



**Ανώτατο Εκπαιδευτικό Ίδρυμα Πειραιά Τ.Τ.
Τμήμα Ηλεκτρονικών Μηχανικών Τ.Ε.**

Πτυχιακή Εργασία

**Υλοποίηση Αυτόνομου Ιπτάμενου Οχήματος βασισμένου σε
πλατφόρμα Ανοικτού Κώδικα και σύγχρονων τεχνολογιών εντοπισμού
θέσης**

Φοιτητής: Λαζαρίδης Νικόλαος

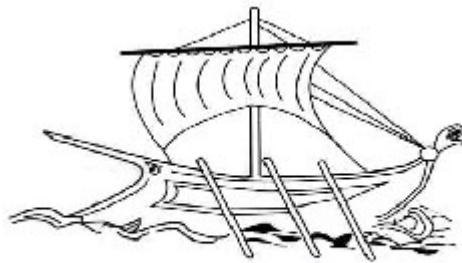
ΑΜ: 41061

Επιβλέπων Καθηγητής

Παναγιώτης Παπαγέωργας

Καθηγητής – Τμήμα Ηλεκτρονικών Μηχανικών Τ.Ε., Α.Ε.Ι. Πειραιά Τ.Τ.

**ΑΘΗΝΑ
ΜΑΡΤΙΟΣ 2016**



**Technological Education Institute
T.E.I. PIRAEUS**

**Piraeus University of Applied Sciences
Department of Electronics Engineering**

Degree Thesis

Implementation of Autonomous Flying Vehicle based on an Open-Source Platform and Modern Positioning Technologies

Student: Nikolaos Lazaridis

Registration Number: 41061

Supervisor

Panagiotis Papageorgas

Professor – Department of Electronics Engineering, Piraeus University of Applied Sciences

ATHENS

MARCH 2016

Όνομα Φοιτητή

Copyright © 2016 Λαζαρίδης Νικόλαος, Μάρτιος 2016

Με επιφύλαξη παντός δικαιώματος, All rights reserved

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τους συγγραφείς.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τους συγγραφείς και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Ανώτατου Εκπαιδευτικού Ιδρύματος Πειραιά.

Student Name

Copyright © 2016 Nikos Lazaridis, March 2016

All rights reserved

Copying, storage and distribution of this work, in whole or part thereof, for any commercial purpose is forbidden. Reproduction, storage and distribution for purposes of non-profit, educational or research nature, provided you indicate the source and to maintain the existing message are allowed. Questions concerning the use of labor for profit should be addressed to the authors.

The views and conclusions contained in this document reflect the authors and should not be interpreted as representing the official position of the Piraeus University of Applied Science.



Τμήμα Ηλεκτρονικών Μηχανικών Τ.Ε.

Ανώτατο Εκπαιδευτικό Ίδρυμα Πειραιά Τ.Τ.

Τμήμα Ηλεκτρονικών Μηχανικών Τ.Ε.

Πτυχιακή Εργασία

Υλοποίηση Αυτόνομου Ιπτάμενου Οχήματος βασισμένου σε πλατφόρμα Ανοικτού Κώδικα και σύγχρονων τεχνολογιών εντοπισμού θέσης

Επιβλέπων Καθηγητής

Παναγιώτης Παπαγέωργας

Καθηγητής – Τμήμα

Ηλεκτρονικών Μηχανικών Τ.Ε., ΑΕΙ Πειραιά Τ.Τ.

Εξεταστής

(Θέση/ Τίτλος)

Εξεταστής

(Θέση / Τίτλος)

Εξεταστής

(Θέση / Τίτλος)

ΑΘΗΝΑ

ΜΑΡΤΙΟΣ 2016

Περίληψη

Η παρούσα πτυχιακή εργασία στοχεύει στη δημιουργία ενός μη επανδρωμένου εναέριου οχήματος τεσσάρων ελίκων, ή αλλιώς τετρακόπτερου (Quadcopter), ή όπως είναι κοινώς γνωστά ενός UAV drone. Το τετρακόπτερο θα έχει την ικανότητα να εκτελεί αυτόνομη πτήση, είτε αυτοματοποιημένη εκτελώντας μια προκαθορισμένη αποστολή, είτε ελεγχόμενη με τη βοήθεια τηλεχειριστηρίου. Για τη δημιουργία του αεροσκάφους πραγματοποιήθηκε πρωταρχική σχεδίαση για εξεύρευση των υλικών, το ηλεκτρολογικό σχέδιο, η συναρμολόγηση και ο προγραμματισμός των ηλεκτρονικών συστημάτων που το απαρτίζουν. Ιδιαίτερη έμφαση δίνεται στο τελευταίο κομμάτι, καθώς απαιτούσε το μεγαλύτερο φόρτο εργασίας.

Δόθηκε ιδιαίτερο βάρος στον τρόπο με τον οποίο θα γίνει η ασύρματη διασύνδεση του υλικού στο τετρακόπτερο με το σύστημα ελέγχου (Ground Control Station, GCS), δηλαδή κινητό, ή H/Y. Αυτή αφορά τη λήψη δεδομένων τηλεμετρίας από το όχημα, αλλά και η αποστολή σε αυτό περιοδικά εντολών για τον σωστό χειρισμό του. Από την αρχή είχε ληφθεί η απόφαση να χρησιμοποιηθεί ανοικτού κώδικα λογισμικού, αλλά και υλικό. Επιλέξαμε λοιπόν ηλεκτρονικές πλατφόρμες που όχι μόνο εκπληρώνουν αυτό το στόχο, αλλά είναι επίσης αρκούντως ισχυροί, αποδοτικοί και ασφαλείς για τις ανάγκες μιας εφαρμογής πραγματικού χρόνου, στην οποία το παραμικρό σφάλμα θα μπορούσε να σημάνει την κατάρριψη του οχήματος, ενδεχόμενο άκρως επικίνδυνο και δαπανηρό.

Θα ξεκινήσουμε όμως με μια γενικότερη επισκόπηση των αεροσκαφών, ειδικεύοντας έπειτα στα μη επανδρωμένα εναέρια οχήματα και μιλώντας για τις έννοιες και φυσικές αρχές που διέπουν αυτές τις ηλεκτρομηχανικές κατασκευές. Αυτή η εισαγωγή κρίνεται απαραίτητη για να μας βοηθήσει στη κατανόηση της θεωρίας και της λειτουργίας αυτών των συστημάτων, προτού βουτήξουμε στη πρακτική εφαρμογή.

Λέξεις – κλειδιά

Μη Επανδρωμένο Αεροσκάφος, Τετρακόπτερο, Τηλεκατευθυνόμενο, Drone, Λειτουργικό Σύστημα Πραγματικού Χρόνου, Ενσωματωμένο Σύστημα, CC3200, APM, Ardupilot Mega, Mission Planner, WiFi, UDP, TCP, Διαδικτυακό Πρωτόκολλο Επικοινωνίας

Abstract

The purpose of this thesis is the development of a 4-helix unmanned aerial vehicle, or Quadcopter, commonly known as UAV Drone. The quadcopter will have the capability to fly autonomously, whether by executing an automatic preconfigured mission, or flying manually under the operator's control. In order to bring this drone to life we had to do the initial planning to order the correct materials, the electrical wiring diagram, the assembly and the programming of the electronic components and modules of the craft. Heavier emphasis was given on the final part, as it demanded the greatest workload.

Increased emphasis was given on the the wireless communication for telemetry data between the quadcopter and the Ground Control Station realized in a PC platform (or a Smartphone). The

visualization platform receives the telemetry data from the vehicle, and in addition has the capability of feedback operation with the transmission of instructions to it, periodically, in order to properly pilot it. From the start of the project we had decided to use open source software as well as open source hardware platforms. As a consequence of this approach we have chosen platforms that suit this goal, but also are quite powerful, efficient, as well as safe for the needs of a real time application. In such a real time application a single tiny mistake could very easily lead to the fall and crash of the aircraft.

We will first begin though with a broader overview of aircrafts, and then focus on unmanned aerial vehicles by discussing the general concepts and physical rules that govern these electromechanical systems. This introduction is compulsory in order to understand the theory and the functionality of these systems before delving into the practical applications.

Keywords

Quadcopter, Multicopter, Remote Control, Radio Control, R/C, Unmanned Aerial Vehicle, UAV, Open Source, RTOS, Real Time Operating System, Embedded System, Internet Protocol, IP

Ευχαριστίες

Θα ήθελα να εκφράσω την ευγνωμοσύνη μου στον Καθηγητή κ. Παναγιώτη Παπαγέωργα που μου έδωσε την ευκαιρία να ασχοληθώ με αυτό το θέμα που με ενδιέφερε πολύ εξαρχής. Οι διάφορες συζητήσεις που είχαμε, οι γνώμες του και η κατεύθυνση που κάθε φορά με προσανατόλιζε οδήγησαν στην επιτυχή ολοκλήρωση αυτής της εργασίας. Επίσης, ένα μεγάλο ΕΥΧΑΡΙΣΤΩ σε πολλές διαδικτυακές εκπαιδευτικές κοινότητες για την ανεκτίμητη βοήθεια που μου προσέφεραν όλα αυτά τα χρόνια των σπουδών μου. Τέτοια βοήθεια θα ανταποδίδω πάντα.

Τέλος, η παρούσα εργασία αφιερώνεται στην οικογένεια μου και ιδιαίτερα στη μητέρα μου, που δε σταμάτησε ποτέ να πιστεύει σ' εμένα.

Περιεχόμενα

1	ΕΙΣΑΓΩΓΗ.....	11
1.1	Αντικείμενο της πτυχιακής εργασίας	12
1.2	Μεθοδολογία	12
1.3	Δομή	14
2	Αεροδυναμική και Βασικές Αρχές Πτήσης.....	15
2.1	Λίγη Ιστορία περί Αεροσκαφών.....	15
2.2	Αεροδυναμική Πτερύγων	17
2.2.1	Αεροτομή	17
2.2.2	Φαινόμενος Άνεμος.....	17
2.3	Αρχή του Bernoulli.....	19
2.4	Φαινόμενο Magnus.....	19
2.5	Φαινόμενο Coriolis	20
2.6	Φαινόμενο του Γυροσκοπίου	21
2.7	Αρχή Επίδρασης του Εδάφους.....	22
2.8	Αυτοπεριστροφή.....	23
2.9	Δυνάμεις που ασκούνται σε ένα Αεροσκάφος	23
2.9.1	Δύναμη Προώθησης (Thrust).....	24
2.9.2	Δύναμη Ανύψωσης (Lift).....	25
2.9.3	Δύναμη Απώθησης (Drag)	25
2.9.4	Βάρος (Weight)	25
2.10	Λειτουργίες Πτήσης Αεροσκαφών.....	26
2.10.1	Pitch.....	26
2.10.2	Roll	26
2.10.3	Yaw	26
2.10.4	Throttle	27
2.11	Αντίδραση Ροπής.....	28
3	Μη Επανδρωμένα Αεροσκάφη.....	30
3.1	Χαρακτηριστικά Τετρακόπτερου	30
3.2	Τρόποι Πτήσης Τετρακόπτερου.....	31
3.2.1	Σταθερή Πτήση (Hovering).....	31
3.2.2	Κατακόρυφη Αναρρίχηση και Κάθοδος	31
3.2.3	Πτήση προς τα Εμπρός, ή προς τα Πίσω	32
3.2.4	Στροφή προς τα Δεξιά, ή προς τα Αριστερά	32
3.2.5	Περιστροφή Οχήματος	32
3.3	Εξισώσεις των Δυνάμεων που εφαρμόζονται σε Αεροσκάφη.....	32
3.3.1	Δυνάμεις Ανύψωσης και Απώθησης	32
3.3.2	Δύναμη Προώθησης.....	33
3.3.3	Βάρος.....	36
3.4	Σχεδίαση για Εκπλήρωση των Προδιαγραφών.....	36
3.5	Ηλεκτρονικές Μονάδες.....	41
3.5.1	Ελεγκτής Πτήσης Ardupilot Mega 2.8	42
3.5.2	Σύστημα Αναφοράς Θέσης και Πορείας	45
3.5.3	Αδρανειακό Σύστημα Πλοϊγησης	45
3.5.4	Σύστημα Ραδιοεπικοινωνίας Turnigy 9x	45
3.5.5	Σύστημα Τηλεμετρίας με τον Μικροελεγκτή CC3200 της T.I	49
3.6	Αισθητήρες.....	55
3.6.1	Επιταχυνσιόμετρο	55
3.6.2	Γυροσκόπιο	55
3.6.3	Μαγνητόμετρο.....	56

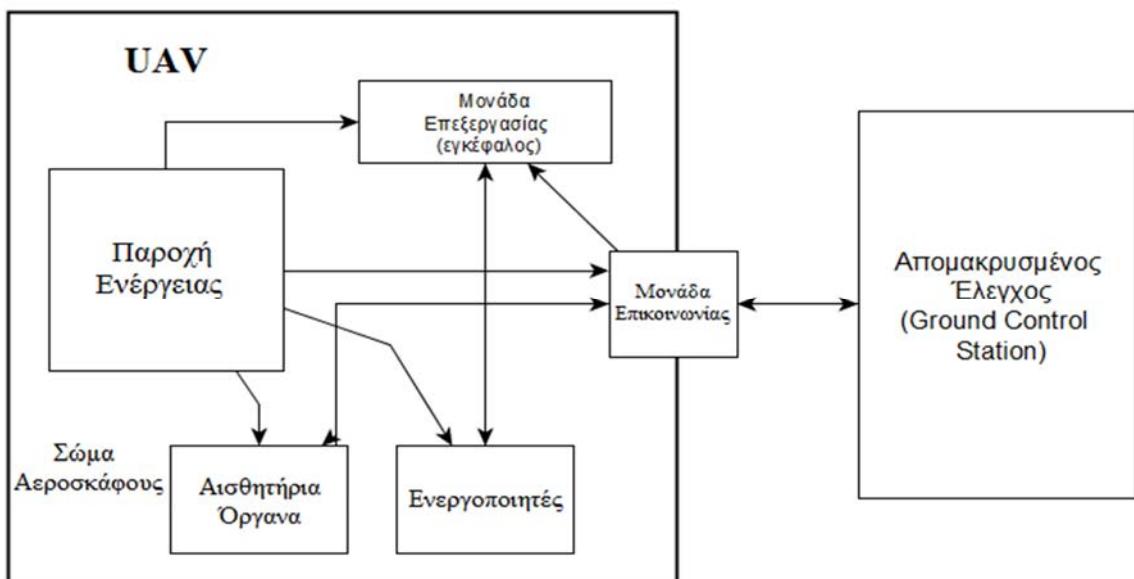
3.6.4	Βαρόμετρο	57
3.6.5	Αισθητήρας Υπερήχων	57
3.7	Μονάδα GPS Δέκτη	58
3.8	Ηλεκτρονικοί Ελεγκτές Ταχύτητας	61
3.9	Ηλεκτρικοί Κινητήρες χωρίς Ψήκτρες	65
3.10	Προπέλες	69
3.11	Τροφοδοσία – Μπαταρία Λιθίου Πολυμερούς	71
3.12	Πλακέτα Διανομής Ισχύος	74
4	Προγραμματισμός των Ηλεκτρονικών Συστημάτων του Μη Επανδρωμένου Αεροσκάφους	77
4.1	Λογισμικό	77
4.2	Λειτουργικά Συστήματα	78
4.2.1	Παρασκηνιακά Συστήματα	79
4.2.2	Λειτουργικά Συστήματα Πραγματικού Χρόνου	79
4.2.3	Πώς Εκκινεί ένα Υπολογιστικό Σύστημα	81
4.2.4	Πυρήνας του Λειτουργικού Συστήματος	82
4.2.5	Προγράμματα, Εφαρμογές, Διεργασίες	83
4.2.6	Πολυνημάτωση	85
4.2.7	Λειτουργικά Συστήματα και Διαχείριση Μνήμης	85
4.2.8	Χρονοπρογραμματισμός Διεργασιών	86
4.2.9	Αντιταραβολή υλοποίησεων με RTOS και χωρίς RTOS	90
4.3	Πρωτόκολλα Επικοινωνίας	91
4.3.1	Το πρωτόκολλο UDP	91
4.3.2	Το πρωτόκολλο TCP	92
4.3.3	Το πρωτόκολλο MAVLINK	93
4.4	Προγραμματισμός του ArduPilot Mega με το Mission Planner	94
4.4.1	Βασικές Ρυθμίσεις	94
4.4.2	Προχωρημένες Ρυθμίσεις	98
4.5	Προγραμματισμός του Μικροελεγκτή CC3200 με το Code Composer Studio	101
5	Επίλογος – Συμπεράσματα	153
6	Κατάλογος Εικόνων:	156
7	Κατάλογος Πινάκων:	158
8	Αναφορές / Links	159

1 ΕΙΣΑΓΩΓΗ

Ο κόσμος των ιπτάμενων μηχανών είναι για λίγους μια καθημερινότητα, ενώ για τους περισσότερους άγνωστος. Αυτό το κείμενο που διαβάζετε θα σας κάνει πιο οικείο με αυτό τον κόσμο και θα σας δείξει όχι μόνο τις βασικές φυσικές αρχές και γνώσεις που χρειάζονται για να τον κατανοήσετε, αλλά και πώς καταφέραμε να υλοποιήσουμε από το μηδέν (προμηθευόμενοι τα διάφορα υλικά βεβαίως) ένα μη επανδρωμένο εναέριο όχημα, με σύγχρονες δυνατότητες αυτοδιοικούμενης πτήσης και εναέριας πορείας. Πρόκειται για την ολοκλήρωση ενός έργου που χρειάστηκε πολύ χρόνο και αγάπη για να συζητηθεί, να σχεδιαστεί, να αναλυθεί, να συναρμολογηθεί και να προγραμματιστεί.

Το αεροσκάφος είναι μηχανική κατασκευή βαρύτερη από τον αέρα, αλλά λόγω της αεροδυναμικής άνωσης που υφίσταται έχει την ικανότητα να πετά. Οι επιστήμονες ανά τους αιώνες παρατηρούσαν τους μηχανισμούς που δίνουν προώθηση και άνωση στα πτηνά και τελικά στις αρχές του 20^{ου} αιώνα κατάφεραν να δημιουργήσουν ένα γοητευτικό, πολύπλοκο μηχανικό κατασκεύασμα που είχε τη δυνατότητα να πετά. Αναφερόμαστε στο αεροπλάνο, αλλά πολύ σύντομα ακολούθησε και το ελικόπτερο. Τα σχήματα, οι μορφές, τα χαρακτηριστικά τους γενικότερα ποικίλουν, όλα όμως στηρίζονται σε παρόμοιες φυσικές αρχές και μηχανισμούς που αναπτύχθηκαν σύμφωνα πάντα με την επιστημονική μέθοδο.

Ακολουθώντας την τεχνολογική «օρμή» που κορυφώθηκε κατά το δεύτερο παγκόσμιο πόλεμο παρουσιάστηκε η ανάγκη κατασκευής μη επανδρωμένων εναέριων οχημάτων, ή UAV's (Unmanned Aerial Vehicle) για συντομία, εμφανώς για πολεμικούς σκοπούς, επειδή.. μπορούσαν να σώσουν ζωές! Τα UAV εξελίχθηκαν παράλληλα με τα συμβατικά επανδρωμένα αεροσκάφη. Η πρόοδος της επιστήμης και της τεχνολογίας επέδρασε θετικά και στα επανδρωμένα και τα μη επανδρωμένα οχήματα. Στην παρακάτω εικόνα παρατηρούμε το δομικό διάγραμμα ενός UAV drone, που απεικονίζει τα βασικά συστατικά του



Εικόνα 1-1 Δομικό Διάγραμμα των στοιχειωδών υποσυστημάτων ενός UAV drone

Ιδιαιτέρως την τελευταία δεκαετία ο κλάδος της ρομποτικής γενικότερα έχει καταστεί πιο προσβάσιμος καθώς η σμίκρυνση των υπολογιστικών μονάδων επεξεργασίας και των αισθητήρων έχει επιτρέψει στον κλάδο της μικροηλεκτρονικής να κάνει μεγάλα άλματα παράγοντας μικρό, ελαφρύ και φθηνό υλικό. Όλα τα παραπάνω έχουν δώσει ώθηση σε ένα νέο τομέα της ρομποτικής ψυχαγωγικό και εκπαιδευτικό ακόμα και για το ευρύ κοινό, τα ραδιοκατευθυνόμενα μοντέλα (Radio Controlled models – R/C).

Τα ιπτάμενα drones είναι ένα νέο καινοτόμο πεδίο της αεροπορίας, που άνοιξε νέους ορίζοντες στις δυνατότητες της. Δημιουργήθηκαν καινούρια ενδιαφέροντα, μια νέα βιομηχανία, νέες θέσεις εργασίας. Οι δυνατότητες των drones είναι τρομακτικά πολλές και πολυποίκιλες, από καθημερινές ψυχαγωγικές δραστηριότητες, έως επιστημονικές μελέτες. Ακραία φωτογραφικά πλάνα, έλεγχος παρασίτων, αθλητική φωτογραφία και βιντεοσκόπηση, ποιοτική φωτογραφία και κινηματογράφηση, μελέτη και παρακολούθηση άγριας ζωής στη φύση, delivery φαγητού, προϊόντων και νέες δυνατότητες επιτήρησης, 3D χαρτογράφηση και τοπογράφηση εκτάσεων, ιατρική και υγειονομική βοήθεια. Υπάρχουν πολλοί οργανισμοί που αναπτύσσουν ιδιόκτητα UAV και τα διαθέτουν προς πώληση σε πολίτες ή κυβερνήσεις, αλλά πέρα από αυτό υπάρχει η διαδικτυακή κοινότητα ανοικτού κώδικα (open source community) με εκατοντάδες χιλιάδες μέλη, που συνασπίζονται, ενώνουν τις γνώσεις, τις ιδέες τους και αναπτύσσουν software και firmware ανοικτό σε όλους, χομπίστες, ερευνητές, επιστήμονες, απλοί πολίτες και γενικότερα όλους όσους έχουν πρόσβαση στο διαδίκτυο.

1.1 Αντικείμενο της πτυχιακής εργασίας

Σκοπός μας ήταν να υλοποιήσουμε ένα αυτόνομο εναέριο όχημα τεσσάρων ελίκων προηγμένων προδιαγραφών και δυνατοτήτων. Εξαρχής επιθυμούσαμε να διαθέτει ικανότητες αυτόνομης πτήσης διαρκείας τουλάχιστον δέκα λεπτών, να μπορεί επίσης να συνδέεται δορυφορικά με το παγκόσμιο σύστημα πλοήγησης (GPS) ώστε να εκτελεί μανούβρες και να λειτουργεί σε αυτόματους τρόπους πτήσης, λαμβάνοντας το δορυφορικό στίγμα για αναγνώριση της θέσης του και για την εξακρίβωση της πορείας και του προσανατολισμού του. Παράλληλα, θέλαμε να εποπτεύουμε και να έχουμε τον πλήρη έλεγχο του τετρακόπτερου. Για την εκπλήρωση αυτού του στόχου προγραμματίσαμε το ηλεκτρονικό σύστημα τηλεμετρίας για να μας παρέχει αυτή τη δυνατότητα.

1.2 Μεθοδολογία

Αρχικά έγινε μια εκτενής μελέτη στην αεροδυναμική των αεροσκαφών κάθε είδους. Επειδή πρόκειται για ένα δύσκολο ζήτημα, με το οποίο δεν είχα ξαναέρθει σε στενή επαφή, ιδιαίτερα από τεχνικής πλευράς που καλούμαι τώρα να υλοποιήσω, έπρεπε πρώτα απ' όλα να κατανοήσω πλήρως το θέμα. Σε αυτό το στάδιο έψαχνα για πληροφορίες σχετικά με τη λειτουργία των αεροσκαφών. Καταρχάς τις αρχές της φυσικής που υφίστανται κατά την πτήση του αεροσκάφους, πχ. τι μετατροπές ενέργειας συμβαίνουν στα τυπικά αεροπλάνα και ελικόπτερα, ποιοί μηχανισμοί τα ανυψώνουν, ποιές είναι οι δυνάμεις που ασκούνται στο αεροσκάφος και μπορούν να το απογειώνουν. Δεν στηρίχθηκα μόνο στους πολύ βασικούς μηχανισμούς, αλλά ήθελα να εμβαθύνω περισσότερο και σε πιο προχωρημένα θέματα στα σύγχρονα ελικόπτερα και αεροπλάνα. Για την εξεύρεση των πληροφοριών αυτών με βοήθησε σημαντικά το διαδίκτυο, διαβάζοντας άρθρα, κομμάτια από e-books και βλέποντας αρκετά βίντεο στο YouTube. Εξεπλάγην στην αρχή όταν αντιλήφθηκα πόσα πολλά εκπαιδευτικά και ψυχαγωγικά βίντεο υπήρχαν για το θέμα, τα οποία διέθεταν μεγάλη ποσότητα γνώσης παρουσιασμένη με γλαφυρό τρόπο. Επιπλέον ο επόπτης καθηγητής μου κ. Παναγιώτης Παπαγέωργας με τροφοδοτούσε με πάρα πολλά άρθρα, αναφορές, αλλά και σχετικές πτυχιακές εργασίες για το θέμα.

Αφού λοιπόν απέκτησα μια σοβαρή κατανόηση του έργου που έπρεπε να επιτελέσω, κλήθηκα να επιλέξω τα κατάλληλα υποσυστήματα που θα απαρτίζουν το αεροσκάφος. Δηλαδή έπρεπε να κάνω εκτενής ανάλυση και σχεδίαση της κάθε βαθμίδας που πρέπει να συμπεριλάβω. Εκτός από το γεγονός ότι όλα αυτά τα προϊόντα και εργαλεία που θα χρειαζόμουν είναι πολλά, ακριβά και δυσεύρετα, έπρεπε να βρω τη χρυσή τομή λαμβάνοντας υπόψη μια πληθώρα παραγόντων ώστε να επιλέξω τα κατάλληλα. Η διαδικασία αυτή χρειάστηκε χρόνο και πολλούς υπολογισμούς. Εφόσον

είχα κατανοήσει πλέον το θεωρητικό υπόβαθρο, έπρεπε να βρω μαθηματικές εξισώσεις, η επίλυση των οποίων θα μου έδινε απάντηση για τις προδιαγραφές που είχαμε συλλέξει για το τετρακόπτερο. Άρθρα, «απόκρυφες» (δηλαδή δυσεύρετες) ιστοσελίδες και έρευνα σε διαδικτυακά forums με οδηγησαν σε όλες αυτές τις πληροφορίες. Όλες αυτές φυσικά θα παρουσιαστούν με λεπτομέρεια στα κατάλληλα κεφάλαια και οι πηγές θα αναφέρονται στο τέλος του εγγράφου.

Είχα αποκτήσει και την τεχνική κατανόηση της εργασίας και ήμουν έτοιμος να ψάξω για τα απαραίτητα εξαρτήματα και συστατικά που θα χρησιμοποιούσα. Πλέον ήξερα τι έπρεπε να ψάξω, να σημειώσω, να απορρίψω και να αποδεχθώ, εφόσον κατείχα τις μαθηματικές γνώσεις. Έπρεπε να προμηθευτώ τον κατάλληλο μηχανολογικό, ηλεκτρολογικό και ηλεκτρονικό εξοπλισμό για να φέρω εις πέρας την εργασία. Ενδεικτικά αναφέρω ότι χρειαζόταν ο σκελετός του οχήματος, οι κινητήρες, ο ελεγκτής πτήσης, το ηλεκτρονικό σύστημα επικοινωνίας κλπ. να μην υπερβαίνουν το προδιαγεγραμμένο βάρος, την κατανάλωση ρεύματος, τις προβλεπόμενες διαστάσεις της κατασκευής, τις απαιτήσεις σε ισχύ και παράλληλα να υποστηρίζουν τις απαιτήσεις για πρωτόκολλα επικοινωνίας και δικτύωσης που είχαμε ως στόχο, την επεξεργαστική ικανότητα των μικροελεγκτών που απαιτείται, την ύπαρξη ανοικτού κώδικα σε μια γλώσσα προγραμματισμού που να γνωρίζω, και πολλά άλλα. Ακόμη και μετά από όλα αυτά και πάλι θα έπρεπε να διατηρήσω ένα περιθώριο ασφαλείας για πιθανά λάθη και δυσκολίες που είναι πολύ δύσκολο να παρατηρηθούν εάν δεν υπάρξει χειρωνακτική ενασχόληση, ακόμα και με τα όσα είχα διαβάσει και πιθανώς προβλέψει ότι θα συμβούν. Και όντως αυτά τα απροσδόκητα κωλύματα υπήρξαν, αλλά εντυχώς ξεπεράστηκαν εγκαίρως, διότι είχε γίνει πολύ καλή σχεδίαση εκ των προτέρων. Εάν δεν είχα αφιερώσει τόσο χρόνο στη έρευνα, ανάλυση και σχεδίαση πιθανόν να ξοδεύονταν διπλάσιοι οικονομικοί πόροι και διπλάσιος χρόνος για την ολοκλήρωση του έργου. Ακόμη και τότε η επιτυχής ολοκλήρωση του δε θα ήταν εγγυημένη. Αφού λοιπόν συνέταξα το Bill Of Materials (BOM), παραγγείλαμε από διάφορα καταστήματα, σχεδόν αποκλειστικά του εξωτερικού, τα υλικά.

Χρειάστηκε ένας μήνας περίπου για να τα παραλάβουμε όλα. Άξιζε όμως η αναμονή διότι όλα τα επιμέρους υλικά (εκτός από τις προπέλες στις οποίες έγινε μια αβλεψία) αποδείχτηκε ότι εκπλήρωναν και πειραματικά τις προδιαγραφές. Δεν υπήρχε σχεδόν κανένα άλλο απροσδόκητο γεγονός το οποίο να είναι άξιο αναφοράς. Κατά τη διάρκεια της αναμονής εγώ γινόμουν περισσότερο οικείος με τη δικτύωση υπολογιστών και τα λειτουργικά συστήματα, τα πρωτόκολλα επικοινωνίας και διάφορα πρακτικά ζητήματα τηλεπικοινωνιών και προγραμματισμού τα οποία θα με ωφελούσαν σημαντικά. Επίσης είχα διαθέσιμη από αρκετά νωρίς την αναπτυξιακή πλατφόρμα CC3200 Launchpad της TI, με την οποία και ασχολήθηκα συστηματικά. Κατά τη διάρκεια των μελετών μου αντιλήφθηκα το ρόλο που θα λάμβανε το CC3200 στο τελικό στάδιο της κατασκευής, καθώς δεν το είχα καταλάβει από την αρχή! Θα έπαιζε το ρόλο του διαμεσολαβητή των δεδομένων τηλεμετρίας μεταξύ του H/Y, το οποίο θα έτρεχε το κατάλληλο λογισμικό Ground Control Station, και του τετρακόπτερου. Παρόλο που δεν διαθέτει μεγάλη εμβέλεια η chip κεραία που ενσωματώνει, θα ήταν επαρκής για τις απαιτήσεις της εργασίας μου. Εφόσον λοιπόν είχα μυηθεί στον προγραμματισμό του CC3200 με τα παραδείγματα που προσέφερε η Texas Instruments για αυτό καθώς και το ιδιόκτητο λειτουργικό σύστημα που παρείχε, ξεκίνησα να αναπτύσσω το πρόγραμμα που θα επιτελεί αυτό το σκοπό, την αποστολή και λήψη της τηλεμετρίας. Ήταν μια επίπονη διαδικασία διότι έπρεπε να κατανοήσω περισσότερο και τις λειτουργίες των λειτουργικών συστημάτων, του πυρήνα (Kernel) και πολλών άλλων λεπτομερειών κάτι που τελικώς κατάφερα με μεγάλη επιτυχία.

Οταν τελικά παραλάβαμε όλα τα προϊόντα εγώ ήμουν προετοιμασμένος αρκετά καλά διαθέτοντας πλέον αυτοπεποίθηση. Παρόλα αυτά όμως υπήρξαν παράγοντες που δεν είχα προβλέψει και μολονότι δίστασα αρχικώς ξεπέρασα τις δυσκολίες σχετικά γρήγορα και προχώρησα στη συναρμολόγηση και συγκόλληση των υποσυστημάτων του οχήματος. Η εγκατάσταση των ηλεκτρονικών υποσυστημάτων χαμηλής και υψηλής ισχύος έγινε γρήγορα αλλά η βελτιστοποίηση

τους απαιτούσε χρόνο και πολλά λάθη τα οποία ήταν μάλλον αναπόφευκτα. Οι πρώτες απόπειρες πτήσης παρόλο που έγιναν κάτω από συνθήκες αυστηρού ελέγχου οδήγησαν σε πολλές συγκρούσεις, στις οποίες ευτυχώς καταστράφηκαν/λύγισαν μόνο μερικές προπέλες, εφόσον είναι οι πιο εκτεθειμένες. Δεν άργησα πολύ να συνηθίσω τη πτήση, αλλά άργησα να βρω και να ρυθμίσω τις κατάλληλες παραμέτρους στον ελεγκτή πτήσης για τη βελτιστοποίηση της.

Εν τω μεταξύ ο προγραμματισμός του CC3200 αποδείχτηκε το πιο προκλητικό έργο αυτής της διατριβής καθώς έπρεπε να κατανοήσω πλήρως σχεδόν έναν ολόκληρο H/Y και τη λειτουργία του λειτουργικού του συστήματος πραγματικού χρόνου TI-RTOS (Texas Instruments Real Time Operating System) το οποίο και έκανα χρήση. Χρειάστηκε πάνω από ένας μήνας εντατικής εργασίας, για να ολοκληρώσω το προγραμματισμό του μικροελεγκτή. Επιπλέον παρουσιάστηκαν και διάφορα λοιπά μικροπροβλήματα στα οποία δεν χρειάζεται να αναφερθώ, ενώ για κάποια άλλα πιο σημαντικά γίνεται αναφορά στο αντίστοιχο επιμέρους εδάφιο. Η επίλυση όλων των προβλημάτων μου παρείχε πείρα και αυτοπεποίθηση. Εντέλει, συνενώνοντας όλα τα επιμέρους κομμάτια της εργασίας επιτεύχθηκε ο στόχος με τεράστια επιτυχία και απολαυστικές στιγμές πτήσης.

1.3 Δομή

Η παρούσα εργασία έχει την ακόλουθη δομή:

Στο Κεφάλαιο 2, ζεκινάμε με μια σύντομη ιστορική αναδρομή στα αεροσκάφη. Μελετάμε τον κλάδο της αεροδυναμικής με λεπτομέρεια, παρουσιάζουμε όλες τις βασικές φυσικές αρχές που λαμβάνονται υπόψη για την πτήση αεροσκαφών.

Στο Κεφάλαιο 3, εστιάζουμε στα μη επανδρωμένα εναέρια οχήματα, παρουσιάζονται διεξοδικά τα υποσυστήματα που απαρτίζουν ένα UAV. Αρχικά εξηγούμε το στάδιο της σχεδίασης και ανάλυσης που έπρεπε να γίνει και αναφέρονται όλες οι εξισώσεις που λάβαμε υπόψη. Έπειτα σε κάθε εδάφιο περιγράφεται η αρχή λειτουργίας της κάθε υπομονάδας ενός μη επανδρωμένου αεροσκάφους, αναφέρεται η επιλογή που κάναμε εμείς, η συναρμολόγηση που ακολουθήθηκε και διάφορα άλλα ψεγάδια που έπρεπε να διορθωθούν. Παρουσιάζονται αρκετές τεχνικές λεπτομέρειες για τη βελτιστοποίηση του κάθε υποσυστήματος.

Στο Κεφάλαιο 4, περιγράφουμε και αναλύουμε τον προγραμματισμό των ηλεκτρονικών συστημάτων του ελεγκτή πτήσης και της βαθμίδας ασύρματης τηλεμετρίας.

Στο Κεφάλαιο 5, ολοκληρώνουμε με τα συμπεράσματα της διατριβής και πιθανά μελλοντικά σχέδια επέκτασης και περαιτέρω βελτίωσης που θα μπορούσαν να γίνουν.

