμένων 3D πτήσεων.
- Σταθερή, υποβοηθούμενη πλατφόρμα πτήσης.

Multi-Ranger Deck [5]:



Εικόνα 2.3 Multi-Ranger Deck

Το Multi-Ranger Deck, όπως παρουσιάζεται στην Εικόνα 2.3, προσθέτει στο Crazyflie 2.1 τη δυνατότητα εντοπισμού εμποδίων γύρω του, με τη μέτρηση της απόστασης σε πέντε κατευθύνσεις—εμπρός, πίσω, αριστερά, δεξιά και πάνω, με ακρίβεια χιλιοστού έως και 4 μέτρα. Χρησιμοποιεί 5 αισθητήρες **VL53L1x ToF** που χρησιμοποιούνται επίσης και στο Flow Deck.

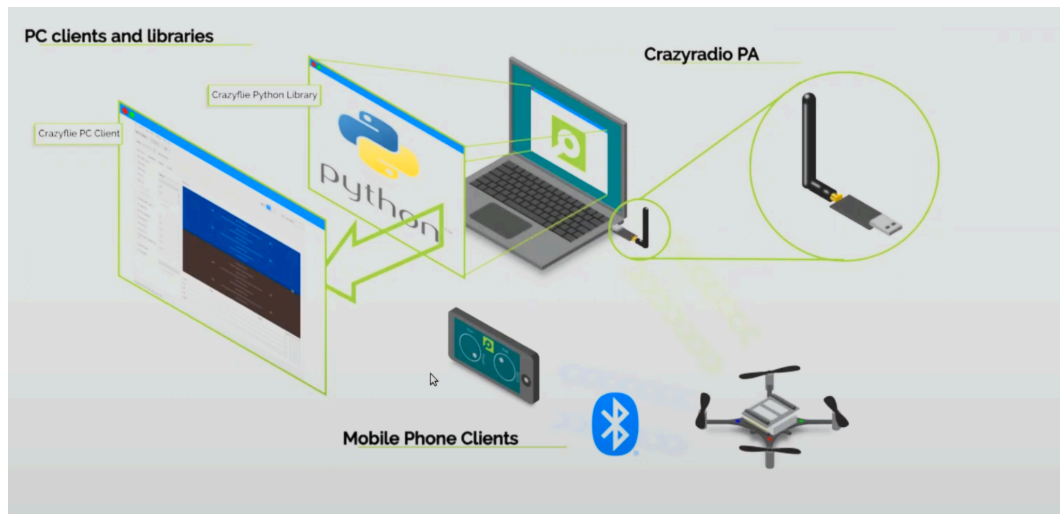
Με το Multi-Ranger Deck, το Crazyflie μπορεί να:

- Ανιχνεύει εμπόδια σε πέντε κατευθύνσεις (εμπρός, πίσω, αριστερά, δεξιά, πάνω).
- Διατηρεί σταθερή απόσταση από τοίχους και αντικείμενα για ασφαλείς πτήσεις κοντά σε επιφάνειες.
- Συνεργάζεται με το Flow Deck για εκτίμηση θέσης και ύψους στο χώρο.

Βασικά Χαρακτηριστικά

- Γρήγορη και ακριβής μέτρηση απόστασης.
- Απόλυτη εμβέλεια έως 4 m σε 5 κατευθύνσεις.
- Ασφαλής για τα μάτια λειτουργία (eye safe).

2.4 Πελάτες



Εικόνα 2.4 Πελάτες Crazyflie

Για τον έλεγχο και την παραμετροποίηση του Crazyflie 2.1 διατίθενται δύο πελάτες:

1. Πελάτης PC

- **Πελάτης Crazyflie:** Εφαρμογή με GUI για διαχείρισης πτήσης, τηλεμετρίας, παραμέτρων και για αναβάθμιση του ενσωματωμένου λογισμικού του Crazyflie.

Επίσης μεταξύ άλλων διαθέτει και κονσόλα η οποία εμφανίζει σε μηνύματα καταγραφής (logs), σφάλματα επικοινωνίας και σφάλματα των decks.

2. Πελάτης Κινητού Τηλεφώνου

- **Εφαρμογή Android/iOS:** Σύνδεση του Crazyflie μέσω Bluetooth με ειδικό application στο τηλέφωνο με γραφικό περιβάλλον χειρισμού, το οποίο προσφέρει on-screen κουμπιά για χειροκίνητο έλεγχο της πτήσης.

Η Εικόνα 2.4 παρουσιάζει τον τρόπο επικοινωνίας με το Crazyflie μέσω PC (GUI & βιβλιοθήκη Python) και κινητών (Bluetooth), χρησιμοποιώντας το Crazyradio ως ενδιάμεσο για ασύρματη σύνδεση.

2.5 Crazyradio 2.0 USB



Εικόνα 2.5 Crazyradio 2.0

Η επικοινωνία του Crazyflie με τον υπολογιστή γίνεται με τη βοήθεια του Crazyradio (Εικόνα 2.5). Το Crazyradio 2.0 είναι ένας προσαρμογέας USB με ενσωματωμένο ενισχυτή ισχύος και κεραία μεγάλης εμβέλειας, που λειτουργεί στα 2.4 GHz.

Όταν στέλνεται μια εντολή από τον υπολογιστή μέσω του Πελάτη Crazyflie ή ενός Python script, το Crazyradio μετατρέπει τα ψηφιακά δεδομένα σε πακέτα *ραδιοσυχνότητων* (radiofrequency - RF).

Στο εσωτερικό του Crazyflie, ένας πομποδέκτης παρακολουθεί τα πακέτα στον αέρα που εκπέμπονται από το Crazyradio, αναγνωρίζει τα απευθυνόμενα σε αυτόν πακέτα και τα διαβιβάζει με τη βοήθεια του **nRF51822**. Ο βοηθητικός αυτός μικροεπεξεργαστής-πομποδέκτης ελέγχει την ακεραιότητα των δεδομένων και στη συνέχεια τις προωθεί στον κύριο επεξεργαστή πτήσης (**STM32F405**) μέσω του διαύλου SPI. Ο STM32F405 επεξεργάζεται τις εντολές και ανάλογα τις εκτελεί το Crazyflie.

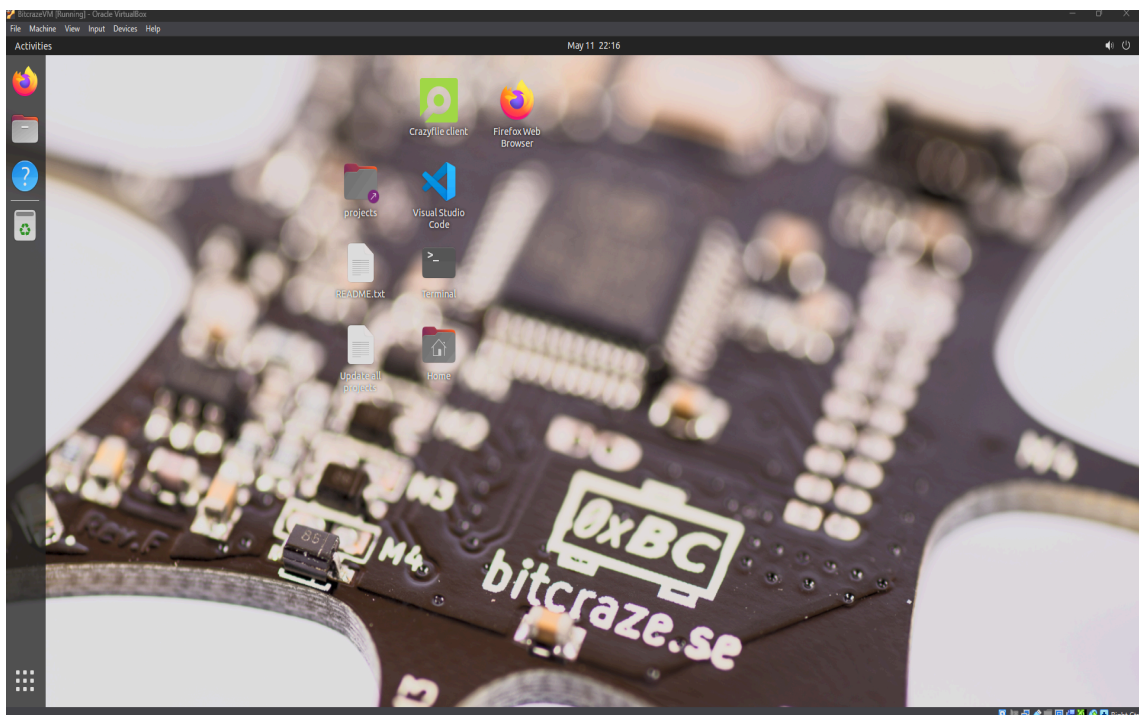
Ταυτόχρονα, τα δεδομένα τηλεμετρίας και οι μετρήσεις των αισθητήρων στέλνονται με την αντίστροφη διαδρομή: ο κύριος επεξεργαστής (**STM32F405**) συσκευάζει τις πληροφορίες, τις στέλνει στον βοηθητικό μικροεπεξεργαστή (**nRF51822**), ο οποίος τις εκπέμπει στο Crazyradio 2.0. Το Crazyradio λαμβάνει αυτές τις απαντήσεις, τις μεταφέρει στον υπολογιστή και ο Πελάτης Crazyflie εμφανίζει σε πραγματικό χρόνο την κατάσταση πτήσης, τις μετρήσεις αισθητήρων ή τυχόν σφάλματα.

Για περισσότερες πληροφορίες στα [6, 7] .

2.6 Αρχική Διάταξη Crazyflie

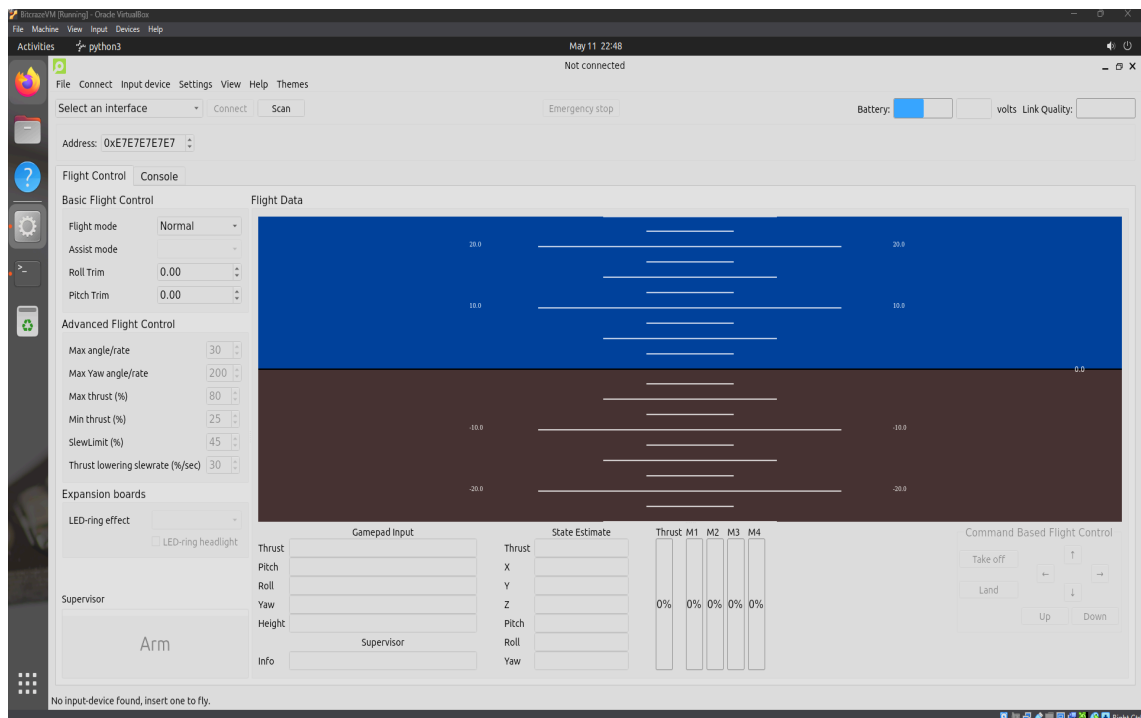
Για να διευκολύνουν τους χρήστες στην εκκίνηση των πτήσεων και της ανάπτυξης λογισμικού, η Bitcraze, οι δημιουργοί του Crazyflie, παρέχουν τη δική τους Εικονική Μηχανή (EM). Η EM αυτή περιλαμβάνει όλα τα απαραίτητα εργαλεία και βιβλιοθήκες, αποφεύγοντας έτσι πολύπλοκες διαδικασίες εγκατάστασης. Στην παρούσα εργασία, το περιβάλλον ανάπτυξης γίνεται σε αυτήν τη EM [8].

Για την εκτέλεση αυτής της εικονικής μηχανής χρησιμοποιήθηκε το Oracle VirtualBox σε περιβάλλον Windows. Η VM παρουσιάζεται στην Εικόνα 2.6



Εικόνα 2.6 Επιφάνεια Εργασίας EM Crazyflie

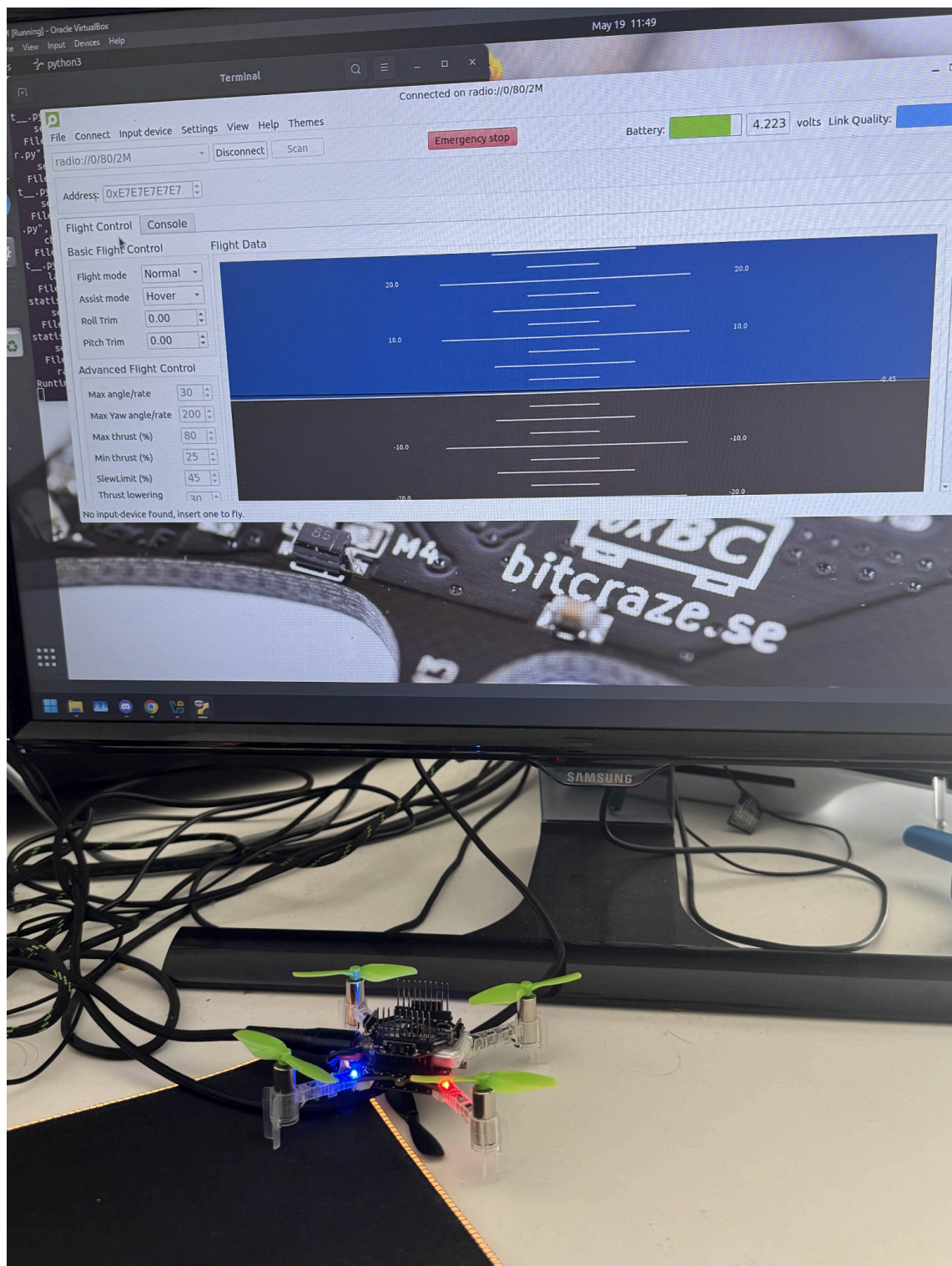
Υπάρχει επίσης ένα script “Update All projects” το οποίο ανακτά αυτόματα τις πιο πρόσφατες ενημερώσεις όλων των αποθετηρίων GitHub και τις εγκαθιστά στην EM. Όταν γίνει η ενημέρωση μπορούμε να τρέξουμε τον Πελάτη Crazyflie (Εικόνα 2.7).