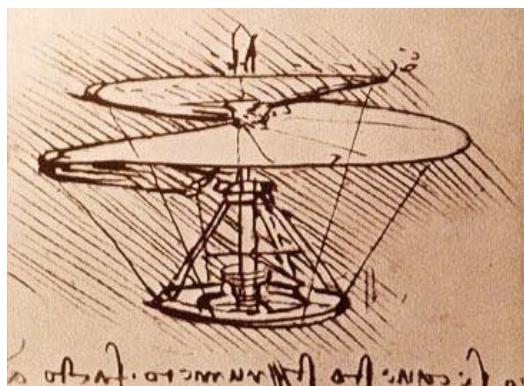
2 Αεροδυναμική και Βασικές Αρχές Πτήσης

Παρόλο που η κατασκευή των πρώτων αεροσκαφών ξεκίνησε χρονικά αρκετά αργά, αναφορικά με την πορεία της ανθρώπινης ιστορίας, το αεροπλάνο και το ελικόπτερο ενσαρκώνουν αρχέγονες αρχές πτήσης που έχουν μελετηθεί από τους επιστήμονες εδώ και αιώνες. Μόλις τον τελευταίο αιώνα ίμως η τεχνολογία και η επιστήμη έφτασαν σε ένα ώριμο στάδιο για να μπορέσουν να κατασκευάσουν αυτές τις ιδιοφυείς επινοήσεις. Ιδιαίτερα η μελέτη των αρχών του ελικοπτέρου έπαιξε σημαντικό ρόλο στην υλοποίηση αυτής της κατασκευής, διότι το εναέριο όχημα τεσσάρων ελίκων, ή αλλιώς τετρακόπτερο (Quadcopter), στηρίζεται περισσότερο σε αρχές αεροδυναμικής ελικοπτέρου και όχι αεροπλάνου, όπως θα εξηγήσουμε με λεπτομέρεια στη συνέχεια που θα γίνει αυτός ο διαχωρισμός.

Η λέξη αεροδυναμική είναι σύνθετη, αποτελούμενη από το πρόθεμα «αερο-» που προέρχεται από το «αέρας» και τη λέξη «δυναμική» που υποδηλώνει κίνηση. Σε αυτή την ενότητα περιγράψουμε όλους τους βασικούς φυσικούς μηχανισμούς που παίζουν ρόλο στην πτήση. Θα περιγραφούν όλοι οι θεμελιώδεις φυσικοί νόμοι που καθορίζουν την πτήση των αεροσκαφών, διότι σε αυτούς έχουμε στηριχθεί και εμείς και οι μηχανικοί και οι τεχνικοί που έχουν κατασκευάσει και σχεδιάσει κάθε εξάρτημα ενός αεροσκάφους. Αυτοί οι νόμοι έχουν ισχύ φυσικά και στα μη επανδρωμένα αεροσκάφη, η υλοποίηση ενός τέτοιου είναι το αντικείμενο αυτής της εργασίας. Πριν όμως μπούμε σε πιο τεχνικές λεπτομέρειες σχετικά με τη σχεδίαση του κάθε τμήματος του οχήματος, και το σωστό συνδυασμό όλων των επιμέρους υλικών πρέπει να περιγράψουμε τον τρόπο που όλα αυτά λειτουργούν, το **πώς**. Αρχίζουμε με λίγη ιστορία περί αεροπλάνων και ελικοπτέρων και έπειτα συνεχίζουμε με την περιγραφή των φυσικών αρχών της αεροδυναμικής. Θα περιγράψουμε τις δυνάμεις που δρούν στο αεροσκάφος ανά πάσα στιγμή, θα εξηγήσουμε το δυναμικό χαρακτήρα μιας πτέρυγας, το φαινόμενο Magnus, την αρχή του Bernoulli, το φαινόμενο της αντιδραστικής ροπής των λεπίδων του ελικοπτέρου, το φαινόμενο Coriolis, το φαινόμενο εδάφους, το φαινόμενο γυροσκοπίου, το φαινόμενο αυτοπεριστροφής κ.α..

2.1 Λίγη Ιστορία περί Αεροσκαφών

Οι παλαιότερες αναφορές ανθρώπινης πτήσης προέρχονται από τη Κίνα, όπου περίπου το 400πΧ. παιδιά έπαιζαν με ιπτάμενα παιχνίδια από μπαμπού. Περιστροφικοί έλικες (rotary wings) σχεδιάζονταν και μελετήθηκαν για πρώτη φορά στις αρχές του 14^{ου} αιώνα, πολλοί από αυτούς ως απομίμηση των φτερών των πουλιών. Τον 15^ο αιώνα ο μεγάλος καλλιτέχνης και εφευρέτης Λεονάρντο Ντα Βίντσι έφτιαξε πολλά σχέδια μιας μηχανής που έμοιαζε με ιπτάμενο σκάφος, η οποία περιστρεφόταν γύρω από έναν κατακόρυφο άξονα. Βλέποντας ένα τέτοιο σχέδιο στην παρακάτω φωτογραφία παρατηρούμε ότι μοιάζει περισσότερο με περιστρεφόμενο κοχλία παρά με οτιδήποτε με δυνατότητες πτήσης.



Εικόνα 2-1 Σχέδιο ιπτάμενης μηχανής του Λεονάρντο Ντα Βίντσι[1]

Για περίπου 3 αιώνες μετά τον Λεονάρντο Ντα Βίντσι το ενδιαφέρον για εναέρια πτήση ήταν στην ουσία μη υπαρκτό. Δεν είχαν επινοηθεί τα μέσα για την προσφορά συνεχούς ηλεκτρικού ρεύματος, το οποίο θα εφοδίαζε το όχημα με επαρκή ποσά ενέργειας κάτι το οποίο φαινόταν ότι θα ήταν απαραίτητο για την ανύψωση ενός μεγάλου ηλεκτρομηχανικού οχήματος στον αέρα, πόσο μάλλον εάν αυτό κουβαλούσε και άνθρωπο.

Παρατηρώντας το χρονοδιάγραμμα της ιστορίας τα αεροπλάνα αναπτύχθηκαν νωρίτερα από τα ελικόπτερα. Γενικότερα τα νεότερα αεροσκάφη κατηγοριοποιούνται ως στροφιόπτερα (Rotor-Craft) όπως τα ελικόπτερα, και σε σταθερών πτερύγων (Fixed-Wing) όπως τα αεροπλάνα. Όταν εφευρέθηκε η ατμομηχανή τον 18^ο αιώνα και ύστερα, υπήρξαν πολλές απόπειρες να κατασκευαστεί ατμοκίνητη πτητική συσκευή. Έτσι, τον 19^ο αιώνα κατασκευάστηκε το πρώτο αεροπλάνο από τον Ρώσο εφευρέτη και ερευνητή Alexander Mozhaysky. Το όχημα έκανε σύντομη πτήση. Αργότερα στο τέλος του αιώνα ο άγγλος Hiram Maxim έκανε δοκιμή αεροπλάνου με ατμομηχανή, αλλά στην πρώτη απόπειρα να απογειωθεί χάλασε η μηχανή. Τελικά οι αδελφοί Ráit, αμερικανοί εφευρέτες και αεροπόροι, κατάφεραν να κατασκευάσουν αεροπλάνο που να επιτυγχάνει σταθερή πτήση και μάλιστα με επιβάτη. Η πρώτη πτήση με αεροπλάνο έγινε από τον Όρβιλ Ράιτ που πέταξε για 12 δευτερόλεπτα σε απόσταση 40 μέτρων. Διαβλέποντας τη σημασία που είχε η εφεύρεση τους, κυβερνήσεις μεγάλων χωρών της εποχής άρχισαν την κατασκευή αυτών των μηχανών, κυρίως για να τις χρησιμοποιήσουν για στρατιωτικούς σκοπούς ως μέρος της επεκτατικής πολιτικής τους. Το αεροπλάνο έπαιξε σημαντικό ρόλο στους δύο παγκόσμιους πολέμους και συνεχώς αναβαθμίζόταν. Μετά το τέλος του 2^{ου} παγκοσμίου πολέμου αεροπλάνα εφοδιάστηκαν με πυραυλοκίνητους κινητήρες και σύντομα το αεροπλάνο έγινε η πρώτη τεχνητή συσκευή που ξεπέρασε το φράγμα της ταχύτητας του ήχου (Mach 1). Σήμερα υπάρχουν αεροπλάνα που πετούν με ταχύτητα που ξεπερνά τα 1000χλμ. την ώρα. Υπάρχουν επίσης αεροπλάνα που προσθαλασσώνονται καθώς και αμφίβια (γιατί μπορούν να προσγειωθούν και να απογειωθούν και στην ξηρά και επί της θάλασσας).

Τα σχήματα και τα μεγέθη είναι πολλά αλλά τα βασικά μέρη ενός αεροπλάνου είναι 1) η πτέρυγα, 2) η άτρακτος ή σώμα, 3) το σύστημα προσγείωσης, 4) το σύστημα προώθησης (έλικας κτλ.). Τα όργανα που υπάρχουν σ' ένα αεροπλάνο είναι περίπου τα εξής: Το χειριστήριο για τον καθορισμό της πορείας, πηδάλιο για το ύψος, το γκάζι, το χειριστήριο των πηδαλίων της κλίσης, το ποδωστήριο. Επίσης πολύ χρήσιμα για τη λειτουργία ενός αεροπλάνου είναι τα όργανα πλεύσης. Αυτά είναι: 1) Τα όργανα ελέγχου, 2) τα αεροναυτιλιακά όργανα (μαγνητική πυξίδα, γυροσκοπικά όργανα κ.α.), 3) τα ραδιοηλεκτρικά όργανα (ραδιοπυξίδα, ραντάρ κ.α.).

Τα ελικόπτερα ως πιο εξεζητημένη μηχανή αναπτύχθηκαν λίγο αργότερα. Στις αρχές του 20^{ου} αιώνα ο γάλλος μηχανικός Paul Cornu και ο, γεννημένος στη Ρωσία, αμερικανός πρωτοπόρος της αεροπορίας Igor Sikorsky πραγματοποίησαν τους αρχικούς αλλά ατελής πειραματισμούς με πρώιμες βενζινοκίνητες μηχανές για να περιστρέψουν τους κινητήρες. Το 1939 ο Igor Sikorsky βρήκε την πρώτη πρακτική επίλυση στο πρόβλημα της αντιδραστικής ροπής (Torque Reaction), με το μοντέλο που εφηύρε το VS-300. Από εκείνο το σημείο –ορόσημο– το ελικόπτερο απέκτησε κυρίαρχο ρόλο στην αεροπορία εν καιρώ και ειρήνης και πολέμου. Το ανυψωτικό του σύστημα αποτελείται από μεγάλη περιστρεφόμενη έλικα, και του παρέχει τη μοναδική δυνατότητα να προσγειώνεται, ή να απογειώνεται σχεδόν κατακόρυφα. Το 1949 το ελικόπτερο πέτυχε ταχύτητα 220χλμ. την ώρα και ανύψωση 7000μ. Ένα βασικό προτέρημα του ελικοπτέρου συγκριτικά με το αεροπλάνο είναι η δυνατότητα του να ίπταται σε μια σταθερή θέση στον αέρα και να μπορεί να απογειωθεί κατακόρυφα, εκμηδενίζοντας την ανάγκη για μεγάλους διαδρόμους απογείωσης/προσγείωσης.

Γενικότερα η ευελιξία του ελικοπτέρου και οι μοναδικές του δυνατότητες να μεταβάλει το ύψος του κάθετα, να ίπταται ακίνητο και να μετακινείται σε οποιαδήποτε κατεύθυνση ανεξαρτήτως πορείας, το έχει καταξιώσει σε αναντικατάστατο μέσο για πάρα πολλές εφαρμογές όπως κατάσβεση

πυρκαγιάς, έρευνας και διάσωσης και σε ξηρά και σε θάλασσα, παρακολούθησης - επιτήρησης, για γρήγορη μεταφορά προσωπικού και αποσκευών σε απρόσιτες περιοχές, ή σοβαρά αρρώστων. Σε σύγκριση με το αεροπλάνο υστερεί σε δυνατότητες μαζικής μεταφοράς, ανύψωσης μεγάλων φορτίων και υψηλής ταχύτητας.

2.2 Αεροδυναμική Πτερύγων

Σε αυτό το εδάφιο θα περιγράψουμε τις δυνάμεις και όλες τις φυσικές αρχές που υφίστανται σε ένα πτερύγιο και γενικότερα σε ένα σώμα το οποίο ίππαται. Κρίνουμε ότι αυτό αποτελεί το πιο σημαντικό και δύσκολο κομμάτι της δυναμικής του αεροσκάφους, γι' αυτό και θα πρέπει να γίνει κατανοητό πρωτίστως.

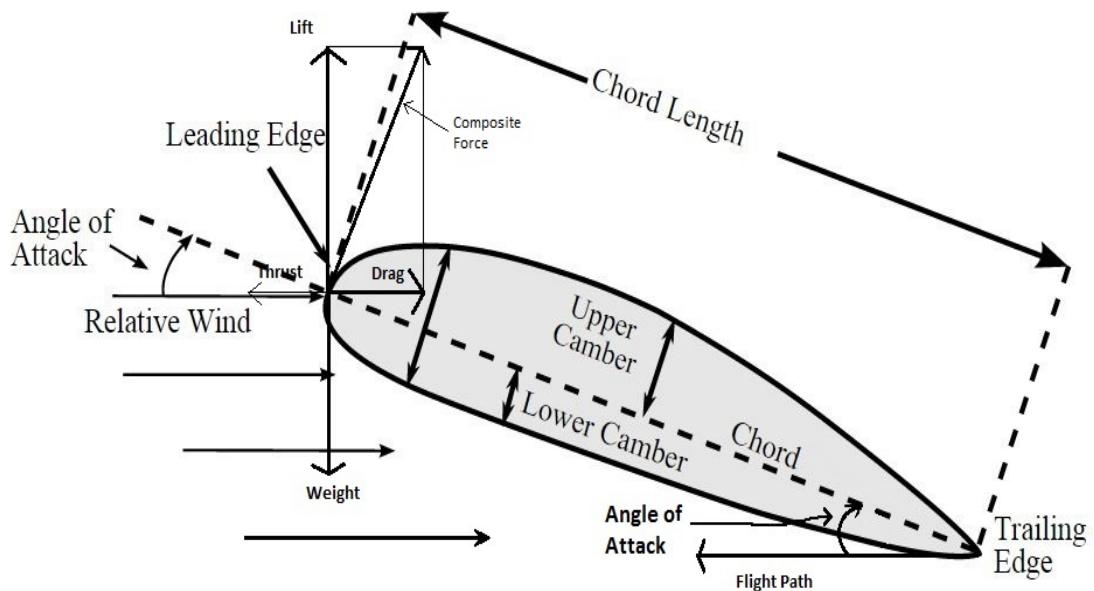
2.2.1 Αεροτομή

Αεροτομή (Airfoil) αποκαλείται οποιαδήποτε επιφάνεια που έχει σχεδιαστεί με τρόπο ώστε όταν προσκρούεται, κατά μέτωπο, από ένα ρεύμα αέρα να παράγει μια χρήσιμη αεροδυναμική αντίδραση. Σχετικά με ένα αεροπλάνο συνήθως εξετάζουμε την αεροτομή των κύριων πτερύγων του, αλλά με μια δεύτερη ματιά βλέπουμε ότι και τα υπόλοιπα βασικά συστατικά κομμάτια του αεροπλάνου, όπως οι οπίσθιες πτέρυγες, οι προπέλες, ακόμα και η άτρακτος έχουν αεροδυναμική μορφή. Συνεπώς όλα θα μπορούσαν θεωρητικά να μοντελοποιηθούν ως μια αεροτομή. Υπάρχουν διάφορες μορφές αεροτομής που έχουν μελετηθεί και σχεδιαστεί, αλλά η πιο κοινή μορφή της είναι αυτή που φαίνεται στο παρακάτω σχήμα (Εικόνα 2-2). Άλλες μορφές αεροτομής μπορεί να έχουν κυρτή μορφή στο άνω μέρος και κούλη στο κάτω μέρος, ή άλλες να έχουν κυρτή μορφή και στα δύο μέρη. Η γραμμή που συνδέει το μπροστινό καμπυλωμένο σημείο, που αποκαλείται οδηγούσα ακμή (Leading Edge) και το οπίσθιο μυτερό σημείο (καταληκτικό σημείο – Trailing Edge) της αεροτομής, είναι γνωστή ως χορδή (Chord). Επίσης κάνουμε λόγο για κυρτώματα εκατέρωθεν της χορδής, το άνω κύρτωμα (Upper Camber) και το κάτω κύρτωμα (Lower Camber). Αυτά απεικονίζουν την καμπυλότητα της αεροτομής.

2.2.2 Φαινόμενος Άνεμος

Ο φαινόμενος άνεμος (Relative Wind) είναι η κίνηση του ανέμου αναφορικά με ένα σώμα. Μπορεί να είναι αποτέλεσμα είτε κινούμενου ανέμου και ενός ακίνητου σώματος, είτε «ακίνητου» ανέμου και ενός κινούμενου σώματος, είτε κινούμενου ανέμου και σώματος με αναφορά κάποιο σταθερό έδαφος, συνήθως τη γη. Στη τελευταία περίπτωση ο σχετικός άνεμος προκύπτει από το αλγεβρικό άθροισμα των δύο ταχυτήτων. Η κατεύθυνση του φαινόμενου ανέμου είναι πάντα παράλληλη και αντίθετης φοράς με τη κατεύθυνση πτήσης του σώματος (Flight Path).

Όταν ένα ρεύμα αέρα προσκρούει σε ένα σώμα, πχ. μια αεροτομή, όπως η πτέρυγα αεροπλάνου, ή η προπέλα ελικοπτέρου, ασκεί σε αυτό δύναμη η κατεύθυνση της οποίας εξαρτάται από τη σχετική θέση του σώματος και της κατεύθυνσης του αέρα που προσκρούει σε αυτό. Αυτή η σύνθετη δύναμη (composite force) μπορεί να αναλυθεί σε δύο συνιστώσες στον κατακόρυφο και οριζόντιο άξονα, αν σχεδιάσουμε το διάγραμμα ελεύθερου σώματος της περίπτωσης. Αυτές θα είναι η δύναμη ανύψωσης (Lift) η οποία δρα κατακόρυφα στον φαινόμενο άνεμο και γι' αυτό το λόγο θα βρίσκεται στον κατακόρυφο άξονα και η δύναμη απώθησης (Drag) η οποία δρα παράλληλα στον φαινόμενο άνεμο, γι' αυτό θα βρίσκεται στον οριζόντιο άξονα. Λόγω του 3^{ου} νόμου της κίνησης του Νεύτωνα, η δύναμη στην οποία αντιτίθεται η ανύψωση θα είναι το βάρος με φορά προς το κέντρο της γης και η δύναμη στην οποία αντιτίθεται η απώθηση θα είναι η δύναμη προώθησης (Thrust) με φορά φυσικά αντίθετη αυτής της δύναμης απώθησης. Το παρακάτω σχήμα αποτυπώνει την κατάσταση για ένα σημείο της αεροτομής στο οποίο προσκρούει ο φαινόμενος άνεμος.



Εικόνα 2-2 Αεροτομή (Airfoil) - Πλευρική Άποψη. Απεικονίζονται τα βασικά χαρακτηριστικά της και οι δυνάμεις που εφαρμόζονται[2]

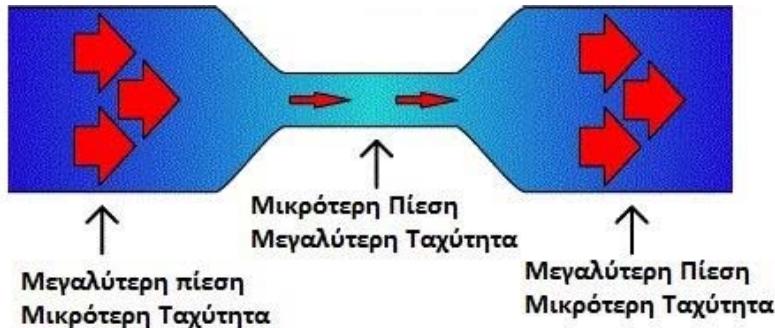
Η οξεία γωνία που σχηματίζεται μεταξύ του φαινόμενου ανέμου και της πρόσοψης της αεροτομής καλείται γωνία επίθεσης (Angle of Attack – AOA). Η γωνία μπορεί να λάβει οποιαδήποτε τιμή μεταξύ 0 και 90 μοιρών και να είναι είτε θετική (ανοδική πορεία του αεροσκάφους), είτε αρνητική (καθοδική πορεία του αεροσκάφους). Όταν μεταβάλλουμε τη γωνία αυτή μεταβάλλεται η διαφορά πίεσης στο άνω και κάτω μέρος της αεροτομής, όπως θα εξηγηθεί λεπτομερώς αργότερα.

Στη γενική περίπτωση, καθώς η γωνία επίθεσης αυξάνεται οι δυνάμεις ανύψωσης και απώθησης αυξάνονται με όμοιο τρόπο. Αυτή η ομοιόμορφη αύξηση στις δυνάμεις ακολουθείται μέχρις ενός σημείου, από το οποίο και έπειτα η δύναμη ανύψωσης μειώνεται απότομα και η δύναμη απώθησης καθίσταται η κύρια συνιστώσα. Η γωνία στην οποία αντιστοιχεί αυτό το κρίσιμο σημείο λέγεται γωνία υπεκφυγής (Stalling Angle), ή κρίσιμη γωνία επίθεσης (Critical Angle of Attack). Η γωνία υπεκφυγής είναι διαφορετική για κάθε αεροτομή, αλλά έχει παρατηρηθεί ότι για τις συνήθεις, τυπικές αεροτομές αντιστοιχεί στις 15° - 20° περίπου. Κάποια μαχητικά αεροσκάφη μπορούν να επιτύχουν πολύ μεγάλη AOA και η κρίσιμη γωνία τους μπορεί να φτάσει μέχρι και τις 45° , με το κόστος όμως μεγάλης δύναμης απώθησης. Από τη γωνία αυτή μέχρι τις 90 μοίρες η δύναμη απώθησης συνεχίζει να αυξάνεται καθώς η δύναμη ανύψωσης καταρρέει. Όταν η AOA γίνει 90° η αεροτομή μοιάζει με ένα σχήμα επίπεδου πιάτου, αντιτιθέμενο πλήρως στον αέρα, με δύναμη απώθησης μέγιστη και δύναμη ανύψωσης αμελητέα.

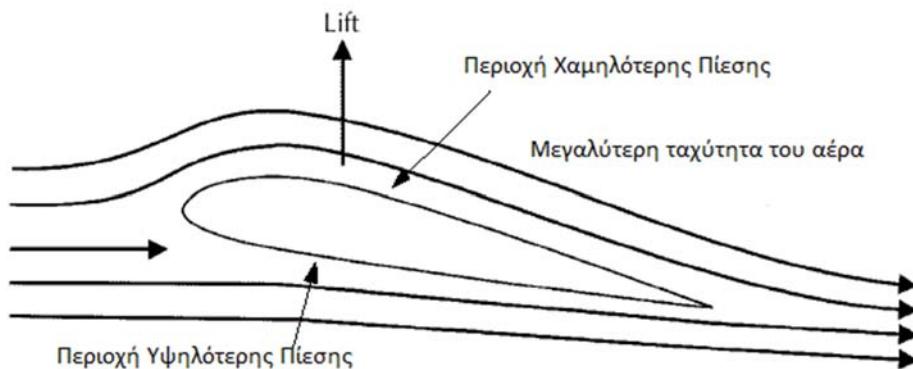
Η πτέρυγα ενός αεροπλάνου σχεδιάζεται με τέτοιο τρόπο ώστε η διατομή του να είναι μεγαλύτερη όσο προχωράμε προς τις ρίζες του, κοντά στην άτρακτο του αεροπλάνου. Αυτό το χαρακτηριστικό συστροφής των φτερών ονομάζεται washout και σκοπό έχει να διαφοροποιήσει τη κατανομή της δύναμης ανύψωσης καθ' όλο το μήκος των φτερών, ώστε η γωνία υπεκφυγής να συμβαίνει αρχικά στις ρίζες του φτερού και με περαιτέρω αύξηση της, στις παρυφές του. Επιπλέον, με αυτό τον τρόπο επιτυγχάνεται μεγαλύτερος έλεγχος του Aileron και δίνεται στον πιλότο χρόνος να αντιδράσει, πιθανώς μέσω ενός είδους συναγερμού για το washout, διότι με την αύξηση της γωνίας επίθεσης πέραν της γωνίας υπεκφυγής συμβαίνουν όλοι και περισσότερες αναταράξεις. Αυτές οι δονήσεις προκαλούνται διότι η πτέρυγα συναντά όλο και μεγαλύτερη αντίσταση από τον αέρα καθώς η γωνία επίθεσης αυξάνεται.

2.3 Αρχή του Bernoulli

Η αρχή του Bernoulli (Bernoulli's Principle) στηρίζεται στη θεμελιώδη αρχή της διατήρησης της ενέργειας. Ο ρυθμός ροής του όγκου ενός ρευστού από μια περιοχή υψηλότερης πίεσης σε μια περιοχή χαμηλότερης πίεσης μεταβάλλει την ταχύτητα του ρευστού. Αυτή η μεταβολή στην ταχύτητα του ρευστού πρέπει να προέλθει από μια δύναμη, η οποία επιταχύνει το υγρό κατά την εισχώρηση στην περιοχή μικρότερης πίεσης και το επιβραδύνει κατά την έξοδο από αυτή. Αυτή η δύναμη δε θα μπορούσε να έρθει από κάπου αλλού, παρά από το ίδιο το ρευστό!



Εικόνα 2-3 Απεικόνιση της αρχής του Bernoulli σε υγρό εντός σωλήνα[3]



Εικόνα 2-4 Η Αρχή του Bernoulli σε μια αεροτομή

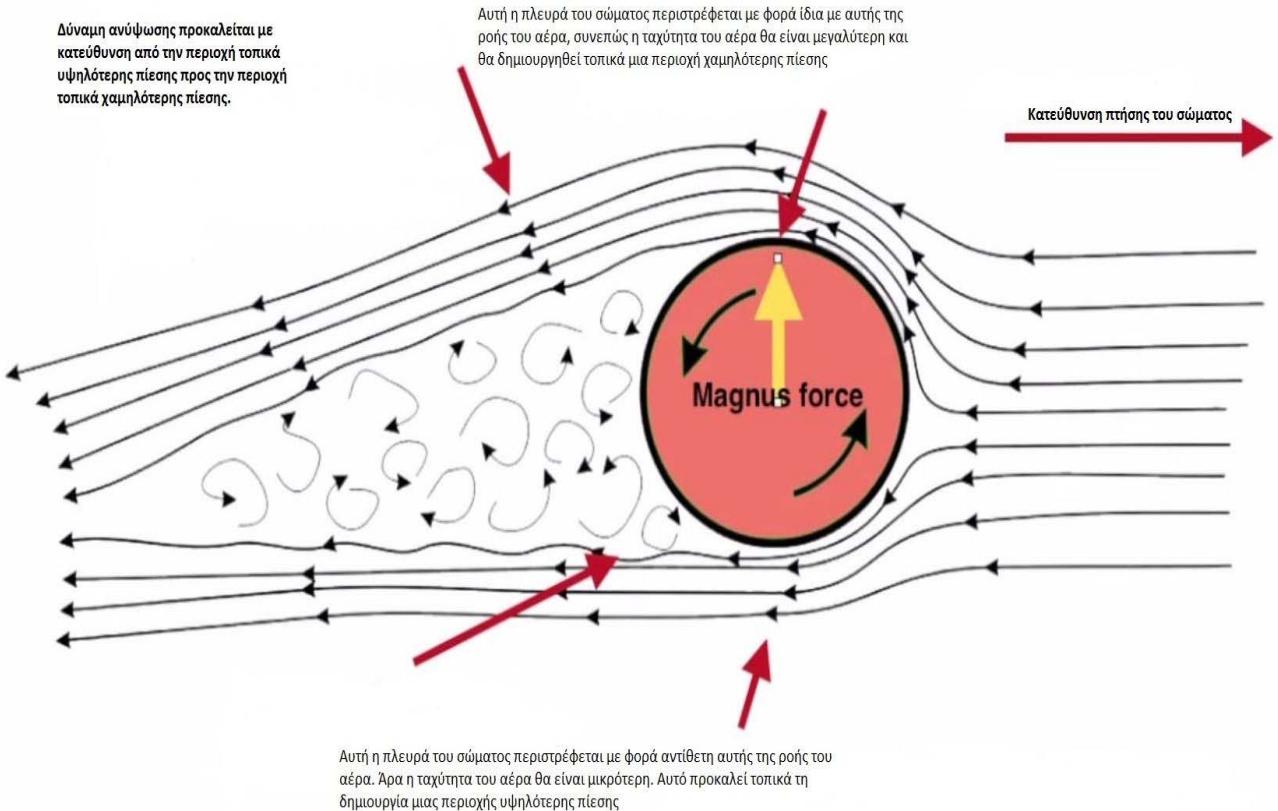
Σε κάθε περίπτωση το σώμα εκτρέπει τα σωματίδια του ρευστού απότομα προς τα πίσω του, καθώς διανύει την πορεία του, ασκώντας δύναμη σε αυτά. Επομένως εξαιτίας του 3^{ου} νόμου του Νεύτωνα το ρευστό θα ασκήσει μια δύναμη ίση σε μέτρο και αντίθετη σε φορά στο σώμα. Αυτή η δύναμη εκδηλώνεται ως εξής: σε μια περιοχή με μικρότερη πίεση αυξάνει την ταχύτητα του ρευστού (και οτιδήποτε βρίσκεται εντός αυτού) και σε μια περιοχή υψηλότερης πίεσης τη μειώνει.

2.4 Φαινόμενο Magnus

Το φαινόμενο, ή η δύναμη, Magnus (Magnus Effect) εφαρμόζεται σε ένα περιστρεφόμενο αντικείμενο που διασχίζει τον αέρα. Ανακαλύφθηκε από τον γερμανό φυσικό Gustav Magnus ο οποίος ήθελε να εξηγήσει γιατί η τροχιά μιας οβίδας κανονιού καμπυλώνει όταν διαδίδεται στον αέρα. Παρατήρησε λοιπόν ότι ο τρόπος περιστροφής μιας μπάλας, ή ενός κυλίνδρου που διαδίδεται σε ένα ρευστό προκαλεί μια δύναμη στο σώμα. Η συμπεριφορά μοιάζει με αυτή μιας αεροτομής στα αεροσκάφη και παίζει σημαντικό ρόλο στη διάδοση όλων τα ιπτάμενων αντικείμενων, καθώς τα υποβοηθά στην πτήση.

Γνωρίζουμε από την αρχή του Bernoulli ότι μεγαλύτερη συγκέντρωση ρευστού (αέρα) σε ένα χώρο (ή γενικότερα στο περιβάλλον) δημιουργεί μια περιοχή αυξημένης (ατμοσφαιρικής) πίεσης. Στο παρακάτω σχήμα βλέπουμε μια μπάλα που ίπταται στον αέρα περιστρεφόμενη με φορά αντίθετη αυτής των δεικτών του ρολογιού. Θα εξετάσουμε τι ακριβώς συμβαίνει κατά την πτήση της. Έστω

ότι η κατεύθυνση κίνησης της είναι από τα αριστερά προς τα δεξιά. Ο φαινόμενος άνεμος έχει φορά αντίθετη αυτής της κίνησης του σώματος. Παρατηρούμε ότι η κάτω πλευρά της μπάλας στριφογυρίζει με φορά αντίθετη αυτής της ροής του αέρα. Άρα τοπικά η ταχύτητα του αέρα θα είναι μικρότερη και αυτό θα προκαλέσει τη δημιουργία μιας περιοχής υψηλότερης πίεσης.



Εικόνα 2-5 Αναπαράσταση της Δύναμης Magnus[4]

Σε αυτό το συμπέρασμα μπορούμε να φτάσουμε και με διαφορετικό τρόπο, παρατηρώντας ότι η περιστροφή τοπικά του σώματος είναι σύμφωνη με την κατεύθυνση της κίνησης του, και εφόσον ο αέρας ρέει με φορά αντίθετη αυτής της κίνησης συναντά μεγαλύτερη αντίσταση και η ταχύτητα του μειώνεται. Αντίθετα, η άνω πλευρά του σώματος στρέφεται με φορά σύμφωνη με αυτή της ροής του αέρα, συνεπώς η ταχύτητα του αέρα θα είναι μεγαλύτερη και έτσι άνωθεν της μπάλας δημιουργείται μια περιοχή χαμηλότερης πίεσης.

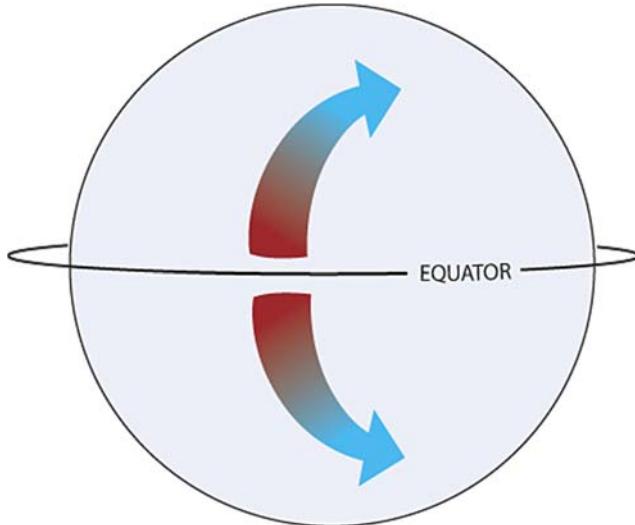
Η περιοχή υψηλότερης πίεσης «σπρώχνει» την περιοχή χαμηλότερης πίεσης, δημιουργώντας έτσι δύναμη ανύψωσης προς τα πάνω στο παράδειγμα μας. Αυτή η δύναμη ανύψωσης είναι η δύναμη Magnus. Όπως και στην αεροτομή, όσος περισσότερος αέρας διασχίζει το σώμα τόσο μεγαλύτερη θα είναι η δύναμη ανύψωσης. Επίσης όσο πιο ελαφρύ είναι το σώμα και όσο μεγαλύτερη είναι η απόσταση που διανύει, τόσο μεγαλύτερη θα είναι η επίδραση του αέρα πάνω του, δηλαδή τόσο περισσότερο θα εκτραπεί από ευθύγραμμη και ομαλή πορεία.

Στη περιοχή πίσω από την μπάλα, ο αέρας δημιουργεί δίνες. Σκεπτόμενοι ότι πρόκειται για μια αεροτομή, αυτές οι δίνες αντιπροσωπεύουν αναταράξεις, οι οποίες αυξάνονται καθώς αυξάνεται η γωνία επίθεσης και παίζουν πολύ σημαντικό ρόλο όταν η γωνία επίθεσης ξεπεράσει τη γωνία υπεκφυγής, όπως ήδη εξηγήσαμε.

2.5 Φαινόμενο Coriolis

Το φαινόμενο, ή η δύναμη, Coriolis (Coriolis Effect) είναι γνωστό και ως αρχή διατήρησης της στροφορμής. Έχει παρατηρηθεί ότι ένα αντικείμενο εν κινήσει φαίνεται να εκτρέπεται από την

πορεία του σαν μια δύναμη να το σπρώχνει πλαγίως. Λόγω της φοράς ιδιοπεριστροφής της γης από τη Δύση προς την Ανατολή, τοποθεσίες πιο κοντά στον ισημερινό περιστρέφονται γύρω από τον άξονα της με μεγαλύτερη ταχύτητα από τοποθεσίες πιο κοντά στους πόλους. Το ίδιο συμβαίνει με τα αντικείμενα τα οποία, επηρεαζόμενα από το φαινόμενο, κινούμενα στο βόρειο και στο νότιο ημισφαίριο της παρεκκλίνουν από την πορεία τους με αντίθετο τρόπο. Στο Βόρειο ημισφαίριο αντικείμενα που κινούνται στον αέρα εκτρέπονται από την ευθύγραμμη κίνηση τους προς τα δεξιά (με σημείο αναφοράς το σημείο που εκκίνησαν). Στο Νότιο ημισφαίριο αντικείμενα παρεκκλίνουν προς τα αριστερά. Αν ένα αντικείμενο ίπταται κατά μήκος της γραμμής του Ισημερινού η τροχιά του δε καμπυλώνει καθόλου. Το φαινόμενο επηρεάζει την κατεύθυνση ενός αντικειμένου, όχι την ταχύτητα του. Το μέγεθος της εκτροπής όμως, είναι ανάλογο της ταχύτητας του αντικειμένου.