Εικόνα 2.7 Πελάτης Crazyflie

Αρχικά ενεργοποιούμε και συνδέουμε το Crazyflie με τον υπολογιστή χρησιμοποιώντας ένα micro-usb καλώδιο. Βάζουμε διεύθυνση 0xE7E7E7E7E7 (η προεπιλεγμένη διεύθυνση), και πατούμε “Scan”. Ο πελάτης θα εντοπίσει το Crazyflie και στη συνέχεια πατούμε “Connect”.

Στην παρακάτω εικόνα παρατηρούμε τη διάταξη και την οθόνη όταν το Crazyflie είναι συνδεδεμένο.



Εικόνα 2.8 Σύνδεση Crazyflie με Πελάτη

Παρατηρούμε ότι με την πρώτη σύνδεση του Crazyflie στον πελάτη, στην κονσόλα εμφανίζονται αμέσως μηνύματα επιβεβαίωσης σύνδεσης και εντοπισμού της μονάδας όπως: Ανίχνευση Μονάδων Επέκτασης (“Flow Deck detected”, “Multi-Ranger Deck

detected”), Έλεγχος ενσωματωμένου λογισμικού (“Firmware version: v2025.05”), κτλ. Η κονσόλα μας βοηθά να παρατηρήσουμε τυχόν σφάλματα που προκύπτουν. Στην πιο κάτω εικόνα βλέπουμε την κονσόλα.



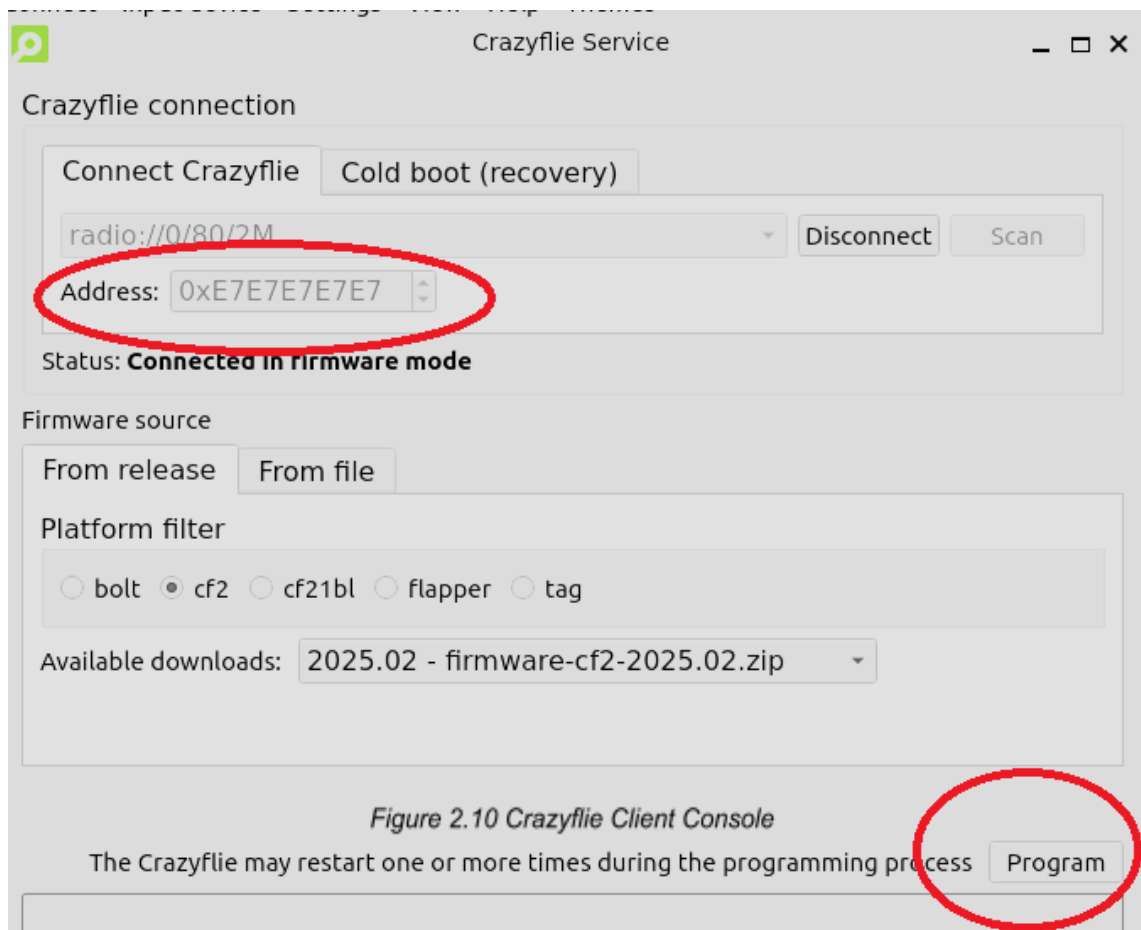
```
Address: 0xE7E7E7E7E7E7E7E7
Flight Control Console
---
SYS: Crazyflie 2.1 is up and running!
SYS: Build 0:16c5630707e9 (2025.02) CLEAN
SYS: I am 0x2030355451315006002C0022 and I have 1024KB of flash!
CFGBLK: v1, verification [OK]
DECK_CORE: 2 deck(s) found
DECK_CORE: Calling INIT on driver bcFlow2 for deck 0
ZR2: Z-down sensor [OK]
PMW: Motion chip id: 0x49:0xB6
DECK_CORE: Calling INIT on driver bcMultiranger for deck 1
IMU: BMI088: Using I2C interface.
IMU: BMI088 Gyro connection [OK].
IMU: BMI088 Accel connection [OK]
IMU: BMP388 I2C connection [OK]
ESTIMATOR: Using Kalman (2) estimator
CONTROLLER: Using PID (1) controller
MTR-DRV: Using brushed motor driver
SYS: About to run tests in system.c.
SYS: NRF51 version: 2025.02 (CF21)
EEPROM: I2C connection [OK].
STORAGE: Storage check [OK].
IMU: BMI088 gyro self-test [OK]
DECK_CORE: Deck 0 test [OK].
MR: Init front sensor [OK]
MR: Init back sensor [OK]
MR: Init up sensor [OK]
MR: Init left sensor [OK]
MR: Init right sensor [OK]
DECK_CORE: Deck 1 test [OK].
SYS: Self test passed!
STAB: Wait for sensor calibration...
SYS: Free heap: 13736 bytes
STAB: Starting stabilizer loop
ESTKALMAN: State out of bounds, resetting
```

Εικόνα 2.9 Κονσόλα Πελάτη Crazyflie

Έπειτα θα δώσουμε στο Crazyflie μια δική του μοναδική διεύθυνση. Κάθε Crazyflie θα έχει τη δική της. Όπως αναφέρθηκε προηγουμένως η προεπιλεγμένη διεύθυνση είναι 0xE7E7E7E7E7. Για το συγκεκριμένο Crazyflie, θα παραμείνει ως έχει, αλλά για τα υπόλοιπα θα αλλάξει. Για να αλλάξει η διεύθυνση ανοίγουμε το μενού **Connect** → **Configure 2.x** για να εμφανιστεί το παράθυρο ρυθμίσεων 2.x, όπου μπορούμε να βρούμε τη διεύθυνση του Crazyflie και να την αλλάξουμε.

Στη συνέχεια, αφού συνδέσουμε το Crazyflie με τον Πελάτη, πρέπει να γίνει ενημέρωση του ενσωματωμένου λογισμικού του Crazyflie και των Μονάδων Επέκτασης που είναι συνδεδεμένες επάνω. Η ενημέρωση γίνεται μέσω του Πελάτη.

Η διαδικασία παρουσιάζεται στην πιο κάτω εικόνα όπου μπορούμε να αλλάξουμε τη διεύθυνση και να διαλέξουμε εκδοση ενσωματωμένου λογισμικού.



Εικόνα 2.10 Ενημέρωση Λογισμικού μέσω Πελάτη Crazyflie

Για την αναβάθμιση του ενσωματωμένου λογισμικού απαιτείται η χρήση του Crazyradio (η αναβάθμιση μέσω USB δεν είναι δυνατή). Συνεπώς, αποσυνδέουμε το Crazyflie από τη θύρα USB και εισάγουμε το Crazyradio στο υπολογιστή. Στον Πελάτη, πληκτρολογούμε τη διεύθυνση του Crazyflie, πατούμε “Scan” για να εντοπιστεί το Crazyflie μέσω Crazyradio, και στη συνέχεια επιλέγουμε “Connect” για να γίνει η σύνδεση.

Έπειτα, μεταβαίνουμε στο μενού. Επιλέγουμε την καρτέλα “From release” που εμφανίζεται στο νέο παράθυρο, για να κατεβάσει αυτόματα την επίσημη έκδοση από το GitHub, φροντίζοντας να έχουμε επιλέξει την πλατφόρμα **cf2** (Crazyflie 2.x). Πατούμε “Program” και περιμένουμε να γίνει η ενημέρωση. (Γίνεται εγγραφή τόσο στον **STM32F405** όσο και στον **nRF51822** και στις Μονάδες Επέκτασης). Μετά το πέρας της ενημέρωσης, για να επαληθεύσουμε την έκδοση του ενσωματωμένου λογισμικού, μπορούμε να κοιτάξουμε στην κονσόλα.

2.7 Βιβλιοθήκη Crazyflie Python

Για τον προγραμματισμό χρησιμοποιούμε τη βιβλιοθήκη Crazyflie Python, η οποία έρχεται προεγκατεστημένη στην εικονική μηχανή που χρησιμοποιείται σε αυτή την εργασία. Τρέχει σε Python 3.11.2 και παρέχει μια διεπαφή που επιτρέπει τη σύνδεση, την αποστολή εντολών πτήσης και όχι μόνο, με μερικές μόνο γραμμές κώδικα. Με τη βοήθεια της βιβλιοθήκης αυτής, γράφονται προγράμματα που υλοποιούν αλγορίθμους. Η χρήση της διεπαφής Python απλοποιεί σημαντικά την ανάπτυξη και επιτάχυνση των δοκιμών, καθώς οι αλλαγές στον κώδικα δοκιμάζονται άμεσα χωρίς να χρειάζεται να αλλάζει το ενσωματωμένο λογισμικό του Crazyflie. [20]

Ένα απλό παράδειγμα σε Python που συνδέεται με το Crazyflie, ελέγχει αν οι Μονάδες Επέκτασης (Flow, Multi-Ranger) είναι συνδεδεμένες και έτοιμες για χρήση. Το πρόγραμμα αυτό αναδεικνύει πώς με λίγες γραμμές κώδικα μπορούμε να ξεκινήσουμε το MEA, να διαβάσουμε την κατάσταση των Μονάδων Επέκτασης και να δώσουμε εντολές.

Στην επόμενη σελίδα παρουσιάζεται ο κώδικας (Εικόνα 2.11).

```

URI = uri_helper.uri_from_env(default='radio://0/80/2M/E7E7E7E7E7')

# Events to track deck attachment
deck_attached_events = {
    'Flow': Event(),
    'Multi-ranger': Event()
}

logging.basicConfig(level=logging.ERROR)

def param_deck_attached(deck_name):
    def callback(_, value_str):
        value = int(value_str)
        if value:
            deck_attached_events[deck_name].set()
            print(f'{deck_name} Deck is attached!')
        else:
            print(f'{deck_name} Deck is NOT attached!')
    return callback

if __name__ == '__main__':
    cflib.crtp.init_drivers()
    with SyncCrazyflie(URI, cf=Crazyflie(rw_cache='./cache')) as scf:
        #register deck attachment callbacks with correct function
        scf.cf.param.add_update_callback(group='deck', name='bcFlow2',
                                         cb=param_deck_attached('Flow'))
        scf.cf.param.add_update_callback(group='deck', name='bcMultiranger',
                                         cb=param_deck_attached('Multi-ranger'))

        time.sleep(1)

```

Εικόνα 2.11 Πρόγραμμα Python για έλεγχο συνδεδεμένων Μονάδων Επέκτασης

Πρώτα ορίζουμε στο πάνω μέρος δύο Event() αντικείμενα σε ένα λεξικό deck_attached_events για να παρακολουθούμε κάθε deck. Η συνάρτηση param_deck_attached(deck_name) επιστρέφει ένα callback το οποίο, όταν μια συγκεκριμένη παράμετρος (deck.bcFlow2 ή deck.bcMultiranger) αλλάξει τιμή, μετατρέπει σε ακέραιο, ελέγχει αν είναι 1 (συνδεδεμένο) ή 0 (αποσυνδεδεμένο) και ενημερώνει αντίστοιχα το deck_attached_events και τυπώνει ένα μήνυμα στην κονσόλα. Στην πιο κάτω εικόνα παρουσιάζεται το αποτέλεσμα στην κονσόλα.

```

shrimp@Andreass-MacBook-Pro Drones % /usr/local/bin/python3 /Users/shrimp/Documents/Drones/tests/check_if_deck_is_attached.py
Flow Deck is attached!
Multi-ranger Deck is attached!

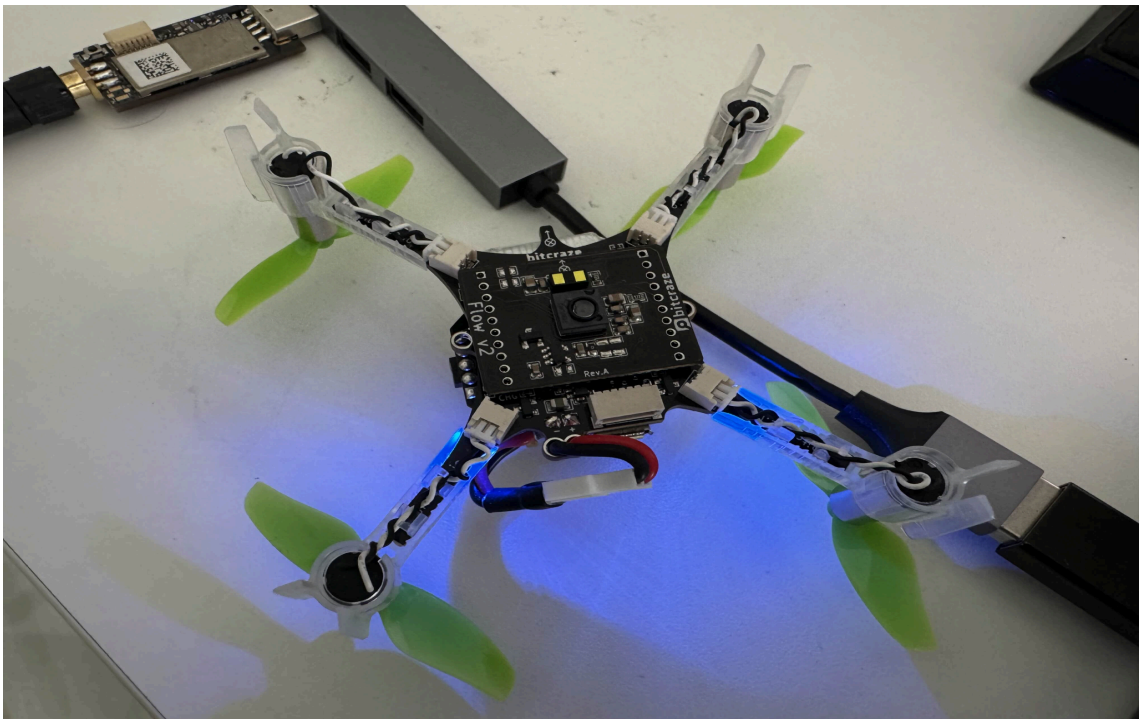
```

Εικόνα 2.12 Έξοδος Προγράμματος

Στις πιο κάτω εικόνες παρουσιάζεται το Crazyflie όπως είναι συναρμολογημένο, με τα Flow Deck (από κάτω) και Multi-Ranger Deck (από πάνω).



Εικόνα 2.13 Crazyflie με συνδεδεμένο το Multi-Ranger Deck από πάνω



Εικόνα 2.14 Crazyflie με συνδεδεμένο το Flow Deck από κάτω

Κεφάλαιο 3

Υλοποίηση Αλγορίθμων

3.1 Έλεγχος Πτήσης: Roll, Pitch, Yaw και Thrust	29
3.2 Εξερεύνηση Δωματίου με ένα Crazyflie	31
3.3 Επίπεδο Εφαρμογής	37
3.4 Εγγραφή του Λογισμικού Πάνω στο Crazyflie	41
3.5 Σμήνος Crazyflie - Η Ιδέα	42
3.6 Διεπαφή P2P	46
3.7 Δοκιμή P2P	46
3.8 Εξερεύνηση Δωματίου με Σμήνος Crazyflie	50

3.1 Έλεγχος Πτήσης: Roll, Pitch, Yaw και Thrust

Προτού περάσουμε στην περιγραφή των αλγορίθμων πλοήγησης, είναι σημαντικό να εξηγήσουμε τις βασικές κινητικές εντολές που ελέγχουν την κίνηση του Crazyflie. Οι τέσσερις αυτές εντολές (roll, pitch, yaw και thrust) αποτελούν τη γλώσσα με την οποία μιλάμε στο MEA, καθορίζοντας την κατεύθυνση και το ύψος της πτήσης.

- **Roll**

Είναι η περιστροφή γύρω από έναν οριζόντιο άξονα που περνά από το πίσω προς το μπροστινό μέρος του τετρακόπτερου. Με την εντολή roll, το Crazyflie “κυλάει” δεξιά ή αριστερά, μετακινώντας αντίστοιχα ολόκληρο το MEA προς τα πλάγια.

- **Pitch**

Είναι η περιστροφή γύρω από έναν οριζόντιο άξονα που διατρέχει το MEA από τα αριστερά προς τα δεξιά. Με την εντολή pitch, το Crazyflie “γέρνει” μπροστά

ή πίσω, επιτυγχάνοντας κίνηση προς τα εμπρός ή προς τα πίσω.

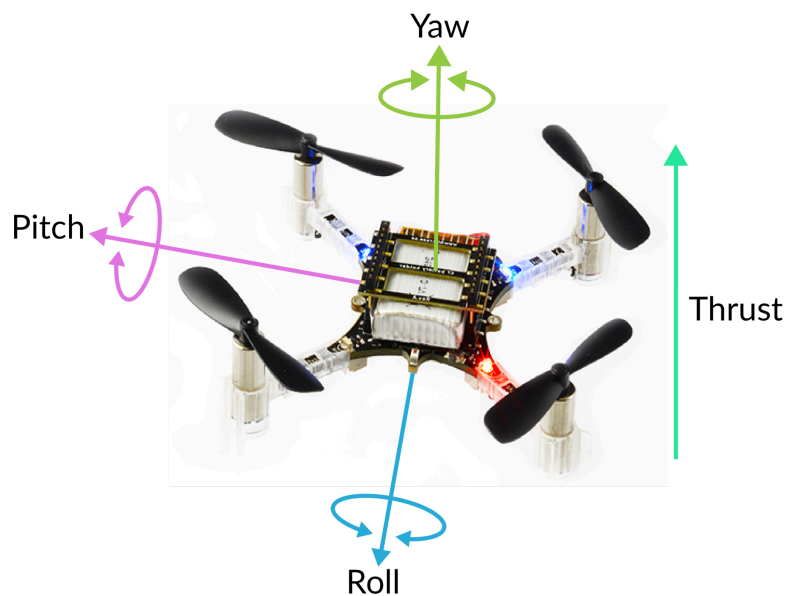
- **Yaw**

Είναι η περιστροφή γύρω από έναν κατακόρυφο άξονα. Με την εντολή yaw, το Crazyflie στρέφεται δεξιά ή αριστερά στον ορίζοντα, επιτρέποντας την αλλαγή της φοράς πτήσης ώστε το πιςινό μέρος να γίνει εμπρός.

- **Thrust**

Ελέγχει την κατακόρυφη ώθηση, ρυθμίζοντας το *ύψος* (altitude) του Crazyflie. Αυξάνοντας ή μειώνοντας την ώθηση, το MEA ανεβαίνει ή κατεβαίνει στον χώρο.

Η πιο κάτω εικόνα απεικονίζει τις τέσσερις βασικές κινήσεις του Crazyflie: Pitch, Roll, Yaw και Thrust.



Εικόνα 3.1 Crazyflie Yaw, Pitch, Roll

Η κίνηση Yaw είναι η κύρια κίνηση που χρησιμοποιείται στους αλγόριθμους μας.

3.2 Εξερεύνηση Δωματίου με ένα Crazyflie

Στο παρόν υποκεφάλαιο περιγράφεται ο απλός αλλά αποτελεσματικός αλγόριθμος WallFollowing για εξερεύνηση δωματίου που υλοποιήθηκε με τη βοήθεια της βιβλιοθήκης Crazyflie Python. Ο στόχος είναι να οδηγήσουμε το Crazyflie σε σταθερή απόσταση από έναν τοίχο, αξιοποιώντας το Multi-Ranger deck για αποφυγή σύγκρουσης με τον τοίχο. Επίσης χρησιμοποιείται το Flow Deck για σκοπούς σταθεροποίησης της πτήσης. Με τον συνδυασμό των μετρήσεων από το Multi-Ranger και το Flow Deck, ο αλγόριθμος διατηρεί σταθερή απόσταση από τον τοίχο, αλλά και σταθερό ύψος και θέση μέσα στον χώρο.

Η WallFollowing διαδικασία συνοψίζεται ως εξής:

1. Απογείωση σε προκαθορισμένο ύψος.
2. Συνεχής μέτρηση της απόστασης εμπρός με το Multi-Ranger και εφαρμογή απλού φίλτρου διάμεσης τιμής για την αφαίρεση *ακραίων τιμών* (outliers) από το σύνολο των μετρήσεων.
3. Όταν η απόσταση πέσει κάτω από ένα *όριο* (threshold), δηλαδή το Crazyflie βρίσκεται κοντά σε τοίχο, περιστροφή 90° για να ευθυγραμμιστεί το MEA παράλληλα στον τοίχο (yaw).
4. Διατήρηση στόχου απόστασης από τον τοίχο κατά το οριζόντιο πέρασμα, μέχρι να φτάσει στην επόμενη γωνιά.
5. Επαναληπτική εκτέλεση μέχρι την ολοκλήρωση της διαδρομής.

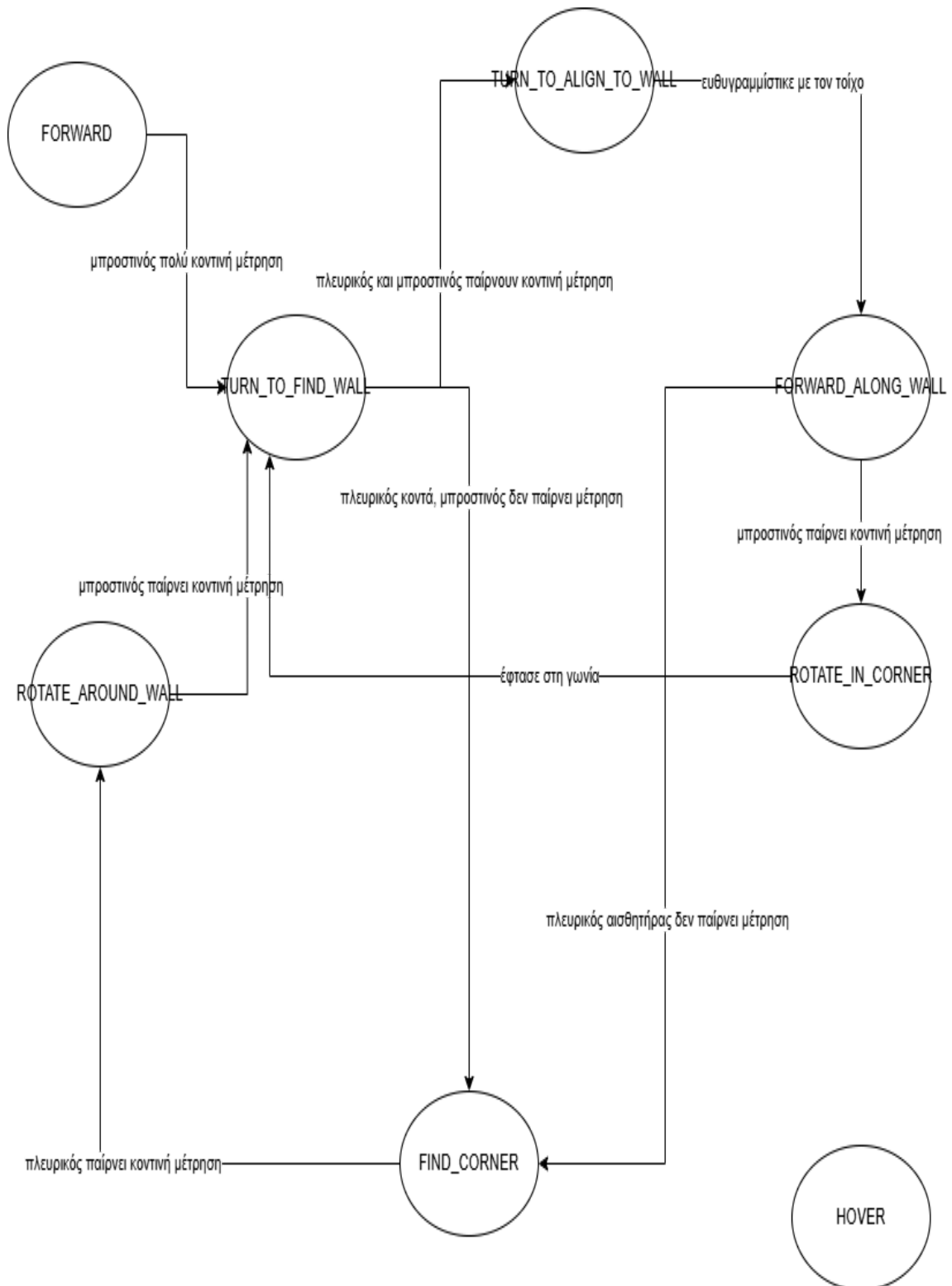
Ως βάση χρησιμοποιήθηκε μια μηχανή καταστάσεων WallFollowing από τη Bitcraze [10]

Έχουμε τις εξής καταστάσεις:

- Κατάσταση Forward: Η αρχική κατάσταση μετά την απογείωση. Το Crazyflie προχωρά μπροστά με ταχύτητα *max_forward_speed*.
- Κατάσταση Hover: Κρατάει θέση, μηδενική ταχύτητα. Χρησιμοποιείται σε παύσεις μέτρησης.
- Κατάσταση Turn To Find Wall: Γυρνα μέχρι 45 μοίρες από τον τοίχο, έτσι ώστε ο μπροστινός και ο πλευρικός αισθητήρας να μπορούν να ανιχνεύσουν τον τοίχο. Αυτό γίνεται για το λόγο ότι σε ένα πραγματικό περιβάλλον το MEA μπορεί να μην έχει πλησιάσει τον τοίχο ακριβώς μετωπικά. Μια στροφή 90 μοίρες αντί 45 μπορεί να μας αφήσει, ας πούμε, σε σχετική γωνία 88° ή 92° και έτσι υπάρχει πιθανότητα ο πλευρικός αισθητήρας μπορεί να βλέπει ελαφρώς έξω από την άκρη του τοίχου και να μην τον βλέπει καθόλου. Μόλις και οι δύο αισθητήρες έχουν μετρήσεις σε 45 μοίρες τότε υπολόγισε το *wall_angle*.
- Κατάσταση Turn To Align To Wall: Συνεχίζει την περιστροφή μέχρι να ευθυγραμμιστεί παράλληλα με τον τοίχο, στο υπολογισμένο *wall_angle*.
- Κατάσταση Forward Along Wall: Τώρα πετάει παράλληλα στο τοίχωμα με σταθερή ταχύτητα μπροστά, ελέγχοντας παράλληλα την απόστασή του από αυτόν.
- Κατάσταση Rotate In Corner: Ο μπροστινός αισθητήρας βρήκε εκ νέου τοίχο. Επομένως βρισκόμαστε σε γωνιά και το Crazyflie πρέπει να γυρίσει 90 μοίρες.
- Find Corner: Σε αυτή την κατάσταση μπαίνει μόλις χάσει ο πλάγιος αισθητήρας τον τοίχο, δηλαδή βρήκε εξωτερική γωνιά. Το Crazyflie θα γυρίσει μέχρι ο πλάγιος αισθητήρας να ξαναπάρει ένδειξη, δηλαδή βρήκε τον επόμενο τοίχο.
- Rotate Around Wall: Μόλις πάρει ένδειξη ο πλάγιος αισθητήρας τότε μπαίνουμε σε αυτή την κατάσταση. Εδώ το Crazyflie γυρνά μέχρι ο μπροστινός

αισθητήρας να πάρει ένδειξη και στη συνέχεια μπαίνουμε στη κατάσταση Turn To Find Wall.

Πιο κάτω παρουσιάζεται το διάγραμμα αυτής της μηχανής καταστάσεων:



Εικόνα 3.2 Διάγραμμα Μηχανής Καταστάσεων WallFollowing

Η μηχανή καταστάσεων είναι υλοποιημένη σε μια κλάση η οποία θα χρησιμοποιηθεί από ένα πρόγραμμα Python το οποίο θα δίνει τις εντολές κίνησης στο Crazyflie. Η κλάση της μηχανής καταστάσεων θα επιστρέφει στο πρόγραμμα 3 τιμές: *velocity_x* (m/s), *velocity_y* (m/s) και *yaw_rate* (rad/s)

Το Python πρόγραμμα που ελέγχει τη μηχανή καταστάσεων:

```
80 if __name__ == '__main__':
81     # Initialize the low-level drivers
82     cflib.crtp.init_drivers()
83
84     # Only output errors from the logging framework
85     logging.basicConfig(level=logging.ERROR)
86
87     keep_flying = True
88
89     wall_following = WallFollowing(
90         angle_value_buffer=0.1, reference_distance_from_wall=0.3,
91         max_forward_speed=0.3, init_state=WallFollowing.StateWallFollowing.FORWARD)
92
93     # Setup logging to get the yaw data
94     lg_stab = LogConfig(name='Stabilizer', period_in_ms=100)
95     lg_stab.add_variable('stabilizer.yaw', 'float')
96
97     cf = Crazyflie(rw_cache='./cache')
98     with SyncCrazyflie(URI, cf=cf) as scf:
99         # Arm the Crazyflie
100         scf.cf.platform.send_arming_request(True)
101         time.sleep(1.0)
102
103         with MotionCommander(scf) as motion_commander:
104             with Multiranger(scf) as multiranger:
105                 with SyncLogger(scf, lg_stab) as logger:
106
107                     # double ended queues, used for filtering the front and left sensor values
108                     front_buf = deque()
109                     left_buf = deque()
110
111                     while keep_flying:
112
113                         # initialize variables
114                         velocity_x = 0.0
115                         velocity_y = 0.0
116                         yaw_rate = 0.0
117                         state_wf = WallFollowing.StateWallFollowing.HOVER
118
119                         # get Yaw
120                         log_entry = logger.next()
121                         data = log_entry[1]
122                         actual_yaw = data['stabilizer.yaw']
123                         actual_yaw_rad = radians(actual_yaw)
124
125                         # get front range in meters
126                         front_range = handle_range_measurement(multiranger.front)
127                         left_range = handle_range_measurement(multiranger.left)
128
129                         # choose here the direction that you want the wall following to turn to
130                         wall_following_direction = WallFollowing.WallFollowingDirection.RIGHT
131                         side_range = left_range
132
133                         front_range_filtered = median_filter(front_buf, front_range)
134                         side_range_filtered = median_filter(left_buf, left_range)
135
136                         # get velocity commands and current state from wall following state machine
137                         velocity_x, velocity_y, yaw_rate = wall_following.wall_follower(
138                             front_range_filtered, side_range_filtered, actual_yaw_rad, wall_following_direction, time.time())
139
140                         # convert yaw_rate from rad to deg
141                         yaw_rate_deg = degrees(yaw_rate)
142
143                         motion_commander.start_linear_motion(
144                             velocity_x, velocity_y, 0, rate_yaw=yaw_rate_deg)
```

Εικόνα 3.3 Πρόγραμμα Python WallFollowing

Με λίγα λόγια, δημιουργεί ένα αντικείμενο *WallFollowing* που περιέχει τη λογική της μηχανής καταστάσεων και μετά μέσα σε έναν ατέρμονο βρόγχο, διαβάζει την τιμή του

μπροστινού αισθητήρα και του πλευρικού αισθητήρα (σε αυτή την περίπτωση του αριστερά, αφού το Crazyflie θα κατευθύνετε δεξιά). Επίσης φιλτράρει καθεμία από αυτές τις μετρήσεις με ένα απλό φίλτρο διάμεσης τιμής (βοηθά στη σταθεροποίηση της πτήσης αφού διαγράφει άκυρες/εσφαλμένες τιμές που μπορεί να στείλει ο αισθητήρας, εξηγείται στην επόμενη παράγραφο). Στη συνέχεια καλεί τη μέθοδο *wall_follower(...)* της μηχανής καταστάσεων, περνώντας τις φιλτραρισμένες αποστάσεις και το τρέχον yaw. Μόλις κάνει τους υπολογισμούς της η μηχανή καταστάσεων, λαμβάνει ως αποτέλεσμα τις τρεις εντολές κινήσεως: *velocity_x*, *velocity_y* (γραμμικές ταχύτητες) και *yaw_rate* (γωνιακή ταχύτητα). Και τέλος, με την εντολή *motion_commander.start_linear_motion(vx, vy, 0, rate_yaw=yaw_rate_deg)* στέλνει στο MEA τις ταχύτητες που επέστρεψε η λογική της μηχανής καταστάσεων.