Εικόνα 2-6 Κατεύθυνση εκτροπής της κίνησης αντικειμένων στο Βόρειο και Νότιο Ημισφαίριο της γης, εξαιτίας του φαινομένου Coriolis[5]

Επίσης, όσο πιο μικρή είναι η απόσταση που διανύει το ρευστό τόσο πιο μικρή είναι η επίδραση του φαινομένου Coriolis πάνω του. Με τον ίδιο τρόπο που επηρεάζονται τα αντικείμενα, το φαινόμενο έχει αντίστοιχη επίδραση στον ίδιο τον αέρα! Δηλαδή ο αέρας (ως ρευστό) παρεκκλίνει από την αρχική του πορεία προς τα δεξιά στο βόρειο ημισφαίριο και προς τα αριστερά στο νότιο ημισφαίριο. Αυτός είναι και ο λόγος που οι τυφώνες στο Βόρειο ημισφαίριο έχουν φορά αντίθετη της φοράς των δεικτών του ρολογιού και στο Νότιο ημισφαίριο συμφωνούν με τη φορά του ρολογιού.

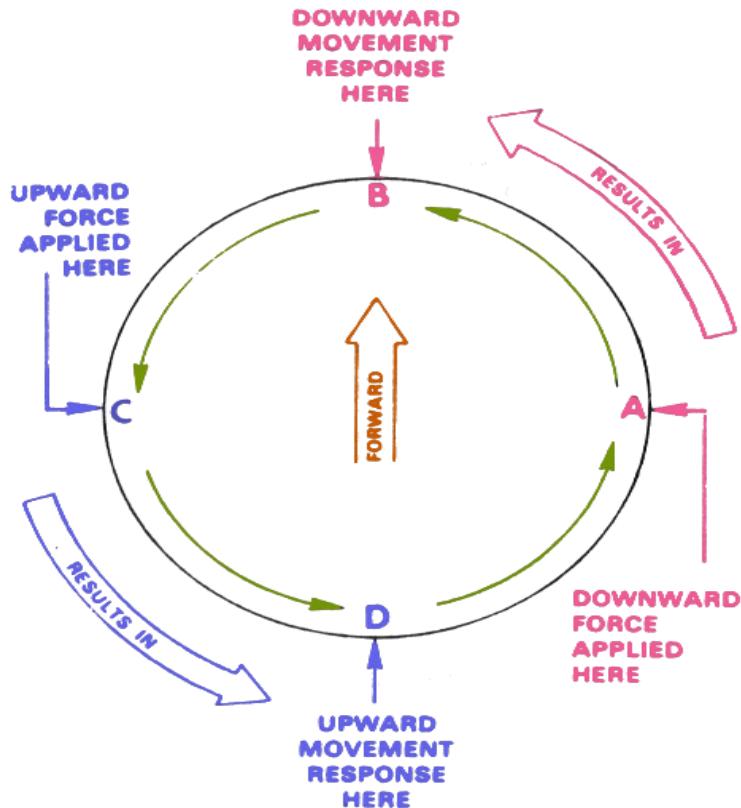
Οι πιλότοι αεροσκαφών πρέπει να λαμβάνουν υπόψη τους το φαινόμενο Coriolis. Ο τελικός τους προορισμός μπορεί να έχει «μετακινηθεί» αρκετά χιλιόμετρα μακριά, από εκεί που θα βρισκόταν εάν έκαναν το ταξίδι τους σε μια ακίνητη σφαίρα. Το ίδιο και οι ναυτικοί και θαλασσοπόροι με τη ροή της παλίρροιας με τους ανέμους, πρέπει να προγραμματίζουν για το μέλλον και να επιλέγουν με ακρίβεια το σημείο εκκίνησης και προορισμού. Οποιαδήποτε άλλη πορεία θα απαιτούσε περισσότερα καύσιμα. Φυσικά για τους πιλότους τηλεκατευθυνόμενων εναέριων οχημάτων το φαινόμενο είναι αμελητέο.

2.6 Φαινόμενο του Γυροσκοπίου

Το φαινόμενο του γυροσκοπίου (Gyroscopic Precession) υφίσταται στα περιστρεφόμενα σώματα στα οποία μια ασκούμενη δύναμη εκδηλώνεται 90 μοίρες αργότερα στην κατεύθυνση της περιστροφής. Συγκεκριμένα για ένα ελικόπτερο, ο περιστρεφόμενος κύριος έλικας του παίζει το ρόλο γυροσκοπίου. Όταν ασκείται μια δύναμη στο ελικόπτερο (δύναμη περιστροφής των ελίκων του) η δύναμη ασκείται ορθογώνια στο επίπεδο περιστροφής και δρα 90° εκτός φάσης από τη χρονική

στιγμή εφαρμογής της. Γι' αυτό το λόγο οι έλικες του ελικοπτέρου λειτουργούν με διαφορά φάσης 90 μοιρών. Το παρακάτω διάγραμμα απεικονίζει πως το φαινόμενο επηρεάζει την προπέλα όταν η δύναμη εφαρμόζεται σε ένα συγκεκριμένο σημείο.

Μια δύναμη προς τα κάτω στο σημείο A εκδηλώνεται 90° αργότερα στο σημείο B. Δηλαδή η συμπεριφορά της προπέλας μεταβάλλεται στο σημείο B, 90° αφότου δόθηκε η οδηγία. Η εφαρμογή μιας δύναμης προς τα πάνω στο σημείο C έχει ως αποτέλεσμα την εκδήλωση της στο σημείο D.



Εικόνα 2-7 Εκδήλωση φαινομένου γυροσκοπίου σε προπέλα ελικοπτέρου[6]

Λάβαμε ως παράδειγμα το ελικόπτερο. Τα συμπεράσματα όμως που εξάχθηκαν εδώ μπορούν να επεκταθούν και στις προπέλες ενός UAV. Παρόλο που το φαινόμενο δεν λαμβάνει κυρίαρχο ρόλο στην αεροδυναμική πορεία του ελικοπτέρου, δεν είναι αμελητέο και πρέπει να λαμβάνεται υπόψη.

Συνήθως τα γυροσκόπια παίρνουν τη μορφή ενός αντικειμένου με μορφή κυκλικού δίσκου, αλλά το φαινόμενο του γυροσκοπίου εκδηλώνεται γενικότερα σε περιστρεφόμενα αντικείμενα.

2.7 Αρχή Επίδρασης του Εδάφους

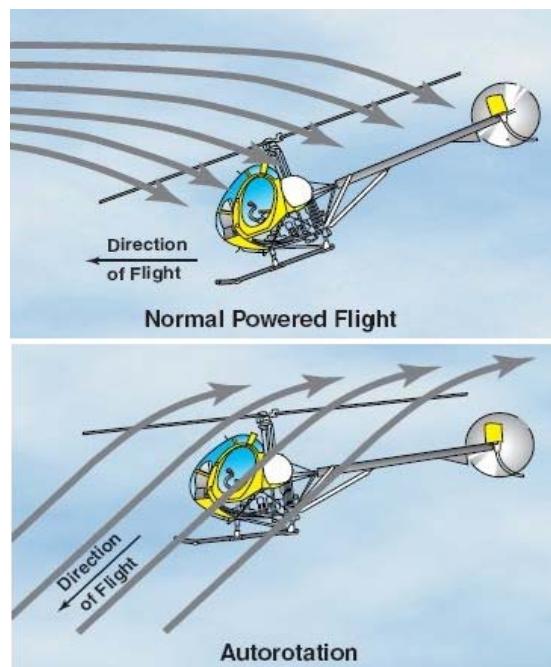
Η αρχή επίδρασης του εδάφους (Ground Effect) είναι το φαινόμενο κατά το οποίο σημειώνεται μείωση των δυνάμεων τριβής στις πτέρυγες ενός αεροπλάνου καθώς αυτό πλησιάζει το έδαφος. Όταν ένα αεροπλάνο ίπταται σε χαμηλή απόσταση από μια σταθερή επιφάνεια υφίσταται αυξημένη δύναμη ανύψωσης και μειωμένη δύναμη απώθησης. Αυτή η μείωση στη δύναμη απώθησης αποτρέπει τη μείωση της ταχύτητας του αεροπλάνου. Ο τύπος της σταθερής επιφάνειας, πχ. έδαφος, επιφάνεια της θάλασσας, διάδρομος αεροπλανοφόρου κα., μεταβάλλει την ισχύ του φαινομένου. Για παράδειγμα ένα αεροπλάνο που αιωρείται κοντά στην επιφάνεια της θάλασσας θα συναντούσε μεγαλύτερη αντίσταση, δηλαδή θα ανέπτυσσε μεγαλύτερη δύναμη απώθησης, σε σύγκριση με το αιωρούνταν κοντά στην επιφάνεια της Γής.

Το φαινόμενο εκδηλώνεται όταν το αεροπλάνο ίπταται σε απόσταση έως και 1.5 φορές το εκπέτασμα των πτερύγων του από τη σταθερή επιφάνεια. Όσο πλησιάζει τη σταθερή επιφάνεια η

ένταση του φαινομένου αυξάνεται εκθετικά. Κατασκευαστές αεροπλάνων (κυρίως ρώσοι) εκμεταλλευόμενοι το φαινόμενο έχουν αναπτύξει ένα εντελώς νέο είδος αεροσκάφους, το ekranoplan, ή Ground-Effect vehicle το οποίο έχει σχεδιασθεί ώστε να μην μπορεί να ξεφύγει από το φαινόμενο εδάφους. Εξαιτίας των υψηλών επιδόσεων της περιοχής αυτής ένα αεροσκάφος αυτού του είδους μπορεί να αναπτύξει αρκετά υψηλές ταχύτητες, ενώ παράλληλα είναι πολύ οικονομικό στην κατανάλωση καυσίμων. Παρ' όλα αυτά τέτοια αεροπλάνα δεν έχουν αναγνωριστεί και χρησιμοποιηθεί ευρύτερα. Το φαινόμενο δεν ισχύει σε στροφιόπτερα, δηλαδή σε αεροσκάφη που η περιστροφή των προπελών τους γίνεται παράλληλα με το επίπεδο του εδάφους, το αντίθετο μάλιστα, αφού τότε προκαλούνται πολλές αναταράξεις και δονήσεις στο όχημα, που αφενός τείνουν να το εκτρέψουν από την πορεία του, αφετέρου μειώνουν τη ταχύτητα του.

2.8 Αυτοπεριστροφή

Αυτοπεριστροφή (Autorotation) είναι μια κατάσταση πτήσης ενός ελικοπτέρου, ή παρόμοιων αεροσκαφών, κατά την οποία το σύστημα τροφοδοσίας της μηχανής είναι απενεργοποιημένο και η περιστροφή των ελίκων επιτυγχάνεται από τη ροή του αέρα διαμέσου αυτών, καθώς το ελικόπτερο κατέρχεται. Τονίζεται ότι κατά τη διάρκεια κανονικής λειτουργίας τροφοδοσίας, ο αέρας απορροφάται (/«τεμαχίζεται») από το πάνω μέρος στις προπέλες κ' έπειτα εξατμίζεται προς τα πίσω και προς τα κάτω, παράγοντας δύναμη προώθησης και ανύψωσης αντίστοιχα. Σε συνθήκες αυτοπεριστροφής η προπέλα περιστρέφεται ελεύθερα με την άτακτη ροή του ανέμου μέσα από αυτή. Χρησιμοποιείται ορισμένες φορές σκοπίμως για εξοικονόμηση καυσίμων και είναι η μοναδική επιλογή σε περίπτωση πλήρης αποτυχίας του συστήματος τροφοδοσίας της μηχανής.

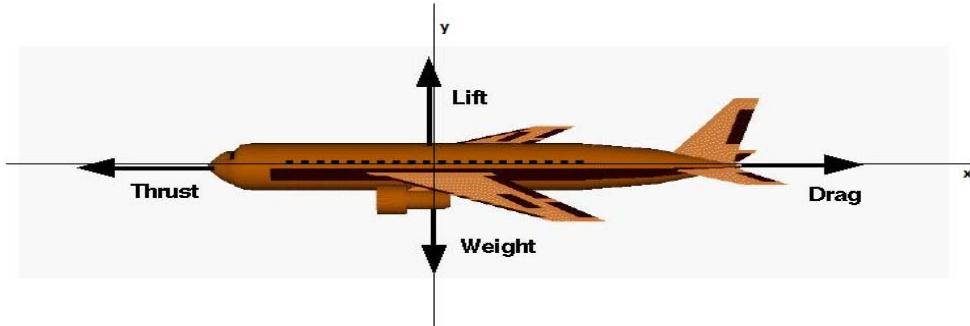


Εικόνα 2-8 Παρουσίαζονται τα ρεύματα αέρα στο ελικόπτερο σε κανονικές συνθήκες λειτουργίας (πάνω) και σε συνθήκες λειτουργίας αυτοπεριστροφής (κάτω)[7]

2.9 Δυνάμεις που ασκούνται σε ένα Αεροσκάφος

Σε αυτό το χωρίο θα περιγραφούν οι δυνάμεις που ασκούνται σε κάθε αεροσκάφος, επανδρωμένο ή όχι. Υφίστανται πάντοτε σε όλα τα ιπτάμενα οχήματα και η κατανόηση τους είναι πολύ σημαντική για τους μηχανικούς και τους πιλότους των αεροσκαφών. Οι μεν θα πρέπει να σχεδιάσουν τα συστατικά στοιχεία του αεροσκάφους βάσει των νόμων που διέπουν τις δυνάμεις

αυτές και οι δε θα πρέπει να κατέχουν τη γνώση για να μπορούν να πιλοτάρουν αποτελεσματικά το όχημα. Στη προκειμένη περίπτωση εμείς έχουμε το ρόλο και ενός μηχανικού, αλλά και του πιλότου. Ως μηχανικοί έχουμε επιλέξει τα κομμάτια του μη επανδρωμένου αεροσκάφους τεσσάρων ελίκων βάσει αυστηρών εξισώσεων που θα παρουσιάσουμε εκτενώς στο επόμενο κεφάλαιο. Επίσης ως πιλότοι του αεροσκάφους η πλήρης κατανόηση των δυνάμεων που του ασκούνται και η γνώση βασικής αεροδυναμικής μας καθιστά ικανούς για την πτήση του.



Εικόνα 2-9 Οι δυνάμεις που εφαρμόζονται σε ένα αεροπλάνο. Εφάμιλλο είναι το διάγραμμα για στροφιόπτερα. Θεωρούμε ότι οι δυνάμεις ασκούνται στο κέντρο βάρους του αεροσκάφους[8]

Σχεδιάζοντας ένα διάγραμμα ελεύθερου σώματος και λαμβάνοντας ορθοκανονικό σύστημα αξόνων αναφοράς το οποίο διέρχεται από το κέντρο βάρους του αεροπλάνου και εκτείνεται στον οριζόντιο άξονα (x) και στον κατακόρυφο άξονα (y), οι δυνάμεις αυτές, σύμφωνα με την παραπάνω εικόνα, είναι οι εξής: η δύναμη ανύψωσης (Lift) προς τα πάνω, η δύναμη προώθησης (Thrust) προς τα αριστερά/μπροστά, η δύναμη του βάρους (Weight) προς τα κάτω και η δύναμη απώθησης (Drag) προς τα δεξιά/πίσω.

Η δύναμη απώθησης είναι η αντίθετη της δύναμης προώθησης και το βάρος είναι αντίθετο της δύναμης ανύψωσης. Λέμε αντίθετες δυνάμεις με την έννοια της φυσικής και συγκεκριμένα αυτής του 3^{ου} νόμου του Νεύτωνα, του νόμου δράσης-αντίδρασης, δηλαδή ότι η μια δύναμη δεν μπορεί να υφίσταται χωρίς την άλλη και ότι η εφαρμογή της μίας συνεπάγεται μία ίσου μέτρου και αντίθετης φοράς δύναμη. Οι δυνάμεις για τις οποίες ενδιαφερόμαστε δηλαδή αυτές που θέλουμε να μεγιστοποιήσουμε είναι η προώθηση και η ανύψωση, όλες όμως είναι απαραίτητες για την ομαλή πορεία του οχήματος, επειδή για να κινείται το αεροσκάφος σε ευθεία γραμμή και με σταθερή ταχύτητα, θα πρέπει η δύναμη ανύψωσης να είναι περίπου ίση με το βάρος και η δύναμη προώθησης περίπου ίση με τη δύναμη απώθησης. Στη πραγματικότητα οι τέσσερις δυνάμεις κατανέμονται σε όλη την έκταση του αεροσκάφους με ανομοιόμορφο και περίπλοκο τρόπο, αλλά εμείς θεωρούμε ότι επιδρούν σε ένα και μοναδικό σημείο, το κέντρο του αεροσκάφους.

2.9.1 Δύναμη Προώθησης (Thrust)

Για την υπερνίκηση της δύναμης απώθησης, τα αεροπλάνα χρησιμοποιούν ένα σύστημα προώθησης για να προωθηθούν στην επιθυμητή κατεύθυνση. Κατ' επέκταση η δύναμη ασκείται με διεύθυνση εκείνη της πορείας του σκάφους και φορά αντίθετη της δύναμης απώθησης. Η προώθηση είναι μηχανική δύναμη, οπότε το σύστημα προώθησης πρέπει να έχει φυσική επαφή με ένα ρευστό μέσο για να παράγει ωφέλιμη προώθηση. Το ρευστό μέσο στο οποίο «επιπλέουν» τα αεροσκάφη είναι ο αέρας της ατμόσφαιρας. Επομένως η προώθηση οφείλεται κατ' ουσίαν σε αέρα που σπρώχνει αέρα! Υπάρχουν δύο συνιστώσες της προώθησης, το μέγεθος της και η ταχύτητα της. Το χαρακτηριστικό pitch της προπέλας ελέγχει την ταχύτητα της προώθησης, ενώ το μήκος της

προπέλας ελέγχει το μέγεθος της προώθησης. Υπάρχουν δύο είδη προωθητικής δύναμης, η στατική προώθηση και η δυναμική προώθηση.

2.9.1.1 Στατική Προώθηση (Static Thrust)

Η προώθηση που παράγεται από μια μηχανή, όταν το αεροσκάφος είναι ακίνητο στο έδαφος, με τη μηχανή σε λειτουργία. Δεν υπάρχει αέρας που μετακινείται προς την προπέλα εξαιτίας της απουσίας εμπρόσθιας ταχύτητας του αεροσκάφους.

2.9.1.2 Δυναμική Προώθηση (Dynamic Thrust)

Είναι η προώθηση που παράγεται από τη μηχανή του, εν κινήσει πλέον, αεροσκάφους με αναφορά προς τη γη.

2.9.2 Δύναμη Ανύψωσης (Lift)

Η δύναμη ανύψωσης παράγεται από την κίνηση του αεροπλάνου για να υπερβεί το βάρος του. Η δύναμη ανύψωσης έχει κατεύθυνση κάθετη στη κατεύθυνση κίνησης του σκάφους με φορά αντίθετη αυτής του βάρους. Το μέτρο της δύναμης ανύψωσης εξαρτάται από πολλούς παράγοντες συμπεριλαμβανομένου του σχήματος, των διαστάσεων του αεροσκάφους και της ταχύτητας του. Όπως και με το βάρος, κάθε τμήμα του αεροσκάφους συνεισφέρει στη δύναμη ανύψωσης. Το κύριο μέρος της δύναμης ανύψωσης προέρχεται από τις πτέρυγες για αεροσκάφη σταθερών πτερύγων, ή τις προπέλες για στροφιόπτερα.

2.9.3 Δύναμη Απώθησης (Drag)

Η δύναμη απώθησης οφείλεται στην αντίσταση που προβάλει ο αέρας στη κίνηση του αεροσκάφους. Έχει κατεύθυνση αντίτιθέμενη στην κατεύθυνση πτήσης. Όπως και με τη δύναμη ανύψωσης υπάρχουν αρκετοί παράγοντες που επηρεάζουν το μέγεθος της δύναμης αυτής, όπως το σχήμα του αεροσκάφους, η πυκνότητα και η πίεση του αέρα και η ταχύτητα του αεροσκάφους. Είναι αποτέλεσμα πολλών επιμέρους «απωθητικών» δυνάμεων -συνιστωσών- που εφαρμόζονται στο αεροσκάφος κατά την πτήση του, όπως profile drag, επαγόμενη (induced) drag, παρασιτική (parasite) drag η ανάλυση των οποίων ξεφεύγει από τους σκοπούς της συγκεκριμένης εργασίας. Συνυπολογίζοντας κατάλληλα όλες αυτές τις συνιστώσες παίρνουμε τη συνισταμένη τους, τη δύναμη απώθησης. Για ευθύγραμμη και ομαλή πτήση η δύναμη απώθησης θα πρέπει να εξουδετερώνεται με τη δύναμη προώθησης, ενώ κατά την απογείωση και τη διεξαγωγή ελιγμών του αεροπλάνου θα πρέπει να είναι μικρότερη.

Επισημαίνεται ότι ο σκοπός της μηχανής ενός αεροπλάνου είναι η υπερνίκηση της δύναμης απώθησης, όχι η πτήση του. Οι μηχανισμοί των πτερύγων του αεροπλάνου το υπερψύωνουν, όχι οι ηλεκτρικές μηχανές του.

2.9.4 Βάρος (Weight)

Σε όλα τα αντικείμενα ασκείται η δύναμη του βάρους εξαιτίας της βαρύτητας. Το βάρος ασκείται πάντα κατακόρυφα από το αεροσκάφος με φορά που δείχνει προς το κέντρο της γης. Το μέτρο της δύναμης εξαρτάται από την ολική μάζα του αεροσκάφους, συμπεριλαμβάνοντας τα καύσιμα, τις αποσκευές και τον αριθμό των επιβατών.

Για κάθε αεροσκάφος οι δυνάμεις προώθησης και ανύψωσης καθορίζουν την κατεύθυνση κίνησης του και για επιθυμητές συνθήκες πτήσης θα πρέπει σχεδόν συνεχώς να είναι μεγαλύτερες από τις δυνάμεις απώθησης και βάρους, αντίστοιχα. Είναι εκείνες οι δυνάμεις που θα θέλαμε να μεγιστοποιούμε με τον ευκολότερο δυνατό τρόπο, δηλαδή δαπανώντας το δυνατό λιγότερη ενέργεια.

Κατά την εκκίνηση του αεροπλάνου οι δυνάμεις ανύψωσης και προώθησης είναι αμελητέες συγκριτικά με τις αντίστοιχες δυνάμεις απώθησης και βάρους και σταδιακά αναπτύσσονται καθώς N. Λαζαρίδης

το αεροπλάνο απογειώνεται. Όταν ο συνδυασμός των δυνάμεων προώθηση και ανύψωση υπερκεράσουν το συνδυασμό των δυνάμεων βάρος και απώθηση τότε το αεροπλάνο δύναται να απογειωθεί.

Στη περίπτωση στροφιοπτέρων, πριν την απογείωση του οι δυνάμεις προώθησης και απώθησης δεν υφίστανται διότι η άτρακτος/ο σκελετός του οχήματος δεν μετακινείται. Τη στιγμή που η δύναμη ανύψωσης γίνει μεγαλύτερη από το βάρος το ελικόπτερο μπορεί να απογειωθεί και τότε μπορούν να εφαρμοστούν όλες οι δυνάμεις ανάλογα με την επιθυμητή διαδρομή.

2.10 Λειτουργίες Πτήσης Αεροσκαφών

Αυτή η ενότητα περιγράφει τις λειτουργίες ελέγχου της πτήσης που ο πιλότος χρησιμοποιεί για να κατευθύνει την πορεία και τον προσανατολισμό του αεροσκάφους. Αναφέρουμε ότι τα συγκεκριμένα χαρακτηριστικά των μηχανισμών ελέγχου πτήσης διαφέρουν ανάλογα με το είδος του αεροσκάφους. Εδώ θα περιγράψουμε τους πιο βασικούς και κοινώς χρησιμοποιούμενους πλέον τρόπους πτήσης ενός αεροσκάφους. Τα πιο βασικά όργανα ελέγχου δρουν μηχανικά, τα χειριζόμαστε με ένα σύνολο μοχλών και πηδαλίων και παρατηρούμε την κατάσταση τους μέσα από μια αλληλουχία οργάνων.

Ο πιλότος αεροσκάφους, αρκεί να προσαρμόζει τέσσερις βασικές λειτουργίες για να έχει τις επιθυμητές συνθήκες πτήσης. Αυτές είναι οι λεγόμενες Pitch, Roll, Yaw και Throttle. Καθεμιά επιφέρει ένα συγκεκριμένο τρόπο κίνησης ζωτικό για την πτήση του αεροσκάφους. Ένας έμπειρος και επιδέξιος πιλότος εφαρμόζει συνήθως πολλαπλούς μηχανισμούς ταυτόχρονα, για να κατευθύνει αποτελεσματικά το όχημα.

2.10.1 Pitch

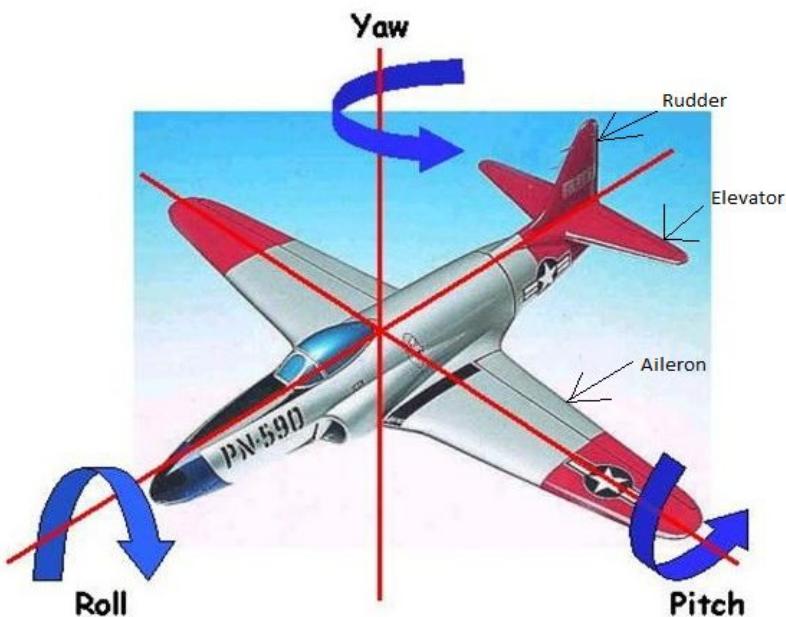
Η λειτουργία ελέγχου πτήσης pitch μεταβάλει την κλίση του αεροσκάφους, σε αναφορά προς το ορθογώνιο σύστημα αξόνων του. Με θετική κλίση το αεροσκάφος αυξάνει το ύψος του από το έδαφος, με αρνητική κλίση «κατηφορίζει» προς χαμηλότερο ύψος. Το pitch εκτελείται μηχανικά από τις δύο οπίσθιες πτέρυγες του αεροπλάνου, όταν ένα τμήμα τους σε σχήμα ορθογωνίου, το οποίο ονομάζεται Elevator, μετακινείται είτε προς τα πάνω οπότε το αεροπλάνο αναρριχάται, είτε προς τα κάτω και τότε το αεροπλάνο κατηφορίζει. Σε κάθε περίπτωση ρυθμίζονται οι δυνάμεις που ασκεί ο αέρας στις οπίσθιες πτέρυγες που τελικά αυτό οδηγεί σε μεταβολή της δύναμης ανύψωσης σε αυτά.

2.10.2 Roll

Η λειτουργία ελέγχου πτήσης roll χρησιμοποιείται για να στρίψει το αεροσκάφος δεξιά ή αριστερά. Εκτελείται μηχανικά με αντίστοιχο τρόπο, με τη διαφορά ότι το τμήμα των πτερύγων που ενεργοποιείται τώρα, βρίσκεται στα μπροστινά κύρια πτερύγια του αεροπλάνου και ονομάζεται Aileron. Για περιστροφή δεξιά το τμήμα του δεξιού φτερού κλίνει προς τα κάτω ενώ ταυτόχρονα το τμήμα του αριστερού φτερού κλίνει προς τα πάνω. Το αντίθετο συμβαίνει για στροφή αριστερά. Το μέγεθος της κλίσης των τμημάτων αυτών και η διάρκεια τους μέχρι να επιστρέψουν στην κανονική ευθύγραμμή θέση καθορίζουν το μέγεθος της στροφής.

2.10.3 Yaw

Η λειτουργία ελέγχου πτήσης yaw χρησιμοποιείται για να στριφογυρίσει το αεροσκάφος γύρω από άξονα κατακόρυφο που διέρχεται από το κέντρο βάρος του. Τότε το αεροσκάφος περιστρέφεται είτε με τη φορά των δεικτών του ρολογιού, είτε με φορά αντίθετη των δεικτών του ρολογιού, διατηρώντας παράλληλα το ίδιο υψόμετρο και κλίση πτήσης.



Εικόνα 2-10 Μηχανισμοί ελέγχου Αεροπλάνου[9]

Αυτό θα μπορούσε να επιτευχθεί με τη μεταστροφή του οχήματος μόνο από ένα σημείο κατά μήκος του. Η λειτουργία επιτυγχάνεται με την οπίσθια πτέρυγα που βρίσκεται κατακόρυφα ως προς την άτρακτο του αεροπλάνου όταν ένα ορθογώνιο τμήμα της, το οποίο ονομάζεται Rudder, εκτρέπεται είτε προς τα δεξιά οπότε έχουμε περιστροφή με τη φορά των δεικτών του ρολογιού (yaw right), είτε προς τα αριστερά οπότε έχουμε περιστροφή με φορά αντίθετη των δεικτών του ρολογιού (yaw left).

Στο ελικόπτερο ο μηχανισμός yaw επιτυγχάνεται με την ενεργή περιστροφή του στροφέα της ουράς, καθώς η κύρια προπέλα περιστρέφεται κανονικά.

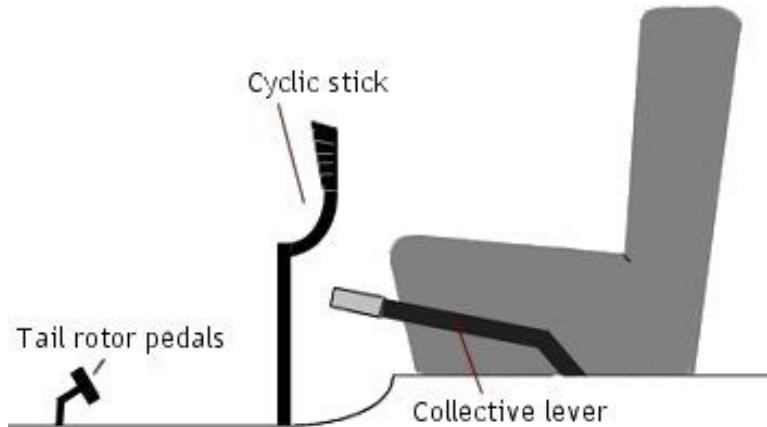
2.10.4 Throttle

Η τελευταία λειτουργία ελέγχου πτήσης είναι ίσως η πιο σημαντική, το throttle, με το οποίο προσαρμόζεται η ισχύς που παρέχεται από τη μηχανή του αεροσκάφους, αυξομειώνοντας το υψόμετρο πτήσης. Εκτελείται με τον μοχλό προώθησης (Thrust Lever) στα αεροπλάνα. Μεγάλα αεροπλάνα έχουν πολλούς τέτοιους μοχλούς ο καθένας για την εκάστοτε μηχανή που κατευθύνει.

Ο πιλότος ελικοπτέρου χρησιμοποιεί δύο βασικούς μοχλούς και πετάλια για ρύθμιση της αντιροπής (Anti-Torque pedals) της οπίσθιας προπέλας. Οι δύο ράβδοι-μοχλοί ονομάζονται Συλλογική ράβδος (Collective Stick) και Κυκλική ράβδος (Cyclic Stick).

Η συλλογική ράβδος βρίσκεται στην πλευρά του πιλότου και είναι αρμόδια για τους συλλογικούς ελέγχους (Collective Control) των σκάφους. Αναγκάζει το ελικόπτερο να μετακινηθεί ανοδικά ή καθοδικά, αυξάνοντας ή μειώνοντας αντίστοιχα την ισχύ τροφοδοσίας στη μηχανή, η οποία έπειτα αυξάνει τις στροφές ανά λεπτό (Revolutions per Minute – RPM) του βασικού στροφέα. Συνεπώς ο μοχλός ελέγχει το υψόμετρο πτήσης δηλαδή το throttle. Ονομάζεται και μοχλός ισχύος (Power Lever) και συνοδεύει και τα αεροπλάνα με παρόμοιο τρόπο.

Η Κυκλική ράβδος παρευρίσκεται στο μπροστινό μέρος του πιλότου και ρυθμίζει τους λεγόμενους Κυκλικούς Ελέγχους (Cyclic Controls). Γέρνοντας τον μοχλό προς μια κατεύθυνση, μεταφέρεται με όμοιο τρόπο ολόκληρο το ελικόπτερο προς την αντίστοιχη κατεύθυνση. Δηλαδή, οι κυκλικοί έλεγχοι ρυθμίζουν την κίνηση μπροστά, πίσω, αριστερά & δεξιά αλλάζοντας κατεύθυνση στο επίπεδο περιστροφής της βασικής προπέλας. Η κυκλική ράβδος σε ένα αεροπλάνο μπορεί να παρομοιαστεί με τιμόνι αυτοκινήτου.



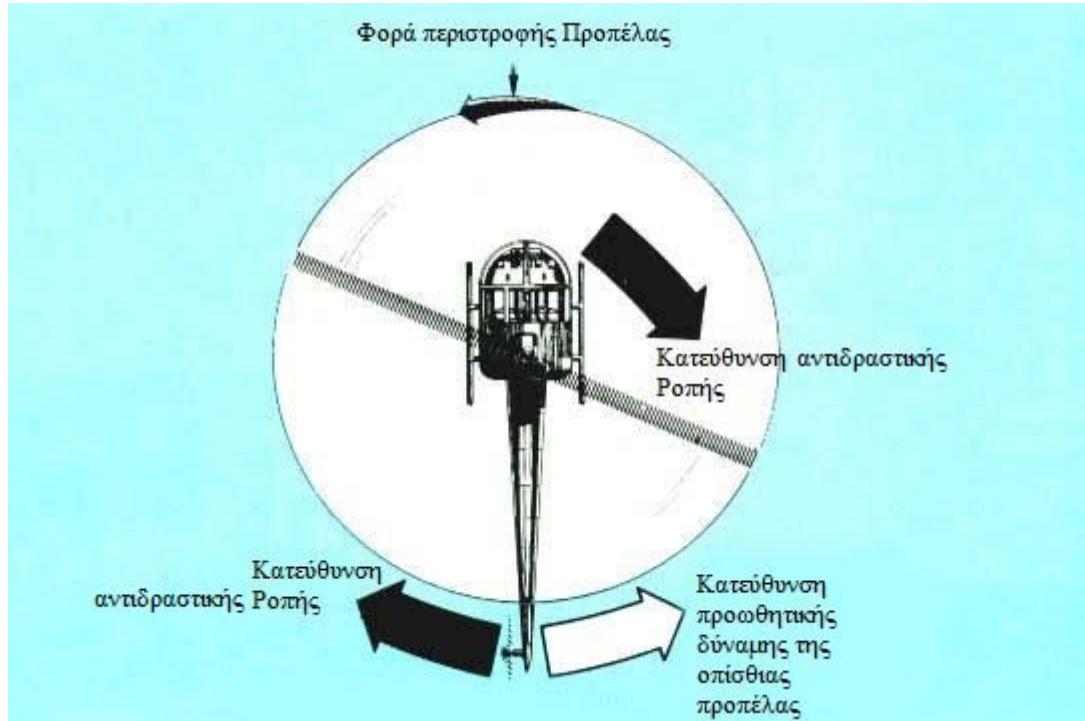
Εικόνα 2-11 Παρουσιάζονται τα πηδάλια ελέγχου πτήσης των ελικοπτέρων. Αντίστοιχοι μηχανισμοί υπάρχουν στα αεροπλάνα και λοιπά αεροσκάφη[10]

2.11 Αντίδραση Ροπής

Το φαινόμενο της αντίδρασης της ροπής (Torque Reaction) είναι αντίστοιχο του 3^{ου} νόμου του Νεύτωνα για τα συστήματα που κινούνται υπό γωνία, στο πεδίο που ασχολούμαστε τα ελικόπτερα και γενικότερα τα στροφιόπτερα (άρα υφίσταται και στα UAV πολυκόπτερα – multicopters). Θα εξηγήσουμε εδώ πως εφαρμόζεται στην κύρια προπέλα του ελικοπτέρου. Ο 3^{ος} νόμος του Νεύτωνα γενικά χρησιμοποιείται για ευθύγραμμες ομαλές δυνάμεις, εφαρμόζεται όμως και σε συστήματα που βρίσκονται υπό περιστροφή. Όπως μια γραμμική δύναμη μπορεί να επιταχύνει μια μάζα ευθύγραμμα και τότε της εφαρμόζεται τριβή προς την αντίθετη κατεύθυνση, με τον ίδιο τρόπο η ροπή (περιστροφική δύναμη) μπορεί να προκαλέσει την γωνιακή επιτάχυνση της μάζας. Η ίση και αντίθετη δύναμη που αντιθέται στην εφαρμοζόμενη ροπή είναι γνωστή ως αντίδραση ροπής.