Φίλτρο διάμεσης τιμής:

```
def median_filter(buf, val):
    buf.append(val)
    if len(buf) > 5: #keep 5 values
        buf.popleft()
    lst = list(buf)
    lst.sort()
    mid = len(lst) // 2
    return lst[mid]
```

Figure 3.4 Μέθοδος Φίλτρου Διάμεσης Τιμής

Η συνάρτηση που απεικονίζεται πιο πάνω εφαρμόζει ένα πολύ απλό φίλτρο διάμεσης τιμής σε μια ροή τιμών, διατηρώντας πάντα τις 5 πιο πρόσφατες. Το *buf* είναι μια *deque* (διπλής όψεως ουρά), που κρατάει τις 5 πιο πρόσφατες μετρήσεις και χρησιμοποιείται γιατί υποστηρίζει αποδοτικές προσθήκες και αφαιρέσεις από τις άκρες. Αρχικά, προσθέτει τη νέα τιμή *val* στο δεξί άκρο της ουράς *buf* και μετά, αν πλέον η ουρά έχει πάνω από 5 στοιχεία, αφαιρεί το παλαιότερο (αριστερά), έτσι ώστε να κρατάμε μόνο τις

5 πιο πρόσφατες τιμές. Στη συνέχεια μετατρέπει την ουρά σε κανονική λίστα και τη ταξινομεί σε αύξουσα σειρά, υπολογίζοντας το δείκτη του μεσαίου στοιχείου (για λίστα μήκους 5 είναι το στοιχείο στην τρίτη θέση) και το επιστρέφει. Αυτός ο διάμεσος απορρίπτει τυχόν μεμονωμένα ακραία σφάλματα στις μετρήσεις. Για παράδειγμα για τιμές [0.2, 0.3, 0.4, 0.5, 0.6, 5.0], δηλαδή μόλις μπει η άκυρη τιμή 5.0 από τον αισθητήρα, ο διάμεσος είναι 0.5 επομένως θα δοθεί η τιμή 0.5 στην μηχανή καταστάσεων. Όπως γίνεται αντιληπτό, αυτό το φίλτρο διάμεσης τιμής απορρίπτει μετρήσεις που δεν έχουν λογική. Εφαρμόζεται σε κάθε επανάληψη του βρόχου ώστε οι τιμές να είναι πολύ κοντά μεταξύ τους και αυτό βοηθά στην αξιοπιστία της πτήσης.

Ακολουθεί ένα σενάριο εκτέλεσης του αλγορίθμου με τιμές *reference_distance_from_wall*=0.3, *max_forward_speed*=0.3. Το Crazyflye απογειώνεται, προχωρά μπροστά και συναντά τοίχο.

Αρχικά δημιουργείται το αντικείμενο *WallFollowing* με αρχική κατάσταση την Forward (γραμμή 91). Ακολούθως ο *MotionCommander* (γραμμή 103) απογειώνει το MEA με διεύθυνση που είναι αποθηκευμένη στη μεταβλητή URI. Ακολούθως μέσα σε έναν ατέρμονο βρόχο, παίρνονται οι μετρήσεις από τους αισθητήρες, φιλτράρονται και δίνονται στη μέθοδο *wall_follower* του αντικειμένου η οποία είναι υπεύθυνη για τη λογική (πότε να γυρίσει, να προχωρήσει κτλ). Οπότε αρχικά είμαστε στην κατάσταση Forward άρα το MEA προχωρά αφού δεν βρήκε ακόμη τον τοίχο μπροστά του. Επιστρέφει σαν *velocity_x* = 0.3 (αφού για να προχωρήσει ευθεία πρέπει να κινηθεί στον άξονα τον x με 0.3 m/s. Το πρόγραμμα παίρνει αυτή την τιμή και με την εντολή *start_linear_motion(vx, vy, 0, rate_yaw=degrees(yaw_rate))* δίνει οδηγία στο Crazyflye να παει μπροστα με ταχύτητα *vx* = 0.3, *vy* = 0 και *yaw_rate* = 0. Τώρα αυτή η διαδικασία επαναλαμβάνεται μέχρι να βρεθεί ο τοίχος μέσα στο όριο (*reference_distance_from_wall*=0.3 m). Μόλις γίνει αυτό αλλάζει η κατάσταση σε Turn To Find Wall. Τώρα η μέθοδος *wall_follower* επιστρέφει την τιμή *yaw_rate* = 0.5 rad/s, *velocity_x*= 0 και *velocity_y*= 0.) έτσι ώστε να μπορεί να γυρίσει το Crazyflye χωρίς να κινείται. Ξανά, μόλις πάρει αυτές τις τιμές το πρόγραμμα την εντολή *start_linear_motion* για να κινηθεί το MEA. Στη συνέχεια, η κατάσταση σε Turn To Align To Wall και πιο μετά σε Forward Along Wall κτλ. Αυτός ο ατέρμονος βρόχος τρέχει συνεχώς, διαβάζοντας τις τιμές των αισθητήρες, καλώντας την *wall_follower* με

τις συγκεκριμένες τιμές και βάση του τι επιστρέφει η *wall_follower* (ταχύτητες προς άξονες και yaw rate) μετακινεί το Crazyflie.

Τα αποτελέσματα των πειραμάτων στο Κεφάλαιο 4

3.3 Επίπεδο Εφαρμογής

Μέχρι στιγμής, ο αλγόριθμος υλοποιήθηκε σε κεντρικό σύστημα μέσω της βιβλιοθήκης Python, με τον υπολογιστή να στέλνει συνεχώς εντολές πτήσης. Για την πλήρη αυτονομία του Crazyflie, όμως, απαιτείται η μεταφορά αυτής της λογικής απευθείας στο ενσωματωμένο λογισμικό του MEA. Στο παρόν υποκεφάλαιο θα περιγράψουμε τη διαδικασία ενημέρωσης και παραμετροποίησης του λογισμικού του Crazyflie, ώστε το WallFollowing να εκτελείται αποκλειστικά από τον ενσωματωμένο επεξεργαστή, χωρίς καμία εξάρτηση από εξωτερικό υπολογιστή κατά τη διάρκεια της πτήσης. Με αυτόν τον τρόπο το Crazyflie αποκτά πλήρη ικανότητα αυτονομίας.

Η μεταφορά του αλγορίθμου από το κεντρικό Python script στο ενσωματωμένο λογισμικό του Crazyflie αποτελεί πρόκληση, καθώς το λογισμικό του MEA είναι γραμμένο σε C/C++ και τρέχει σε ένα περιβάλλον περιορισμένων πόρων. Ευτυχώς, στο αποθετήριο της Bitcraze στο GitHub υπάρχει ήδη ένα παράδειγμα ενσωματωμένου λογισμικού που υλοποιεί αντίστοιχη λειτουργία, προσφέροντας μια έτοιμη βάση για τροποποιήσεις. Σε αυτή τη βάση μπορούμε να προσαρμόσουμε τις παραμέτρους όπως εμείς θέλουμε, π.χ. μέτρηση απόστασης [10].

Η τροποποίηση του ενσωματωμένου λογισμικού στο Crazyflie είναι μια πιο πολύπλοκη διαδικασία σε σύγκριση με το απλό πάτημα του “Run” σε ένα πρόγραμμα Python. Η γλώσσα του ενσωματωμένου λογισμικού κώδικα είναι σε C/C++ και πρέπει να γίνει φόρτωση απευθείας στη μνήμη του MEA.

Για να διευκολυνθεί η προσθήκη δικών μας λειτουργιών στο firmware του Crazyflie, η Bitcraze πρόσφατα λάνσαρε ένα πειραματικό Επίπεδο Εφαρμογής (App Layer) [11], το οποίο είναι ένα σύνολο λειτουργιών και APIs. Επιτρέπει την ενσωμάτωση δικού μας κώδικα χωρίς να χρειαστεί να τροποποιηθεί ο κύριος ενσωματωμένος κώδικας. Όταν ο κώδικας μεταγλωττιστεί με τη ρύθμιση *CONFIG_APP_ENABLE=y*, μετά την εκκίνηση

του Crazyflie καλείται η συνάρτηση `void appMain()`. Έτσι, με το Επίπεδο Εφαρμογής μπορούμε να υλοποιήσουμε και να δοκιμάσουμε απευθείας στον επεξεργαστή του Crazyflie τον δικό μας αλγόριθμο.

Για να μπει ο αλγόριθμος μέσα στο Crazyflie firmware θα χρειαστεί να γίνει συνδυασμός του Επιπέδου Εφαρμογής με **out-of-tree build** [12], δηλαδή δικός μας φάκελος ξεχωριστά (“out of tree”) από το αποθετήριο του επίσημου λογισμικού.

Το “out-of-tree build” γίνεται με τη βοήθεια του “*oot.mk*”, το οποίο ένα μέρος του επίσημου συστήματος build του Crazyflie που βρίσκεται στο φάκελο *tools/make/oot.mk* του αποθετηρίου του λογισμικού. Το *oot.mk* αναλαμβάνει να ενσωματώσει τον δικό μας φάκελο όπου είναι μέσα ο δικός μας κώδικας (με την συνάρτηση `appMain()`) στο κύριο σύστημα build του Crazyflie. Συγχωνεύει τις δικές μας ρυθμίσεις (π.χ. *CONFIG_APP_ENABLE=y*) με το βασικό *.config* του λογισμικού, ενεργοποιώντας έτσι το Επίπεδο Εφαρμογής. Στη συνέχεια αναζητά αυτόματα στον τρέχοντα φάκελο ένα αρχείο *Kbuild* και προσθέτει τις εγγραφές του (π.χ. *obj-y += my_app.o*) στα αρχεία του πηγαίου κώδικα του λογισμικού. Όλα αυτά γίνονται με τη βοήθεια ενός “Makefile”, το οποίο δηλώνει πού βρίσκεται το κύριο λογισμικό, δείχνει το δικό μας *app-config* αρχείο το οποίο περιλαμβάνει τη ρύθμιση *CONFIG_APP_ENABLE=y* και φορτώνει το “build system *oot.mk*”. Τέλος, το “*oot.mk*” καλεί το κεντρικό “Makefile” του Crazyflie, το οποίο με τη σειρά του ξεκινά τον μηχανισμό *Kbuild*. Ο *Kbuild* σκανάρει όλα τα *Kbuild* αρχεία: πρώτα αυτά του επίσημου λογισμικού, μετά τα δικό μας *Kbuild* με τον δικό μας κώδικα. Το δικό μας *Kbuild* περιλαμβάνει *obj-y += my_app.o* και έτσι γίνεται η συσχέτιση του δικού μας κώδικα με το επίσημο λογισμικό.

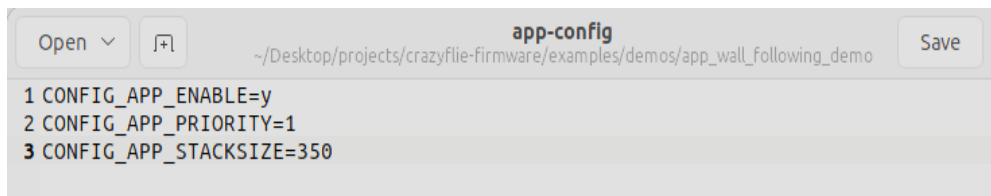
Έτσι, τρέχοντας την εντολή “make clean”, δημιουργείται ένα ένα ενσωματωμένο λογισμικό που περιλαμβάνει αυτόματα το δικό μας κώδικα χωρίς να γίνουν οποιεσδήποτε τροποποιήσεις στο λογισμικό.

Δηλαδή από το παράδειγμα που βρίσκεται στο αποθετήριο GitHub για το WallFollowing app [10] :

1. Δημιουργούμε έναν νέο φάκελο “**demo**” μέσα στο κύριο repository του firmware. Εκεί θα συγκεντρώσουμε όλο τον custom κώδικά μας, τις ρυθμίσεις

και τα αρχεία build.

2. Μέσα στο φάκελο “demo” δημιουργούμε ένα αρχείο **app-config** όπου ενεργοποιούμε το Επίπεδο Εφαρμογής (**CONFIG_APP_ENABLE=y**). (Εικόνα 3.5).



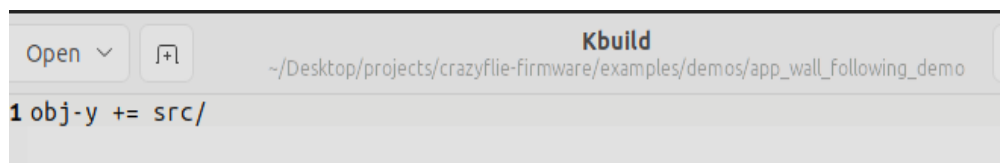
Εικόνα 3.5 App-config

3. Στον ίδιο φάκελο τοποθετούμε ένα Makefile που δείχνει στο κύριο λογισμικό και επίσης καλεί το σύστημα κατασκευής του Crazyflie και του λέει να χρησιμοποιήσει τις ρυθμίσεις του **app-config**. (Εικόνα 3.6).

```
12 # We want to execute the main Makefile for the firmware project,
13 # it will handle the build for us.
14 #
15 CRAZYFLIE_BASE := ../../..
16
17 #
18 # We override the default OOT_CONFIG here, we could also name our config
19 # to oot-config and that would be the default.
20 #
21 OOT_CONFIG := $(PWD)/app-config
22
23 include $(CRAZYFLIE_BASE)/tools/make/oot.mk
```

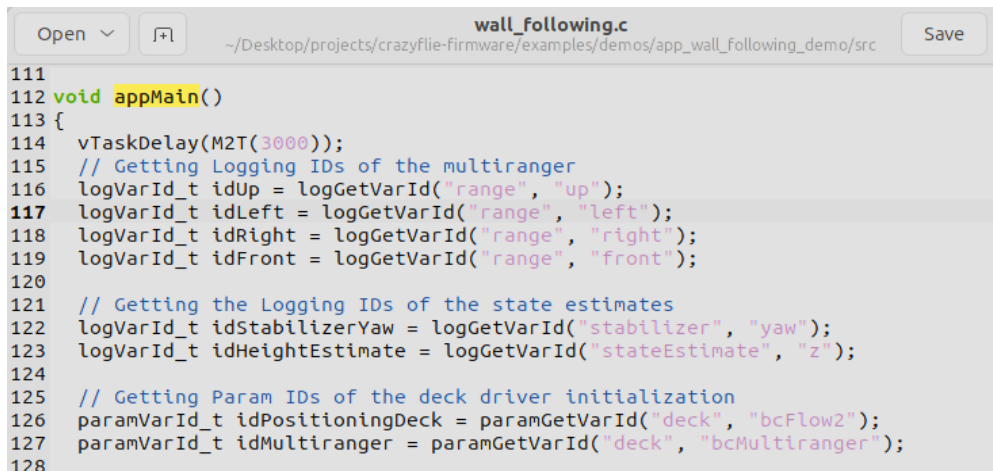
Εικόνα 3.6 Makefile

4. Δημιουργούμε ένα φάκελο “src”.
5. Δημιουργούμε ένα αρχείο Kbuild και μέσα βάζουμε “obj-y += src/” (Εικόνα 3.7).



Εικόνα 3.7 KBuild

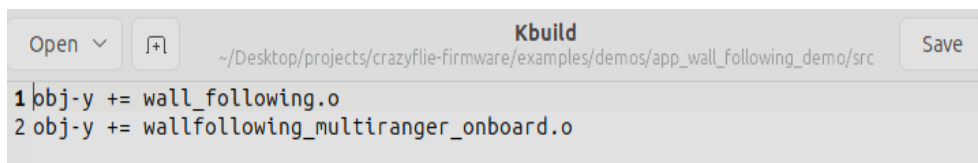
6. Μέσα στον φάκελο src βάζουμε τον C κώδικά μας, με την κύρια συνάρτηση appMain() (Εικόνα 3.8).



```
111
112 void appMain()
113 {
114     vTaskDelay(M2T(3000));
115     // Getting Logging IDs of the multiranger
116     logVarId_t idUp = logGetVarId("range", "up");
117     logVarId_t idLeft = logGetVarId("range", "left");
118     logVarId_t idRight = logGetVarId("range", "right");
119     logVarId_t idFront = logGetVarId("range", "front");
120
121     // Getting the Logging IDs of the state estimates
122     logVarId_t idStabilizerYaw = logGetVarId("stabilizer", "yaw");
123     logVarId_t idHeightEstimate = logGetVarId("stateEstimate", "z");
124
125     // Getting Param IDs of the deck driver initialization
126     paramVarId_t idPositioningDeck = paramGetVarId("deck", "bcFlow2");
127     paramVarId_t idMultiranger = paramGetVarId("deck", "bcMultiranger");
128
```

Εικόνα 3.8 Παράδειγμα WallFollowing appMain()

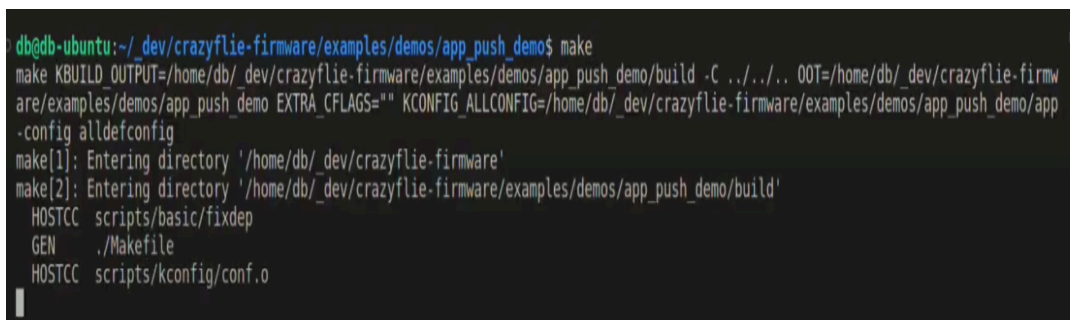
7. Μέσα στον φάκελο src δημιουργούμε ένα αρχείο Kbuild και δηλώνουμε μέσα του ποιο αρχείο αντικειμένου (object) θα συμπεριληφθεί στο τελικό λογισμικό (Εικόνα 3.9).



```
1 obj-y += wall_following.o
2 obj-y += wallfollowing_multiranger_onboard.o
```

Εικόνα 3.9 src KBuild

8. Τρέχουμε την εντολή “make clean” μέσα στο φάκελο “demo” (Εικόνα 3.10).



```
db@db-ubuntu:~/dev/crazyflie-firmware/examples/demos/app_push_demo$ make
make KBUILD_OUTPUT=/home/db/dev/crazyflie-firmware/examples/demos/app_push_demo/build -C ../../.. OOT=/home/db/dev/crazyflie-firmware/examples/demos/app_push_demo EXTRA_CFLAGS="" KCONFIG_ALLCONFIG=/home/db/dev/crazyflie-firmware/examples/demos/app_push_demo/app-config alldefconfig
make[1]: Entering directory '/home/db/dev/crazyflie-firmware'
make[2]: Entering directory '/home/db/dev/crazyflie-firmware/examples/demos/app_push_demo/build'
HOSTCC scripts/basic/fixdep
GEN ./Makefile
HOSTCC scripts/kconfig/conf.o
```

Εικόνα 3.10 Γραμμή Εντολών μετά το “make”

```

CC      vendor/libdw1000/src/libdw1000.o
CC      vendor/libdw1000/src/libdw1000Spi.o
LD      vendor/built-in.o
LD      firmware
COPY    cf2.hex
COPY    cf2.bin
Build for the cf2 platform!
Build 85:3d5129ff6afb (2023.11 +85) MODIFIED
Version extracted from git
Flash | 271052/1032192 (26%), 761140 free | text: 263956, data: 7096, ccmdata: 0
RAM   | 85620/131072 (65%), 45452 free | bss: 78524, data: 7096
CCM   | 62120/65536 (95%), 3416 free | ccmbss: 62120, ccmdata: 0
make[2]: Leaving directory '/home/db/ dev/crazyflie-firmware/examples/demos/app_push_demo/build'

```

Εικόνα 3.11 Γραμμή Εντολών κατά την ολοκλήρωση

Όταν τελειώσει η διαδικασία, στην τελική έξοδο παρατηρούμε με έντονα κόκκινα γράμματα τη λέξη **MODIFIED** (Εικόνα 3.11). Αυτό σημαίνει ότι το λογισμικό έχει τροποποιηθεί επιτυχώς, δηλαδή περιέχει τον δικό μας κώδικα που προστέθηκε μέσω του Επιπέδου Εφαρμογής.

3.4 Εγγραφή του Λογισμικού Πάνω στο Crazyflie

Η διαδικασία εγγραφής λογισμικού πάνω στο Crazyflie γίνεται μέσω του Crazyradio 2.0. Επίσης προαπαιτείται το Crazyflie να είναι σε λειτουργία “bootloader”. Για να βάλουμε σε λειτουργία “bootloader” το Crazyflie, πρώτα το απενεργοποιούμε αν είναι ενεργοποιημένο και στη συνέχεια πατούμε το κουμπί που το ενεργοποιούμε για περίπου 3 δευτερόλεπτα μέχρι τα δύο μπλε LED να αρχίσουν να αναβοσβήνουν.

Στη συνέχεια, αφού συνδέσουμε το Crazyradio στον υπολογιστή, τρέχουμε την εντολή **“make cload”** στη γραμμή εντολών. Το εργαλείο θα αναζητήσει αυτόματα ένα Crazyflie σε λειτουργία “bootloader” εντός εμβέλειας και θα γράψει το νέο λογισμικό στη μνήμη του (Εικόνα 3.12). Μετά από αυτό, αν η εγγραφή ήταν επιτυχής, το Crazyflie θα κάνει επανεκκινήσει αυτόματα και θα αρχικοποιηθεί με το νέο ενσωματωμένο λογισμικό.


```

python3 -m cfloader flash cf2.bin stm32-fw
Restart the Crazyflie you want to bootload in the next
10 seconds ...
nRF51 has: sd-s130 and requires None and upgrade provides None. Current bootload
er version is [2024.2.0] and upgrade provides [None]
No need to flash nRF soft device
Flashing 1 of 1 to stm32 (fw): 278803 bytes (273 pages) .....10.....10
.....10.....10.....10.....10.....10.....10.....10.....
..10.....10.....10.....10.....10.....10.....10.....10....
.....10.....10.....10.....10.....10.....10.....10.....10
.....10.....10.....10.....10.....10.....10...3

```

Εικόνα 3.12 Γραμμή Εντολών κατά την εγγραφή λογισμικού στη μνήμη του Crazyflie

Περισσότερες πληροφορίες για το “building και flashing” στο [13].

3.5 Σμήνος Crazyflie - Η Ιδέα

Η προηγούμενη ενότητα έδειξε πώς καταφέρνουμε να βάλουμε custom firmware πάνω στο Crazyflie, συγκεκριμένα το παράδειγμα του WallFollowing. Στο επόμενο βήμα, θα επεκτείνουμε αυτή τη διαδικασία σε πολλαπλά Crazyflie, γράφοντας σε καθένα από αυτά το ίδιο βασικό Επίπεδο Εφαρμογής WallFollowing. Παράλληλα, θα ενσωματώσουμε έναν απλό μηχανισμό τοπικής επικοινωνίας (peer-to-peer), ώστε κάθε MEA να ενημερώνει τους γείτονές του για τη θέση του. Με αυτόν τον τρόπο, τα μέλη του σμήνους θα μπορούν όχι μόνο να ακολουθούν τους τοίχους, αλλά και να αποφεύγουν συγκρούσεις μεταξύ τους. Στην παρούσα εργασία θα χρησιμοποιήσουμε ως αφετηρία και βασικό πλαίσιο μελέτης την εργασία “Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment” McGuire, k.; De Wagter, C.; Tuyls, K; Kappen, H.J.; de Croon, G. C. H. E. 2019, ScienceRobotics [14].

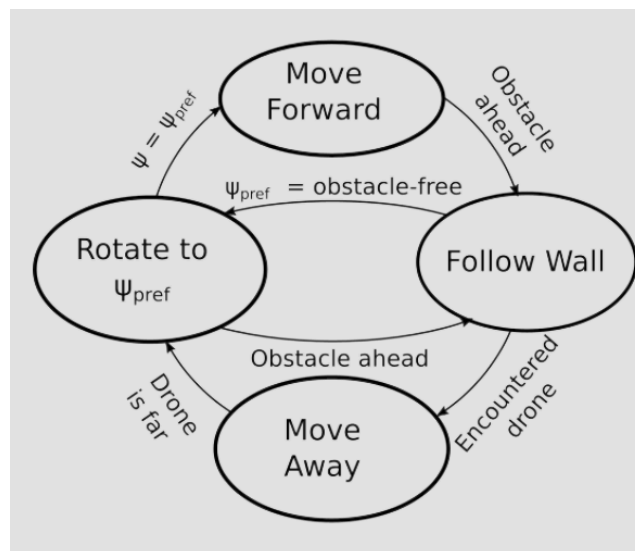
Οι συγγραφείς της προαναφερθέντας εργασίας [14], ανέπτυξαν έναν αλγόριθμο Ελάχιστης Πλοήγησης για σμήνη Crazyflie 2.0 (η προηγούμενη έκδοση του Crazyflie 2.1). Εμπνευσμένοι από τη φύση και βιολογικές στρατηγικές πλοήγησης (π.χ. μέλισσες) οι συγγραφείς ανέπτυξαν τον **Swarm Gradient Bug Algorithm (SGBA)**. Ο αλγόριθμος αντιδρά σε εμπόδια και ακολουθεί τοίχους (WallFollowing) και στηρίζεται σε:

- Προτεινόμενη κατεύθυνση (preferred heading) από το σημείο εκκίνησης.

- Wall following όταν ανιχνεύεται εμπόδιο.
- Αποφυγή συγκρούσεων και προσαρμογή προτιμώμενης κατεύθυνσης.
- Επιστροφή πίσω στην εκκίνηση, βάσει έντασης σήματος από ένα φάρο (beacon) βάσης. Ο φάρος εκπέμπει περιοδικά σήμα το οποίο λαμβάνουν τα Crazyflie.

Ο αλγόριθμος αυτός χρησιμοποιεί τα Flow Deck και Multi-Ranger deck.

Οι συγγραφείς υιοθέτησαν μια *Πεπερασμένη Μηχανή Καταστάσεων - ΠΜΚ* (finite state machine - FSM) για την ανάπτυξη του αλγορίθμου. Η ΠΜΚ χωρίζει τη συμπεριφορά σε τέσσερις καταστάσεις και ορίζει απλούς κανόνες μετάβασης μεταξύ τους.



Εικόνα 3.13 ΠΜΚ Αλγορίθμου SGBA

Η Εικόνα 3.13 παρουσιάζει τη ΠΜΚ του Αλγορίθμου SGBA, με τέσσερις διακριτές φάσεις που εναλλάσσονται ανάλογα με τα δεδομένα:

1. Κατάσταση 1: Move Forward

- Το MEA κινείται ευθύγραμμα προς την τρέχουσα προτεινόμενη κατεύθυνση (ψ_{pref}).
- Μετάβαση σε “Follow Wall” όταν ανιχνευθεί εμπόδιο μπροστά (multi-ranger value < threshold).

2. Κατάσταση 2: Follow Wall

- Εφαρμόζεται WallFollowing γύρω από το εμπόδιο (τοίχος), διατηρώντας σταθερή απόσταση.
- Μετάβαση σε “Rotate to ψ_{pref} ” μόλις ο αισθητήρας δείξει πως προς την προτεινόμενη κατεύθυνση (ψ_{pref}) δεν υπάρχει κάποιο εμπόδιο.
- Μετάβαση σε “Move Away” αν κατά την πορεία εντοπιστεί πολύ κοντινό MEA (σήμα υπερβαίνει το όριο).

3. Κατάσταση 3: Move Away

- Το Crazyflie απομακρύνεται για να δώσει προτεραιότητα στο γειτονικό MEA.
- Μετάβαση σε “Rotate to ψ_{pref} ” μόλις απομακρυνθεί αρκετά και η απόσταση γίνει ξανά ασφαλής.

4. Κατάσταση 4: Rotate to ψ_{pref}

- Το MEA στρέφεται ώστε να επανέλθει ακριβώς στην προτεινόμενη κατεύθυνση ψ_{pref} .

- Μετάβαση σε “Move Forward” όταν η στροφή ολοκληρωθεί ($\psi = \psi_{pref}$).

Όταν δοκίμαζαν αυτόν τον αλγόριθμο σμήνους, δεν υπήρχε ακόμη P2P υποστήριξη, γι’ αυτό επέλεξαν την λύση με τον φάρο. Για να αποφύγουν τα MEA να εκπέμπουν ταυτόχρονα στο ίδιο κανάλι, κάθε Crazyflie ρυθμίστηκε να έχει το δικό του κανάλι. Ο ενσωματωμένος πομποδέκτης **nRF51822** προγραμματίστηκε να σαρώνει όλα τα κανάλια και να στέλνει σε κάθε ένα ένα μικρό πακέτο με την ταυτότητά του.

Ο συγχρονισμός γινόταν από ένα κεντρικό φάρο, ο οποίος ήταν ένα Crazyradio: πριν από κάθε κύκλο εκπομπής, ο φάρος έστελνε ένα απλό πακέτο που ενημέρωνε όλα τα Crazyflie πότε να περάσουν σε λειτουργία εκπομπής και πότε να επανέλθουν σε λειτουργία λήψης. Έτσι, σε καθορισμένες στιγμές, κάθε MEA εξέπεμπε για μια σύντομη στιγμή στο κανάλι του και αμέσως μετά έπαιρνε σήματα από τα γειτονικά MEA. Μόλις ένα Crazyflie μετρούσε σήμα υψηλότερο από το όριο στο επιθυμητό του μονοπάτι, εκτελούσε άμεσα μια μικρή επιβράδυνση ή αλλαγή πορείας, ώστε να διατηρήσει επαρκή απόσταση.

Στα πειραματικά αποτελέσματα χρησιμοποιήθηκαν 6 Crazyflie σε ένα διάδρομος με κενά δωμάτια και παρατηρήθηκε ότι παρά την ύπαρξη παρεμβολών, hardware malfunctions και λανθασμένες μετρήσεις από τους αισθητήρες, η αποφυγή συγκρούσεων παρέμεινε εξαιρετικά αξιόπιστη: από 54 συνολικά περιπτώσεις στις οποίες MEA ήρθαν κοντά, καταγράφηκε μόνο μία σύγκρουση (98% επιτυχία). Όμως παρατηρήθηκε μειωμένη συνολική κάλυψη και μειωμένο ποσοστό επιστροφής των MEA πίσω στη βάση και αυτό οφείλεται σε πραγματικούς περιορισμούς, όπως η εξαντλημένη μπαταρία και σήμα που χάθηκε σε απομακρυσμένα δωμάτια.

Όλος ο κώδικας του Αλγορίθμου SGBA είναι ανοικτού κώδικα και έχει αναρτηθεί στο GitHub. Παρότι η αρχική υλοποίηση ήταν σχεδιασμένη για πολύ παλαιό firmware του Crazyflie 2.0 και δεν είναι απευθείας συμβατή με την τρέχουσα έκδοση του Crazyflie 2.1, μας παρέχει ένα στέρεο σημείο εκκίνησης.

3.6 Διεπαφή P2P

Όπως αναφέρθηκε νωρίτερα, η αρχική μέθοδος επικοινωνίας μεταξύ των Crazyflie ήταν σχετικά πολύπλοκη, καθώς απαιτούσε αλλαγές απευθείας στον κώδικα του firmware για να υλοποιηθούν τα broadcasts. Πρόσφατα όμως η Bitcraze κυκλοφόρησε την επίσημη *Διεπαφή P2P (Peer-to-Peer API)*, η οποία συνδυάζεται με το **Επίπεδο Εφαρμογής** και το **out-of-tree build**, επιτρέποντάς μας να προσθέτουμε κώδικα επικοινωνίας σε ξεχωριστά αρχεία εφαρμογής χωρίς να πειράζουμε τον πυρήνα του λογισμικού. Αυτή η απλοποιημένη προσέγγιση μειώνει δραστικά τον χρόνο ανάπτυξης και επιτρέπει τη γρήγορη και εύκολη συντήρηση του κώδικα μας.

Σήμερα η υποστήριξη Peer-to-Peer στο Crazyflie βρίσκεται ακόμη υπό ανάπτυξη. Η Διεπαφή P2P επιτρέπει προς το παρόν την αποστολή P2P πακέτων σε λειτουργία “**broadcast**” (δηλαδή στέλνει το ίδιο πακέτο σε όλα τα MEA ταυτόχρονα και δεν απευθύνεται σε συγκεκριμένο παραλήπτη και έτσι· οποιοδήποτε MEA είναι αρκετά κοντά και στο ίδιο κανάλι μπορεί να λάβει και να επεξεργαστεί το μήνυμα), ενώ στο μέλλον σχεδιάζεται η επέκταση σε λειτουργία “**unicast**” (δηλαδή να μπορεί να στέλνει πακέτο σε ένα μόνο, συγκεκριμένο παραλήπτη και το μήνυμα φέρει τη διεύθυνση ή ID του αποστολέα και του παραλήπτη, και μόνο ο στοχευμένος δέκτης το αποκωδικοποιεί και το επεξεργάζεται). Για περισσότερες πληροφορίες [15].

3.7 Δοκιμή P2P

Στη συνέχεια θα παρουσιάσουμε έναν απλό πείραμα για τη Διεπαφή P2P που θα χρησιμοποιηθεί στον κώδικα του σμήνους. Δύο Crazyflie που θα στέλνουν περιοδικά P2P πακέτα το ένα στο άλλο, μεταφέροντας επιλεγμένα δεδομένα (π.χ. ID) και ένα τρίτο Crazyflie, το οποίο θα κάνει τη δουλειά του “sniffer”, δηλαδή θα λαμβάνει όλα τα πακέτα P2P από τα άλλα δύο MEA κι απλώς θα καταγράφει και θα εμφανίζει στην κονσόλα λεπτομέρειες για τα πακέτα που λαμβάνει.

Καταρχάς, να αναφέρουμε τι είναι το σήμα RSSI (Received Signal Strength Indicator): πρόκειται για μια ένδειξη της ισχύος του ασύρματου σήματος που λαμβάνει ένας δέκτης και μετριέται σε dBm (decibel-milliwatts). Οι χαμηλότερες τιμές dBm

υποδεικνύουν ισχυρότερο σήμα. Στα Crazyflie, η τιμή RSSI παρέχεται άμεσα από τον ενσωματωμένο πομποδέκτη (**nRF51822**) κάθε φορά που λαμβάνεται ένα πακέτο. Θα αξιοποιήσουμε το RSSI για αποφυγή συγκρούσεων.