Ο σωστός χειρισμός αυτής της αντίδρασης είναι αναπόσπαστο κομμάτι πτήσης ενός ελικοπτέρου. [11] Το ελικόπτερο παράγει δύναμη ανύψωσης περιστρέφοντας ένα ζευγάρι από έλικες, που σπρώχνουν τον αέρα προς τα κάτω από την βασική προπέλα. Η ίση και αντίθετη αντίδραση σε αυτή την δύναμη, είναι η δύναμη από τον αέρα προς το ελικόπτερο που το προωθεί προς τα πάνω. Δεν τελειώνει όμως εκεί. Οι έλικες καθώς περιστρέφονται προς μια κατεύθυνση με τεράστια ταχύτητα, λόγω της αντίδρασης ροπής το σώμα του ελικοπτέρου αποκτά την τάση να περιστραφεί προς την αντίθετη κατεύθυνση. Ο ρόλος της οπίσθιας προπέλας του ελικοπτέρου είναι να αντιδράσει στην αντίδραση ροπής που προκαλείται από την κύρια προπέλα. Η οπίσθια προπέλα του ελικοπτέρου συνήθως είναι σχεδιασμένη να κατευθύνει τον αέρα οριζόντιως. Όταν η δύναμη από την οπίσθια προπέλα είναι ακριβώς ίση με τη ροπή που παράγεται από την κύρια προπέλα, οι δύο ροπές εξουδετερώνονται και τότε το ελικόπτερο μπορεί να επιτύχει σταθερή, ευθύγραμμη και ασφαλή πτήση. Σε ένα τυπικό ελικόπτερο χωρίς οπίσθια προπέλα θα περιστρεφόταν η άτρακτος του ανεξέλεγκτα με φορά αντίθετη αυτής με την οποία στριφογυρίζει η βασική προπέλα. Η παρακάτω Εικόνα 2-12 προσφέρει μια πιο διαισθητική απεικόνιση του ζητήματος.

Όπως φαίνεται στο διάγραμμα, ο κύριος στροφέας περιστρέφεται με φορά αντίθετη αυτής των δεικτών του ρολογιού, άρα η άτρακτος θα προσπαθήσει να περιστραφεί με τη φορά των δεικτών του ρολογιού. Αυτή η φορά απεικονίζεται με το μεγάλο μαύρο βέλος στο διάγραμμα. Άρα το στροφείο της ουράς θα πρέπει να παράγει προωθητική δύναμη που θα τείνει να στρέψει τον σκελετό στην αντίθετη κατεύθυνση, η οποία απεικονίζεται με το άσπρο βέλος. Όταν η ροπή με τη δύναμη αποκτήσουν το ίδιο μέτρο οι επιδράσεις που έχουν στο σκελετό θα εξισωθούν προσφέροντας σταθερή πτήση.



Εικόνα 2-12 Κατεύθυνση ροπής της κύριας προπέλας και κατεύθυνση της πρωθητικής δύναμης της οπίσθιας προπέλας αντιτιθέμενη στην αντίδραση ροπής που προκαλείται[12]

Υπάρχουν κάποια ελικόπτερα που αντί για οπίσθια προπέλα, για να αντισταθούν στην αντίδραση ροπής χρησιμοποιούν δύο βασικές προπέλες, δεξιά και αριστερά από τον επίμηκες σκελετό, που περιστρέφονται με αντίθετες φορές. Έτσι λύνουν το πρόβλημα της αντίδρασης ροπής με διαφορετικό τρόπο.

Το ζήτημα μπορεί σχετικά εύκολα να εξηγηθεί και μαθηματικά. Γνωρίζουμε ότι η ροπή δίνεται από τον ακόλουθο τύπο:

$$\vec{T} = \vec{r} \times \vec{F} \quad (1)$$

Οπου r η ακτίνα περιστροφής, F η δύναμη και T η προκύπτουσα αντιροπή.

Χρησιμοποιώντας το νόμο του δεξιού χεριού για το εξωτερικό γινόμενο και λαμβάνοντας ως δύναμη F αυτή την οποία ασκεί η κύρια προπέλα στον αέρα (προς τα κάτω), ο μέσος, δηλαδή η ροπή θα δείχνει προς την κατεύθυνση υποδεικνυόμενη από το μεγάλο μαύρο βέλος του άνω σχήματος. Συνεπώς για να αντισταθμιστεί αυτή η ροπή, η προπέλα της ουράς του ελικοπτέρου θα πρέπει να ασκήσει μια πρωθητική δύναμη περίπου ίσου μέτρου και αντίθετης φοράς με τη ροπή αυτή, όπως υποδεικνύεται από το άσπρο βέλος του σχήματος, για να εξισορροπήσει το σύστημα.

3 Μη Επανδρωμένα Αεροσκάφη

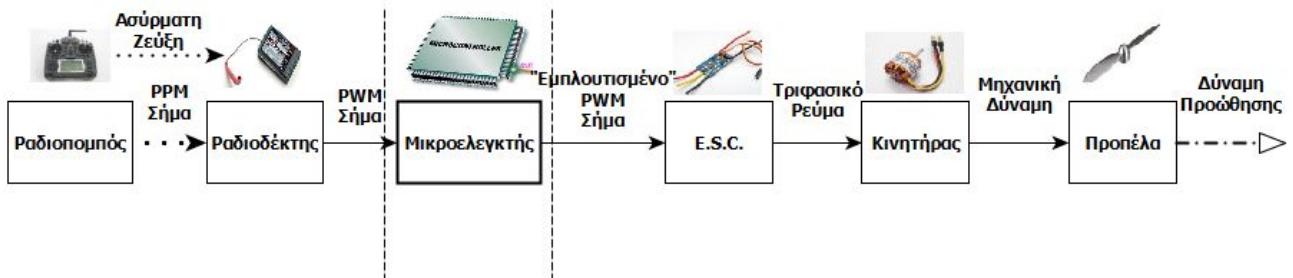
Σε αυτό το κεφάλαιο θα μιλήσουμε για τα μη επανδρωμένα εναέρια οχήματα. Κάθε εδάφιο του κεφαλαίου θα αφιερώνεται σε μια συγκεκριμένη υπομονάδα ενός UAV, θα περιγράφεται η λειτουργία της, θα αναλύονται τα χαρακτηριστικά της και ο ρόλος της στο τελικό αεροσκάφος και σε κάθε περίπτωση θα λαμβάνουμε την επιλογή για το δικό μας τετρακόπτερο ως αναφορά. Σε μια προκαταρκτική ενότητα θα περιγραφεί το στάδιο της σχεδίασης που ακολουθήσαμε για να επιλέξουμε τα κατάλληλα υλικά για το τετρακόπτερο. Θα γίνει μια εμπεριστατωμένη ανάλυση όλων των εξισώσεων που λάβαμε υπόψη.

Ιδιαίτερα τα τελευταία χρόνια τα drones έχουν γίνει πολύ δημοφιλή και συχνά γίνονται αντικείμενο έρευνας για στρατιωτικές, κοινωνικές, περιβαλλοντολογικές εφαρμογές, αλλά πλέον διανέμονται προς πώληση ακόμη και ως εμπορικά παιχνίδια. Τα UAV για αποτελεσματική, ασφαλή πτήση απαιτούν τέσσερις βαθμούς ελευθερίας, ο κάθε ένας για να ελέγχει μία από τις βασικές λειτουργίες πλοϊγησης του αεροσκάφους. Αυτές είναι, όπως ήδη αναφέραμε, οι pitch, roll, yaw και throttle. Διάφορες μορφές μη επανδρωμένων αεροσκαφών σε πολυποίκιλα μεγέθη και σχήματα έχουν κατακλύσει τις αγορές. Εκτός από quadcopters υπάρχουν εξακόπτερα (hexacopter), οκτακόπτερα (octocopter), τρικόπτερα (tricopter) κτλ. Δεδομένου ότι τουλάχιστον τέσσερις βαθμοί ελευθερίας είναι αναγκαίοι για αξιόπιστη πτήση, τα εξακόπτερα, οκτακόπτερα κτλ. αποτελούν πλεονασμό για τις ανάγκες και δυνατότητες της παρούσας εργασίας. Παρ' όλα αυτά τα τελευταία είναι ελκυστικά για πλήθος εφαρμογών, κυρίως μεγαλύτερης ισχύος και για δυνατότητες διαμετακόμισης φορτίων. Από την άλλη μεριά τα τρικόπτερα έχουν αποδειχτεί αρκετά ευσταθή, όμως στερούνται υψηλών επιδόσεων. Τα quadcopters λοιπόν έχουν μάλλον κυριαρχήσει για τις περισσότερες εφαρμογές διότι αποτελούν τη χρυσή τομή από άποψη οικονομίας, ενεργειακής απόδοσης, αυτονομίας και επιδόσεων.

3.1 Χαρακτηριστικά Τετρακόπτερου

Σε αυτή την ενότητα εισάγουμε μια περιεκτική σύνοψη των χαρακτηριστικών των μη επανδρωμένων αεροσκαφών με στροφείς (Multi-Rotors), και συγκεκριμένα ενός τετρακόπτερου. Μια γενική ανασκόπηση είναι χρήσιμη για να κατανοήσουμε τις ομοιότητες και τις διαφορές τους με τα κλασσικά επανδρωμένα αεροσκάφη. Τα v-κόπτερα αποτελούνται από v-κινητήρες και v-προπέλες. Δεν απαιτούν περίπλοκα μηχανική μέρη για τον έλεγχο της πτήσης. Τα ελέγχουμε χρησιμοποιώντας πρωτίστως ένα ηλεκτρονικό υπολογιστικό σύστημα επεξεργασίας μαζί μ' ένα σύστημα ασύρματου ραδιοτηλεχειρισμού, και προαιρετικά μια ζεύξη ασύρματης τηλεμετρίας. Έπειτα υπάρχει πληθώρα αισθητήρων για βελτίωση της πτήσης.

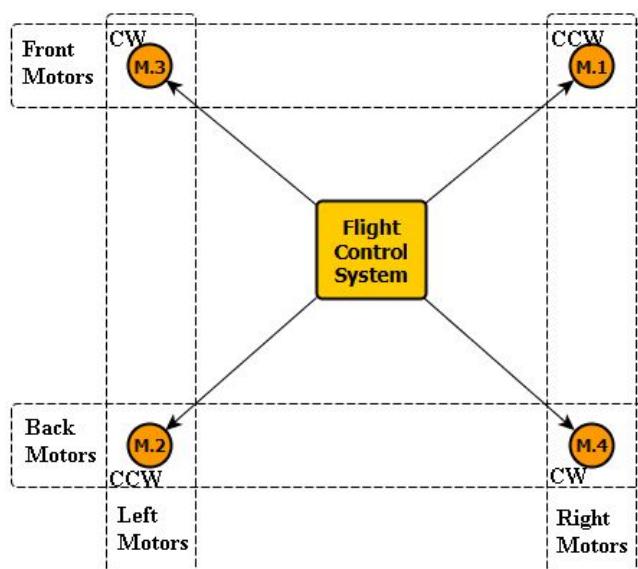
Από τον ραδιοπομπό που χρησιμοποιούμε εκπέμπουμε ραδιοκύματα ψηφιακά διαμορφωμένα κατά θέση παλμού (Pulse Position Modulation – PPM), στη συσκευή του ραδιοδέκτη τηλεχειρισμού στο όχημα. Ο δέκτης αποδιαμορφώνει και αποκωδικοποιεί το σήμα και τελικά καθοδηγεί PWM ηλεκτρικούς παλμούς στο κύριο υπολογιστικό σύστημα – τον ελεγκτή πτήσης –, που συνήθως είναι ένας μικροελεγκτής. Ο ελεγκτής πτήσης τελικά στέλνει τους παλμούς, διορθωμένους / βελτιστοποιημένους με τη συγχώνευση τιμών από διάφορους αισθητήρες, στους κινητήρες. Δηλαδή στα UAV μεταβάλλουμε τα RPM των κινητήρων, ανάλογα με τον κύκλο καθήκοντος (Duty Cycle) του PWM σήματος που φτάνει σε αυτούς, για να ελέγχουμε τις δυνάμεις που εφαρμόζονται στο αεροσκάφος ώστε να επιτύχουμε την πορεία και τον προσανατολισμό της αρεσκείας μας. Δεν υπάρχουν συγκεκριμένα πτερύγια και τμήματα –Elevator, Aileron, Rudder– για να εφαρμοστούν οι λειτουργίες ελέγχου πορείας Pitch, Roll, Yaw και Throttle, όπως συμβαίνει στα fixed-wing τύπου αεροσκάφη. Περισσότερη ανάλυση θα γίνει σε κάθε επιμέρους ενότητα αργότερα. Ωστόσο το παρακάτω μπλόκ διάγραμμα απεικονίζει τη διαδικασία.



Εικόνα 3-1 Μπλόκ Διάγραμμα διαδικασίας περιστροφής στροφέων

3.2 Τρόποι Πτήσης Τετρακόπτερου

Στη παρούσα ενότητα θα εξηγήσουμε πώς υλοποιούνται σε ένα quadcopter οι λειτουργίες ελέγχου πτήσης Pitch, Roll, Yaw & Throttle. Θα εξηγήσουμε τι λειτουργία ακριβώς εκτελεί το σύστημα ελέγχου πτήσης (Flight Control System – FCS) όταν λαμβάνει οδηγίες, μέσω του ραδιοδέκτη, για τον προσδιορισμό της κατεύθυνσης του οχήματος.



Εικόνα 3-2 Μποκ διάγραμμα ενός τετρακόπτερου που παριστάνει το Σύστημα Ελέγχου Πτήσης και τους κινητήρες

3.2.1 Σταθερή Πτήση (Hovering)

Το FCS διατηρεί τα RPM και των τεσσάρων κινητήρων στο ίδιο επίπεδο. Το RPM πρέπει να είναι επαρκές για να παρέχει αρκετή δύναμη ανύψωσης για την πτήση του αεροσκάφους. Η ροπή βρίσκεται σε ισορροπία, οπότε το quadcopter δεν περιστρέφεται γύρω από τον άξονα του (yaw). Για σωστό hovering το FCS πρέπει να λαμβάνει ακριβείς τιμές από τους αισθητήρες που το εφοδιάζουν με τιμές για την τωρινή του θέση, δηλαδή γυροσκοπικά όργανα και πυξίδα. Μια μονάδα GPS μπορεί να παίξει επικουρικό ρόλο στη διαδικασία.

3.2.2 Κατακόρυφη Αναρρίχηση και Κάθοδος

Εννοούμε τη λειτουργία throttle η οποία ελέγχει τη στάθμη ισχύος που οδηγείται στους τέσσερις κινητήρες. Για αύξηση του υψομέτρου (throttle up) πρέπει να αυξηθεί τα RPM και στους τέσσερις κινητήρες ταυτόχρονα. Για μείωση του υψομέτρου (throttle down) μειώνουμε RPM και στους τέσσερις κινητήρες ταυτόχρονα.

3.2.3 Πτήση προς τα Εμπρός, ή προς τα Πίσω

Πρόκειται για τη λειτουργία ελέγχου πορείας Pitch. Για πτήση προς τα εμπρός (pitch forward) αυξάνουμε τα RPM στους οπίσθιους κινητήρες 2 και 4 (βλέπετε Εικόνα 3-2) και μειώνουμε κατάλληλα τα RPM στους μπροστινούς κινητήρες 1 και 3. Για πτήση προς τα πίσω (pitch backwards) αυξάνουμε τα RPM στους μπροστινούς κινητήρες και μειώνουμε RPM στους οπίσθιους κινητήρες.

3.2.4 Στροφή προς τα Δεξιά, ή προς τα Αριστερά

Με πτήση δεξιά ή αριστερά αναφερόμαστε στη λειτουργία ελέγχου πτήσης Roll. Για στροφή προς τα δεξιά (roll right) αυξάνουμε τα RPM στους αριστερούς κινητήρες (2 και 3) και μειώνουμε RPM στους δεξιούς κινητήρες (1 και 4), πάντα αντισταθμίζοντας κατάλληλα για ευσταθή πτήση. Για πτήση προς τα αριστερά (roll left) αυξάνουμε RPM στους δεξιούς κινητήρες (1 και 4) και μειώνουμε RPM στους αριστερούς κινητήρες (3 και 2).

3.2.5 Περιστροφή Οχήματος

Μιλάμε για τη λειτουργία ελέγχου πτήσης yaw. Δηλαδή την περιστροφή του αεροσκάφους, ως προς άξονα που διέρχεται κατακόρυφα από το κέντρο βάρους του, είτε με τη φορά τον δεικτών του ρολογιού (CW), είτε αντίθετα (CCW). Αυξάνοντας, ή μειώνοντας RPM στα ζευγάρια κινητήρων που βρίσκονται στην ίδια διαγώνιο ταυτόχρονα, αυξάνουμε, ή μειώνουμε τη ροπή που παράγεται από αυτό το ζεύγος κινητήρων προς τη φορά περιστροφής τους. Δηλαδή το αποτέλεσμα αυτής της λειτουργίας εξαρτάται από τη φορά περιστροφής των κινητήρων στην ίδια διαγώνιο. Με αναφορά λοιπόν στην εικόνα παραπάνω, που απεικονίζει τη σύνθεση του δικού μας quadcopter, για περιστροφή με τη φορά των δεικτών του ρολογιού (yaw clockwise) αυξάνουμε RPM στους κινητήρες 3 και 4 (οι οποίοι περιστρέφονται CW) και/ή μειώνουμε RPM στους κινητήρες 1 και 2 (οι οποίοι περιστρέφονται CCW). Για περιστροφή με φορά αντίθετη των δεικτών του ρολογιού (yaw counterclockwise) αυξάνουμε RPM στους κινητήρες 1 και 2 και/ή μειώνουμε RPM στους κινητήρες 3 και 4.

3.3 Εξισώσεις των Δυνάμεων που εφαρμόζονται σε Αεροσκάφη

Σε αυτό το εδάφιο αναλύουμε τις εξισώσεις των δυνάμεων που εφαρμόζονται σε ένα αεροσκάφος και στο επόμενο θα τις χρησιμοποιήσουμε σε συνδυασμό με άλλες, για να επιλέξουμε τα κατάλληλα εξαρτήματα που θα αποτελέσουν το δικό μας τετρακόπτερο.

3.3.1 Δυνάμεις Ανύψωσης και Απώθησης

Μετρήσεις έχουν δείξει ότι δοθέντος οποιασδήποτε αεροτομής οι δυνάμεις ανύψωσης και απώθησης είναι ευθέως ανάλογες 1) με το εμβαδόν της αεροτομής, 2) με το τετράγωνο της ταχύτητας του φαινόμενου ανέμου και 3) με την πυκνότητα του αέρα. Θα φέρουμε ένα παράδειγμα για κάθε περίπτωση ξεχωριστά ώστε να γίνει καλύτερα κατανοητό. 1) Εάν το εμβαδόν διπλασιαστεί, οι δυνάμεις ανύψωσης και απώθησης επίσης διπλασιάζονται. 2) Εάν η ταχύτητα του φαινόμενου ανέμου διπλασιαστεί, οι δυνάμεις ανύψωσης και απώθησης τετραπλασιάζονται. 3) Εάν η πυκνότητα του αέρα διπλασιαστεί οι τιμές των δυνάμεων ανύψωσης και απώθησης επίσης διπλασιάζονται.

Οι δυνάμεις ανύψωσης και απώθησης χαρακτηρίζονται και από έναν συντελεστή που ονομάζεται συντελεστής ανύψωσης (lift coefficient) και απώθησης (drag coefficient) αντίστοιχα και εξαρτώνται από τη γωνία επίθεσης και από τη διατομή της αεροτομής. Οι δυνάμεις δίνονται με τους παρακάτω τύπους.

$$Lift = c_L \cdot p \cdot A \cdot V^2 \quad (2)$$

$$Drag = c_D \cdot p \cdot A \cdot V^2 \quad (3)$$

Όπου, c_L και c_D οι συντελεστές ανύψωσης και απώθησης αντίστοιχα, p η πυκνότητα του αέρα, A το εμβαδόν της αεροτομής (ή των πτερύγων) και V η ταχύτητα του φαινόμενου ανέμου.

Για λόγους πληρότητας θα πρέπει να αναφέρουμε ότι οι τιμές αυτών των δυνάμεων δίνονται και από εναλλακτικούς τύπους που εμπλέκουν την αληθής ταχύτητα (True Airspeed - TAS), αντί της ταχύτητας του φαινόμενου ανέμου. Η αληθής ταχύτητα είναι, όπως υπονοεί και το όνομα, η πραγματική ταχύτητα του ιπτάμενου αεροσκάφους κατά τη πτήση. Διαφέρει από την μετρούμενη ταχύτητα του αεροσκάφους (Indicated Airspeed - IAS) – η ταχύτητα όπως μετράται από το ταχύμετρο του – στο ότι η τελευταία μετράει τη διαφορά πίεσης, ή την πίεση του αέρα που ρέει πάνω από τα φτερά και μετατρέπεται ύστερα σε μονάδες ταχύτητας. Η δυναμική αυτή πίεση όμως διαφέρει με το υψόμετρο. Η πυκνότητα του αέρα μειώνεται με την αύξηση του υψομέτρου, οπότε η πίεση του θα μειώνεται με το υψόμετρο. Από αυτό συνεπάγεται ότι η υποδεικνύμενη ταχύτητα θα μειώνεται καθώς το υψόμετρο αυξάνεται, ενώ η αληθής ταχύτητα θα παραμένει σταθερή. Γενικά, η IAS είναι πάντα μικρότερη από την TAS.

Με αυτή την εισαγωγή στις έννοιες των ταχυτήτων των αεροσκαφών μπορούμε να δώσουμε τους νέους τύπους για τις δυνάμεις ανύψωσης και απώθησης ως εξής.

$$Lift = \frac{1}{2} c_L \cdot p \cdot A \cdot U^2 \quad (4)$$

$$Drag = \frac{1}{2} c_D \cdot p \cdot A \cdot U^2 \quad (5)$$

όπου εδώ U είναι η αληθής ταχύτητα – TAS.

Οι τύποι (2) και (3) είναι αυτοί που προτείνονται στον υπολογισμό των δυνάμεων ανύψωσης και απώθησης, αφού ο ακριβής υπολογισμός της TAS είναι πολύπλοκος.

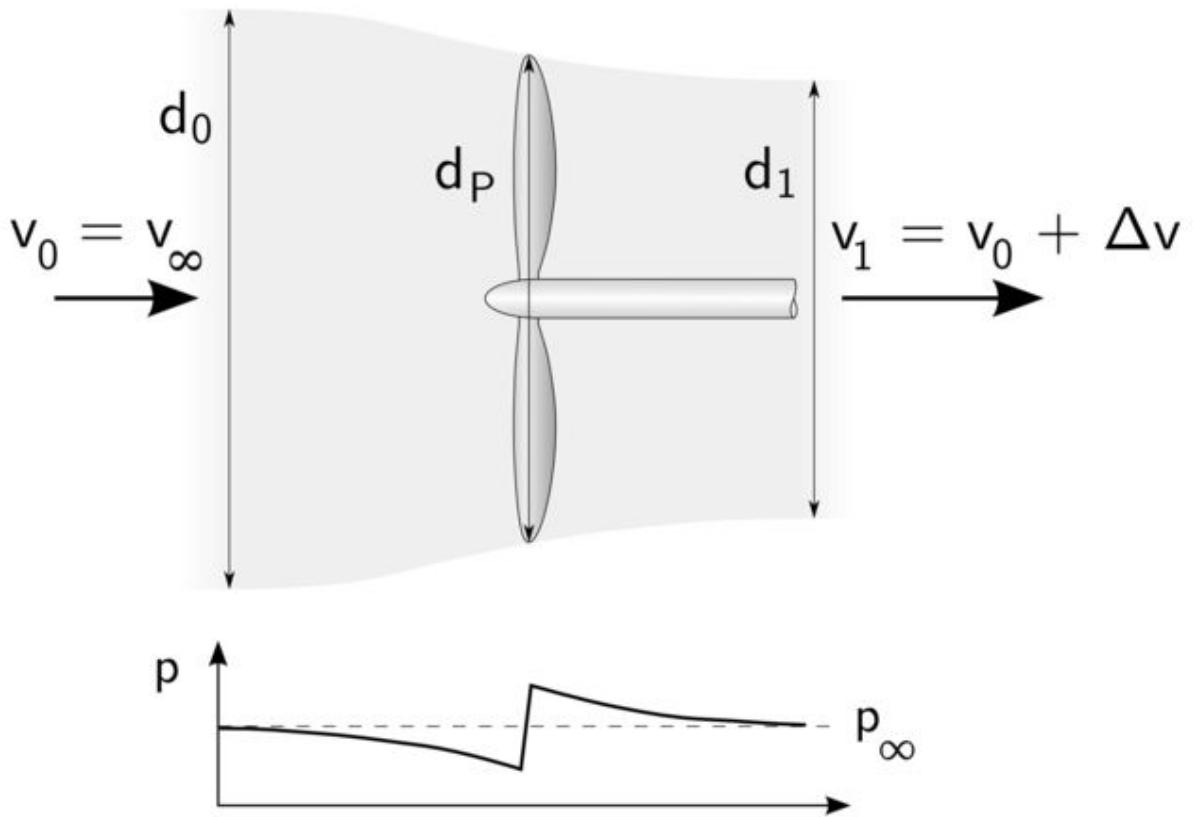
3.3.2 Δύναμη Προώθησης

Θα υπολογίσουμε τη δύναμη στατικής προώθησης, δηλαδή τη δύναμη που παράγει η προπέλα όταν είναι ακίνητη στο έδαφος. Η δύναμη δυναμικής προώθησης είναι μάλλον αδύνατο να προσδιοριστεί από έναν και μόνο γενικό τύπο καθώς είναι συνάρτηση πολλών και δυναμικών παραγόντων.

Μια προπέλα διαμέτρου d_p επιταχύνει αέρα πυκνότητας ρ ο οποίος ρέει διαμέσου του δίσκου που σχηματίζει. Η διαδικασία απεικονίζεται στην Εικόνα 3-3.

[13]Η ταχύτητα του αέρα που εισέρχεται μπροστά από τη προπέλα είναι $V_0 = V_\infty$. Η ταχύτητα του αέρα στην έξοδο επιταχύνθηκε από την προπέλα, άρα αυξήθηκε κατά Δv . Η ταχύτητα του αέρα που εξέρχεται πίσω από την προπέλα θα είναι $V_1 = V_0 + \Delta v$. Η προπέλα προκαλεί μια διαφορά πίεσης η οποία απορροφά τον αέρα μπροστά της και τον σπρώχνει από πίσω της. Εφόσον η ροή της μάζας πρέπει να είναι ίση και μπροστά και πίσω από την προπέλα, ο «νοητός» σωλήνας αέρα που σχηματίζει η προπέλα πρέπει να είναι μεγαλύτερος από μπροστά και μικρότερος προς τα πίσω της, για την εξισορρόπηση της πίεσης στα δύο μέρη. Απλοποιώντας κάπως τις εξισώσεις, θεωρώντας ότι η ταχύτητα του αέρα είναι πανομοιότυπη κατά μήκος της διατομής του δίσκου της προπέλας, χωρίς όμως να χάνουμε σημαντική πληροφορία έχουμε ότι η ροή της μάζας είναι η εξής:

$$\frac{dm}{dt} = \pi \rho \frac{d_p^2}{4} (V_\infty + \frac{\Delta v}{2}) \quad (6)$$



Εικόνα 3-3 Δυνάμεις προώθησης που παράγονται από μια προπέλα διαμέτρου d_p [13]

Η δύναμη προώθησης ισούται με τη ροή της μάζας επί την μεταβολή της ταχύτητας σύμφωνα με το δεύτερο νόμο του Νεύτωνα:

$$T = \pi \rho \frac{d_p^2}{4} \left(V_\infty + \frac{\Delta v}{2} \right) \cdot \Delta v \quad (7)$$

Επιλύοντας αυτή την εξίσωση ως προς την επιτάχυνση του αέρα Δv , με όρους στατικής προώθησης, δηλαδή $T = T_0$ και $V_0 = 0$ (εφόσον το αεροσκάφος είναι ακίνητο στο έδαφος, με τη μηχανή σε λειτουργία, για λεπτομέρειες δείτε ενότητα για Static Thrust) έχουμε ότι:

$$\Delta v = 2 \sqrt{\frac{2T_0}{\pi \rho d_p^2}} \quad (8)$$

Η δύναμη προώθησης είναι η ενεργός ισχύς διαιρεμένη με την ταχύτητα του αέρα στο δίσκο της προπέλας (εφόσον ισχύει ότι:

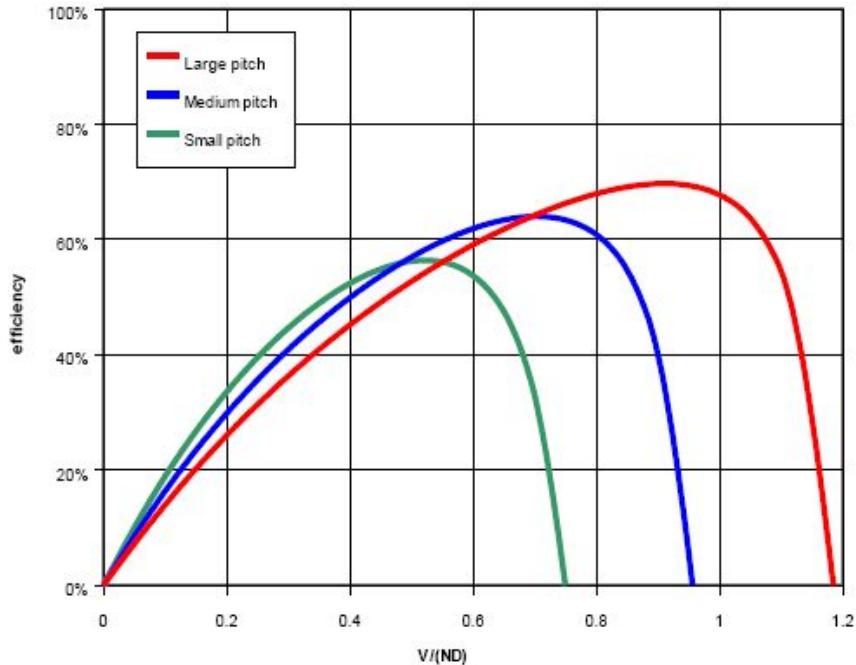
$$P = \vec{F} \cdot \vec{V} = FV \cos(\hat{\theta}) \quad (9)$$

\vec{F} : δύναμη, \vec{V} : ταχύτητα, $\hat{\theta}$ η γωνία που σχηματίζεται μεταξύ των δύο διανυσμάτων. Μπορούμε να θεωρήσουμε ότι η ταχύτητα του αεροσκάφους και η προωθητική δύναμη έχουν την ίδια κατεύθυνση, άρα $\hat{\theta} = 0^\circ$). Εάν η μηχανή του αεροσκάφους παράγει συνολική ισχύ $P_{o\lambda}$, τότε η ωφέλιμη ισχύς P που δέχεται η προπέλα και παράγει μηχανικό έργο, θα πρέπει να πολλαπλασιασθεί με την αποδοτικότητα της προπέλας n_p και με την ηλεκτρική αποδοτικότητα n_e . Δηλαδή:

$$P = P_{o\lambda} n_p n_e \quad (10)$$

Η ηλεκτρική αποδοτικότητα είναι ένας παράγοντας βελτίωσης που προσδιορίζει το ποσοστό της συνολικής ισχύος που απορροφά η προπέλα από τον ηλεκτρικό κινητήρα για να τη μετατρέψει σε προωθητική δύναμη. Η αποδοτικότητα της προπέλας αφορά την ποιότητα των υλικών που τη δομούν

και είναι συνάρτηση αρκετών παραγόντων. Η τιμή της συνήθως δίνεται από τους κατασκευαστές. Εάν δε δίνεται θα πρέπει να προσεγγιστεί σύμφωνα με το παρακάτω γράφημα.



Εικόνα 3-4 Σχέση αποδοτικότητας προπέλας μικρού, μεσαίου και μεγάλου pitch με το λόγο προπόρευσης(Advance Ratio). Ο λόγος προπόρευσης στα UAV λαμβάνει υψηλές τιμές[14]

Αντικαθιστώντας τη δύναμη προώθησης και την ταχύτητα στην (9) προκύπτει ότι:

$$T = \frac{P}{V_\infty + \frac{\Delta v}{2}} \Leftrightarrow T = \frac{P_{o\lambda} n_p n_e}{V_\infty + \frac{\Delta v}{2}} \quad (11)$$

Από την εξίσωση αυτή βλέπουμε ότι η δύναμη προώθησης αυξάνεται όταν το Δv μειώνεται. Μια μεγαλύτερη προπέλα συλλαμβάνει περισσότερο αέρα και περιστρέφεται πιο αργά από μια μικρότερη προπέλα. Από τη προηγούμενη πρόταση λαμβάνονται δύο συμπεράσματα. Όσο πιο αργά περιστρέφεται η προπέλα, τόσο μικρότερη η επιτάχυνση του αέρα πίσω από τη προπέλα (Δv μειώνεται). Επίσης εφόσον η μεγαλύτερη προπέλα αιχμαλωτίζει περισσότερο αέρα, θα χρειαστεί μικρότερο Δv για να παράξει την ίδια δύναμη προώθησης. Επομένως μια μεγάλη, αργά περιστρεφόμενη προπέλα είναι αποδοτικότερη από μια μικρή και γοργά περιστρεφόμενη.

Τελικώς προκύπτει η δύναμη στατικής προώθησης από τις (11) και (8):

$$T_0 = \frac{P_{o\lambda} n_p n_e}{2 \sqrt{\frac{2T_0}{\pi \rho d_p^2}}} \Leftrightarrow T_0 = 0.794 \cdot \sqrt[3]{\pi \cdot \rho \cdot d_p^2 \cdot P_{o\lambda}^2 \cdot n_e^2 \cdot n_p^2} \quad (12)$$

Οπου:

$P_{o\lambda}$: Ισχύς μηχανής (W),

$\rho = 1.18 \left(\frac{kg}{m^3}\right)$: πυκνότητα αέρα @ 25°C,

n_e : αποδοτικότητα μηχανής,

n_p : αποδοτικότητα προπέλας και

d_p : διάμετρος προπέλας (m).

3.3.3 Βάρος

Η δύναμη του βάρους υπολογίζεται βεβαίως από το δεύτερο νόμο του Νεύτωνα:

$$W = m \cdot g \quad (13)$$

Οπου:

m : μάζα του οχήματος (kg) και

$g = 9.81(\frac{m}{s^2})$: επιτάχυνση της βαρύτητας.

3.4 Σχεδίαση για Εκπλήρωση των Προδιαγραφών

Πριν επιλέξουμε τα διάφορα υποσυστήματα που απαρτίζουν ένα quadcopter, έπρεπε να καταλήξουμε σε αποφάσεις για τα εξής:

- Το μέγεθος του.
- Το χρόνο πτήσης του.
- Τις δυνατότητες που επιθυμούμε να πληρεί.

Κάθε κατηγορία θέτει πολλούς περιορισμούς σχετικά με το τι μπορούμε να πετύχουμε. Το μέγεθος του Quadcopter θέτει περιορισμούς για το βάρος του. Όσο μεγαλύτερη είναι η μάζα ενός οχήματος τόση περισσότερη ενέργεια θα χρειαστεί για να πετάξει, άρα θα είναι και πιο δαπανηρό.

Ο χρόνος πτήσης του αεροσκάφους καθορίζει τη διάρκεια ζωής της μπαταρίας και η επιλογή της μπαταρίας είναι κρίσιμο συστατικό για την όλη κατασκευή. Οι προδιαγραφές ισχύος της μπαταρίας πρέπει να υποστηρίζονται από όλα τα άλλα εξαρτήματα που τροφοδοτούνται από αυτή. Τέτοια είναι οι κινητήρες, οι ηλεκτρονικοί ελεγκτές ταχύτητας και οι ελεγκτές πτήσης και τηλεμετρίας. Επίσης μια μπαταρία μεγάλης χωρητικότητας είναι αρκετά πιο βαριά και η μπαταρία είναι συνήθως το πιο βαρύ συστατικό στοιχείο των μη επανδρωμένων αεροσκαφών.

Οι δυνατότητες ενός μη επανδρωμένου αεροσκάφους εξαρτώνται από την εργασία για την οποία προορίζονται. Δηλαδή, ένα quadcopter θα μπορούσε να έχει δυνατότητες όπως αυτόματος πιλότος (απαιτεί GPS), αντισφαλματικά συστήματα (Failsafe) σε περίπτωση εξάντλησης μπαταρίας, ή κάποιας άλλης έκτακτης ανάγκης, δυνατότητες ανύψωσης φορτίων, δυνατότητες παρακολούθησης σε πραγματικό χρόνο και πολλές άλλες. Εμείς στα πλαίσια αυτής της πτυχιακής εργασίας θέλουμε ένα quadcopter μικρού μεγέθους και μάζας (<0.65kg), με χρόνο πτήσης τουλάχιστον 10 λεπτά σε κανονικές συνθήκες λειτουργίας με δυνατότητες GPS, τηλεμετρίας, αυτόνομης ευέλικτης πτήσης, αλλά και ικανότητα ανύψωσης φορτίου έως και 100gr αν χρειαστεί. Ακολουθούν κατά σειρά όλες οι εξισώσεις που λάβαμε υπόψη για να επιλέξουμε τα εξαρτήματα του τετρακόπτερου. Υστερα από έρευνα τελικά καταλήξαμε στην επιλογή υλικών με τις ακόλουθες προδιαγραφές:

❖ Προδιαγραφές Προπέλας

Διάμετρος: 5 ίντσες $\rightarrow d_p = 0.127\text{m}$

Pitch: 3 ίντσες $\rightarrow p_p = 0.0762\text{m}$

Σύμφωνα με την Εικόνα 3-4 επιλέγουμε την αποδοτικότητα προπέλας σε $n_p = 0.6$.

❖ Προδιαγραφές Μπαταρίας

Ονομαστική τιμή Τάσης: $V = 11.1\text{ Volt}$

Ρυθμός εκφόρτισης: $C = 20$

Χωρητικότητα: $capacity = 2700 \text{ mAh}$

❖ Προδιαγραφές Κινητήρα

Σταθερά ταχύτητας κινητήρα (Motor Velocity Constant): $K_v = 2200$

Μέγιστο απορροφούμενο ρεύμα: $I_{Mm} = 6 \text{ A}$

Η ηλεκτρική αποδοτικότητα του brushless κινητήρα είναι πολύ υψηλή: $n_e = 0.95$

❖ Απορρόφηση Ρεύματος ελεγκτή πτήσης και περιφερειακών συσκευών

Καταλισκόμενο ρεύμα F.C.APM 2.8, ραδιοδέκτη Turnigy, PM, GPS+Mag, CC3200: $I_e = 0.45 \text{ A}$

Υπολογισμοί

Η Μπαταρία LiPo που χρησιμοποιούμε συντίθενται από τρία κύτταρα (cells) των 3.7 V εν σειρά. Έχει το χαρακτηριστικό 3S, το οποίο πρέπει να υποστηρίζεται τους ηλεκτρονικούς ελεγκτές ταχύτητας. Επίσης πρέπει να βεβαιωθούμε ότι η μπαταρία μπορεί να προσφέρει αρκετό ρεύμα ώστε να τροφοδοτήσει επαρκώς τους 4 κινητήρες και τα υπόλοιπα ηλεκτρονικά συστήματα επι του σκάφους ταυτόχρονα. Ο ελεγκτής πτήσης λαμβάνει ισχύ 5.37V/2.25A από το τροφοδοτικό του APM Power Module, το οποίο συνδέεται με την μπαταρία και τροφοδοτεί όλες τις περιφερειακές συσκευές, δηλαδή τον ραδιοδέκτη, την πλατφόρμα τηλεμετρίας CC3200 και τη βαθμίδα GPS + Πυξίδα. Μετρήσαμε ότι όλες οι περιφερειακές συσκευές και ο ελεγκτής πτήσης σε πλήρη φόρτο απορροφούν ~450mA. Το μέγιστο ρεύμα που μπορεί να αποδίδει η LiPo μπαταρία σε κανονικές συνθήκες λειτουργίας είναι:

$$\bullet I_{Bm} = capacity \cdot C = 2700 \cdot 20 = 54 \text{ A.}$$

Χρησιμοποιώντας τον εμπειρικό τύπο που λέει ότι το μέγιστο ρεύμα που μπορεί να αποδώσει η μπαταρία μας πρέπει να είναι μεγαλύτερο από το μέγιστο ρεύμα που μπορούν να καταναλώσουν οι κινητήρες, συν το μέγιστο ρεύμα που μπορούν να καταναλώσουν τα υπόλοιπα ηλεκτρονικά στο σκάφος, συν ένα περιθώριο ασφάλειας (έστω το περιθώριο ασφάλειας είναι 10A που είναι τυπική τιμή για quadcopters μικρής ισχύος), έχουμε ότι:

$$I_{Bm} \geq 4 \cdot I_{Mm} + I_e + 10 \Leftrightarrow 54 \geq 42.25 \text{ το οποίο αληθεύει.}$$

- Μέγιστη καταναλισκόμενη ισχύς κινητήρα: $P_m = I_{Mm} \cdot V = 66.6 \text{ W}$
- Χρησιμοποιούνται 4 κινητήρες επομένως η μέγιστη καταναλισκόμενη ισχύς τους είναι: $4 \cdot P_m = 266.4$
- Συνολική καταναλισκόμενη ισχύς λοιπών ηλεκτρονικών: $P_e = I_e \cdot 5 = 2.25 \text{ W}$
- Ολική καταναλισκόμενη ισχύς εξαρτημάτων μπαταρίας LiPo: $P_o = 4 \cdot P_m + P_e = 268.65 \text{ W}$
- Στροφές ανά λεπτό (RPM) που παράγονται από τον κινητήρα: $RPM = K_v \cdot V = 24420$
- Η ολική στατική προώθηση που παράγεται, σύμφωνα με τον τύπο (12), θα είναι:

$$T_o = 4 \cdot 0.794 \cdot \sqrt[3]{\pi \cdot \rho \cdot 0.127^2 \cdot 88.8^2 \cdot 0.95^2 \cdot 0.6^2} \cong 4 \cdot 3.50698 = 14.02792 \text{ N}$$

- Συνολική μάζα που μπορεί να ανυψωθεί σύμφωνα με το δεύτερο νόμο του Νεύτωνα: $F = m \cdot g$, θα είναι: $m_T = \frac{Thrust}{g} \cong 1.42996 = 1.43 \text{ kg.}$

Υπολογίζουμε προσεγγιστικά τη μάζα του κάθε εξαρτήματος που βρίσκεται πάνω στο όχημα και αθροίζουμε ώστε να προκύψει η μάζα του τετρακόπτερου:

4 ESC's = $4 \cdot 13 = 52 \text{ g}$, 4 κινητήρες = $4 \cdot 24 = 96 \text{ g}$, 4 προπέλες = $4 \cdot 4 = 16 \text{ g}$, Turnigy R/C Rx = 15 g , σκελετός = 110 g , PM = 25 g , μπαταρία = 198 g , APM ArduPilot + Βάση αντικραδασμικής

προστασίας = 30g, CC3200 = 30g, GPS + Mag. + Στύλος προστασίας = 35g, Πλακέτα διανομής = 7.6 g, διάφορα καλώδια & λουριά & μικροπράγματα + δείκτης ασφαλείας = 20g.

Συνολικό βάρος: $m_q \sim 634.6\text{g}$.

Χρησιμοποιώντας τον εμπειρικό, αλλά αρκετά ακριβή τύπο, που αναφέρει ότι η (στατική) προώθηση θα πρέπει να είναι τουλάχιστον 2.1 φορές μεγαλύτερη από τη μάζα του οχήματος:

$$m_T \geq 2.1 \cdot m_q \quad (14)$$

Έχουμε ότι $1.43 \geq 1.33266$ που ισχύει, επομένως εκπληρώνεται η προδιαγραφή μας για το βάρος.

Επίσης, ισχύει ότι αν η μάζα προώθησης είναι μεγαλύτερη ή ίση με 3 φορές τη μάζα του οχήματος, το όχημα είναι ιδιαίτερα ευέλικτο και ικανό για πολλές ακροβατικές μανούβρες και ανήκει σε μια θα λέγαμε «sport» κατηγορία. Εάν όμως η μάζα προώθησης είναι πάνω από 4 φορές τη μάζα του οχήματος, τότε το όχημα γίνεται ασταθές, δηλαδή δύσκολο για (χεροκίνητη) πτήση.

Μετά την τελική συναρμολόγηση του τετρακόπτερου διαπιστώσαμε ότι το βάρος ήταν τελικά 663g. (<5% διαφορά) πολύ κοντά στο ιδανικό! Γι' αυτό το λόγο αλλάξαμε μια παράμετρο του ελεγκτή πτήσης, που θα αναφερθεί αργότερα. Αφορά το επίπεδο βάρους που θεωρεί ως μέσο, και από 50% το θέσαμε 55%. Προς μεγάλη μας χαρά οι πτήσεις που επιτύχαμε ήταν μάλλον άριστες, όπως ακριβώς φαινόταν από τη σχεδίαση.

Η επόμενη κρίσιμη παράμετρος που έπρεπε να υπολογίσουμε είναι ο χρόνος πτήσης του τετρακόπτερου μας. Ο ελάχιστος χρόνος πτήσης ενός μη επανδρωμένου οχήματος δίνεται από τη σχέση:

$$t = n \cdot \frac{60}{1000} \cdot \frac{\text{capacity} \cdot V}{4 \cdot P_m + P_e} \quad (15)$$

Οπου:

capacity: χωρητικότητα μπαταρίας σε (mAh),

V: τάση λειτουργίας μπαταρίας σε (Volt),

P_m : Μέγιστη ισχύς κινητήρα σε (W) και

P_e : Ισχύς υπόλοιπων ηλεκτρονικών υποσυστημάτων στο σκάφος σε (W).

n: συντελεστής αποδοτικότητας.

Ο χρόνος t που παίρνουμε είναι σε λεπτά. Ο πολλαπλασιασμός με $\frac{60}{100}$ γίνεται για να μετατραπούν τα mAhours σε Aminutes. Σημειώνεται ότι η μονάδα μέτρησης mAh δεν υπονοεί mA/h μιλιαμπέρ ανά ώρα. Σε αντίθεση προς την κοινή πρακτική, το mAh υποδηλώνει mA · h. Επίσης ο συντελεστής αποδοτικότητας n συνυπολογίζει πολλούς παράγοντες, όπως: 1) Ένα μέρος του αέρα που εκτοπίζεται από τις προπέλες χτυπώντας το σκελετό συντελώντας στην αύξηση της δύναμης απώθησης του αεροσκάφους. Αυτό αναγκάζει τη μηχανή/μπαταρία να προσφέρει περισσότερο ρεύμα σε άτακτες χρονικές στιγμές για να αντισταθμίσει το φαινόμενο. Όσο χειρότερες είναι οι περιβαλλοντολογικές και οι καιρικές συνθήκες τόσο εντονότερη είναι η επίδραση του φαινομένου. Ούτως η άλλως όμως γενικώς δε συνιστάται η πτήση UAV με κακές καιρικές συνθήκες, 2) Οι μπαταρίες LiPo δεν πρέπει να εκφορτίζονται 100% διότι θα αρχίσουν να «διαρρέουν» (leakage), δηλαδή το φορτίο στη μπαταρία θα πρέπει να είναι μεγαλύτερο από μια τιμή κατωφλίου, που για τις μπαταρίες LiPo προτείνεται το 15%, για να αποφευχθούν πιθανές ζημιές στην μπαταρία, 3) η αποδοτικότητα προπέλας, 4) η αποδοτικότητα των κινητήρων κ.α. Γενικώς η τιμή n = 0.8 είναι πρακτικά μια καλή τιμή για ικανοποιητικές καιρικές συνθήκες (χρησιμοποιώντας βέβαια brushless

κινητήρες). Άρα σύμφωνα με την εξίσωση (15) ο ελάχιστος χρόνος πτήσης για το δικό μας τετρακόπτερο είναι:

$$t = 0.8 \cdot \frac{60}{1000} \cdot \frac{2700 \cdot 11.1}{4 \cdot 66.6 + 2.25} \cong 5.35477 \text{ min}$$

Η παραπάνω εξίσωση θα δώσει τη χειρότερη δυνατή περίπτωση για μέγιστη ισχύ κινητήρα. Αυτό όμως συμβαίνει μόνο κατά την εκκίνηση (ρεύμα εκκίνησης - inrush current) και χωρίς μια τροποποίηση δεν θα είναι αξιόπιστη. Στην πραγματικότητα η ισχύς του κινητήρα που απαιτείται για μια τυπική πτήση κυμαίνεται μεταξύ 25% ($0.25 \cdot P_m$) και 45% ($0.45 \cdot P_m$). Έστω λοιπόν $0.4 \cdot P_m$, οπότε:

$$t = 0.8 \cdot \frac{60}{1000} \cdot \frac{2700 \cdot 11.1}{4 \cdot 0.4 \cdot 66.6 + 2.25} \cong 13.221 \text{ min}$$

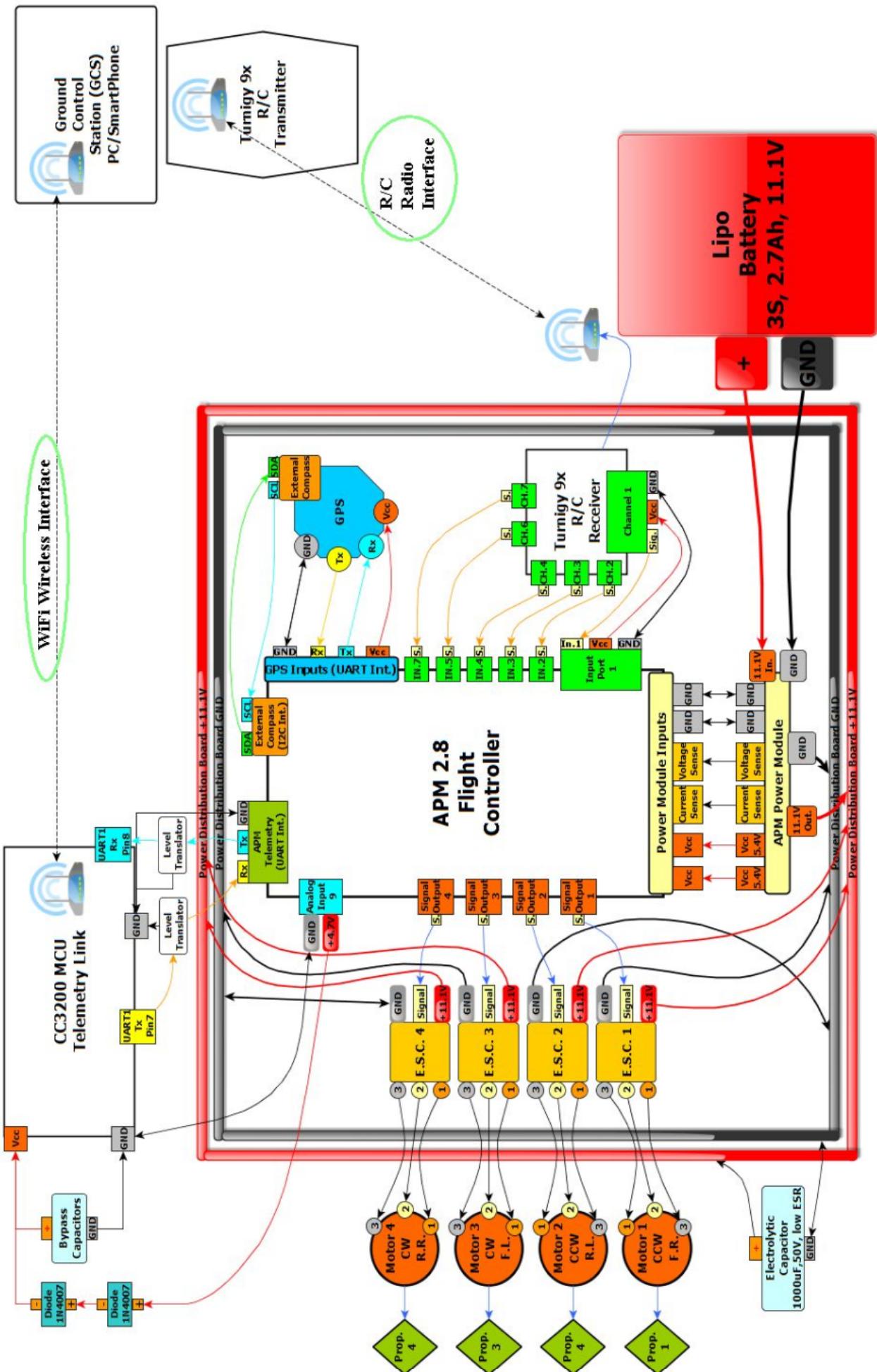
Μια ακόμα διόρθωση για τις μπαταρίες LiPo, ώστε να φροντίσουμε να μην ξοδέψουν πάνω από το 85% της χωρητικότητας τους. Οπότε έχουμε ότι ο χρόνος πτήσης θα είναι:

$$t = 0.8 \cdot \frac{60}{1000} \cdot \frac{0.85 \cdot 2700 \cdot 11.1}{4 \cdot 0.4 \cdot 66.6 + 2.25} \cong 11.238 \text{ min}$$

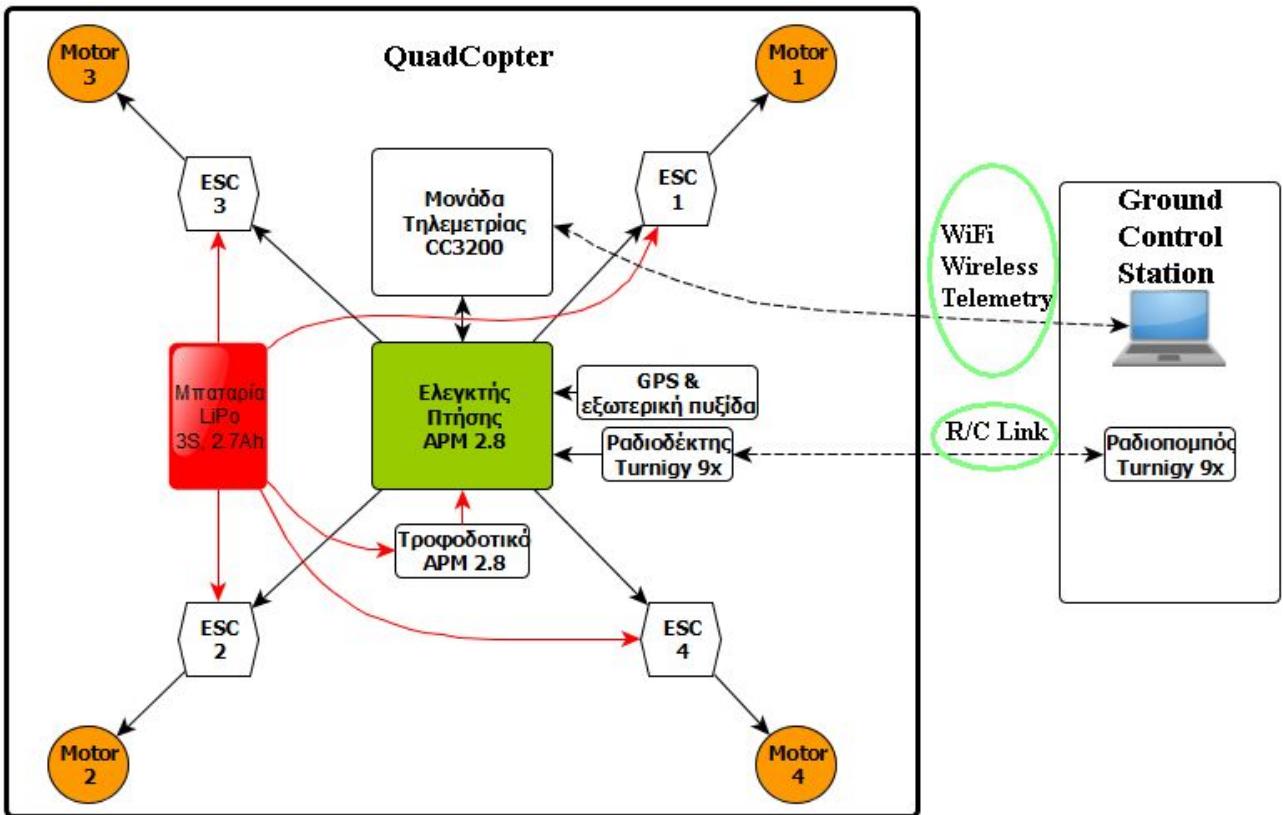
που εκπληρώνει τη προδιαγραφή μας για το χρόνο πτήσης. Έπειτα από κάποιες χρονομετρημένες πτήσεις διαπιστώσαμε ότι αυτός ο χρόνος αποτελεί σχεδόν ακριβώς το πραγματικό χρόνο πτήσης (~12min) του τετρακόπτερου, ακόμα και με ευέλικτη πτήση που ήταν ο αρχικός μας στόχος!

Τέλος, σαν να μη μην έφταναν αρκετές επιβεβαιώσεις, για τον υπολογισμό αυτής της πολύ σημαντικής παραμέτρου χρησιμοποιήσαμε και αυτή[15] την ιστοσελίδα για να βεβαιωθούμε για τους υπολογισμούς μας και επαληθευτήκαμε. Αυτή είναι η διαδικασία μετρήσεων για να βεβαιωθούμε για τις προδιαγραφές και να επιλέξουμε τελικά τα κατάλληλα υποσυστήματα.

Παρακάτω βλέπουμε το αναλυτικό block διάγραμμα ολόκληρου του τετρακόπτερου + Ground Control Station + R/C Ραδιοπομπός ύστερα από τη συναρμολόγηση, που περιγράφει κάθε σύνδεση και συστατικό. Αμέσως μετά ακολουθεί και ένα απλοποιημένο διάγραμμα block. Η κάθε υπομονάδα θα εξηγηθεί στις ενότητες που ακολουθούν. Αλλά αυτό το διάγραμμα θα χρησιμεύει ως βάση για γρήγορη αναφορά, αφού απεικονίζει με κάθε λεπτομέρεια όλες τις συνδέσεις που έγιναν. Όσες θύρες δεν προβάλλονται σημαίνει ότι δε χρησιμοποιήθηκαν.



Εικόνα 3-5 Αναλυτικό Μπλοκ Διάγραμμα Quadcopter + GCS + R/C Tx



Εικόνα 3-6 Απλοποιημένο Μπλοκ Διάγραμμα Quadcopter + GCS + R/C Tx

3.5 Ηλεκτρονικές Μονάδες

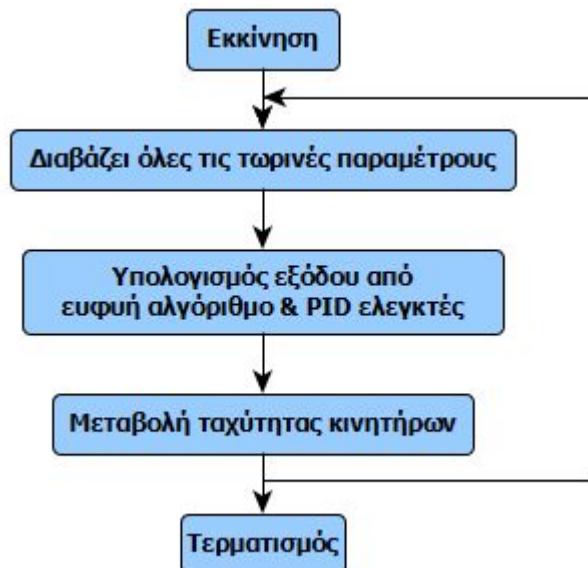
Τα μη επανδρωμένα εναέρια οχήματα μπορούν να χρησιμοποιούν πολλά και διάφορα ηλεκτρονικά συστήματα ελέγχου, από τα οποία το πιο σημαντικό είναι η κύρια μηχανή επεξεργασίας, ο ελεγκτής πτήσης (Flight Controller). Στα σύγχρονα UAV, όπως και στο δικό μας δηλαδή, ο ελεγκτής πτήσης ενσωματώνει σε μια πλατφόρμα ηλεκτρονικές υπομονάδες που εκτελούν διάφορες λειτουργίες. Αυτά είναι τα λεγόμενα όργανα καθορισμού πορείας και προσανατολισμού του αεροσκάφους, γυροσκόπια, επιταχυνσιόμετρα, βαρόμετρα, μαγνητόμετρα κλπ, τα οποία ο ελεγκτής πτήσης διαχειρίζεται και λαμβάνει απ' αυτά περιοδικά readings, που χρησιμοποιεί για να βελτιώσει τη πτήση. Με το μικρό τους μέγεθος και την ευελιξία τους, τα τετρακόπτερα μπορούν να πετούν και εντός και εκτός κτιρίων και εγκαταστάσεων. Θεωρητικά μπορούν να ανέλθουν σε πολύ μεγάλο υψόμετρο, όπως οι «γονείς» τους, τα επανδρωμένα αεροσκάφη.

Χρησιμοποιούνται τρείς ηλεκτρονικές πλατφόρμες στο δικό μας τετρακόπτερο. Η πρώτη είναι ο «εγκέφαλος» του οχήματος, εφόσον ενεργοποιεί κατάλληλα τα υποσυστήματα υψηλής ισχύος του οχήματος, δηλαδή τους κινητήρες και τους ηλεκτρονικούς ελεγκτές ταχύτητας. Ονομάζεται ArduPilot Mega, ή APM έκδοση 2.8. Πρόκεται για μια τροποποίηση της πασίγνωστης πλατφόρμας Arduino, που αρχικώς αναπτύχθηκε για ένα πανεπιστημιακό πρότζεκτ, για τον έλεγχο μικρών μη επανδρωμένων εναέριων οχημάτων. Η δεύτερη πλατφόρμα είναι ένα αναπτυξιακό board της Texas Instruments, ο μικροελεγκτής CC3200 με δυνατότητα ασύρματης δικτύωσης με το πρότυπο WiFi 802.11b/g/n με ποικίλους τρόπους, ως AP, P2P (WiFi direct), ή σύνδεση σε Wlan. Ο ρόλος της είναι να μεταφέρει τα δεδομένα τηλεμετρίας που λαμβάνει από το APM και να τα επανεκπέμπει στο συνδεδεμένο H/Y (ή SmartPhone), αλλά και το αντίστροφο. Πρόκειται για πλήρως αμφίδρομη (Full-Duplex) επικοινωνία. Στο CC3200 δόθηκε η μεγαλύτερη έμφαση, αφού έχει προγραμματισθεί πλήρως σε γλώσσα C. Η Τρίτη ηλεκτρονική πλατφόρμα στο τετρακόπτερο είναι ο δέκτης της ραδιοεπικοινωνίας (R/C), το Turnigy 9x Rx. Ο ραδιοδέκτης λειτουργεί στα 2.4Ghz και λαμβάνει τα

σήματα από τον αντίστοιχο πομπό που ελέγχει ο χειριστής και τα κατευθύνει στο APM. Τα σήματα αυτά τελικά θα χρησιμοποιηθούν από τον ελεγκτή πτήσης για τον καθορισμό της πορείας του αεροσκάφους.

3.5.1 Ελεγκτής Πτήσης ArduPilot Mega 2.8

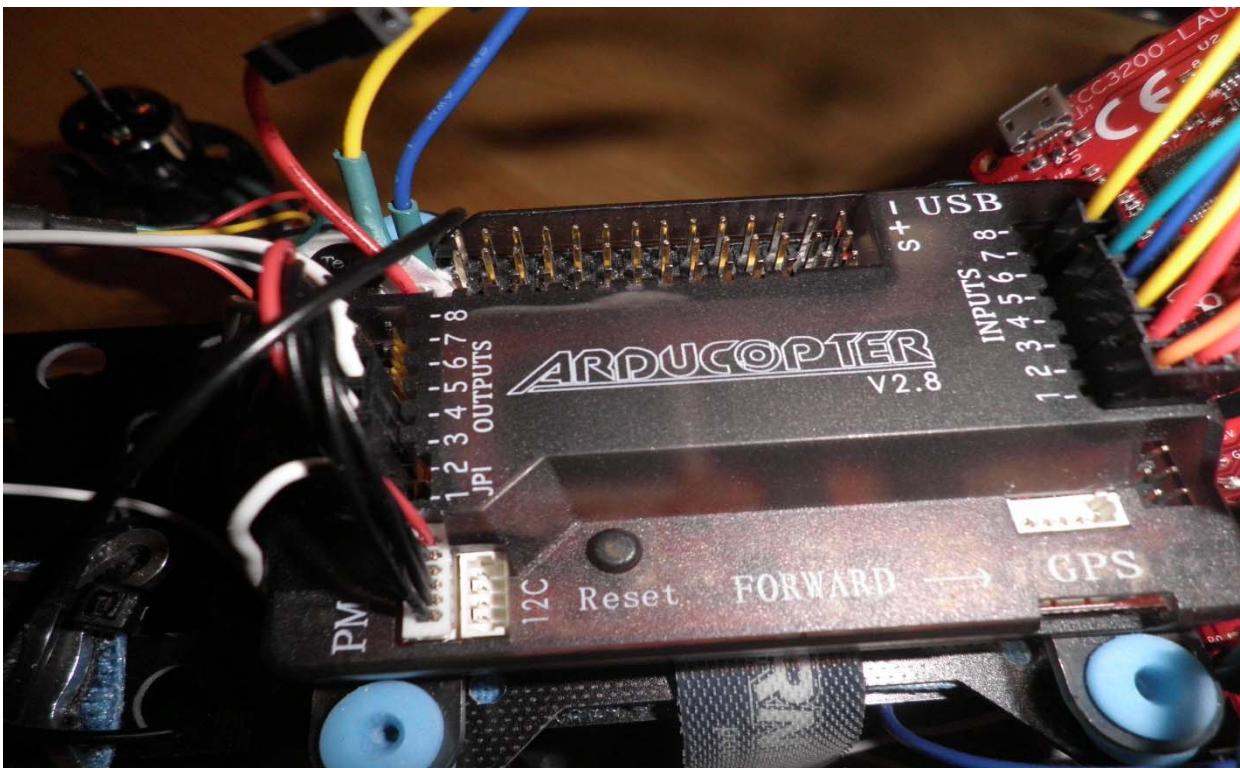
Ο ελεγκτής πτήσης είναι ένα μικρό τυπωμένο κύκλωμα με πρωταρχική λειτουργία για ένα μη επανδρωμένο όχημα. Η σκοπιμότητα του είναι να διαμοιράζει το σήμα που λαμβάνει από το ραδιοδέκτη με κατάλληλο τρόπο στους ηλεκτρονικούς ελεγκτές ταχύτητας (Electronic Speed Controller – ESC), ως απόκριση στην είσοδο του, την οποία καθορίζει ο πιλότος/χειριστής με τον αντίστοιχο ραδιοπομπό. Ο ελεγκτής πτήσης αφού εκκινήσει τη λειτουργία του και αρχικοποιήσει τις ρυθμίσεις του, εκτελεί περιοδικά τα εξής: 1) Διαβάζει τις οδηγίες του πιλότου από τον ραδιοδέκτη και τις τιμές των αισθητήρων, 2) Υπολογίζει τις τιμές των σημάτων που θα σταλούν στους κινητήρες μέσω ενός ευφυούς αλγορίθμου συγχώνευσης όλων των τιμών εισόδου και εμπλουτισμού τους με ελεγκτές PID, 3) Στέλνει τα σήματα στους κινητήρες, αλλάζοντας κατάλληλα την ταχύτητα τους. Τα βήματα 1, 2, 3 επαναλαμβάνονται συνεχώς μέχρι να τεθεί εκτός λειτουργίας.



Εικόνα 3-7 Μπλοκ Διάγραμμα Λειτουργίας Ελεγκτή Πτήσης

Η πλειοψηφία των ελεγκτών πτήσης μεταχειρίζονται αισθητήρες οι οποίοι συμπληρώνουν το σήμα πληροφορίας που στέλνεται στα ESC. Οι αισθητήρες είναι ένας μηχανισμός ανατροφοδότησης, λαμβάνοντας το ρόλο σταθεροποίησης του οχήματος ανά πάσα στιγμή, πάντα σε συμφωνία όμως με τις οδηγίες του πιλότου. Ο ρόλος τους είναι επικουρικός, αλλά δύσκολα θα φανταζόταν κανείς έναν ελεγκτή πτήσης χωρίς αισθητήρια όργανα. Ο ελεγκτής πτήσης εκτός από αισθητήρες συνίσταται από το Σύστημα Αναφοράς Θέσης και Πορείας (Attitude and Heading Reference System – AHRS) και ορισμένες φορές και από Αδρανειακό Σύστημα Πλοήγησης (Inertial Navigation System – INS).

Ο ελεγκτής πτήσης APM 2.8 που επιλέξαμε είναι ένας κλώνος του Arducopter Mega 2.6 της εταιρείας 3D Robotics. Όλα τα χαρακτηριστικά του είναι τα ίδια εκτός από κάποιες μετατροπές στην τοποθεσία διάφορων βραχυκυκλωτήρων και στη θύρα υποδοχής της μονάδας GPS+Εξωτερική Πυξίδα. Φαίνεται στη παρακάτω εικόνα, που είναι εγκατεστημένος πάνω σε ειδική αντικραδασμική βάση, με ελαστικά για να απορροφούν τις δονήσεις του UAV. Τα ελαστικά μετριάζουν την απορρόφηση μηχανικής ενέργειας που παράγεται από ταλαντώσεις και είναι σημαντικό να χρησιμοποιείται τουλάχιστον για τον ελεγκτή πτήσης, αφού η βαθμίδα βασίζει τη λειτουργία της σε ακριβείς μετρήσεις των αισθητήρων που διαθέτει για να εξομαλύνει την πορεία και τον προσανατολισμό του οχήματος. Εάν το επίπεδο δονήσεων είναι υψηλό αυτές οι τιμές αποκλίνουν με αποτέλεσμα την αποσταθεροποίηση του οχήματος.