Πως χρησιμοποιουμε τη Διεπαφή P2P για τα δύο Crazyflie που ανταλλάσσουν πακέτα (Εικόνα 3.14):

```
void p2pCallbackHandler(P2PPacket *p)
{
    // extract the sender's ID
    uint8_t sender_id = p->data[0];

    // Measure the RSSI of the received packet
    measured_rssi = (p->rssi);

    // Debug print
    DEBUG_PRINT("Received packet from CF %d with RSSI %d dBm\n", sender_id, measured_rssi);
}
```

Εικόνα 3.14 Συνάρτηση P2P Callback

Κάθε φορά που φτάνει ένα P2P πακέτο, θα καλείτε αυτή η συνάρτηση. Εξάγετε το ID του αποστολέα και καταγράφεται η RSSI τιμή, ώστε να τη χρησιμοποιήσουμε αργότερα.

```

void appMain()
{
    // get the address of the Crazyflie and obtain the last byte to use as the ID
    uint64_t address = configblockGetRadioAddress();
    my_id = (uint8_t)(address & 0x000000FF);

    //register the callback function to receive P2P packets
    p2pRegisterCB(p2pCallbackHandler);

    // initialize the P2P packet
    static P2PPacket p_packet;
    p_packet.port = 0x00;

    while (true)
    {
        //prepare the data payload
        p_packet.data[0] = my_id; // sender's ID

        p_packet.data[1] = measured_rssi; // Measured RSSI from last received packet
        p_packet.size = 2;

        // Send the P2P packet
        radiolinkSendP2PPacketBroadcast(&p_packet);

        DEBUG_PRINT("Sent packet to CF %d with measured RSSI %d dBm\n", other_id, measured_rssi);

        // random delay between 900 ms and 1100 ms
        int delay = 1000 + (rand() % 200) - 100;
        vTaskDelay(M2T(delay));
    }
}

```

Εικόνα 3.15 Συνάρτηση appMain() για P2P

Πιο πάνω παρουσιάζεται η συνάρτηση appMain (Εικόνα 3.15). Αυτή η συνάρτηση είναι μέρος του **Επίπεδο Εφαρμογής** του Crazyflie και θα εκτελεστεί αμέσως μετά την εκκίνηση του Crazyflie. Αρχικά μόλις γίνη κλήση διαβάζουμε τη διεύθυνση του Crazyflie και χρησιμοποιούμε το τελευταίο byte ως το *my_id*. Η εντολή *p2pRegisterCB(p2pCallbackHandler)* δηλώνει ότι κάθε φορά που θα φτάνει ένα P2P πακέτο, το λογισμικό θα καλεί τη συνάρτηση *p2pCallbackHandler()* για επεξεργασία του πακέτου. Στη συνέχεια φτιάχνουμε το *P2PPacket* που θα γεμίζουμε επανειλημμένα με δεδομένα και θα στέλνουμε. Η θύρα που χρησιμοποιείται είναι η 0. Μετά, σε ένα ατέρμονο βρόχο προετοιμάζεται το *φορτίο* (payload) που θα περιέχει το id του MEA, το RSSI από την προηγούμενη παραλαβή πακέτου (αυτό θα το χρησιμοποιήσει ο sniffer για να μπορούμε να δούμε τις τιμές RSSI που λαμβάνουν τα 2 Crazyflie). Επίσης δηλώνουμε το μέγεθος του πακέτου. Τέλος στέλνουμε το πακέτο με την εντολή *radiolinkSendP2PPacketBroadcast(&p_packet)* και προσθέτουμε τυχαία καθυστέρηση

μέχρι να ξαναστείλει το Crazyflie άλλο πακέτο ώστε να αποφύγουμε ταυτόχρονες εκπομπές.

```
void p2pCallbackHandler(P2PPacket *p)
{
    // extract data from the packet
    uint8_t sender_id = p->data[0]; //id of the drone that sent the packet
    int8_t measured_rssi = (int8_t)p->data[1]; // RSSI measured by the sender

    // Print the information to the console
    consolePrintf("CF %d to CF %d: Measured RSSI: %d dBm\n", sender_id, receiver_id, measured_rssi);
}

void appMain()
{
    consolePrintf("Starting P2P Sniffer Application...\n");

    // Register the callback function to receive P2P packets
    p2pRegisterCB(p2pCallbackHandler);

    while (true)
    {
        // The sniffer waits and processes incoming P2P packets
        vTaskDelay(M2T(100)); // small delay to prevent CPU overuse
    }
}
```

Εικόνα 3.16 P2P Sniffer

Παρόμοια, ο sniffer ακούει για τα εισερχόμενα P2P πακέτα αλλά δεν στέλνει κανένα δικό του πακέτο. Μπαίνει σε ατέρμονο βρόχο, όπου απλώς μια σύντομη καθυστέρηση κάθε 100 ms για να μην φορτώνεται αχρείαστα ο επεξεργαστής, ενώ το API καλεί αυτόματα το callback για κάθε πακέτο που φτάνει. Στη συνάρτηση *p2pCallbackHandler*, διαβάζει από το φορτίο το *sender_id* και το *measured_rssi* που έστειλε ο αποστολέας, και το εκτυπώνει στην κονσόλα του Πελάτη Crazyflie έτσι ώστε να μπορούμε να βλέπουμε πληροφορίες για τα πακέτα που ανταλλάσσονται και να έχουμε μια πιο ξεκάθαρη εικόνα (Εικόνα 3.16).

Με αυτή τη δομή, το **Επίπεδο Εφαρμογής** αναλαμβάνει την peer-to-peer επικοινωνία. Δηλαδή κάθε MEA ακούει RSSI, στέλνει το δικό του ID και την τελευταία μετρούμενη RSSI, χωρίς να πειράζουμε το ενσωματωμένο λογισμικό του Crazyflie.

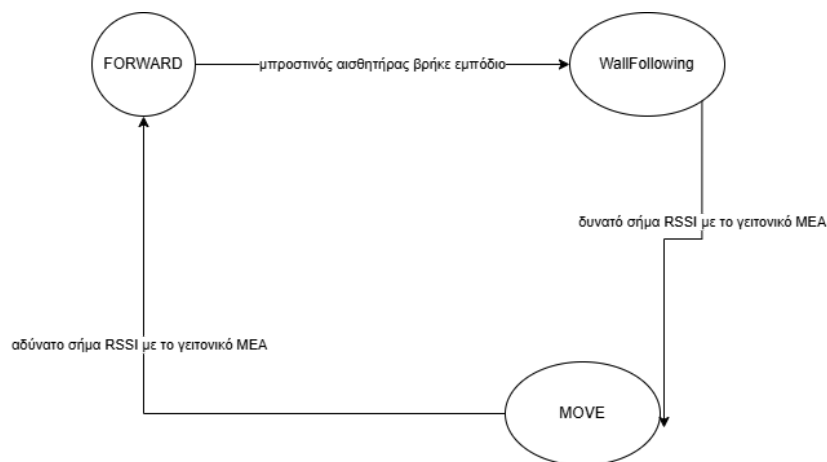
3.8 Εξερεύνηση Δωματίου με Σμήνος Crazyflie

Λόγω των περιορισμών στον χώρο δοκιμών και των προβλημάτων με την αξιοπιστία των πτήσεων, επιλέξαμε να υλοποιήσουμε μια πιο απλή έκδοση του συγκεκριμένου αλγορίθμου που παρουσιάζεται στο Κεφάλαιο 3.5 Κάθε Crazyflie εκτελεί WallFollowing σε συνδυασμό με P2P επικοινωνία για την αποφυγή σύγκρουσης.

Θα χρησιμοποιηθεί μια μηχανή καταστάσεων παρόμοια με αυτή που παρουσιάζεται στο Κεφάλαιο 3.2. Οι καταστάσεις είναι η ακόλουθες:

1. Κατάσταση Forward: Το MEA κινείται προς τα εμπρός. Μετάβαση σε Wall Following όταν ο αισθητήρας μπροστά ανιχνεύσει τοίχο σε απόσταση μικρότερη από $\text{ref_distance_from_wall} + 0.2\text{m}$
2. Κατάσταση Wall Following: Μόλις το state machine μεταβεί σε αυτή την κατάσταση, καλείται η συνάρτηση wall_follower που είναι port του state machine που είχαμε αναλύσει στο Κεφάλαιο 3.2 από Python σε C. Πρόκειται για ακριβώς την ίδια υλοποίηση.
3. Κατάσταση Move: το MEA εκτελεί μόνο περιστροφή στον άξονα yaw ($\text{vel}_w = 0.5$) ώστε να απομακρυνθεί από την πορεία του υψηλότερης προτεραιότητας MEA. Επιστροφή στην κατάσταση Forward όταν το RSSI επανέλθει πάνω από το όριο.

Παρακάτω παρουσιάζεται το διάγραμμα της μηχανής καταστάσεων.



Εικόνα 3.17 Διάγραμμα Μηχανής Καταστάσεων για WallFollowing και αποφυγή

Πιο κάτω παρουσιάζεται ο κώδικας της μηχανής καταστάσεων:

```
int wall_follower_and_avoid_controller(float *vel_x, float *vel_y, float *vel_w, float front_range, float left_range,
float right_range, float current_heading, uint8_t rssi_other_drone, uint8_t other_priority)
{
    // Initialize static variables
    static int state = 1;
    static int rssi_collision_threshold = 43;

    // if it is reinitialized
    if (first_run) {
        state = 1;
        float t = usecTimestamp() / 1e6;
        state_start_time = t;
        first_run = false;
    }

    if (state == 1) { //FORWARD
        // if front range is close, start wallfollowing
        if (front_range < ref_distance_from_wall + 0.2f) {
            wall_follower_init(ref_distance_from_wall, 0.5, 3);
            state = transition(2); //wall_following
        }
    } else if (state == 2) { //WALL_FOLLOWING
        if (rssi_other_drone < rssi_collision_threshold) {
            if (other_priority > my_priority) //if other priority is higher than mine then MOVE
                state = transition(3);
        }
    } else if (state == 3) { //MOVE
        if (rssi_other_drone > rssi_collision_threshold) {
            state = transition(1);
        }
    }

    float temp_vel_x = 0;
    float temp_vel_y = 0;
    float temp_vel_w = 0;

    if (state == 1) { //FORWARD
        // forward max speed
        temp_vel_x = 0.3;
    } else if (state == 2) { //WALL_FOLLOWING
        //Get values from wall_follower
        wall_follower(&temp_vel_x, &temp_vel_y, &temp_vel_w, front_range, left_range, current_heading, local_direction);
    } else if (state == 3) { //MOVE
        temp_vel_w = 0.5f; //just turn clockwise
    }

    *vel_x = temp_vel_x;
    *vel_y = temp_vel_y;
    *vel_w = temp_vel_w;
}
```

Εικόνα 3.18 Μηχανή Καταστάσεων για Wall Following και αποφυγή

Αρχικά η μέθοδος παίρνει σαν είσοδο τα ακόλουθα:

- *vel_x, vel_y, vel_w*: ταχύτητες (x, y, yaw) που θα κινηθεί το MEA σε m/s και rad/s αντίστοιχα.
- *front_range, left_range, right_range*: μετρήσεις απόστασης σε κάθε πλευρά.

- *current_heading*: η τρέχουσα κατεύθυνση (yaw).
- *rss_i_other_MEA*: το τελευταίο μετρούμενο RSSI του γειτονικού MEA.
- *other_priority*: η προτεραιότητα του γειτονικού MEA.

Γίνεται αρχικοποίηση του *static int rss_i_collision_threshold* = 43 (Το όριο RSSI κάτω από το οποίο θεωρούμε ότι ένα άλλο MEA είναι πολύ κοντά).

Πρώτη κατάσταση είναι η Forward και οι μεταβάσεις γίνονται με εξής τρόπο. Όταν η τιμή του αισθητήρα μπροστά είναι *front_range* < *ref_distance_from_wall* + 0.2, τότε καλούμε *wall_follower_init* για να θέσουμε τις παραμέτρους για το Wall Following και μεταβαίνουμε στη κατάσταση 2. Ενώ βρισκόμαστε στη κατάσταση 2, δηλαδή κάνουμε Wall Following, αν το RSSI του άλλου MEA πέσει κάτω από το όριο και η δική μας προτεραιότητα είναι χαμηλότερη (*other_priority* > *my_priority*), σημαίνει ότι πρέπει να κάνουμε χώρο στο άλλο MEA και έτσι γίνεται η μετάβαση στην κατάσταση 3. Ενώ βρισκόμαστε στην κατάσταση 3, όταν το RSSI επανέλθει πάνω από το όριο (*rss_i_other_MEA* > *rss_i_collision_threshold*) τότε επιστρέφουμε στην κατάσταση 1.

Σε κάθε κατάσταση εκτελούνται και ενέργειες και ανάλογα με την τρέχον κατάσταση, γεμίζουμε τις μεταβλητές *temp_vel_x*, *temp_vel_y*, *temp_vel_w*. Στην κατάσταση 1, θετούμε *temp_vel_x* = 0.3 (κίνηση 0.3 m/s στον άξονα τον X, δηλαδή ευθεία). Στην κατάσταση 2, καλούμε την συνάρτηση *wall_follower(&temp_vel_x, &temp_vel_y, &temp_vel_w, ...)* που υλοποιεί την μηχανή κατάσταση WallFollowing (Κεφάλαιο 3.2) σε C. Λειτουργεί με παρόμοιο τρόπο, δηλαδή γίνεται η είσοδος με δείκτες (π.χ. *&temp_vel_x*) επειδή η συνάρτηση *wall_follower* δεν επιστρέφει απευθείας τιμές, αλλά γράφει τις νέες εντολές ταχύτητας μέσα στις μεταβλητές που της δίνουμε, όπως θα γίνει και με τη μηχανής καταστάσεων του αλγορίθμου WallFollowing. Τέλος, στη κατάσταση 3, δίνουμε μόνο yaw (*temp_vel_w* = 0.5 rad/s) ώστε να στρίψουμε επί τόπου.

Αφού ολοκληρωθούν οι υπολογισμοί για τις ταχύτητες, οι τιμές αυτές ανατίθενται στις μεταβλητές εξόδου που δέχονται οι δείκτες *vel_x*, *vel_y* και *vel_w*. Αυτές οι παράμετροι της συνάρτησης έχουν δοθεί ως είσοδοι στην συνάρτηση της Μηχανής Καταστάσεων,

δηλαδή εισάγονται με την κλήση, τροποποιούνται εντός της συνάρτησης και μετά επιστρέφονται στον κώδικα που την κάλεσε, ο οποίος παρουσιάζεται πιο κάτω.

```
#define MY_PRIORITY 2 // για το δεύτερο drone

void appMain(void *param) {

    // register P2P callback
    p2pRegisterCB(p2pcallbackHandler);

    // calculate own ID from radio address
    uint64_t address = configblockGetRadioAddress();
    uint8_t my_id = (uint8_t)(address & 0xFF);

    // Prepare broadcast packet containing ID and priority
    static P2PPacket p_reply;
    p_reply.port = 0;
    p_reply.data[0] = my_id;
    p_reply.data[1] = MY_PRIORITY;
    p_reply.size = 2;

    // Initialize controller
    init_wall_follower_and_avoid_controller(0.5f, 0.5f, 1);

    systemWaitStart();

    while (1) {

        // Read range sensors and heading
        float front = rangeGet(rangeFront) / 1000.0f;
        float left = rangeGet(rangeLeft) / 1000.0f;
        float right = rangeGet(rangeRight) / 1000.0f;
        int varid = logGetVarId("stabilizer", "yaw");
        float heading = logGetFloat(varid) * M_PI / 180.0f;

        // Compute velocity commands via wall-following with avoid
        float vx=0, vy=0, vw=0;
        wall_follower_and_avoid_controller(
            &vx, &vy, &vw,
            front, left, right,
            heading,
            rssi_inter,
            other_priority
        );

        // Convert yaw rate to deg/s
        float yaw_rate = vw * 180.0f / M_PI;

        // Send motion commands
        vel_command(&setpoint_BG, vx, vy, yaw_rate, nominal_height);
        commanderSetSetpoint(&setpoint_BG, STATE_MACHINE_COMMANDER_PRI);

        // Periodically broadcast ID packet
        if (usecTimestamp() >= radioSendBroadcastTime + 500000) {
            radiolinkSendP2PPacketBroadcast(&p_reply);
            radioSendBroadcastTime = usecTimestamp();
        }
    }
}
```

Εικόνα 3.19 Συνάρτηση appMain() για WallFollowing και αποφυγή

Εξηγώντας το κύριο πρόγραμμα (**appMain**), αρχικά καταχωρούμε το callback για τα P2P πακέτα με την κλήση *p2pRegisterCB(p2pcallbackHandler)*; ώστε να δεχόμαστε τα μηνύματα από τα άλλα MEA. Έπειτα διαβάζουμε τη διεύθυνση του Crazyflie και εξάγουμε το τελευταίο byte, που χρησιμοποιείται ως μοναδικό *my_id*. Στη συνέχεια, ετοιμάζουμε το πακέτο P2P (*p_reply*), θέτοντας τη θύρα σε 0, και αποθηκεύοντας στις δύο πρώτες θέσεις (*data[0]* και *data[1]*) το *my_id* και την προτεραιότητά μας (*MY_PRIORITY*). Προτού ξεκινήσει ο ατέρμονος βρόχος, καλούμε την *init_wall_follower_and_avoid_controller(0.5f, 0.5f, 1)*, για να αρχικοποιήσουμε τις παραμέτρους.