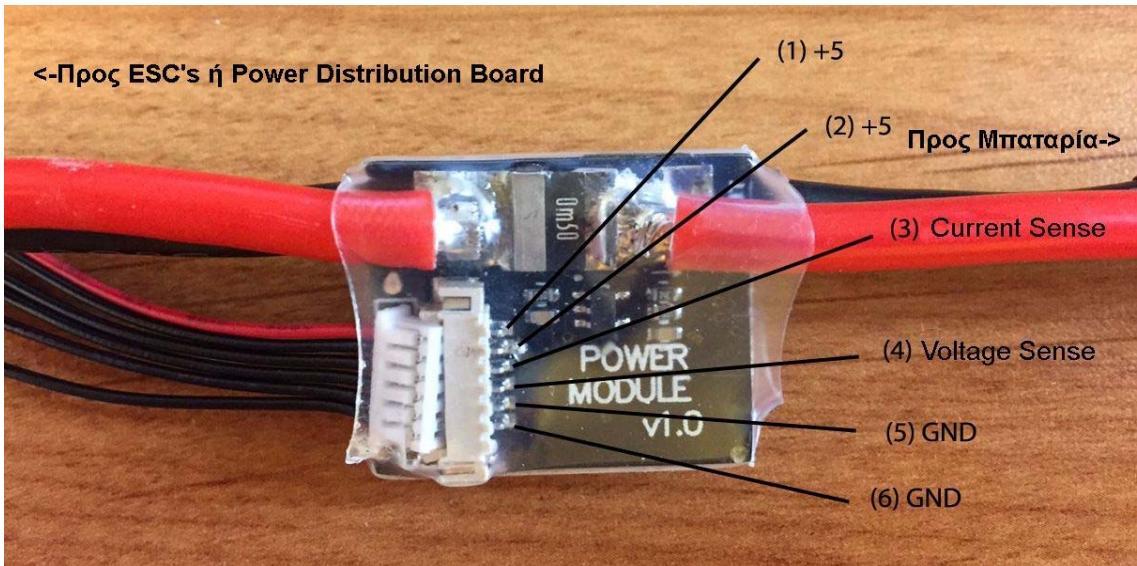


Εικόνα 3-8 Στην εικόνα φαίνεται η πλατφόρμα ελεγκτή πτήσης APM 2.8

Οι προδιαγραφές του APM 2.8 είναι [16]:

- 3-αξόνων γυροσκόπιο + επιταχυνσιόμετρο στη μονάδα 6DOF (Degrees of Freedom) MPU-6000 της Invensense. Εσωτερικά η μονάδα διαθέτει επεξεργαστή DMP Sensor fusion, ο οποίος συγχωνεύει τις τιμές του επιταχυνσιομέτρου και του γυροσκοπίου παράγοντας τιμές με τα πλεονεκτήματα και των δύο, αλλά χωρίς τα μειονεκτήματα κανενός.
- 3 αξόνων μαγνητόμετρο, το οποίο θα απενεργοποιηθεί μετακινώντας έναν βραχυκυκλωτήρα. Θα προτιμηθεί το εξωτερικό μαγνητόμετρο που βρίσκεται μαζί με τη μονάδα δέκτη GPS απομακρυσμένο από υψηλά ρεύματα και επαγόμενο θόρυβο.
- Υψηλής ευκρίνειας βαρόμετρο MS5611-01BA03 της Measurement Specialties. Πρόκειται για αισθητήρα πίεσης που μπορεί να ανιχνεύει το υψόμετρο στηριζόμενος στην ατμοσφαιρική πίεση του περιβάλλοντος.
- Επεξεργαστική μονάδα ATMEGA2560. Είναι ο μικροελεγκτής του ελεγκτή πτήσης. Τα χαρακτηριστικά του συνοπτικά είναι: επεξεργαστής 8-bit AVR αρχιτεκτονικής RISC, 256kBytes Flash, 16Mhz συχνότητα λειτουργίας, μονάδες επικοινωνίας SPI, TWI(I2C – σήμα κατατεθέν), UART, πολλαπλά κανάλια ADC, PWM, ρολόι πραγματικού χρόνου, Watchdog Timer κ.α.
- 4MB DataFlash chip για αυτόματη καταγραφή δεδομένων πτήσης (datalogging)
- ATMEGA32U-2 chip για συνδεσιμότητα USB.
- LP2985-3.3 Low Dropout ρυθμιστής τάσης. Μπορεί να δεχτεί τάση μέχρι και 16V και εξάγει τάση 3.3V, που είναι η τάση λειτουργίας του ATMEGA2560.
- Έχει προστεθεί πυκνωτής τανταλίου 47uF για το φιλτράρισμα υψίσυχων παρεμβολών αυξάνοντας έτσι την αξιοπιστία του συστήματος τροφοδοσίας.
- Σε αντίθεση με το APM 2.6 οι υποδοχές GPS και εξωτερικής πυξίδας έχουν τοποθετηθεί δίπλα-δίπλα επιτρέποντας πιο εύκολες συνδέσεις.
- Σε αντίθεση με το APM2.6 η υποδοχή I2C έχει περάσει από πολυπλέκτη και πλέον μπορεί να ρυθμιστεί ως θύρα UART0, UART2, I2C ή OSD.
- Μια δίοδο Zener με προστασία υπέρτασης, υπερέντασης και ανάστροφης τάσης. Στοιχίζει 0.37V σε πτώση τάσης, τα οποία πρέπει να αντισταθμιστούν από την τροφοδοσία του.

Ο ελεγκτής πτήσης απαιτεί τροφοδοσία στα 5V, 2A για τον εαυτό του. Επειδή όμως συνδέουμε και άλλες συσκευές σε αυτό, οι απαιτήσεις ρεύματος αυξάνονται. Σε πλήρη φόρτο, με όλες τις εξωτερικές συσκευές συνδεδεμένες, διαπιστώσαμε ότι το APM 2.8, απορροφάει έως και 450mA[17].



Εικόνα 3-9 APM Power Module. Τροφοδοτικό του APM 2.8.

Γι' αυτό το λόγο τροφοδοτούμε το APM με τη “dedicated” μονάδα τροφοδοσίας που το συνοδεύει, το APM Power Module, η οποία συνδέει τη μπαταρία με τη πλακέτα διανομής ισχύος για τροφοδοσία των εξαρτημάτων υψηλής ισχύος του τετρακόπτερου και παράλληλα διαθέτει ρυθμιστή τάσης που παρέχει καθαρή, εξομαλυμένη τάση 5.37V – το τέλειο επίπεδο – και ρεύμα έως 2.25A.

Το τροφοδοτικό στην είσοδο του αποδέχεται τάση μέχρι και 18V (μέχρι και 4S μπαταρία LiPo) και μέγιστο ρεύμα 90A (αλλά εφόσον το ίδιο το καλώδιο XT-60 αντέχει ρεύμα έως 60A τότε μάλλον και το ίδιο δε θα αντέχει περισσότερο ρεύμα!). Εκτός από την τροφοδοσία το PM εφοδιάζει τον ελεγκτή πτήσης με χρήσιμες μετρήσεις για την ποσότητα ρεύματος που απορροφάει κάθε χρονική στιγμή από τη μπαταρία, καθώς και την τάση της. Αυτές οι πληροφορίες είναι πολύ χρήσιμες καθώς όπως θα αναφέρουμε και αργότερα τις χρησιμοποιούμε ως δεδομένα τηλεμετρίας για να βλέπουμε κάθε φορά την κατάσταση της μπαταρίας. Επίσης χρησιμοποιώντας το πρόγραμμα Mission Planner βαθμονομήσαμε το APM PM με αναφορά την μπαταρία που χρησιμοποιούμε για να παίρνουμε ακριβείς μετρήσεις. Γενικά, για βέλτιστα αποτελέσματα στο πραγματικό περιβάλλον υψηλής ισχύος, στο οποίο θα λειτουργεί η πλατφόρμα APM, συχνά συμβαίνουν βυθίσεις τάσεις στη τροφοδοσία του εξαιτίας της ροής υψηλού ρεύματος σε κοντινή απόσταση που επάγει σε αυτό υψηλό θόρυβο, οπότε θέλουμε η τάση εισόδου στο APM να είναι από 4.8 έως 5.4 Volt. Κάτι που το πετύχαμε οριακά καθώς μετρήσαμε ότι η τάση του σε πλήρη φόρτιση αυτού και όλων των περιφερειακών συσκευών είναι ~4.8V.

Τέλος το APM διαθέτει επίσης 3 ενδεικτικά LED's ανάλογα με τη δραστηριότητα που εκτελεί κάθε φορά[18]. Η γνώση τους μας επιτρέπει να αντιλαμβανόμαστε εύκολα και γρήγορα την κατάσταση του σκάφους, δηλαδή παρέχονται για λόγους εκσφαλμάτωσης.

- Το κόκκινο A led όταν είναι σταθερά αναμμένο σημαίνει ότι το τετρακόπτερο είναι οπλισμένο, όπως λέμε χαρακτηριστικά, εννοώντας ότι εάν κουνήσουμε τον μοχλό από τον ραδιοπομπό, οι κινητήρες θα περιστραφούν. Όταν αναβοσβήνει διαδοχικά με μία μόνο λάμψη (single blink) το σκάφος είναι αφοπλισμένο, ενώ όταν αναβοσβήνει με διπλή λάμψη (double blink) οι κινητήρες δεν μπορούν να περιστραφούν, λόγω μη πληρότητας διαφόρων προϋποθέσεων.
- Το δεύτερο B κίτρινο led αναβοσβήνει μόνο κατά τη διάρκεια βαθμονόμησης, ή όταν έχει ενεργοποιηθεί ο τρόπος πτήσης (flight mode) Autotune.

- Το τρίτο C μπλέ led αφορά το GPS. Όταν είναι σταθερά αναμμένο ο δέκτης GPS έχει την καλύτερη λήψηστίγματος - 3D fix, όταν αναβοσβήνει το στίγμα δεν είναι ακριβές - 2D fix (υψηλό GPS HDOP – Horizontal Dilution of Precision, που είναι ένας δείκτης ποιότητας του γεωμετρικού στίγματος που παρέχεται από έναν δορυφόρο GPS – το «υψηλό» είναι παραμετροποιήσιμο), ενώ όταν δεν ανάβει καθόλου τότε δεν υπάρχει μονάδα GPS εγκατεστημένη, ή υπάρχει αλλά δυσλειτουργεί.

3.5.2 Σύστημα Αναφοράς Θέσης και Πορείας

Το σύστημα αναφοράς θέσης και πορείας (Attitude and Heading Reference System – AHRS) περιλαμβάνει αισθητήρες σε τρείς άξονες που παρέχουν πληροφορίες προσανατολισμού και πορείας για το αεροσκάφος. Σχεδιάστηκαν για να αντικαταστήσουν παραδοσιακά μηχανικά γυροσκοπικά όργανα και για να προσφέρουν μεγαλύτερη αξιοπιστία και ακρίβεια. Παλαιότερα το σύστημα λεγόταν Αδρανειακή Μονάδα Μέτρησης (Inertial Measurement Unit – IMU). Η διαφορά κλειδί μεταξύ των δύο είναι ότι στο AHRS περιλαμβάνονται συστήματα επεξεργασίας σήματος που επεξεργάζονται τα δεδομένα εισόδου, χρησιμοποιώντας ειδικούς αλγορίθμους, για βέλτιστη εκτίμηση της πορείας του οχήματος. Ένα δημοφιλές είδος μη γραμμικού αλγορίθμου, που τυπικά χρησιμοποιείται για να συγχωνεύσει τα αποτελέσματα από τους διάφορους on-board αισθητήρες και για να παράγει τη βέλτιστη εκτίμηση θέσης του αεροσκάφους, είναι το Διευρυμένο Φίλτρο Κάλμαν (Extended Kalman Filter – EKF). Το IMU απλώς διανέμει τους παλμούς των αισθητήρων σε μια άλλη συσκευή, η οποία παράγει τα τελικά αποτελέσματα για τον καθορισμό της πορείας. Εκτός αυτού το AHRS μπορεί να συγκροτήσει μέρος ενός αδρανειακού συστήματος πλοήγησης.

3.5.3 Αδρανειακό Σύστημα Πλοήγησης

Το αδρανειακό σύστημα πλοήγησης (Inertial Measurement System – IMS) είναι ένας βοηθός πλοήγησης που χρησιμοποιεί ένα σύγχρονο υπολογιστικό σύστημα, αισθητήρες κίνησης (επιταχυνσιόμετρα), [19]αισθητήρες περιστροφής (γυροσκόπια), ώστε να υπολογίζει συνεχώς την τοποθεσία, το προσανατολισμό και την ταχύτητα ενός κινούμενου αντικειμένου χωρίς την ανάγκη εξωτερικών πληροφοριών, μέσω της μεθόδου του νεκρού υπολογισμού (Dead Reckoning). Νεκρός υπολογισμός είναι μια διαδικασία υπολογισμού της τρέχουσας θέσης της κινητής μονάδας, που βασίζεται σε μια αμέσως προηγούμενη γνωστή, ή καθορισμένη γεωγραφική θέση και προωθώντας αυτή τη θέση, γνωρίζοντας τη τωρινή ταχύτητα επί τον χρόνο που θα παρέλθει στην πορεία ($\Delta x = U \cdot t$). Με αυτό τον τρόπο μπορεί να προβλεφθεί προσεγγιστικά και η επόμενη θέση. Αδρανειακά Συστήματα Πλοήγησης χρησιμοποιούνται σε αεροσκάφη, υποβρύχια, κατευθυνόμενους πυραύλους και διαστημόπλοια.

3.5.4 Σύστημα Ραδιοεπικοινωνίας Turnigy 9x

Ο τηλεχειρισμός ενός μη επανδρωμένου οχήματος πραγματοποιείται με το σύστημα ραδιοεπικοινωνίας το οποίο μπορεί να λειτουργεί σε διάφορες συχνότητες, αλλά στη δική μας περίπτωση είναι στα 2.4GHz. Πρόκειται για τη συμβατική επικοινωνία ραδιοκυμάτων (RF). Για τη λήψη ραδιοσημάτων χρησιμοποιείται μια κεραία. Επειδή όμως η κεραία λαμβάνει χιλιάδες διαφορετικά κύματα από πολλαπλές πηγές, πρέπει να χρησιμοποιηθεί ένα ηλεκτρονικό φίλτρο το οποίο να επιλέγει τα κύματα εκείνα που προέρχονται από την επιθυμητή πηγή, δηλαδή τον ραδιοπομπό. Αυτό επιτυγχάνεται στην απλούστερη περίπτωση με ένα κύκλωμα ενός πυκνωτή και ενός επαγωγέα που σχηματίζουν ένα συντονισμένο φίλτρο. Ο συντονιστής (resonator), όπως ονομάζεται, προσαρμόζοντας κατάλληλα την επαγωγή και χωρητικότητα του επαγωγέα και του πυκνωτή αντίστοιχα, ενισχύει τις συχνότητες μιας ζώνης ραδιοκυμάτων που επιθυμούμε και παράλληλα εξασθενεί τα ηλεκτρομαγνητικά κύματα πέρα από αυτό το φάσμα συχνοτήτων.

Εμείς λοιπόν επιλέξαμε το πολύ δημοφιλές σύστημα 9x της Turnigy[20], επειδή είναι εξαιρετικά χαμηλού κόστους, προτιμάται από τους περισσότερους αρχάριους χομπίστες, είναι πλήρως παραμετροποιήσιμο και στο software και στο hardware (πλέον) και έχει δυνατότητες frequency hopping, άρα είναι ανθεκτικό στις παρεμβολές (οι οποίες στα 2.4Ghz είναι συχνά πολλές!). Με τη διασπορά φάσματος αναπήδησης συχνότητας (FHSS) το ραδιοκύμα εκπέμπεται σε μια φαινομενικά ψευδοτυχαία αλληλουχία συχνοτήτων, αναπηδώντας από τη μια συχνότητα στην άλλη ανά σταθερά χρονικά διαστήματα. Μόνος όταν ένας δέκτης μεταπηδάει και εκείνος στη συχνότητα σε συγχρονισμό με τον πομπό θα μπορέσει να λάβει το σήμα. Ο αλγόριθμος που γεννά την ψευδοτυχαία αλληλουχία συχνοτήτων είναι γνωστός μόνο από τον πομπό και τον δέκτη, οπότε πιθανοί υποκλοπές ακούν μόνο ακατανόητους ήχους. Επίσης οι προσπάθειες παρεμβολής παρασίτων στο σήμα το μόνο που πετυχαίνουν είναι να εξαλείψουν μερικά bit[21].

Το πρώτο μας καθήκον μετά το unboxing του πομπού και του δέκτη είναι να τα συν-δέσουμε (bind) μεταξύ τους. Αυτό θα επιτρέψει στο πομπό να μπορεί να «μιλήσει» με τον συγκεκριμένο δέκτη. Επιπλέον ο δέκτης θα συγχρονιστεί με τον πομπό και έπειτα θα είναι δυνατή η επικοινωνία μεταξύ των δύο συσκευών. Η διαδικασία δέσμευσης (binding) είναι πολύ απλή. Αρκεί να ακολουθήσουμε τα παρακάτω βήματα που ισχύουν για το σύστημα Turnigy 9x, αλλά θα πρέπει να είναι παρόμοια (αν όχι ίδια) για οποιοδήποτε σύστημα ραδιοεπικοινωνίας ενός πομπού – ενός δέκτη:

1. Καταρχάς έχουμε τον ραδιοπομπό Turnigy 9x απενεργοποιημένο.
2. Τοποθετούμε το βύσμα δέσμευσης (binding plug) στη θύρα BAT του δέκτη. Ουσιαστικά είναι ένα καλώδιο που βραχυκυκλώνει τον ακροδέκτη σήματος με τον ακροδέκτη γείωσης στην υποδοχή BAT.
3. Ενεργοποιούμε τον ραδιοδέκτη. Μπορούμε να τοποθετήσουμε μπαταρία μέχρι 6.5V στη θυρίδα BIND. Τότε ένα (κόκκινο) led θα πρέπει να αναβοσβήνει.
4. Πατάμε το κουμπί BIND στον ραδιοπομπό για ~1 δευτερόλεπτο.
5. Ενεργοποιούμε τον ραδιοπομπό.
6. Παρατηρούμε το led του ραδιοδέκτη, όταν σταματήσει να αναβοσβήνει (συνήθως στιγμαία, αλλά μπορεί να χρειαστεί ως και 10 δευτερόλεπτα) η διαδικασία binding έχει ολοκληρωθεί.

Τώρα πλέον ο δέκτης μπορεί να επικοινωνήσει μόνο με τον συγκεκριμένο πομπό, ακόμα και στην περίπτωση που υπάρχουν άλλοι πανομοιότυποι δέκτες. Στην ουσία ο πομπός έχει καταχωρήσει στη μνήμη του ένα μοναδικό αναγνωριστικό για τη συσκευή του ραδιοδέκτη.



Εικόνα 3-10 Δέκτης Turnigy 9x



Εικόνα 3-11 Πομπός Turnigy 9x V2

Από τις εικόνες γίνεται εμφανές ότι και ο πομπός και ο δέκτης χρησιμοποιούν μια Whip κεραία. Είναι η πιο κοινώς χρησιμοποιούμενη μονοπολική κεραία, αφού είναι και η πιο φθηνή. Αποτελείται από ένα εύκαμπτο καλώδιο, ή μια επιμήκης ράβδο. Χρησιμοποιούνται ευρέως στην FM ραδιοφωνία, στα ραδιόφωνα αυτοκινήτων, συσκευές WiFi, κινητά τηλέφωνα κλπ. Η whip κεραία μπορεί να θεωρηθεί ως μισή διπολική κεραία γεγονός που αντικατοπτρίζεται και στο διάγραμμα ακτινοβολίας της, το οποίο είναι πανκατευθυντικό. Το μήκος τους είναι το $1/4^{\circ}$ του μήκους κύματος που χρησιμοποιείται, ή και ακόμα μικρότερο. Δηλαδή ιδανικά το μήκος της κεραίας στα 2.4Ghz που χρησιμοποιούμε εμείς είναι 12.5cm. Τα μειονεκτήματα της είναι ότι έχει μικρή απολαβή, περιορισμένη εμβέλεια και είναι επιρρεπής στη διάδοση πολλαπλών διαδρομών. Φυσικά η περιορισμένη εμβέλεια είναι σχετική. Η εμβέλεια του συστήματος 9x ανέρχεται σε παραπάνω από 1 χιλιόμετρο, με ισχύ εκπομπής μικρότερη από 20dbm/100mW[22], που υπερβαίνει κατά πολύ τις απαιτήσεις της παρούσας εργασίας. Επιπλέον η πόλωση των κεραιών είναι οριζόντια, γι' αυτό φροντίζουμε ώστε στο τετρακόπτερο η κεραία του ραδιοδέκτη να στέκεται παράλληλα σε αναφορά με το έδαφος. Τέλος οι ίδιες οι κεραίες είναι παθητικές δηλαδή δε διαθέτουν στάδια ενίσχυσης του μεταδιδόμενου σήματος.

Στο σύστημα Turnigy 9x η πληροφορία μεταδίδεται ψηφιακά χρησιμοποιώντας διαμόρφωση θέσης παλμού (PPM). Οι παλμοί των καναλιών μεταφέρονται σειριακά (ο ένας μετά τον άλλο) και αρκεί σε κάθε κύκλο να μεταβάλλεται μονάχα η θέση του δεξιού άκρου του κάθε παλμού για να γίνει διάκριση για την ένταση του εκάστοτε καναλιού. Ο ραδιοδέκτης PPM πρέπει να είναι συγχρονισμένος με το ίδιο ρολόι του πομπού. Σε κάθε κύκλο επαναλαμβάνεται νέα σειρά παλμών για κάθε κανάλι. Ο δέκτης αντιλαμβάνεται το τέλος του κύκλου όταν λάβει έναν καθορισμένο αρνητικό παλμό. Ο δέκτης αποδιαμορφώνει το σήμα, ξεχωρίζει το λαμβανόμενο παλμό για κάθε κανάλι, τον αποκωδικοποιεί και τελικά προωθεί αναλογικό σήμα διαμορφωμένο κατά εύρος παλμού

(Pulse Width Modulated – PWM)[23] στον ελεγκτή πτήσης. Η PPM είναι μια μέθοδος ψηφιακής διαμόρφωσης για μεταφορά απλών δεδομένων – εντολών[24] (όχι όγκου δεδομένων πχ. αρχεία). Οι ψηφιακές διαμορφώσεις προτιμούνται καθώς διευκολύνουν την υλοποίηση πολύπλοκων τηλεπικοινωνιακών συστημάτων χρησιμοποιώντας ψηφιακά σήματα που είναι εύκολο να επεξεργαστούν, επειδή κάθε φορά αρκεί να γίνεται διάκριση μόνο μεταξύ δύο τιμών, είτε υψηλής, είτε χαμηλής τιμής και όχι μιας ευρείας ζώνης συνεχών τιμών – αναλογική διαμόρφωση. Ακόμα και υπό την παρουσία θορύβου, η ύπαρξη, ή η απουσία ψηφιακών παλμών ξεχωρίζεται εύκολα. Αρκεί να τηρείται το θεώρημα δειγματοληψίας στο οποίο βασίζονται όλα τα ψηφιακά ηλεκτρονικά.

Στη συνέχεια, ο ελεγκτής πτήσης χρησιμοποιώντας κάποιον ευφυή αλγόριθμο, που στο APM 2.8 είναι το φίλτρο Kalman, εμπλουτίζει το ληφθέν σήμα, τοποθετώντας διορθωτικές τιμές που λαμβάνονται από τους αισθητήρες και τελικά το μοιράζει με κατάλληλο τρόπο στους ηλεκτρονικούς ελεγκτές ταχύτητας. Οι τελευταίοι λαμβάνουν το σήμα πληροφορίας και βάσει αυτού τροφοδοτούν τους brushless κινητήρες με AC τριφασικό ρεύμα προερχόμενο από την βασική πηγή ισχύος, την μπαταρία. Το AC σήμα είναι εναλλασσόμενο κύκλου καθήκοντος για να ανταποκρίνεται στο ληφθέν PWM σήμα. Περισσότερα γι' αυτό στο εδάφιο των ηλεκτρονικών ελεγκτών ταχύτητας.

Το σύστημα Turnigy 9x διαθέτει 8 κανάλια επικοινωνίας. Εμείς από τα 8 κανάλια αναθέσαμε λειτουργίες στα 6 και έχουμε απενεργοποιήσει τα υπόλοιπα 2. Κάθε κανάλι επιδέχεται εύρος παλμού του σήματος μεταξύ κάποιων τιμών που καθορίζονται όταν πραγματοποιούμε τη βαθμονόμηση της ραδιοεπικοινωνίας, στο πρόγραμμα Mission Planner. Το πρόγραμμα Mission Planner κανονικοποιεί αυτό το εύρος τιμών που παρουσιάζονται στον παρακάτω πίνακα. Αναφέρουμε τα κατανεμημένα κανάλια, το εύρος παλμού που επιδέχεται το καθένα, ο μοχλός (stick) που έχουμε ορίσει εμείς για την ενεργοποίηση του και η αντίστοιχη λειτουργία που επιτελείται.

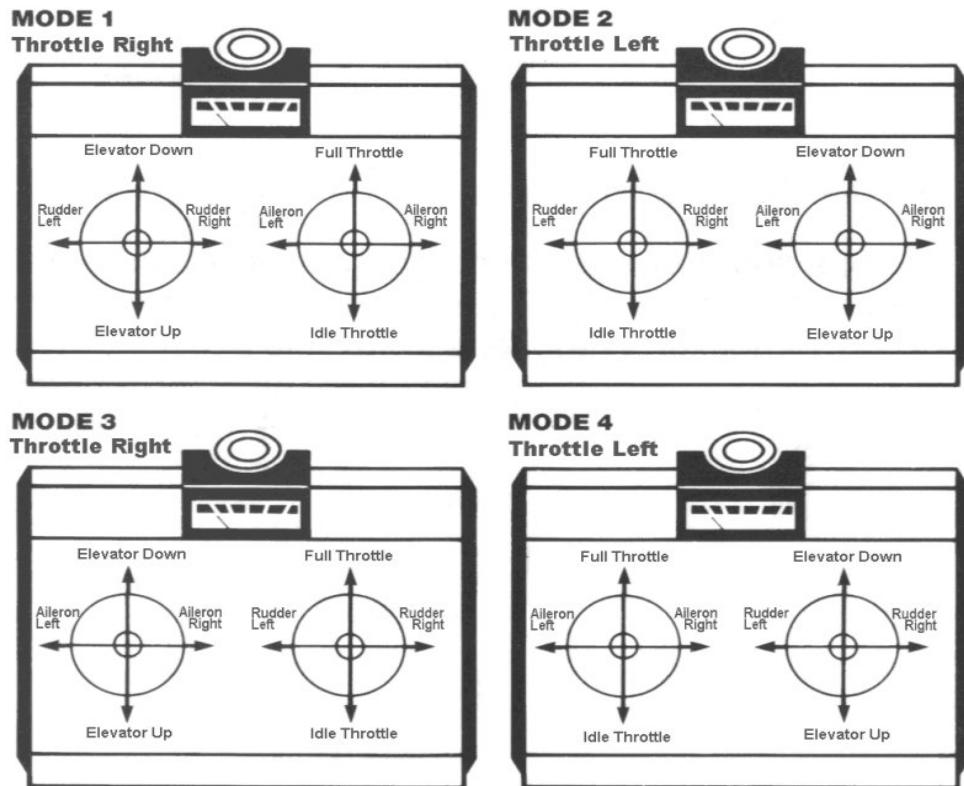
Κανάλι	Ελάχιστη Τιμή	Μέγιστη Τιμή	Λειτουργία
1	1048	1880	Roll
2	1050	1879	Pitch
3	1056	1883	Throttle
4	1055	1878	Yaw
5	1463	1473	Επιλογή Flight Mode
7	1497	1498	Βοηθητική λειτουργία

Πίνακας 1 Κανάλια Ελέγχου Ελεγκτή Πτήσης

Παρατηρούμε ότι με τα κανάλια 1-4 καθορίζουμε τη πορεία πτήσης του τετρακόπτερου όπως εξηγήσαμε στο προηγούμενο κεφάλαιο. Για παράδειγμα όταν μετακινούμε τον μοχλό για το κανάλι 3 στο ραδιοπομπό από το low στο high τόσο αυξάνεται η τιμή προς το 1883, που σημαίνει ότι τόσο αυξάνεται ο κύκλος καθήκοντος του PWM σήματος που αποστέλλεται στους κινητήρες, άρα τόσο πιο γρήγορα θα περιστρέφονται οι κινητήρες, τελικά αυξάνοντας τις δυνάμεις προώθησης και ανύψωσης του αεροσκάφους. Αυτό το απλό παράδειγμα θα πρέπει να κατέστησε σαφές το πως ακριβώς ανταποκρίνονται οι κινητήρες στην οδηγεία του χειριστή και εν ολίγοις πώς πετάει ένα μη επανδρωμένο εναέριο όχημα. Το κανάλι 5 δεσμεύεται για την εναλλαγή μεταξύ των διάφορων τρόπων πτήσης (Flight Modes) του τετρακόπτερου. Το κανάλι 7 σε κανονική λειτουργία είναι low (τιμή ~1497) και όταν μεταβάλλουμε τον μοχλό από τον ραδιοπομπό μας μεταβαίνει σε high (τιμή

~1498), όπου ενεργοποιείται μια εναλλακτική λειτουργία που έχουμε θέσει εμείς, η οποία μπορεί να είναι οτιδήποτε, από τον ήχο ενός piezobuzzer, μέχρι την κίνηση ενός ρομποτικού βραχίονα.

Επιπλέον ας σημειωθεί ότι υπάρχουν 4 τρόποι διαμόρφωσης που μπορεί κανείς να επιλέξει για τα sticks του ραδιοπομπού, ανάλογα με τα χειριστήρια που χρησιμοποιούνται για να καθοριστούν οι βασικές λειτουργίες πτήσης Roll, Pitch, Throttle και Yaw. Αυτοί οι τρόποι φαίνονται παρακάτω.



Εικόνα 3-12 Τρόποι ανάθεσης λειτουργιών πτήσης με τον R/C πομπό

Εμείς έχουμε επιλέξει τον δεύτερο τρόπο (MODE 2), δηλαδή Throttle και yaw με τον αριστερό μοχλό και Roll, Pitch με το δεξί, με τις κατευθύνσεις να εξηγούνται καλύτερα από την εικόνα.

Τέλος αναφέρουμε ότι κάθε ραδιοπομπός πρέπει να διαθέτει μια θύρα προπόνησης (training port), από την οποία μπορούμε να λάβουμε το PPM διαμορφωμένο σήμα για να ελέγξουμε κάποια άλλη συσκευή, πχ. έναν μικροελεγκτή, ή για να δούμε πως ανταποκρίνεται το σήμα στη μεταβολή των μοχλών στον παλμογράφο! Για να λάβουμε το σήμα από αυτή τη θύρα πρέπει να απενεργοποιήσουμε τον ήδη υπάρχον ραδιοπομπό. Στο σύστημα Turnigy 9x η training port βρίσκεται ακριβώς από πίσω και για να λάβουμε σήμα από αυτή αρκεί να αφαιρέσουμε τον ραδιοπομπό που βρίσκεται ακριβώς πάνω απ' τη θύρα, ουσιαστικά δηλαδή κόβοντας το μονοπάτι του με τη μπαταρία.

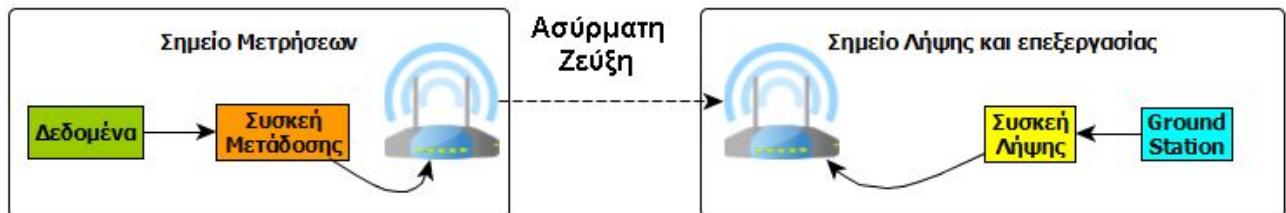
3.5.5 Σύστημα Τηλεμετρίας με τον Μικροελεγκτή CC3200 της T.I.

Συστήματα ασύρματης τηλεμετρίας επιτρέπουν τη δυνατότητα λήψης μετρήσεων από απροσπέλαστα σημεία και περιοχές, οι οποίες μπορεί να είναι απόμακρες, να μετακινούνται δυναμικά και απρόοπτα, ή επικίνδυνες. Έτσι μεταφέρουν πληροφορίες σε εμάς χωρίς υψηλό κόστος. Η πρόσφατη έξαρση με την πρόοδο των ηλεκτρονικών και τηλεπικοινωνιακών συστημάτων έχει καταστήσει τα συστήματα ασύρματης/απομακρυσμένης τηλεμετρίας πολύ αξιόπιστα και αποδοτικά. Ένα σύστημα τηλεμετρίας βασικά συντίθενται από τρία υποσυστήματα:

1. Ένα σύστημα λήψης δεδομένων (Data Acquisition System - DCP) που αποτελείται από τις πλατφόρμες συλλογής δεδομένων. Περιλαμβάνει όλο τον εξοπλισμό που χρειάζεται σε κάθε σταθμό επιτήρησης για να συλλέξει, αποθηκεύσει, κωδικοποιήσει και μεταδώσει τα δεδομένα. Ο

εξοπλισμός αυτός συνίσταται από αισθητήρες, τροφοδοτικό σύστημα, σύστημα αποθήκευσης και καταχώρισης των δεδομένων (logger) και την κεραία.

2. Ενα σύστημα μετάδοσης του σήματος. Εξοπλισμός απαραίτητος για τη μετάδοση των δεδομένων από το DCP στη βάση του χειριστή, ή στο ground station. Τις περισσότερες φορές η διάδοση του σήματος γίνεται υπό τη μορφή ενέργειας ραδιοκυμάτων (RF). Παρ’όλα αυτά μπορεί να γίνει χρήση ασύρματης τηλεμετρίας στη μικροκυματική, ή οπτική ζώνη συχνοτήτων αλλά αυτό είναι κάτι που συμβαίνει σπάνια. Συστήματα τηλεμετρίας στην ακουστική ζώνη συχνοτήτων αρκετές φορές χρησιμοποιούνται, για μικρή ποσότητα δεδομένων / μετρήσεων.
3. Το σύστημα ανάλυσης και επεξεργασίας των δεδομένων, τον σταθμό ελέγχου, που αναφέρεται και ως Ground Control Station (GCS). Το GCS συνίσταται κυρίως από έναν υπολογιστή, laptop, ή κάποια άλλη έξυπνη ηλεκτρονική συσκευή καθώς και από τη συσκευή λήψης του σήματος. Ο ασύρματος δέκτης λαμβάνει τα δεδομένα από το σημείο μέτρησης και τα δίνει στο GCS για αποθήκευση και επεξεργασία.



Εικόνα 3-13 Μπλοκ Διάγραμμα συστήματος ασύρματης τηλεμετρίας

Συστήματα τηλεμετρίας λειτουργούν σε πολλαπλές ζώνες συχνοτήτων με διάφορους τρόπους. Έτσι μπορούμε να συναντήσουμε[25]:

- VHF/UHF συστήματα τηλεμετρίας. Ο εξόπλισμός αυτών των συστημάτων δεν είναι ιδιαίτερα ακριβός (εάν δεν χρειάζονται επαναλλήπτες). Πρόκειται για καλή επιλογή εάν το DCP και το GCS απέχουν λιγότερο από 15 χιλιόμετρα. Μερικά μειονεκτήματα αυτά του είδους είναι ότι η εγκατάσταση δεν είναι εύκολη (μεγάλο βάρος) και πιθανόν να χρειαστεί η απόκτηση άδειας, ανάλογα με τους ισχύοντες κανονισμούς της χώρας.
- Κυτταρική (Cellular) τηλεμετρία, η οποία χρησιμοποιεί το κυτταρικό δίκτυο κινητής τηλεφωνίας. Χρησιμοποιείται σε τοποθεσίες με ισχυρή και αξιόπιστη κάλυψη (Cell Phone Coverage). Το hardware που απαιτείται δεν είναι δαπανηρό και η εγκατάσταση του δεν είναι πολύ δύσκολη. Μειονεκτήματα αφορούν τα μηνιαία τέλη που απαιτούνται και την ποιότητα του σήματος. Πρέπει να διασφαλιστεί εκ των προτέρων ότι η ποιότητα μεταφοράς δεδομένων είναι επιτρεπτή για τις ανάγκες της εφαρμογής. Στην αρχή της πορείας αυτής την εργασίας σκεφτήκαμε ότι θα μπορούσαμε να υλοποιήσουμε το σύστημα τηλεμετρίας χρησιμοποιώντας το κυτταρικό δίκτυο 3G, αλλά εγκαταλείφθηκε η ιδέα δεδομένου του αρκετού περιττού βάρους του κινητού τηλεφώνου και των επιπλέον εξόδων που αιτούνται.
- Σύστημα τηλεμετρίας με Ραδιοεπικοινωνία που χρησιμοποιεί τεχνολογία αναπήδησης συχνότητας. Μια κατάλληλη ζώνη συχνοτήτων γι' αυτό είναι τα 2.4GHz επειδή είναι ελεύθερη, δεν απαιτείται άδεια. Επίσης ο εξόπλισμός είναι αρκετά πιο εύκολο να εγκατασταθεί, σε σχέση με VHF/UHF, αλλά η εμβέλεια της επικοινωνίας είναι περιορισμένη, κατά μέσο όρο το πολύ μέχρι 10 χλμ. Τα συστήματα ραδιοεπικοινωνίας που δεν χρησιμοποιούν τεχνολογία FHSS στη 2.4GHz μπάντα συχνοτήτων είναι πολύ επιρρεπή σε αλλοιώσεις σήματος και παρεμβολές, καθώς σε αυτή τη ζώνη λειτουργούν κάθε λογής ασύρματες ηλεκτρονικές συσκευές. Για να καταπολεμηθεί το πρόβλημα των παρεμβολών θα μπορούσε, αντί για 2.4GHz, να γίνει χρήση της 5-5.8 GHz ζώνης. Σε αυτήν όμως μειώνεται η εμβέλεια της επικοινωνίας επειδή μειώνεται το μήκος κύματος. Είναι αλήθεια ότι θα μπορούσαμε να λαμβάνουμε δεδομένα τηλεμετρίας στο

ραδιοπομπό μας, αλλά δεδομένου ότι τέτοιοι πομποί κοστίζουν αρκετά περισσότερο από το αρκετά φθηνό σύστημα μας, δεν στηριχτήκαμε σε αυτή την επιλογή.

- Σύστημα τηλεμετρίας περιορισμένης εμβέλειας μέσω WiFi, χρησιμοποιώντας πρωτόκολλα δικτύωσης H/Y, όπως TCP/IP. Αυτή η επιλογή προσφέρει τη δυνατότητα μεταφοράς μεγάλου όγκου πληροφοριών, με αρκετά καλή ποιότητα επικοινωνίας – χαμηλό Bit Error Rate (BER) – κάτι που ισχύει ιδιαίτερα αν χρησιμοποιηθούν πρωτόκολλα επικοινωνίας παραπλήσια του TCP. Αυτή ήταν και η επιλογή μας, αφού δεν ενδιαφερόμασταν για πτήση σε μεγάλες αποστάσεις. Επιπλέον εφόσον δεν τοποθετούμε κάμερα στο μη επανδρωμένο αεροσκάφος, το πεδίο όρασης μας θα ήταν μηδενικό ύστερα από λίγες εκατοντάδες μέτρα, ούτως η άλλως.
- Δορυφορικό σύστημα τηλεμετρίας, που είναι το καλύτερο για
 - 1) Απομακρυσμένα σημεία παρακολούθησης, που τις περισσότερες φορές που δεν υπάρχει line-of-sight και παράλληλα η εμβέλεια της ραδιοεπικοινωνίας δεν είναι επαρκής,
 - 2) Τοποθεσίες χωρίς κάλυψη κυτταρικού δικτύου,
 - 3) Πρακτικά η κάλυψη είναι παγκόσμια, εκτός αν πρόκειται για υπόγειες, ή εσωτερικές περιοχές,
 - 4) Άλλες επιλογές τηλεμετρίας δεν είναι οικονομικά δυνατές, ή ωφέλιμες, πχ. το κόστος ποιοτικής επικοινωνίας δεν είναι αρκετό, ή απαιτείται εγκατάσταση επαναληπτών (repeaters).

Για την εγκατάσταση ενός ιδανικού ασύρματου συστήματος τηλεμετρίας, το σύστημα θα πρέπει[26]:

- Να είναι απλό στην εγκατάσταση.
- Να είναι αρκετά εύρωστο ώστε να επιβιώσει στο περιβάλλον στο οποίο θα χρησιμοποιηθεί.
- Να είναι συμβατό με διάφορα είδη αισθητήρων.
- Να μπορεί να μεταδίδει τα δεδομένα απευθείας στο GCS, ή να τα αποθηκεύει για μετέπειτα χρήση, ή και τα δύο.
- Να μη χρειάζεται μεγάλο απόθεμα ισχύος.
- Να μπορεί να τροφοδοτείται με ανανεώσιμη πηγή ενέργειας, πχ. με φωτοβολταϊκό σύστημα. Αν αυτό δεν είναι δυνατό τότε με επαναφορτιζόμενες μπαταρίες.
- Να μην καθιστά αναγκαία την απόκτηση άδειας.
- Να υπάρχει επαρκής ραδιοκάλυψη για τις ανάγκες της εφαρμογής.
- Να είναι επεκτάσιμο και παραμετροποίησιμο.
- Να υπάρχει καλό λογισμικό διάγνωσης, ανάλυσης και επεξεργασίας.
- Να υπάρχει δυνατότητα εύκολου διαμοιρασμού των δεδομένων σε άλλους κόμβους.

Εμείς αποφασίσαμε να χρησιμοποιήσουμε την αναπτυξιακή πλατφόρμα CC3200LP αναθεώρηση 4.1 της Texas Instruments, για να εγκαθιδρύσουμε μια ζεύξη ασύρματης WiFi τηλεμετρίας μεταξύ του τετρακόπτερου και του GCS. Κάποιες σημαντικές προδιαγραφές του μικροελεγκτή CC3200[27] που αξίζει να αναφέρουμε είναι:

- Διαθέτει μικροελεγκτή για τις εφαρμογές, ξεχωριστό επεξεργαστή για τις δυνατότητες δικτύωσης WiFi και το υποσύστημα διαχείρισης ενέργειας,
- Ο μικροελεγκτής για τις εφαρμογές αποτελείται από:
 - ARM Cortex M4 πυρήνα με συχνότητα λειτουργίας 80Mhz,
 - 32 κανάλια Direct Memory Access controller (μDMA),
 - 2 μονάδες UART,
 - Προηγμένη βαθμίδα ασφαλείας σε hardware, Crypto Engine που υποστηρίζει αλγόριθμους κωδικοποίησης AES, DES, SHA2, MD5, CRC, 3DES και Checksum,
 - 1 δίαυλο σειριακής περιφερειακής επικοινωνίας SPI,
 - 1 δίαυλο I^2C ,

- 4 χρονιστές γενικού σκοπού με παραγωγή κυματομορφών PWM ακρίβειας 16bit,
- 1 μονάδα χρονιστή επιτήρησης (Watchdog Timer),
- Αναλογοψηφιακοί μετατροπείς ακρίβειας 12bit 4 καναλιών,
- Μέχρι 27 προγραμματιζόμενους πολυπλεγμένους ακροδέκτες γενικού σκοπού,
- Κανάλια I2S,
- Ενσωματωμένη μνήμη
 - Ram (μέχρι 256KB),
 - Serial Flash για τη φόρτωση του Bootloader και των οδηγών περιφερειακών συσκευών,
- Ο WiFi επεξεργαστής δικτύωσης περιλαμβάνει:
 - ARM MCU, επικοινωνεί με τον μικροελεγκτή εφαρμογών μέσω υψηλής ταχύτητας θύρας SPI στα 20MHz,
 - Wi-Fi και διαδικτυακά πρωτόκολλα στη ROM,
 - TCP/IP Stack,
 - BSD Socket API's,
 - 8 παράλληλα TCP, ή UDP sockets,
 - 2 παράλληλα TLS και SSL sockets,
 - Τρόποι λειτουργίας ως Station, Access Point και WiFi Direct,
- Ευφυείς τρόποι χαμηλής ισχύος (Low Power Modes),
- Clock 40MHz με εσωτερικό ταλαντωτή,
- 32.768kHz κρύσταλλο, ή εξωτερικό ρολόι πραγματικού χρόνου.

Παρατηρούμε δηλαδή ότι υποστηρίζει όλα τα σύγχρονα πρότυπα ασύρματης WiFi δικτύωσης με πρακτικό μέγεθος και με πολύ μικρό βάρος (μόλις 29gr), γεγονός πολύ ελκυστικό για χρήση σε ένα μη επανδρωμένο αεροσκάφος, με αυστηρούς περιορισμούς σε όγκο, βάρος, τροφοδοσία και τρόπους επικοινωνίας.

Το CC3200 έχει προγραμματιστεί σε γλώσσα C ώστε να συνδέεται με τον ελεγκτή πτήσης APM 2.8, να λαμβάνει από αυτόν τα δεδομένα τηλεμετρίας μέσω της μονάδας σειριακής επικοινωνίας UART1 και έπειτα να τα εκπέμπει ασύρματα στο GCS εντός του WiFi δικτύου. Ο μικροελεγκτής έχει προγραμματισθεί ώστε να μπορεί να δημιουργεί το δικό του σημείο πρόσβασης στο WiFi δίκτυο, ως Software Access Point και έπειτα μπορεί να συνδεθεί μόνο ένας κόμβος σε αυτόν. Αφού συνδεθεί αρχίζει η μεταφορά δεδομένων.

Εκτός όμως από AP τρόπο λειτουργίας το CC3200 μπορεί να συνδεθεί και σε ένα ήδη υπάρχων ασύρματο τοπικό δίκτυο (WLAN) και έπειτα να αποστείλει τα δεδομένα στον δικτυακό κόμβο με κατάλληλο εγκατεστημένο λογισμικό GCS. Έχουμε καταστήσει εφικτές και τις δύο δυνατότητες. Για σύνδεση σε WLAN μπορεί κάποιος να συνδέσει το CC3200 σε μια υπολογιστική βαθμίδα μέσω USB και να εκκινήσει ένα πρόγραμμα σειριακού τερματικού εξομοιωτή (Serial Terminal Emulator), πχ. εμείς χρησιμοποιήσαμε το Tera Term, ώστε για επικοινωνία με το μικροελεγκτή, ώστε να εισάγει τις παραμέτρους του Wlan και του GCS.

Πιο συγκεκριμένα όπως μπορούμε να διαπιστώσουμε και από το μπλοκ διάγραμμα του τετρακόπτερου (**Error! Reference source not found.**) και από την παρακάτω Εικόνα 3-14 έχουμε προγραμματίσει το CC3200 ώστε οι ακροδέκτες 7 και 8 να χρησιμοποιούν τη θύρα σειριακής επικοινωνίας UART1 Tx/Transmitter (αποστολέας) και Rx/Receiver (παραλήπτης) αντίστοιχα. Αναφέραμε ότι όλοι οι ακροδέκτες διαθέτουν πολυπλέκτες για να επιλέγουν κάθε φορά μία από διάφορες πιθανές λειτουργίες. Οι ακροδέκτες 7 και 8 συνδέονται με την αντίστοιχη θύρα UART του ελεγκτή πτήσης σταυρωτά, δηλαδή Rx->Tx και Tx->Rx, όπως φαίνεται και στο μπλοκ διάγραμμα. Το BAUD Rate της επικοινωνίας έχει καθοριστεί στα 57600 σύμβολατο δευτερόλεπτο (εδώ το ένα σύμβολο-baud- αποτελείται από ένα bit). Ο ακροδέκτης UART Tx του ελεγκτή πτήσης στέλνει την

πλειοψηφία των δεδομένων, ενώ ο UART Tx του CC3200 στέλνει περιοδικά στο APM διάφορες εντολές που προέρχονται από το GCS.

Τα πρωτόκολλα επικοινωνίας που χρησιμοποιούνται για τη μεταφορά των δεδομένων είναι τα UDP και TCP. Κατά την τροφοδοσία του CC3200 δίνεται δυνατότητα επιλογής ενός από τα δύο πρωτόκολλα επικοινωνίας, με το πάτημα κατάλληλων κουμπιών. Επίσης πρέπει να διασφαλιστεί ότι κάποιο software ή hardware firewall στον στόχο που θα αποσταλούν τα δεδομένα δεν μπλοκάρει το TCP, ή UDP traffic. Έχουμε φροντίσει και για αυτό. Θα πούμε περισσότερα για τα πρωτόκολλα επικοινωνίας που χρησιμοποιούνται στο επόμενο κεφάλαιο.

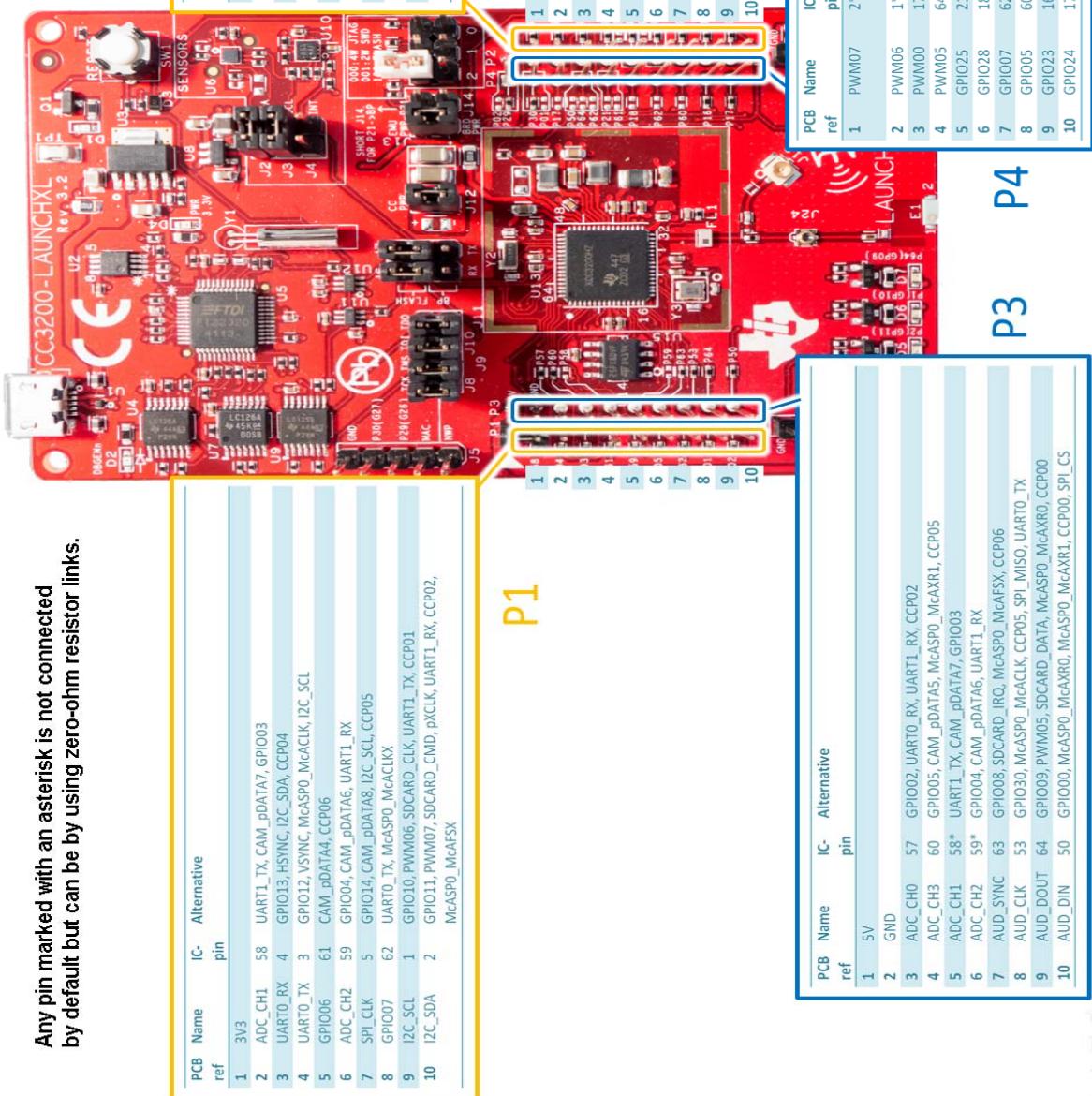
Το CC3200 απαιτεί τροφοδοσία σταθεροποιημένη στα ~3.3V. Όταν στέλνει &όταν λαμβάνει δεδομένα στο WiFi δίκτυο η κατανάλωση ρεύματος ανέρχεται έως και τα 200mA. Μερικές φορές φτάνει και τα 300mA, σε ξαφνικές ριπές όγκου δεδομένων (data-bursts) που τις συνηθίζει. Μπορούμε να μετρήσουμε την κατανάλωση ρεύματος της πλατφόρμας, εάν αφαιρέσουμε τον βραχυκυκλωτήρα J12 και τοποθετήσουμε κατά μήκος του τις άκρες ενός αμπερομέτρου. Με αυτό τον τρόπο μετράμε το ρεύμα που καταναλώνει ο μικροελεγκτής των εφαρμογών και ο μικροελεγκτής της δικτύωσης καθώς και φορτία «υψηλής» ισχύος (όπως LED) που ενεργοποιεί η διάταξη[28]. Όταν η πλατφόρμα είναι συνδεδεμένη σε ασύρματο τοπικό δίκτυο και στέλνει/λαμβάνει δεδομένα συμβαίνουν βυθίσεις τάσεις στην τροφοδοσία της, όπως αναμενόταν εφόσον απορροφά πολύ περισσότερο ρεύμα. Τότε πρέπει να φροντίζουμε ότι η τάση της δε θα μειωθεί περισσότερο από 2.1V, που είναι το επίπεδο για Brownout Reset. Εάν η τάση μειωθεί κάτω από αυτό το κατώφλι, ενώ παράλληλα είναι πάνω από 1.4V (επίπεδο τροφοδοσίας Blackout) όλη η ψηφιακή λογική μετασχηματίζεται σε power gated. Power gating είναι μια τεχνική που χρησιμοποιείται στα ολοκληρωμένα κυκλώματα η οποία μειώνει την κατανάλωση ισχύος, απενεργοποιώντας τη ροή ρεύματος σε τμήματα του τσιπ που δεν χρησιμοποιούνται και κατά συνέπεια μειώνεται η κατανάλωση ισχύος της συσκευής (σε επίπεδο μΑ).

Τροφοδοτούμε το CC3200 από τον ελεγκτή πτήσης APM 2.8 παρεμβάλλοντας δύο διόδους 1N4007 (αντοχή μέχρι και 1A συνεχόμενο ρεύμα) εν σειρά, οι οποίες έχουν πτώση τάσης ορθής πόλωσης ~1.4V και τελικά το CC3200 λαμβάνει τάση ~3.4V, που σε κατάσταση πλήρους λειτουργίας φτάνει τα 3.1V – απολύτως κατάλληλο επίπεδο. Επίσης χρησιμοποιούμε πυκνωτές αποσύζευξης για εξομάλυνση της τάσης τροφοδοσίας του μικροελεγκτή. Επομένως είναι εξασφαλισμένο ότι ο μικροελεγκτής δουλεύει όπως πρέπει. Αρχικώς αντιμετωπίσαμε σημαντικά προβλήματα με την τροφοδοσία της βαθμίδας τηλεμετρίας CC3200 και θα θέλαμε να αναφερθούμε σε αυτά λίγο περισσότερο και να εξηγήσουμε πώς βρήκαμε τις λύσεις.

Αρχικώς χρησιμοποιήσαμε δύο AA μπαταρίες, χωρίς πυκνωτές αποσύζευξης, που παρείχαν τάση 2.7V που σε κατάσταση πλήρους λειτουργίας και των κινητήρων επαγόταν ισχυρός ηλεκτρομαγνητικός θόρυβο γεγονός που επανατοποθετούσε τη διάταξη από μόνη της, με αποτέλεσμα να χάνουμε τη ζεύξη τηλεμετρίας! Εκτός από το ότι η τάση δεν ήταν αρκετή σε πλήρης λειτουργία του τετρακόπτερου, το βασικό σφάλμα ήταν ότι δεν ήταν σταθεροποιημένη. Συνέβαιναν στιγμιαίες βυθίσεις τάσεις κατά τη διάρκεια πλήρης λειτουργίας – πτήσης του τετρακόπτερου κάτω από το επίπεδο Brownout Reset. Για να καταπολεμήσουμε αυτό το φαινόμενο τοποθετήσαμε 3 πυκνωτές παράκαμψης, έναν τανταλίου 10μF και δύο κεραμικούς 0.1μF και 10nF για καλή αποσύζευξη σε μεγάλο εύρος συχνοτήτων. Με την προσθήκη των πυκνωτών, οποτεδήποτε πρόκειται να συμβεί στιγμιαία πτώση τάσης, ανταποκρίνεται ο πυκνωτής πολύ πιο γρήγορα από την τροφοδοσία και «καλύπτει» εγκαίρως το κενό τροφοδοσίας που παρουσιάζεται. Είναι σημαντικό επίσης οι πυκνωτές να τοποθετηθούν όσο πιο κοντά γίνεται στον ακροδέκτη που «παρακάμπτεται», για να αυξηθεί η αποτελεσματικότητα τους.

Με αυτές τις τροποποιήσεις, την αντικατάσταση με 2 AA μπαταρίες και την προσθήκη πυκνωτών, δεν ξαναπαρουσιάστηκε πρόβλημα. Όμως τα επιπλέον 50g των μπαταριών θα μπορούσαν

Any pin marked with an asterisk is not connected by default but can be by using zero-ohm resistor links.



Εικόνα 3-14 Πρόσοψη του CC3200 αναπτυξιακού με τους τρόπους χρήσης όλων των ακροδεκτών του[29]

να παραλειφθούν, παρόλο που το τετρακόπτερο πέταγε ικανοποιητικά. Το επιπλέον βάρος προξενεί περαιτέρω φόρτο στους κινητήρες με αποτέλεσμα να χρειάζεται να απορροφούν περισσότερο ρεύμα για να λειτουργούν ελαττώνοντας πιο γρήγορα τη διάρκεια ζωής της μπαταρίας. Τελικά αφαιρέσαμε τις μπαταρίες και χρησιμοποιήσαμε τις δύο διόδους για τροφοδοσία του CC3200 από τον ελεγκτή πτήσης. Η ποιότητα πτήσης από εκεί και ύστερα ήταν εξαιρετική και η διάρκεια μιας πλήρης πτήσης αυξήθηκε τουλάχιστον κατά 2 λεπτά, φτάνοντας έως και τα ~12 λεπτά με την μπαταρία LiPo φορτισμένη στο ~15%.

3.6 Αισθητήρες

Ένα μη επανδρωμένο ιπτάμενο αεροσκάφος θα πρέπει να είναι εφοδιασμένο με αφθονία αισθητήρων. Αισθητήρες όπως επιταχυνσιόμετρο, γυροσκόπιο και μαγνητόμετρο είναι οι ελάχιστοι απαραίτητοι για προσανατολισμό του οχήματος. Όλοι αυτοί πρέπει να λειτουργούν και στους τρείς ευκλείδειους άξονες, ώστε να δίνουν αξιόπιστες μετρήσεις στο χώρο. Γενικά όσο περισσότεροι αισθητήρες χρησιμοποιούνται τόσο περισσότερο ελέγχουμε την κατάσταση του οχήματος και με αυτό τον τρόπο είναι δύνατό να προσθέτουμε περισσότερους βαθμούς ελευθερίας στο UAV, συντελώντας σε μια πλήρως αυτόνομη πτήση.

Ακόμη, αισθητήρες GPS είναι πιθανοί με τους οποίους το όχημα δύναται να λαμβάνει οδηγίες για την πορεία που θα ακολουθήσει χωρίς την ανάγκη χειροκίνητης επέμβασης. Με GPS είναι δυνατή η εκτέλεση προσχεδιασμένων αποστολών από το διαχειριστή του συστήματος, χρησιμοποιώντας το πρόγραμμα GCS. Τέτοιοι αισθητήρες είναι συνήθως μέρος του αυτόματου πιλότου του οχήματος και συνήθως τέτοια οχήματα χρησιμοποιούνται για εφαρμογές παρακολούθησης καταστροφών, πυρκαγιών, για έλεγχο χώρων κλπ. Παράλληλα όμως επιπλέον αισθητήρες αυξάνουν το βάρος και το κόστος. Συχνά χρησιμοποιούνται αισθητήρες υπερήχων για τη βέλτιστη ρύθμιση του υψομέτρου. Σε αυτό το εδάφιο θα κάνουμε μια σύντομη περιγραφή των επιμέρους αισθητήρων και θα μιλήσουμε περισσότερο για αυτούς που εμείς χρησιμοποιήσαμε στο τετρακόπτερο.

3.6.1 Επιταχυνσιόμετρο

Το επιταχυνσιόμετρο (Accelerometer) είναι μια ηλεκτρομηχανική (Microelectromechanical – MEMS) συσκευή που έχει την ικανότητα να μετρά δυνάμεις επιτάχυνσης. Είτε στατικές όπως η επιτάχυνση της βαρύτητας, είτε δυναμικές όταν προκαλούνται – προέρχονται από αλλαγές στην ταχύτητα ή στη διεύθυνση της κίνησης. Αυτό επιτυγχάνεται με μια μάζα αναρτημένη μεταξύ δύο αγώγιμων πλακών, οι οποίοι σχηματίζουν έναν πυκνωτή. Η μάζα αντιτίθεται σε δυνάμεις που προσπαθούν να τη μετακινήσουν εξαιτίας της αδράνειας της. Μια επιτάχυνση στη μετρούμενη κατεύθυνση θα επιφέρει μετατόπιση αυτής της μάζας, από τη θέση ισορροπίας της, με αποτέλεσμα τη μεταβολή της χωρητικότητας σύμφωνα με τη μεταβολή της επιτάχυνσης. Επιπρόσθετα κυκλώματα μετατρέπουν αυτή τη μεταβολή της χωρητικότητας σε τάση, η οποία έπειτα θα ψηφιοποιηθεί και θα προμηθεύσει μια προσέγγιση της απόλυτης στάσης του αεροσκάφους στο χώρο. Η μέτρηση τριαξονικής επιτάχυνσης (a_x, a_y, a_z στο APM 2.8) (ουσιαστικά πρόκειται για τρία μικροσκοπικά επιταχυνσιόμετρα τοποθετημένα κατάλληλα μεταξύ τους) είναι αναπόσπαστο συστατικό τμήμα του AHRS των αεροσκαφών. Παρ' όλα αυτά τα επιταχυνσιόμετρα έχουν και μειονεκτήματα. Σε γενικές γραμμές δε διαθέτουν ιδιαίτερα γρήγορη απόκριση και είναι επιρρεπή σε θόρυβο και μηχανικές δονήσεις.

3.6.2 Γυροσκόπιο

Το κλασικό μηχανικό γυροσκόπιο (Gyroscope) είναι ένας τροχός που περιστρέφεται γοργά γύρω από τον άξονα του. Η αρχή λειτουργίας του γυροσκοπίου στηρίζεται στην αρχή διατήρησης

της στροφορμής. Το μεγαλύτερο μέρος της μάζας του γυροσκοπίου θα πρέπει να βρίσκεται όσο είναι δυνατό πιο μακριά από τον άξονα περιστροφής. Όταν το γυροσκόπιο δεν περιστρέφεται συμπεριφέρεται σαν ένα οποιοδήποτε αντικείμενο. Όταν όμως τεθεί υπό περιστροφή με μεγάλη ταχύτητα, γύρω από τον κύριο άξονα του, αντιστέκεται σε απόπειρες κίνησης του προς διάφορες άλλες κατευθύνσεις. Ένα περιστρεφόμενο γυροσκόπιο μπορεί να συγκρατήσει υψηλά ποσά ενέργειας. Ο πρώτος νόμος του Νεύτωνα δηλώνει ότι ένα αντικείμενο έχει την τάση να διατηρεί τη κατάσταση κίνησης του σταθερή (είτε ακίνητο, είτε εν κινήσει), λόγω αδράνειας, μέχρις ότου επαρκείς εξωτερικές δυνάμεις τη μεταβάλλουν. Εξαιτίας των δυνάμεων που δρούν όταν το γυροσκόπιο στρέφεται, το τελευταίο θα προσπαθήσει να τις αντισταθμίσει, χρησιμοποιώντας την εσωτερική του ενέργεια, ώστε να διατηρήσει την κατάσταση του. Η πιο συνήθης χρήση του γυροσκοπίου είναι για τη σταθεροποίηση αντικειμένων, που είναι ακριβώς ο λόγος που χρησιμοποιείται στα αεροσκάφη.

Σε μορφή αισθητήρα είναι μια μικροσκοπική ηλεκτρομηχανική συσκευή που περιλαμβάνει ένα δονούμενο στοιχείο περιτριγυρισμένο από χωρητικούς αισθητήρες. Οποιαδήποτε περιστροφή στους ανιχνευόμενους (τρείς) ευκλείδιους άξονες θα επιφέρει μια απόκλιση στο δονούμενο στοιχείο, η οποία μεταφράζεται σε μεταβολή στη χωρητικότητα. Ο βαθμός αυτής της μεταβολής στη χωρητικότητα είναι ανάλογος της γωνιακής ταχύτητας της συσκευής σύμφωνα με την αρχή λειτουργίας του. Στη συνέχεια η χωρητικότητα μετατρέπεται σε παλμό τάσης, ενισχύεται και ψηφιοποιείται (οι τιμές του παρουσιάζονται ως g_x, g_y, g_z στο APM 2.8). Τα γυροσκόπια είναι ανθεκτικά στις δονήσεις και δεν επηρεάζονται από τη μεταλλική άτρακτο του σκάφους συμπληρώνοντας έτσι τη λειτουργία των επιταχυνσιομέτρων.



Εικόνα 3-15 Μηχανικό γυροσκόπιο σε περιστροφή[30]

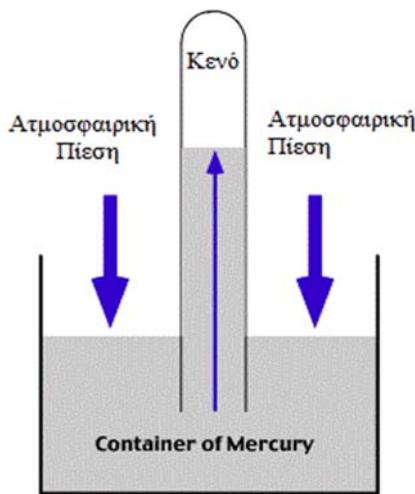
3.6.3 Μαγνητόμετρο

Το μαγνητόμετρο (Magnetometer) είναι ουσιαστικά μια πυξίδα σε ένα μικροσκοπικό ηλεκτρομηχανικό περίβλημα. Χρησιμοποιούνται είτε για να μετρήσουν τη μαγνήτιση ενός υλικού, για να προσδιορίσουν αν είναι μαγνητικά, είτε για να μετρήσουν την ένταση και συχνά τη κατεύθυνση του μαγνητικού πεδίου σε ένα σημείο στο χώρο. Ως αισθητήρας στο αεροσκάφος το μαγνητόμετρο τριών αξόνων μετρά την ένταση του μαγνητικού πεδίου κατά μήκος των τριών ευκλείδιων αξόνων, χρησιμοποιώντας τρείς λωρίδες μαγνητοευαίσθητου υλικού, ένα είδος υλικού του οποίου η αντίσταση αλλάζει ανάλογα με την ένταση του μαγνητικού πεδίου. Αυτές οι μετρήσεις στους τρείς άξονες συγκρίνονται με το γνωστό διάνυσμα του γήινου μαγνητικού πεδίου, οδηγώντας στον υπολογισμό του προσανατολισμού. Βοηθά στη λειτουργία του επιταχυνσιομέτρου παράγοντας τιμές πορείας και προσανατολισμού (m_x, m_y, m_z στο APM 2.8), ιδιαιτέρως στη λειτουργία ελέγχου yaw στην οποία το τελευταίο υστερεί.

Εμείς χρησιμοποιούμε πυξίδα εξωτερική του ελεγκτή πτήσης, που βρίσκεται μαζί με το GPS σε υπερυψωμένη θέση για να αποφύγουμε τις ηλεκτρομαγνητικές παρεμβολές που είναι αναπόφευκτες εξαιτίας των υψηλών ρευμάτων που ρέουν στον ελεγκτή πτήσης και στα γειτονικά ηλεκτρονικά. Η πυξίδα «αισθάνεται» συνεχώς το βόρειο μαγνητικό πόλο. Πριν όμως τη χρησιμοποιήσουμε στο τετρακόπτερο για να μας παρέχει ακριβείς μετρήσεις θέσης και προσανατολισμού θα πρέπει να τη βαθμονομήσουμε. Η διαδικασία βαθμονόμησης έγινε με τη βοήθεια του προγράμματος Mission Planner και θα περιγραφεί στο επόμενο κεφάλαιο.

3.6.4 Βαρόμετρο

Το βαρόμετρο (Barometer) είναι όργανο μέτρησης ατμοσφαιρικής πίεσης και κατ' επέκταση προσφέρει ικανοποιητικές προσεγγίσεις για τη θερμοκρασία του περιβάλλοντος. Η ατμοσφαιρική πίεση είναι μεγαλύτερη κατά τη διάρκεια της ημέρας, κατά τη διάρκεια καλών καιρικών συνθηκών και σε μικρότερα υψόμετρα, γι' αυτό το λόγο ασκεί μεγαλύτερη δύναμη στο υγρό που εναποτίθεται στο δοχείο του οργάνου (Εικόνα 3-16), σπρώχνοντας το υγρό πιο ψηλά στον αεροστεγώς σφραγισμένο σωλήνα. Στη κορυφή του σωλήνα υπάρχει κενό (ή καλύτερα δεν υπάρχει τίποτα). Το υγρό στο δοχείο είναι συνήθως μερκούριο, εξαιτίας της μεγάλης πυκνότητας του. Έτσι το ύψος του σωλήνα δύναται να είναι μικρό. Συνεπώς, σε συνθήκες υψηλής ατμοσφαιρικής πίεσης η στάθμη του υγρού στο σωλήνα αυξάνεται και σε συνθήκες χαμηλής ατμοσφαιρικής πίεσης η στάθμη του υγρού μειώνεται. Η στάθμη μπορεί να αυξηθεί με ακρίβεια cm.



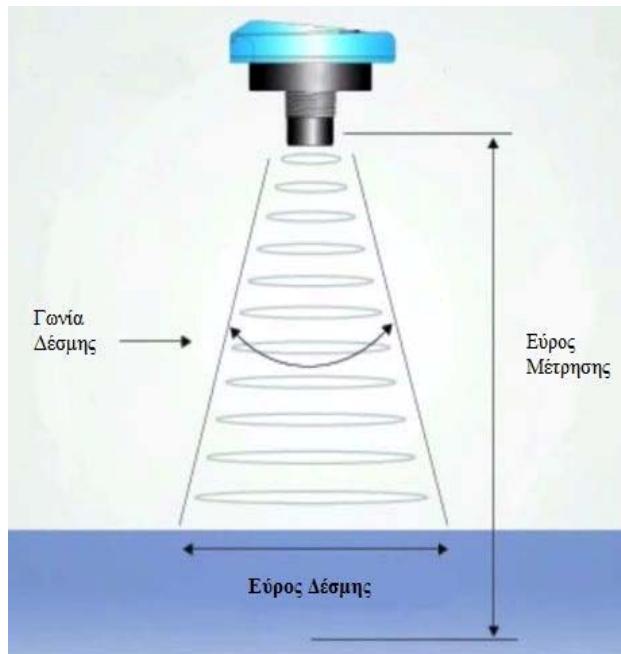
Εικόνα 3-16 Τυπικό Βαρόμετρο[31]

Ως αισθητήρας σμικρύνεται και αυτό σε μια μικροηλεκτρομηχανική συσκευή που επιτελεί το ίδιο έργο με το κλασικό βαρόμετρο! Ως παράδειγμα χρησιμότητας του σ' ένα μη επανδρωμένο αεροσκάφος, αναφέρεται ότι επιτρέπει στον πιλότο να μην ανησυχεί συνεχώς για τη ρύθμιση του υψομέτρου του οχήματος, εφόσον έχει αποκτήσει ένα επιθυμητό ύψος. Ας σημειωθεί όμως ότι επειδή η μεταβολή στην ατμοσφαιρική πίεση είναι σχεδόν αμελητέα για μικρά υψόμετρα πτήσης, το βαρόμετρο αποδεικνύεται αναξιόπιστο σε τέτοια ύψη, γι' αυτό χρησιμοποιείται πολλές φορές σε συνδυασμό με αισθητήρα υπερήχων, ο οποίος συμπληρώνει το ρόλο του σε χαμηλά υψόμετρα.