Στη συνέχεια μέσα στον ατέρμονο βρόχο, μετράμε τις αποστάσεις των αισθητήρων και τα άλλα δεδομένα που θα χρειαστούμε. Ακολουθώς, καλούμε τη *wall_follower_and_avoid_controller* (τη μηχανή καταστάσεων), περνώντας δείκτες σε τρεις μεταβλητές ταχυτήτων (*vx*, *vy*, *vw*), τις μετρήσεις αποστάσεων, τον προσανατολισμό, το μετρούμενο RSSI του κοντινότερου MEA (*rss_i_inter*) και την προτεραιότητα του άλλου (*other_priority*). Το *rss_i_inter* και *other_priority* υπολογίζονται από τον *p2pcallbackHandler* που θα εξηγηθεί αργότερα). Η συνάρτηση αυτή επιστρέφει τις απαιτούμενες γραμμικές ταχύτητες στον άξονα X και Y και τη γωνιακή ταχύτητα *yaw* (σε rad/s) όπως εξηγήθηκε προηγουμένως. Μετατρέπουμε τη γωνιακή ταχύτητα από rad/s σε deg/s επειδή η *commanderSetSetpoint* δέχεται τη γωνιακή ταχύτητα σε μοίρες το δευτερόλεπτο και τέλος, γίνεται η αποστολή εντολών πτήσης: Γεμίζουμε τη δομή *setpoint_BG* με τις υπολογισμένες ταχύτητες και το επιθυμητό ύψος (*nominal_height*), και την υποβάλλουμε στον Flight Commander με *commanderSetSetpoint* για να πετάξει το MEA. Κάθε περίπου 500 ms, στέλνουμε το *p_reply* μέσω του *radiolinkSendP2PPacketBroadcast*, ενημερώνοντας έτσι τα υπόλοιπα MEA για το *my_id* και το *MY_PRIORITY*.

```

void p2pcallbackHandler(P2PPacket *p) {
    // Extract sender ID and measured RSSI
    id_inter_ext = p->data[0];
    other_priority = p->data[1];
    rssi_inter    = p->rssi;
}

```

Εικόνα 3.20 Χειριστής P2P για Wal Following και αποφυγή

Πιο πάνω (Εικόνα 3.20) παρουσιάζεται ο χειριστής *p2pcallbackHandler*; ο οποίος πολύ απλά όταν λάβει το πακέτο, εξάγει το ID και την προτεραιότητα του αποστολέα αλλά και υπολογίζει την τιμή RSSI και τα αποθηκεύει όλα στις αντίστοιχες μεταβλητές.

Παράλληλα, ένα Crazyflie “sniffer” τρέχει δικιά του εφαρμογή που απλώς καταχωρεί το ίδιο callback αλλά δεν στέλνει πακέτα. Ο sniffer θα εκτυπώνει στην κονσόλα όλα τα P2P πακέτα που λαμβάνει, εμφανίζοντας ποιος στέλνει, δίνοντάς μας εικόνα των ανταλλαγών μεταξύ των δύο (ή περισσότερων) MEA που πετούν.

Κεφάλαιο 4

Πειραματική Εφαρμογή

4.1 Ετοιμασία Χώρου	56
4.2 Προκλήσεις	58
4.3 Αποτελέσματα WallFollowing Ενός Crazyflie	60
4.4 Αποτελέσματα Swarm	63

4.1 Ετοιμασία Χώρου

Στο πλαίσιο των πειραμάτων μας, αξιοποιήσαμε ένα χώρο ο οποίος είναι ένα δωμάτιο του σπιτιού μου, διαμορφωμένος με χάρτινα κουτιά ώστε να προσομοιάζει ένα τετραγωνικό περιβάλλον κλειστού χώρου με τοίχους και γωνίες.

Οι δοκιμές WallFollowing και χαρτογράφησης εκτελέστηκαν σε έναν κλειστό, τετραγωνικό χώρο διαστάσεων περίπου 4×4 m, διαμορφωμένο με χάρτινες συσκευασίες (κουτιά) και σεντόνια, ώστε να προσομοιώνει τοίχους και πιθανές γωνίες. Η επιφάνεια του δαπέδου ήταν επίπεδη, χωρίς χαλιά για να είναι σταθερές οι μετρήσεις από το Flow Deck. Στις γωνίες τοποθετήθηκαν ελαφρά εμπόδια, όπως στήλες από κουτιά και πρόχειρες διαχωριστικές κατασκευές, για να ελεγχθεί η ικανότητα του αλγορίθμου να αναγνωρίζει αλλαγές διεύθυνσης όταν πλησιάζει σε γωνία. Ο φωτισμός παρέμεινε σταθερός σε όλη τη διάρκεια των δοκιμών. Τα χάρτινα κουτιά ή οι “τοίχοι” είχαν ύψος 0,5 m, αρκετό για να διασφαλίζεται ότι ο μπροστινός αισθητήρας θα ανιχνεύει έγκαιρα την προσέγγιση, αλλά χωρίς να συγκρούεται με υπερβολικά υψηλά εμπόδια. Η απλή αυτή διαμόρφωση επέτρεψε την επαναληψιμότητα των πτήσεων και την εύκολη αναδιαρρύθμιση του χώρου για περαιτέρω πειράματα.

Ωστόσο, οι δοκιμές δεν ήταν τόσο απλές όσο θα φανταζόταν κανείς. Γρήγορα διαπιστώσαμε ότι τόσο το Crazyflie όσο και ο αισθητήρας Multi-Ranger αντιμετωπίζουν δυσκολίες όταν προσπαθούν να ακολουθήσουν τοίχους με αντικείμενα ή έπιπλα ενδιάμεσα, ειδικά σε στενούς χώρους ή κοντινές γωνίες και μικρά ανοίγματα μεταξύ καναπέδων και τραπεζιών. Γι' αυτόν ακριβώς τον λόγο επιλέξαμε να χρησιμοποιήσουμε χάρτινα κουτιά και σεντόνια για τη διαμόρφωση των τοίχων, δημιουργώντας ευθείες και καθαρές επιφάνειες χωρίς εμπόδια στο ενδιάμεσο. Ο διαμορφωμένος χώρος παρουσιάζεται στις Εικόνες 4.1 και 4.2.



Εικόνα 4.1 Ο χώρος (βόρεια πλευρά)



Εικόνα 4.2 Ο χώρος (νότια πλευρά)

4.2 Προκλήσεις

Ήδη αναφέρθηκε μία από τις βασικές προκλήσεις, που ήταν η εύρεση ενός χώρου χωρίς πολλά εμπόδια. Έπρεπε να έχουμε ένα άδειο δωμάτιο με ευθείες επιφάνειες, γι' αυτό χρησιμοποιήσαμε χαρτοκιβώτια και σεντόνια για τους τοίχους. Πέρα όμως από τη διαμόρφωση του περιβάλλοντος, πολλά άλλα ζητήματα εμφανίστηκαν κατά τη διάρκεια των πειραμάτων.

Καταρχάς, όταν το Crazyflie πλησίαζε το έδαφος σε ύψος μικρότερο των 30–40 cm, παρουσίαζε το φαινόμενο του “ground effect”, δηλαδή αστάθειες λόγω συμπίεσης του αέρα ανάμεσα στους έλικες και το έδαφος. Συχνά προκαλούσε “drift”, δηλαδή απρόβλεπτη μετατόπιση δεξιά και αριστερά κατά την απογείωση. Πολλές φορές, δεν απογειωνόταν ομαλά και έπεφτε πριν καν προλάβει να πετάξει. Επίσης όπως αναφέρθηκε και προηγουμένως το Multi-Ranger χειριζόταν επιτυχώς ευθείες, επίπεδες επιφάνειες, δεν κατάφερνε όμως να ανιχνεύσει λεπτά ή ανώμαλα εμπόδια (π.χ. φυτά).

Ακόμη το Flow Deck v2, σε χαμηλό φωτισμό και δάπεδο χωρίς μοτίβα δυσκολεύεται να κρατήσει τη πτήση σταθερή.

Σε ορισμένες πτήσεις, το Crazyflie παρέκκλινε αριστερά ή δεξιά κατά την πτήση. Ελέγξαμε την τοποθέτηση της μπαταρίας (έπρεπε να είναι κεντραρισμένη), τη σωστή ελεύθερη περιστροφή και ισορροπία των έλικων καθώς πολλές φορές συλλέγει σκόνη και τρίχες μεταξύ έλικων και κινητήρα.

Επίσης οι μετρήσεις RSSI επηρεάζονταν έντονα από παρεμβολές και αντανάκλαση σήματος. Οι ασταθής τιμές RSSI οδήγησαν σε λανθασμένες αποφάσεις και τοπικούς ατέρμονους βρόγχους, σε αρκετές περιπτώσεις το MEA απαιτούσε επανεκκίνηση.

Οι πειραματικές δοκιμές απαιτούσαν διαρκείς επαναλήψεις και προσαρμογές προκειμένου το σύστημα να λειτουργεί, δεδομένων των ατελειών τόσο του υλικού όσο και των αισθητήρων. Σε πολλές περιπτώσεις τα Crazyflie εξετράπηκαν στην πορεία τους, χτυπώντας ανεπιτυχώς στους τοίχους ή εκτελώντας ξαφνικές αστοχίες ελέγχου που τα έφερναν εκτός πορείας.

Ταυτόχρονα, οι μπαταρίες εξαντλούνταν γρήγορα. Ενώ η μπαταρία διαφημίζεται πως έχει διάρκεια 7 λεπτά, διαπιστώθηκε πως πολλές φορές η διάρκεια είναι 3-4 λεπτά. Αυτό έχει να κάνει με τις Μονάδες Επέκτασης: 2 decks (Multi-Ranger και Flow Deck καταναλώνουν πιο γρήγορα την μπαταρία).

Επίσης, όταν το MEA επιχειρούσε επανεκκίνηση μετά από σύγκρουση ή “drift”, έπρεπε συχνά να περιμένουμε πλήρη φόρτιση (περίπου 40 λεπτά) για να συνεχίσουμε. Πολλές φορές δοκιμάστηκε άλλο ύψος πτήσης και μειωμένη ταχύτητα για να μπορέσουμε να αντιμετωπίσουμε τα προβλήματα.

Κατά τη διάρκεια των δοκιμών συναντήσαμε επίσης και ένα πιο τεχνικό πρόβλημα με τους drivers της τρέχουσας έκδοσης λογισμικού (Φεβρουάριος 2025). Το Multi-Ranger Deck ξαφνικά σταμάτησε να λειτουργεί, κάτι που αποδείχθηκε ότι οφείλεται σε σύγκρουση οδηγών (drivers). Μετά από αρκετές αναζητήσεις και ανταλλαγή μηνυμάτων με την ομάδα υποστήριξης της Bitcraze, μας βοήθησαν να εντοπίσουμε ένα

σφάλμα στη γραμμή κώδικα των οδηγιών στο λογισμικό (*vl53l1x.c*) όπου η προεπιλεγμένη διεύθυνση των αισθητήρων του Multi-Ranger είχε σύγκρουση με άλλη διεύθυνση. Η λύση ήταν να αλλάξει η συγκεκριμένη αυτή γραμμή σε: *static int nextI2CAddress = VL53L1X_DEFAULT_ADDRESS + 10;* (δηλαδή προστέθηκε το 10 (τυχαίο νούμερο) για να μην υπάρχει πλέον σύγκρουση).

Ένα από τα σημαντικότερα προβλήματα όμως και η δυσκολία, ήταν ότι δεν υπάρχει κανένας άμεσος τρόπος για debug. Γράφεται ο κώδικας, μεταγλωττίζεται, ανεβάζεται στο MEA και ελπίζεις ότι κάτι θα δουλέψει, αλλά συχνά δεν δουλεύει και δεν ξέρουμε ακριβώς γιατί και ειδικά σε συνδυασμό τα προβλήματα που έχω αναφέρει προηγουμένως είναι πολύ δύσκολο να βρεθεί το λάθος. Για να δοκιμαστεί οποιαδήποτε αλλαγή, πρέπει να γίνει ξανά η διαδικασία “build”, να εγγραφεί ξανά το λογισμικό πάνω στο MEA και αυτή η διαδικασία διαρκεί πάνω από 10 λεπτά μέχρι να είναι ξανά έτοιμο για πτήση. Αυτό σημαίνει ότι κάθε μικρή διόρθωση στον κώδικα κοστίζει σε χρόνο και υπομονή, καθώς μπορεί να απαιτηθούν πολλές επαναλήψεις προσπάθειας πριν να υπάρξει επιτυχία.

Γενικά παρά τις δυσκολίες αυτές, κάθε αποτυχία μας βοήθησε στο να βελτιώσουμε τη σταθερότητα και την αξιοπιστία του συστήματος, λαμβάνοντας πολύτιμες πληροφορίες για τα όρια και τις απαιτήσεις του Crazyflie, των αισθητηρίων και του P2P πρωτοκόλλου σε πραγματικές συνθήκες.

4.3 Αποτελέσματα WallFollowing Ενός Crazyflie

Σε αυτό το υποκεφάλαιο θα παρουσιάσουμε την αξιολόγηση της συμπεριφοράς ενός μεμονωμένου Crazyflie κατά την εκτέλεση WallFollowing. Παρουσιάζουμε τα δεδομένα και τις παρατηρήσεις από τις δοκιμές πτήσης, αναλύοντας την ακρίβεια διατήρησης σταθερής απόστασης από τον τοίχο, την ανταπόκριση του αλγορίθμου στις γωνίες και την επίδραση των εξωτερικών παραγόντων. Στην πιο κάτω εικόνα παρουσιάζεται το Crazyflie στο κέντρο του χώρου.



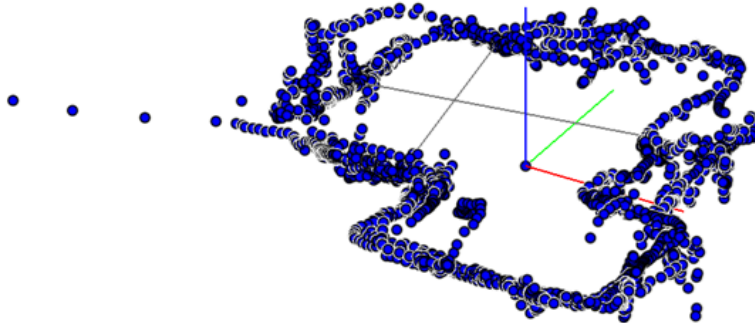
Εικόνα 4.3 Το Crazyflie στο κέντρο του χώρου

Όπως αναφέρθηκε προηγουμένως, το δωμάτιο στο οποίο πραγματοποιήθηκαν οι δοκιμές ήταν μικρό, με λίγα έπιπλα. Επιπλέον, τοποθετήσαμε σεντόνια πάνω στο γραφείο, προκειμένου να προσομοιώσουμε μια ομαλή επιφάνεια για να ακολουθήσει το Crazyflie. Με αυτόν τον τρόπο διατηρήθηκε η απλότητα του πειραματικού περιβάλλοντος, ώστε οι αισθητήρες να βλέπουν πάντα καθαρά ευθείες γραμμές χωρίς απρόβλεπτα σχήματα ή επιφάνειες. Στην πιο κάτω εικόνα παρουσιάζεται το Crazyflie να πετά και να ακολουθεί τους τοίχους.



Εικόνα 4.4 Το Crazyflie εξερευνεί

Απο την συγκεκριμένη πτήση πήραμε το ακόλουθο γράφημα:



Εικόνα 4.5 Χαρτογράφηση χώρου

Η εικόνα παρουσιάζει τις παραγόμενες τιμές αναφοράς (setpoints) από τον αλγόριθμο κατά την εξερεύνηση του δωματίου, όπως καταγράφηκαν από τις μετρήσεις των αισθητήρων τοίχου (multi-ranger). Αυτή η γραφική παράσταση απεικονίζει αυτούσια τα δεδομένα απόστασης και κατεύθυνσης που έστειλε το Multi-Ranger Deck και όχι την πραγματική τροχιά του Crazyflie στον χώρο. Κάθε σημείο αντιστοιχεί σε ένα σημείο αναφοράς στο οποίο το MEA στοχεύει να διατηρήσει σταθερή απόσταση από τον τοίχο, βάσει των αισθητήρων.

Το γεγονός ότι το σχήμα που προκύπτει από τα σημεία αναφοράς ταιριάζει με την πραγματική διάταξη του δωματίου, τότε το συγκεκριμένο πείραμα αποτελεί επιτυχία, ειδικά μετά από τις πολλές αποτυχημένες προσπάθειες. Παρά τις αρχικές πτώσεις και τα θέματα ισορροπίας, η τελική αντιστοίχιση των μετρήσεων με το περιβάλλον επιβεβαιώνει ότι ο αλγόριθμος αναγνωρίζει ορθά τους τοίχους και διατηρεί την πορεία του με συνέπεια.

4.4 Αποτελέσματα Σμήνους

Δεδομένων των πολλαπλών προσκρούσεων και των προβλημάτων που αναφέρθηκαν προηγουμένως, η αξιοπιστία των MEA μειώθηκε δραματικά. Για κάποιο λόγο πολλές φορές απλώς ένα από τα 4 μοτέρ σταματούσαν να λειτουργούν. Εφόσον η πτήση με έναν μόνο Crazyflie ήταν από μόνη της δύσκολη, θα ήταν σχεδόν θαύμα να τα καταφέρουμε με δύο ή περισσότερα. Μετά από αμέτρητες ανεπιτυχείς προσπάθειες να υλοποιήσουμε ακόμη και τα πολύ βασικά (απλώς να πετάξει το MEA) καταλήξαμε στην απόφαση να προσομοιώσουμε το πείραμα.

Στην προσομοίωση τοποθετήσαμε έναν Crazyflie στο κέντρο του δωματίου και έναν δεύτερο έξω από την περιοχή. Η ιδέα είναι το εξωτερικό MEA, με χαμηλότερη προτεραιότητα, να εισέλθει στο πεδίο του πρώτου και να περάσει το όριο RSSI, οπότε, επειδή έχει χαμηλότερη προτεραιότητα, να περιστραφεί και να απομακρυνθεί.

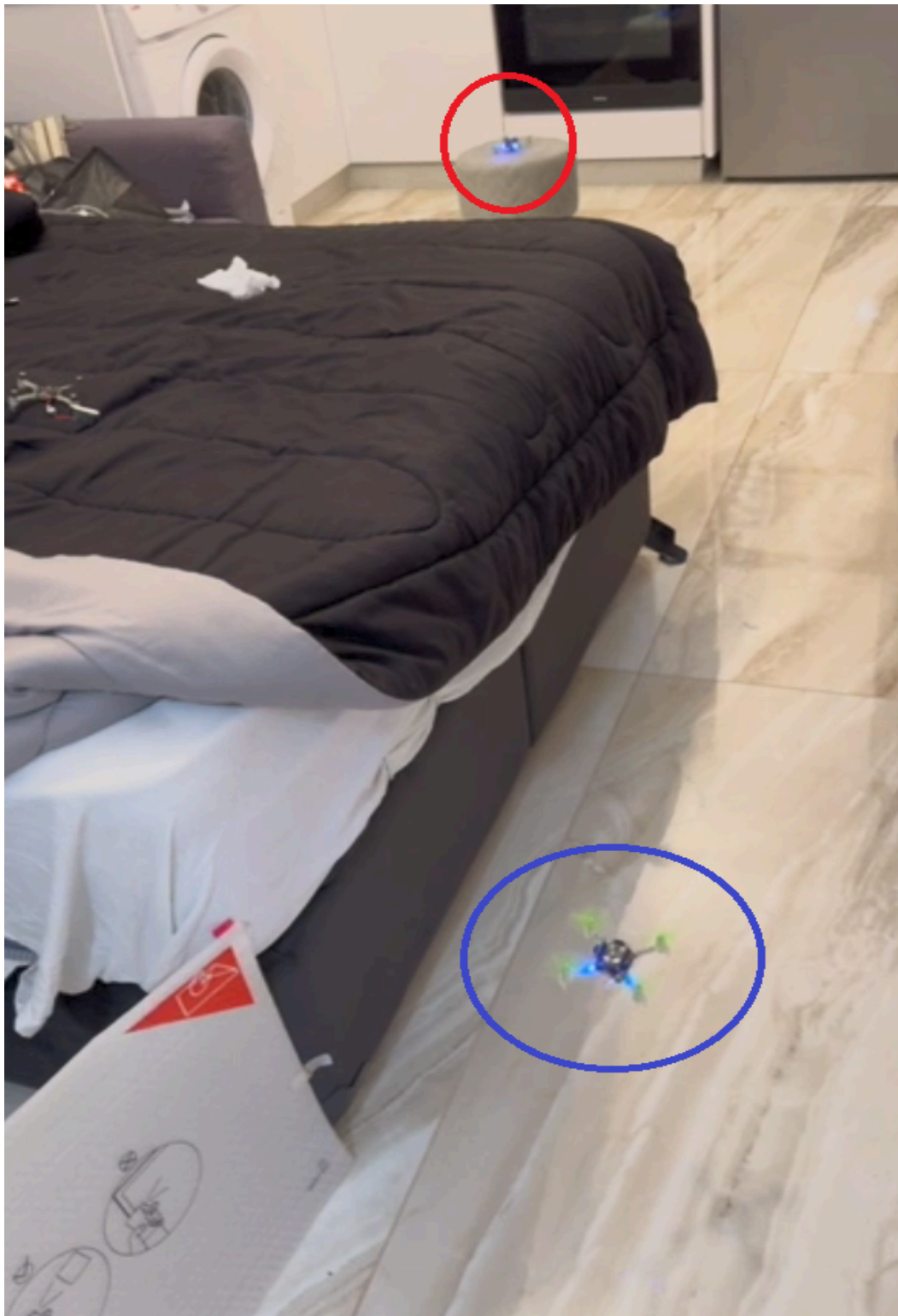
Αν και αυτή η διαδικασία δεν αποκαλύπτει πλήρως τη συμπεριφορά του ολοκληρωμένου αλγορίθμου Σμήνους, τουλάχιστον μας επέτρεψε να επιβεβαιώσουμε ότι η επικοινωνία P2P λειτουργεί σωστά και ότι οι μηχανισμοί ελέγχου κατάστασης ανταποκρίνονται στο σήμα RSSI.

Το Crazyflie στο κέντρο του δωματίου πάνω σε απεικονίζεται στην πιο κάτω εικόνα.



Εικόνα 4.6 Το Crazyflie στο κέντρο του χώρου πάνω σε σκαμπό

Στην πιο κάτω εικόνα απεικονίζεται η διάταξη του πειράματος. Σε κόκκινο κύκλο το 1ο Crazyflie στο κέντρο του χώρου και σε μπλε κύκλο το 2ο Crazyflie που θα επιχειρήσει να μπει στο χώρο του 1ου.



Εικόνα 4.7 Διάταξη πειράματος σμήνους

Στην πιο κάτω εικόνα απεικονίζεται το 2ο Crazyflie. Κάπου εδώ ακολουθεί τον αριστερό τοίχο (το κρεβάτι) και σε αυτό το σημείο η τιμή RSSI που λαμβάνει από το 1ο MEA πέφτει κάτω από τα 45 dB. Συνεπώς, σύμφωνα με τον αλγόριθμο, θα πρέπει τώρα να εκτελέσει περιστροφή για να αποφύγει τη σύγκρουση.



Εικόνα 4.8 Το 2ο Crazyflie εξερευνεί

Και όντως έτσι γίνεται. Τουλάχιστον το 2ο Crazyflie επεξεργάζεται σωστά το P2P πακέτο που το έστειλε το 1ο Crazyflie και αυτόνομα γυρίζει προς τα δεξιά, όπως και απεικονίζεται στην πιο κάτω εικόνα.



Εικόνα 4.9 Το 2ο Crazyflie αλλάζει κατεύθυνση μόλις κοντέψει του 1ου

Για αυτό το πείραμα δεν υπάρχει γράφημα, καθώς όλος ο κώδικας τρέχει απευθείας σε C στο λογισμικό του Crazyflie και δεν υπάρχει εύκολος τρόπος εξαγωγής και επεξεργασίας των δεδομένων σε Python, όπως κάναμε στην περίπτωση του μεμονωμένου ΜΕΑ. Εκεί, χάρη στη χρήση του Python API, μπορούσαμε άμεσα να συλλέξουμε τα σημεία αναφοράς και να τα σχεδιάσουμε, κάτι που εδώ δεν ήταν πρακτικά εφικτό.

Κεφάλαιο 5

Συμπεράσματα και Μελλοντικές Εργασίες

5.1 Συμπεράσματα	67
5.2 Βελτιώσεις	68
5.3 Μελλοντικές Εργασίες	68

5.1 Συμπεράσματα

Στην παρούσα διπλωματική εργασία παρουσιάσαμε την υλοποίηση και πειραματική αξιολόγηση αποκεντροποιημένης πλοήγησης Σμήνους από MEA Crazyflie. Μέσα από τη φάση πρωτοτύπων σε Python και την τελική ενσωματωμένη εφαρμογή σε C/C++ με το Επίπεδο Εφαρμογής και τη Διεπαφή P2P, καταφέραμε να υλοποιήσουμε και να δοκιμάσουμε ένα απλό αλγόριθμο WallFollowing, ικανό να χαρτογραφεί και να εξερευνά πραγματικούς εσωτερικούς χώρους.

Παρά την υψηλή ευελιξία και το χαμηλό κόστος των Crazyflie, συναντήσαμε σημαντικές προκλήσεις: αστοχίες αισθητήρων σε χαμηλό φωτισμό, “ground effect” σε μικρό ύψος πτήσης και συγκρούσεις διευθύνσεων οδηγών των Μονάδων Επέκτασης, χαλασμένοι κινητήρες κτλ. Πολλές πτήσεις τελικά απέβησαν ανεπιτυχείς, όμως οι συνεχείς βελτιώσεις στον αλγόριθμο και οι επαναλήψεις των πειραμάτων έφεραν σταδιακά σταθερότητα και αξιοπιστία.

Η αξιολόγηση των τελικών αποτελεσμάτων δείχνει ότι δεν καταφέραμε να υλοποιήσουμε πλήρως ένα σμήνος από MEA για να δοκιμάσουμε καλά τους αλγορίθμους και να πετύχουμε τον στόχο της εργασίας. Ωστόσο, με δύο Crazyflie προσομοιώσαμε τη συμπεριφορά σύγκρουσης και αποφυγής, επιτυγχάνοντας μερικά πρώτα αποτελέσματα που επιβεβαιώνουν τη σωστή λειτουργία της επικοινωνίας P2P και της βασικής λογικής αποφυγής με βάση το σήμα RSSI.

5.2 Βελτιώσεις

Καθ' όλη τη διάρκεια των πειραμάτων μας επισημάνθηκε ότι η χωρική διάταξη και το μέγεθος του δωματίου περιορίζουν σημαντικά την απόδοση του αλγορίθμου. Ένας μεγαλύτερος, άδειος χώρος (όπως ένας μεγάλος διάδρομος ή αίθουσα χωρίς έπιπλα, συσκευές ή άλλα εμπόδια) θα επέτρεπε στα MEA να εκτελέσουν πιο γρήγορα και ομαλά τις διαδρομές WallFollowing, μειώνοντας τις ανεπιτυχείς προσπάθειες. Για την επέκταση της αυτονομίας, κρίνεται απαραίτητη η υιοθέτηση μεγαλύτερων μπαταριών ώστε τα Crazyflie να πετούν πάνω από επτά λεπτά χωρίς ανάγκη επανεκκίνησης και φόρτισης.

Όσον αφορά την πλοήγηση, πέρα από τα ήδη χρησιμοποιημένα Flow και Multi-Ranger Decks, η ενσωμάτωση πρόσθετων αισθητηρίων όπως το “**Lighthouse positioning system deck**” ή το “**AI Deck**” με ενσωματωμένη κάμερα θα βελτιώσει δραστικά την ακρίβεια θέσης και την αποφυγή λεπτών εμποδίων. Ωστόσο, πρέπει να σημειωθεί ότι το συνολικό κόστος αυξάνεται γρήγορα: ένα Crazyflie 2.1 κοστίζει σχετικά λίγο, αλλά κάθε Μονάδα Επέκτασης (Flow, Multi-Ranger, Lighthouse, AI) προσθέτει μερικές δεκάδες ευρώ, ενώ ένα πλήρες σμήνος με 6–10 MEA πολλαπλασιάζει το κόστος.

Αν απαιτείται κορυφαία ακρίβεια και ομαλή πτήση, η καλύτερη λύση παραμένει ένα σύστημα *Καταγραφής Κίνησης* (Motion-Capture - MoCap) το οποίο κοστίζει χιλιάδες ευρώ αλλά παρέχει ακριβή θέση σε πραγματικό χρόνο. Σε συνδυασμό με το Crazyflie, εξασφαλίζει σχεδόν τέλεια σταθερότητα, καθιστώντας το ιδανικό για ερευνητικά εργαστήρια αλλά μη βιώσιμο για ευρεία εμπορική εφαρμογή.

Σε μελλοντικές υλοποιήσεις, ένας συνδυασμός μεγαλύτερου χώρου, βελτιωμένων μπαταριών, έξυπνων Μονάδων Επέκτασης και συστήματος Καταγραφής Κίνησης, θα καταστήσει το σύστημα πιο αξιόπιστο, ανθεκτικό και έτοιμο για σύνθετες αποστολές σε πραγματικό περιβάλλον.

5.3 Μελλοντικές Εργασίες

Ένα φυσικό επόμενο βήμα είναι η κλιμάκωση του συστήματος σε μεγαλύτερους σχηματισμούς, ένα σμήνος δεκάδων Crazyflie που συνεργάζονται σε πραγματικό χρόνο. Με περισσότερα MEA, μπορούμε να καλύπτουμε ταχύτερα εκτεταμένους

χώρους και να υλοποιούμε πολύπλοκες αποστολές, όπως επιχείρηση έρευνας-διάσωσης σε κατεστραμμένα κτίρια: τα ρομπότ θα εξερευνούν κάθε δωμάτιο, θα ανιχνεύουν θύματα με χρήση θερμικών ή οπτικών αισθητήρων (όπως το AI Deck) και θα μετέφερον σε πραγματικό χρόνο video και συντεταγμένες στις ομάδες διάσωσης. Ακόμη κι αν είχαμε στην διάθεσή μας έναν επιπλέον χρόνο για να δουλέψουμε αυτό το έργο, θα επιλέγαμε και πάλι αυτό το MEA. Με την εμπειρία που αποκτήσαμε, θεωρώ ότι αν είχαμε στη διάθεσή μας καινούρια Crazyflie, οι συντριβές θα ήταν πολύ λιγότερες και το σύστημα πιο αξιόπιστο, καθώς πλέον έχουμε μάθει πώς να το χειριζόμαστε και υπάρχει μεγάλη προοπτική. Κατά τα πρώτα στάδια εκμάθησης είχαμε πολλές συντριβές και αυτό έπαιξε ρόλο στην αξιοπιστία του συστήματος και στα αποτελέσματα.

Επιπλέον, μπορούμε να επιχειρήσουμε εφαρμογές βιομηχανικής επιθεώρησης, όπως ανίχνευση διαρροών αερίου ή νερού μέσα σε αγωγούς, χρησιμοποιώντας κατάλληλα expansion decks και αλγόριθμους μηχανικής μάθησης για αυτοματοποιημένη επεξεργασία των μετρήσεων.

Οι προκλήσεις εδώ είναι η έξυπνη δρομολόγηση των MEA και η διαχείριση της ασφάλειας του σμήνους, καθώς, σε επικίνδυνες αποστολές, απαιτούνται μηχανισμοί ασφαλούς λειτουργίας.

Τέλος, η έρευνα μπορεί να επεκταθεί στην ενσωμάτωση deep learning, όπου το AI Deck θα τροφοδοτεί νευρωνικά δίκτυα ώστε το σμήνος να αναγνωρίζει αντικείμενα και να προσαρμόζει την εξερεύνηση δυναμικά. Αυτό θα επιτρέψει την πραγματοποίηση πολύπλοκων, αυτοματοποιημένων ερευνών σε περιβάλλοντα που αλλάζουν συνεχώς, ανοίγοντας τον δρόμο για εφαρμογές στην ασφάλεια, την περιβαλλοντική παρακολούθηση και τη γεωργία ακριβείας.

Βιβλιογραφία

[1] Bitcraze wiki

<https://wiki.bitcraze.io/>

[2] Bitcraze GitHub Repository

<https://github.com/bitcraze>

[3] Expansion Decks

<https://www.bitcraze.io/documentation/system/platform/cf2-expansiondecks/>

[4] Flow Deck v2

<https://www.bitcraze.io/products/flow-deck-v2/>

[5] Multi-Ranger Deck

<https://www.bitcraze.io/products/multi-ranger-deck/>

[6] Crazyradio

<https://www.bitcraze.io/products/crazyradio-2-0/>

[7] Crazyflie Communication

<https://www.bitcraze.io/2020/08/crazyflie-radio-communication/>

[8] Bitcraze Virtual Machine

<https://github.com/bitcraze/bitcraze-vm>

[9] Crazyflie Python Library

<https://www.bitcraze.io/documentation/repository/crazyflie-lib-python/master/api/cflib/>

[10] Wall-following source code

https://github.com/bitcraze/crazyflie-firmware/tree/master/examples/demos/app_wall_following_demo

[11] App layer guide

https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/userguides/app_layer/

[12] Out-of-tree build process

<https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/development/oout/>

[13] Building and Flashing

<https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/building-and-flashing/build/>

[14] Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment McGuire, k.; De Wagter, C.; Tuyls, K; Kappen, H.J.; de Croon, G. C. H. E. 2019 ScienceRobotics

<https://www.science.org/doi/10.1126/scirobotics.aaw9710>

[15] P2P API

https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/p2p_api/

[16] BitCraze Discussions

<https://github.com/orgs/bitcraze/discussions>

[17] Crazyflie 2.1 Specifications

https://www.bitcraze.io/documentation/hardware/crazyflie_2_1/crazyflie_2_1-datasheet.pdf

[18] Swarms of Unmanned Aerial Vehicles — A Survey Anam Tahir,

Jari Böling, Mohammad-Hashem Haghbayan, Hannu T. Toivonen, Juha Plosila, 2018, ScienceDirect

<https://www.sciencedirect.com/science/article/pii/S2452414X18300086>

[19] Search and rescue with autonomous flying robots through behavior-based cooperative intelligence, Ross D. Arnold, Hiroyuki Yamaguchi & Toshiyuki Tanaka, 2018, Journal of International Humanitarian Action
<https://jhumanitarianaction.springeropen.com/articles/10.1186/s41018-018-0045-4>

[20] Drone swarm technology is shaping the agricultural industry, Michael Burt, 2024, The Farmer Magazine
https://thefarmermagazine.com.au/drone-swarm-technology/?utm_source=chatgpt.com

[21] Fireworks are out, drones are in? Why a July 4th tradition is slowly evolving, Erin Blake, 2024, National Geographic
<https://www.nationalgeographic.com/science/article/fireworks-drones-swarm-danger-health-technology>