3.6.5 Αισθητήρας Υπερήχων

Οι αισθητήρες υπερήχων (Ultrasonic Sensor) είναι μορφομετατροπείς που μετατρέπουν υπερήχους σε ηλεκτρικά σήματα και αντίστροφα. Τοποθετούνται σε ένα μη επανδρωμένο όχημα κοιτώντας το έδαφος. Εκπέμπουν υπερηχητικά κύματα και λαμβάνουν την ανάκλαση (echo) από την επιφάνεια κάτωθεν τους. Μπορούν να μετρήσουν την απόσταση από το έδαφος, λαμβάνοντας υπόψη τη γωνία της δέσμης που εκπέμπουν και το χρονικό διάστημα που απαιτείται για να λάβουν την ανάκλαση σε μια χρονική στιγμή. Λαμβάνοντας τον αντίλαλο, τον μετατρέπουν σε ηλεκτρικό σήμα

το οποίο ύστερα οδηγείται σε μια συσκευή επεξεργασίας σήματος και τελικά λαμβάνεται από τον ελεγκτή πτήσης που χρησιμοποιεί τη μέτρηση για ρύθμιση του υψομέτρου του αεροσκάφους.



Εικόνα 3-17 Τα βασικότερα χαρακτηριστικά ενός αισθητήρα υπερήχων[32]

Πολλές φορές η «όραση» του μπορεί να μπλοκάρεται από εμπόδια, και ορισμένοι κατασκευαστές εκπέμπουν εκτός από το σήμα υπερηχών μια αρκετά πιο λεπτή δέσμη, με την ελπίδα αυτή να διαπεράσει τα πλευρικά εμπόδια, και να παρέχει πιο οξιόπιστη καταμέτρηση της απόστασης από το έδαφος. Η εμβέλεια μέτρησης τους είναι συνήθως μικρή, περιοριζόμενη σε μερικά μόλις μέτρα για τύπου UAV (συνήθως $<10\text{m}$), αφού η δέσμη ακρίβειας τους αυξάνεται σημαντικά με την απόσταση, όπως φαίνεται και στην Εικόνα 3-17. Σε συνδυασμό όμως με το βαρόμετρο αλληλοεξουδετερώνονται τα μειονεκτήματα τους.

3.7 Μονάδα GPS Δέκτη

Το παγκόσμιο σύστημα εντοπισμού θέσης είναι γνωστό ως “GPS” (Global Positioning System). Το GPS είναι ουσιαστικά ένα σύστημα πλοήγησης που βασίζεται σε σήματα που εκπέμπονται από ένα δίκτυο δορυφόρων που βρίσκονται σε διαφορετικές τροχιές γύρω από τη γη. Οι δορυφόροι κινούνται διαρκώς, κάνοντας δύο πλήρεις περιφορές γύρω από τη γη σε λιγότερο από 24 ώρες. Το εν χρήσει [33]GPS δίκτυο εκπέμπει σε δύο συχνότητες, από τις οποίες η μία χρησιμοποιείται μόνο για στρατιωτικούς σκοπούς, ενώ η δεύτερη, που είναι ανοιχτή και για κοινή χρήση, διαθέτει μειωμένη ανάλυση.

Ο κάθε δορυφόρος ζυγίζει λιγότερο από 1 τόνο και το πλάτος του δεν ξεπερνά τα 5 μέτρα με τις ηλιακές κυψέλες σε ανάταση. Οι GPS δορυφόροι έχουν μέση διάρκεια ζωής 10 χρόνια. Η αντικατάσταση τους γίνεται κανονικά εδώ και χρόνια με νέους δορυφόρους. Οι τροχιές των δορυφόρων περνούν περίπου 60 μοίρες βόρεια μέχρι 60 μοίρες νότια, παρέχοντας επαρκές σήμα για τον υπολογισμό της θέσης σε οποιοδήποτε σημείο πάνω στη γη και οποιαδήποτε στιγμή. Η τροχιά των δορυφόρων στους πόλους της γης ξεπερνά τις 60° , με αποτέλεσμα να περιορίζεται η ακρίβεια του στίγματος.

Ο κάθε δορυφόρος μεταδίδει κατ' επανάληψην πληροφορίες για την ώρα και τη θέση του, επιτρέποντας σε έναν κατάλληλο δέκτη GPS, που το πιο πιθανό είναι να βρίσκεται υπό την κάλυψη πολλαπλών δορυφόρων, να υπολογίσει με τη μέθοδο του τριπλευρισμού (trilateration) την εκάστοτε

δική του θέση. Η χρονομετρική ακρίβεια είναι κρίσιμης σημασίας έτσι κάθε GPS δορυφόρος είναι εφοδιασμένος με τέσσερα ατομικά ρολόγια, παρέχοντας κυριολεκτικά αστρονομική ακρίβεια. Αρχικά ο GPS δέκτης εντοπίζει τους ορατούς δορυφόρους στον ουράνιο θώλο για να τους παρακολουθήσει. Οι GPS δέκτες αναλύουν το υψηλής συχνότητας και χαμηλής ισχύος ραδιοσήμα του δορυφόρου, αποκωδικοποιώντας τις πληροφορίες θέσης και χρόνου που διαθέτουν, αλλά επιπρόσθετα υπολογίζουν και την ποιότητα του λαμβανόμενου σήματος (RSSI). Πρώτον η ώρα του δέκτη GPS συγχρονίζεται με την ώρα του δορυφόρου. Δεύτερον ο δέκτης γνωρίζοντας το χρόνο (τον υπολογίζει από τη διαφορά στη δική του ώρα και την ώρα του δορυφόρου) και τη θέση ενός δορυφόρου καθώς και την ταχύτητα μετάδοσης του σήματος, που είναι η ταχύτητα μετάδοσης του φωτός, μπορεί να υπολογίζει την απόσταση (ακτίνα) που απέχει από τον δορυφόρο, με τον τύπο: $S = t \times C$, όπου C η ταχύτητα του φωτός, t ο χρόνος που έκανε το σήμα να φτάσει απ' το δορυφόρο στο δέκτη και S η απόσταση. Με έναν όμως δορυφόρο δεν μπορεί να υπολογίζει τη θέση του, απαιτούνται το λιγότερο 3 δορυφόροι. Αυτή είναι συνοπτικά η μέθοδος του τριπλευρισμού η οποία, σε αντίθεση με τη μέθοδο του τριγωνισμού η οποία μετρά γωνίες, μπορεί να μετρά αποστάσεις. Η πληροφορία του στίγματος εμφανίζεται στην οθόνη του GPS δέκτη, εκφρασμένη σε γεωγραφικές συντεταγμένες. Επίσης μιλάμε για στίγμα δύο διαστάσεων (2D Fix) όταν η συσκευή «καλύπτεται» από μόνο 3 δορυφόρους, ενώ για στίγμα τριών διαστάσεων (3D Fix) όταν υπάρχει κάλυψη από 4 τουλάχιστον δορυφόρους.

Εμείς χρησιμοποιούμε τη μονάδα δέκτη GPSNEO-6M της Ublox η οποία επικοινωνεί με τον ελεγκτή πτήσης μέσω μιας διεπαφής UART. Η μονάδα GPS δέχεται τάση 5V από το APM, αλλά λειτουργεί σε τάση 3.3V, αφού περάσει μέσα από έναν LDO ρυθμιστή τάσης. Επίσης η συσκευασία διαθέτει και πυξίδα την οποία και προτιμήσαμε από την εσωτερική πυξίδα του ελεγκτή πτήσης. Προτού όμως τη συνδέσουμε έπρεπε να απενεργοποιήσουμε την εσωτερική πυξίδα και αυτό το κάναμε αφαιρώντας τον βραχυκυκλωτήρα “MAG”, αμέσως πάνω από τη δεξιά θέση της υποδοχής GPS. Η εξωτερική μονάδα GPS επικοινωνεί με τον ελεγκτή πτήσης μέσω αφιερωμένης διεπαφής I^2C . Εγκαθιστούμε τη βαθμίδα GPS+πυξίδα σε μια υπερυψωμένη θέση στο τετρακόπτερο προσέχοντας το βελάκι που βρίσκεται εσωτερικά στο θόλο της συσκευασίας GPS να κοιτάει στην ίδια κατεύθυνση με το βελάκι του ελεγκτή πτήσης. Το αποτέλεσμα φαίνεται στην Εικόνα 3-18.

Ίσως θα παρατηρήσατε το μικρό φύλλο από αλουμινόχαρτο που έχουμε τοποθετήσει στη βάση του. Αυτό έγινε επειδή συχνά αντιμετωπίζαμε προβλήματα με τη λήψη του GPS και τελικά καταλάβαμε ότι οφείλεται σε κακές συνθήκες γείωσης. Οδηγηθήκαμε σε αυτό το συμπέρασμα επειδή όταν συνδέαμε τον ελεγκτή πτήσης με USB στον H/Y το GPS ενεργοποιούνταν σχεδόν αμέσως (αναβόσθηνε το μπλέ ενδεικτικό led στο άνω μέρος), αλλά όταν τροφοδοτούσαμε το APM μονάχα με την μπαταρία, το GPS πολλές φορές δεν λάμβανε ποτέ σήμα και άλλοτε απλώς καθυστερούσε πολύ. Καταλάβαμε λοιπόν ότι πιο πιθανή αιτία είναι η γείωση, εφόσον η γραμμή γείωσης μέσω του USB του υπολογιστή οδηγείται σε αυτή του σπιτιού η οποία προσφέρει φυσικά τις καλύτερες συνθήκες grounding, σε αντίθεση με την μπαταρία. Βέβαια το πρόβλημα αυτό μετριαζόταν σε εξωτερικούς χώρους, αλλά και πάλι χωρίς εμφανή διαφορά και ικανοποιητική απόδοση. Σε εσωτερικούς χώρους το GPS δεν λειτουργεί και κανονικά θα πρέπει να απενεργοποιείται. Εάν λειτουργήσει (υπάρχει κάποιο fix) αυτό θα οφείλεται σε λήψη σήματος μέσω πολλαπλών διαδρομών που δεν είναι αξιόπιστο. Με αυτή λοιπόν την τροποποίηση της προσθήκης αλουμινόχαρτου αυξήθηκε (μάλλον τριπλασιάστηκε) το ενεργό groundplane της συσκευής και από τότε και στο εξής η λήψη σήματος συμβαίνει σχεδόν ακαριαία με τη τροφοδοσία. Σε γενικές γραμμές διαπιστώσαμε ότι για δέκτες GPS τέτοιου μεγέθους groundplane εμβαδού 5cm x 5cm, ή μεγαλύτερου είναι ιδανικό.

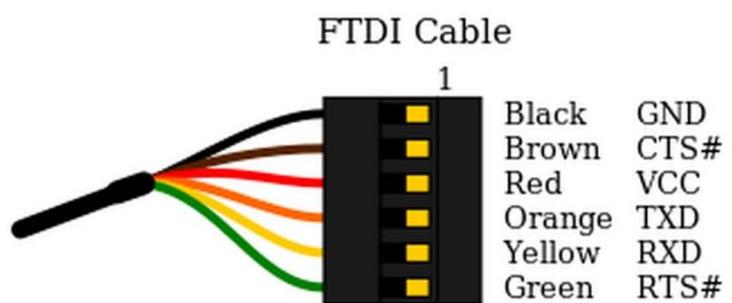
Αφού συνδέσαμε το GPS με τον ελεγκτή πτήσης APM 2.8 έπρεπε και να το προσαρμόσουμε ώστε οι δύο συσκευές να γίνουν συμβατές (διότι δεν είναι από προεπιλογή) και να λειτουργούν σε αρμονία. Όταν το πετύχουμε αυτό η μονάδα GPS θα τροφοδοτεί το APM με γεωγραφικές



Εικόνα 3-18 Μονάδα GPS δέκτη +πυξίδα υπερυψωμένη με ειδικό στύλο στο τετρακόπτερο

συντεταγμένες ακριβείας από GPS, ακριβής ώρα κα., τα οποία μπορούμε να τα βλέπουμε εμείς ως διαχειριστές με τη ζεύξης τηλεμετρίας στο GCS. Η διαδικασία λοιπόν που ακολουθήσαμε για να επαναπρογραμματίσουμε το firmware του GPS δέκτη είναι η εξής[34][35]:

1. Κατεβάζουμε το πρόγραμμα u-Center της uBlox.
2. Κατεβάζουμε το πηγαίο hex αρχείο 3DR-Ublox.txt που περιέχει τις ρυθμίσεις που πρέπει να εφαρμοστούν.
3. Συνδέουμε το δέκτη GPS με τον H/Y χρησιμοποιώντας ένα FTDI καλώδιο. Η μονάδα GPS χρησιμοποιεί σήματα τύπου TTL και το FTDI καλώδιο είναι απαραίτητο για τη κλιμάκωση τους σε μορφή σειριακών δεδομένων μορφής RS-232. Έπρεπε να προσέξουμε πώς θα γίνει η σύνδεση μεταξύ των καλωδίων της μονάδας GPS και του FTDI καλωδίου. Γι' αυτό το σκοπό χρησιμοποιήσαμε ως αναφορά μας την παρακάτω εικόνα που δείχνει τη διαρρύθμιση των ακροδεκτών εξόδου (pinout) και των δύο διατάξεων. Θυμόμαστε πάντα Rx->Tx και Tx->Rx.



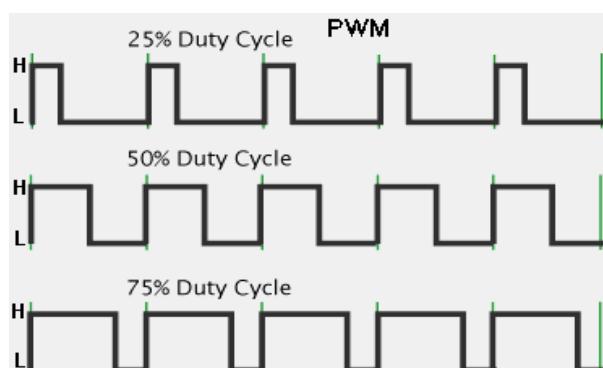
Εικόνα 3-19 Σύνδεση μονάδας GPS Neo-6M με FTDI. Προβάλλονται τα pinouts της κάθε συσκευής[36]

4. Ανοίγουμε το πρόγραμμα u-Center, πατάμε το πλήκτρο με το μαγικό ραβδί και επιλέγουμε την κατάλληλη σειριακή θύρα επικοινωνίας θέτοντας την ταχύτητα σε 9600 baud rate.
5. Φορτώνουμε το αρχείο ρυθμίσεων πηγαίνοντας Tools->βρίσκουμε το path του αρχείου 3DR-Ublox.txt που το αποθηκεύσαμε, κάνουμε κλικ στο “Store configuration into BBR/Flash (non-volatile memory)” και τελικά πατάμε “File >> GPS”.
6. Εάν η ενημέρωση πέτυχε πρέπει να δούμε κάτω δεξιά στο παράθυρο του u-Center τη ταχύτητα επικοινωνίας να άλλαξε σε 38,400 baud. Εάν όχι, τότε έχει συμβεί κάποιο λάθος.
7. Εάν είναι όλα καλά επιλέγουμε από το κεντρικό κατάλογο το Receiver->Action->Save Config για να αποθηκευτούν οι ρυθμίσεις στη μνήμη της συσκευής. Ο προγραμματισμός της βαθμίδας έχει ολοκληρωθεί και είναι έτοιμη για λειτουργία με τον ελεγκτή πτήσης APM.

3.8 Ηλεκτρονικοί Ελεγκτές Ταχύτητας

Ο ηλεκτρονικός ελεγκτής ταχύτητας (Electronic Speed Controllers–ESC's) είναι ένα ηλεκτρονικό κύκλωμα ισχύος που χρησιμοποιείται για να μεταβάλει την ταχύτητα και την κατεύθυνση των κινητήρων. Τα ESC's χρησιμοποιούνται συχνά σε ηλεκτρικώς τροφοδοτούμενα ραδιοελεγχόμενα μοντέλα, όπως τα UAV drones, με την πλειονότητα τους να προσφέρουν τριφασική παροχή ισχύος για τους τριφασικούς κινητήρες χωρίς βούρτσα (brushless) που οδηγούν. Υπάρχουν διάφορες παραλλαγές. Μπορεί να χρησιμοποιείται ένα μόνο ESC το οποίο συνδέεται κατευθείαν με τον ραδιοδέκτη, ή πολλαπλά ESC's, ένα για κάθε κινητήρα που χρησιμοποιείται στη διάταξη, όπως κάνουμε εμείς. Στην πρώτη περίπτωση το ESC είναι υπεύθυνο εξολοκλήρου για την επεξεργασία του λαμβανόμενου σήματος και έπειτα την τροφοδότηση του στον κινητήρα, ενώ στη δεύτερη περίπτωση τα σήματα του ραδιοδέκτη οδηγούνται πρώτα στον ελεγκτή πτήσης, ο οποίος κατανέμει το εμπλουτισμένο PWM σήμα στα ESC και τα τελευταία τροφοδοτούν τους DC κινητήρες.

Ανεξαρτήτως του είδους ένα ESC μεταφράζει το λαμβανόμενο σήμα πληροφορίας σε ρυθμό μεταγωγής των τρανζίστορ επίδρασης πεδίου που διαθέτει (FET). Τα περισσότερα πλέον ESC δομούνται με FET's (όπως και η πλειονότητα των ηλεκτρονικών κυκλωμάτων) αφού είναι τα πιο αποδοτικά. Ο ρυθμός μεταγωγής των τρανζίστορ ποικίλλει αλλά σε γενικές γραμμές όσο υψηλότερος είναι, τόσο λιγότερη ποσότητα ισχύος σπαταλείται ως θερμότητα. Ρυθμοί μεταγωγής μεταξύ 1000 και 5000Hz είναι συχνοί και αποδεκτοί για τις περισσότερες εφαρμογές. Η γρήγορη μεταγωγή των τρανζίστορ παράγει έξοδο, τριφασικού AC ρεύματος με την τεχνική PWM. Η PWM είναι τεχνική μετατροπής ψηφιακού σήματος σε αναλογικό, και παίζει πρωταρχικό ρόλο στο κόσμο των σύγχρονων ηλεκτρονικών. Τα σήματα διαμορφωμένα κατά εύρος παλμού μεταβάλλουν τον κύκλο καθήκοντος (Duty Cycle) του ρεύματος που παράγουν για τους κινητήρες, ανάλογα με το Duty Cycle του PWM σήματος πληροφορίας που λαμβάνουν από τον ελεγκτή πτήσης. Όσο μεγαλύτερο είναι το Duty Cycle τόσο πιο γρήγορα περιστρέφεται ο κινητήρας και τόσο περισσότερη ισχύς σπαταλείται. Η επόμενη εικόνα δείχνει σήματα PWM με κύκλο καθήκοντος 25%, 50% και 75%.



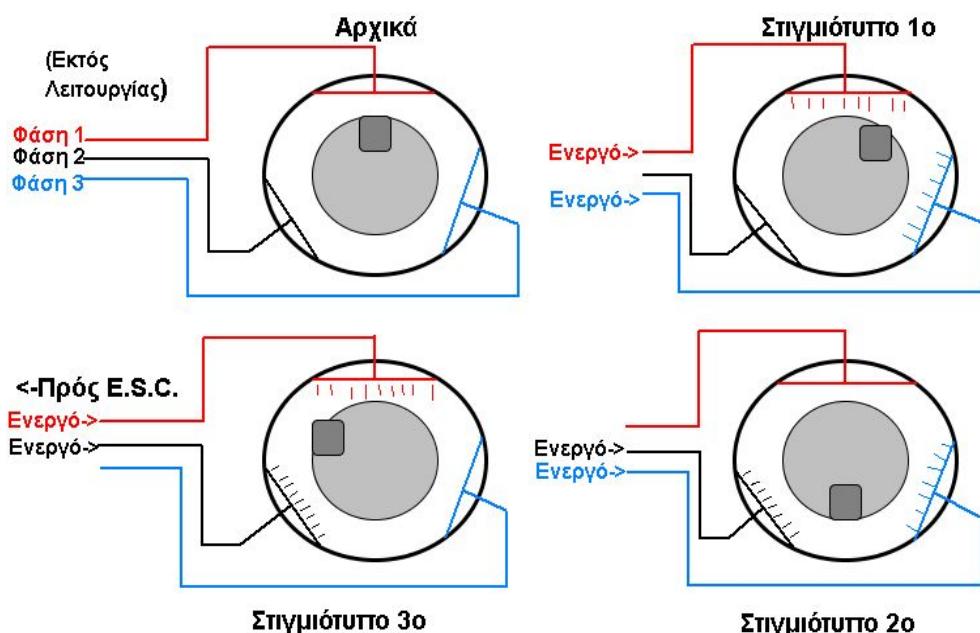
Εικόνα 3-20 Παλμοί διαμορφωμένοι κατά Εύρος (τεχνική PWM)[37]

Ο κινητήρας ως ηλεκτρομηχανική συσκευή δεν μπορεί να ανταποκριθεί τόσο γρήγορα μεταξύ της κάθε μετάπτωσης της τάσης μεταξύ υψηλής και χαμηλής. Τελικά αυτό που αντιλαμβάνεται είναι μια σταθερή εξομαλυμένη DC τάση, που βρίσκεται μεταξύ του υψηλού και χαμηλού επιπέδου της τάσης της πηγής τροφοδοσίας και το επίπεδο της εξαρτάται από τον κύκλο καθήκοντος του PWM παλμού. Όσο αυξάνεται το duty cycle τόσο περισσότερη ισχύς τροφοδοτείται στο κινητήρα και τελικά περιστρέφεται πιο γρήγορα. Επίσης όσο πιο γρήγορα περιστρέφεται τόσο περισσότερη ισχύς καταναλώνει. Το σήμα που λαμβάνει το ESC από τον ελεγκτή πτήσης είναι επίσης διαμορφωμένο κατά PWM (πολύ μικρότερης ισχύος με το σήμα του κινητήρα).

Ο μικροελεγκτής του ESC είναι υπεύθυνος για την αναγνώριση του σήματος που λαμβάνει από τον ελεγκτή πτήσης και στηριζόμενος σε αυτό, την τροφοδότηση του κινητήρα με AC τριφασικό ρεύμα από την μπαταρία. Το υλικολογισμικό (firmware) του μικροελεγκτή του ESC είναι άλλες φορές ιδιόκτητο προεγκατεστημένο από τον κατασκευαστή και δεν επιτρέπει εγκατάσταση ενημερώσεων, ενώ άλλες φορές έχει αναπτυχθεί από την κοινότητα ανοιχτού κώδικα που το αναβαθμίζει και ενημερώνει σε τακτικά χρονικά διαστήματα. Το πρώτο είδος δεν προτιμάται πλέον ιδιαίτερα στο κόσμο των R/C, καθώς μόνο με την παρέμβαση με κολλητήρι κάνοντας αλλαγές στις συνδέσεις του υλικού, μπορεί να επιτευχθεί η ενημέρωση του firmware με τη σύνδεση μιας συσκευής προγραμματισμού. Το δεύτερο είδος προτιμάται ιδιαίτερα και μάρκες όπως η SimonK, η AfroESC και η Turnigy Plush (που χρησιμοποιείται και από εμάς) είναι πλέον πολύ δημοφιλείς.

Οι ηλεκτρονικοί ελεγκτές ταχύτητας τροφοδοτούνται απευθείας από την μπαταρία, τη βασική πηγή ενέργειας του συστήματος. Θεωρούνται λοιπόν συσκευές υψηλής ισχύος, εφόσον οδηγούν τις κατεξοχήν συσκευές υψηλής ισχύος, που δεν είναι άλλες από τους κινητήρες. Λαμβάνουν λοιπόν DC τάση και με έναν πολύ απλό και έξυπνο τρόπο τροφοδοτούν τους τριφασικούς κινητήρες. Θα εξηγήσουμε στη συνέχεια και με τη βοήθεια της παρακάτω εικόνας 3-21 πως υλοποιείται αυτή η διαδικασία.

Στην Εικόνα 3-21 βλέπουμε ένα τριφασικό brushless κινητήρα με 3 μαγνητικούς πόλους, που ουσιαστικά είναι τρείς περιελίξεις πηνιοσύρματος. Στο κέντρο βρίσκεται ένας μόνιμος μαγνήτης που παριστάνεται με μικρό τετράγωνο σχήμα. Οι δικοί μας κινητήρες διαθέτουν 18 πόλους και γενικά οι περισσότεροι πλέον διαθέτουν πολλούς, αλλά η αρχή λειτουργίας τους υφίσταται γενικότερα για κινητήρες με $n \times 3$ πόλους. Απεικονίζεται η χρονική εξέλιξη της διαδικασίας, αρχικά $t = 0$,



Εικόνα 3-21 Αρχή λειτουργίας ESC. Τρόπος που ένα ESC τροφοδοτεί έναν τριφασικό brushless κινητήρα

πάνω αριστερά, για $t = 1$ πάνω δεξιά, για $t = 2$ κάτω δεξιά και για $t = 3$ κάτω αριστερά. Οι γραμμές με μπλέ, μαύρο και κόκκινο αφορούν μια από τις τρείς φάσεις του κινητήρα. [38] Κάθε φάση τροφοδοτεί 1 πόλο και κάθε χρονική στιγμή μόνο δύο από τις 3 φάσεις είναι ενεργές.

1. Αρχικά: η διάταξη είναι εκτός λειτουργίας.
2. Στιγμιότυπο 1^o: Ενεργοποιούνται οι δύο από τις 3 φάσεις συγκεκριμένα η κόκκινη και η μπλέ, άρα οι πόλοι ενεργοποιούνται (ένας βόρειος και ένας νότιος) (δηλαδή αποκτούν μαγνητικές ιδιότητες όταν ρέει από αυτούς ηλεκτρικό ρεύμα) και μαγνητίζουν τον μόνιμο μαγνήτη, ο οποίος κατά συνέπεια παίρνει θέση μεταξύ αυτών.
3. Στιγμιότυπο 2^o: Απενεργοποιείται η κόκκινη φάση και ενεργοποιείται η μαύρη. Άρα ενεργοποιούνται οι ηλεκτρομαγνήτες που ελέγχονται από τις νέες φάσεις και ο μόνιμος μαγνήτης σπεύδει αναμεταξύ τους.
4. Στιγμιότυπο 3^o: Απενεργοποιείται η μπλέ φάση και ενεργοποιείται η κόκκινη. Και πάλι ο μόνιμος μαγνήτης μετακινείται μεταξύ των δύο νέων φάσεων.

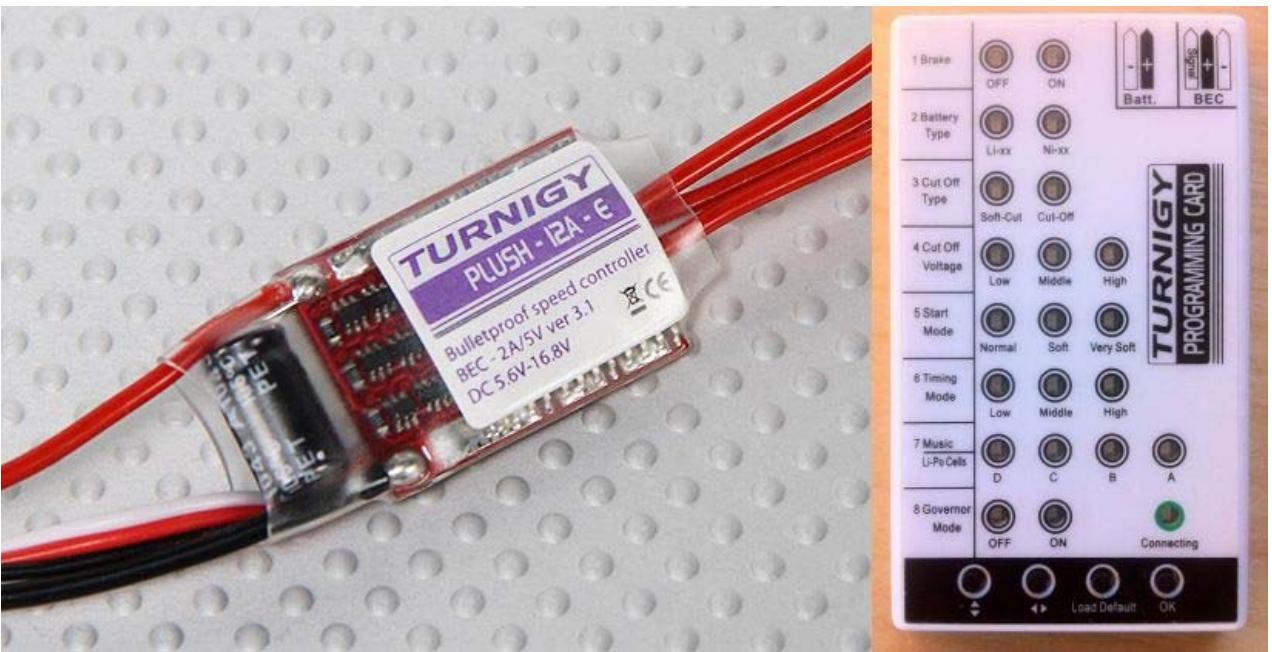
Το ESC επαναλαμβάνει αυτό τον κύκλο πολλές φορές το δευτερόλεπτο (όπως αναφέραμε, ανάλογα με το ρυθμό μεταγωγής του ESC και τον κύκλο καθήκοντος του PWM παλμού) και έτσι ο οπλισμός του κινητήρα περιστρέφεται.

Τα περισσότερα μοντέρνα ESC εκτός από τον μικροελεγκτή για την παραγωγή του PWM παλμού, περιλαμβάνουν και ένα επιπλέον κύκλωμα που είναι γνωστό ως Battery Eliminated Circuit (BEC). Πρόκειται απλώς για έναν ρυθμιστή τάσης για την τροφοδοσία τυχόν άλλων ηλεκτρονικών συσκευών και προσφέρει εξομαλυμένη τάση συνήθως στα 5V. Για παράδειγμα στη δική μας περίπτωση θα μπορούσαμε να τροφοδοτήσουμε τον ελεγκτή πτήσης από το ESC, αλλά τελικά προτιμήσαμε το APM power module για την τροφοδοσία, εφόσον παρέχει και αισθητήρια όργανα μέτρησης της τάσης και του ρεύματος που απορροφάται από τη μπαταρία κάθε χρονική στιγμή. Η ονομασία Battery Eliminated Circuit υπονοεί ότι εξαλείφει την ανάγκη ύπαρξης μπαταρίας για τα άλλα ηλεκτρονικά κυκλώματα χαμηλής ισχύος, εφόσον θα τροφοδοτούνται πλέον όλα από το BEC. Υπάρχουν διάφορα είδη BEC. Τα τυπικά BEC είναι γραμμικοί ρυθμιστές τάσης, δηλαδή από μια υψηλότερη τάση, πχ. 11.1V, εξάγουν μικρότερη τάση, πχ. 5V, και η υπόλοιπη ισχύς που δεν χρειάζεται σπαταλάται όλη σε έναν αντιστάτη, αυξάνοντας τη θερμική ενέργεια της διάταξης. Προφανώς αυτό δεν είναι καθόλου αποδοτικό, γι' αυτό προτιμούνται κυρίως παραλλαγές BEC τα SBEC και UBEC τα οποία είναι μεταγωγικοί (switching) ρυθμιστές τάσης. Προσφέρουν διακοπτόμενη τροφοδοσία. Με αυτό τον τρόπο όταν το σήμα είναι ON, προσφέρεται ρεύμα (άρα και ενέργεια), ενώ όταν το σήμα είναι OFF το ρεύμα είναι 0, άρα: Ισχύς = Τάση × Ρεύμα = Τάση πχ. 5(V) × 0(A) = 0 (Watt)! Συνεπώς τα SBEC και UBEC είναι λιγότερο ενεργοβόρα και αυτός είναι ο ύστατος στόχος.

Σε γενικές γραμμές όμως τα ESC θεωρούνται πολύ αποδοτικές συσκευές. Μάλιστα μερικά διαθέτουν τη δυνατότητα αναγεννητικής πέδησης (Regenerative Braking), με την οποία όποτε ο κινητήρας επιβραδύνει, η τάση – αντιηλεκτρεγερτική δύναμη – που παράγεται από την πέδηση οδηγείται από το ESC πίσω στη μπαταρία του οχήματος φορτίζοντας τη. Αυτό το χαρακτηριστικό εμφανίζεται κυρίως στα επιβατηγά οχήματα, αλλά πρόσφατα έχει αρχίσει σταδιακά να κάνει την εμφάνιση του και στα ραδιοκατευθυνόμενα μοντέλα.

Όπως μπορούμε να δούμε και στο μπλοκ διάγραμμα του τετρακόπτερου (**Error! Reference source not found.**) τα τέσσερα δικά μας ESC συνδέονται κατευθείαν με την πλακέτα διανομής ενέργειας που λαμβάνει ισχύ από την μπαταρία. Έπρεπε πριν εγκαταστήσουμε τα Turnigy Plush ESC's να τα ρυθμίσουμε, ώστε να ανταποκρίνονται στις ανάγκες της εφαρμογής. Αυτό το κάναμε χρησιμοποιώντας την κάρτα προγραμματισμού Turnigy BESC Programming Card. Στην παρακάτω

εικόνα φαίνεται το ESC Turnigy Plush 12A-E και η κάρτα προγραμματισμού του Firmware που χρησιμοποιήσαμε.



Εικόνα 3-22 ESC Turnigy Plush-12A-E & κάρτα προγραμματισμού Turnigy BESC Programming Card[39]

Συνδέσαμε λοιπόν κάθε ESC πρώτα με την κάρτα προγραμματισμού και έπειτα με την μπαταρία από το άλλο άκρο και αφού λάμψουν οι προκαθορισμένες ρυθμίσεις του ESC στην κάρτα κάναμε τις ακόλουθες μετατροπές.

- Brake -> OFF. Εάν είναι ενεργοποιημένο το ESC θα φρενάρει τον κινητήρα όταν το throttle είναι στο 0, εάν είναι OFF θα τον αφήσει να επιβραδύνει μόνος του λόγω τριβής (windmill).
- Battery Type -> NI-XX. Παρόλο που χρησιμοποιούμε μπαταρία χημείας LiPo επιλέγουμε NI-XX εδώ επειδή θέλουμε την τάση αποκοπής (voltage cutoff) να είναι στο 0% (δηλαδή απενεργοποιημένη).
- Cut Off Voltage -> Low. Τάση αποκοπής σημαίνει ότι κάτω από εκείνο το ποσοστό τάσης της μπαταρίας το ESC θα διακόψει τη λειτουργία του κινητήρα. Οι τιμές της τάσης αποκοπής είναι Low (0% voltage cutoff, δηλαδή απενεργοποιημένη), Medium (45% voltage cutoff), High (60% voltage cutoff) της αρχικής τάση τροφοδοσίας. Ο συνδυασμός των δύο προηγούμενων επιλογών είναι αυτό ακριβώς που θέλουμε.
- Cut off Type -> Soft Cut. Αυτή η παράμετρος δεν έχει σημασία στη περίπτωση όπως εξηγήθηκε παραπάνω, αλλά γενικώς σημαίνει ότι η επιβράδυνση του κινητήρα θα είναι σταδιακή, όταν η τάση μειωθεί πιο κάτω από την προκαθορισμένη.
- Start Mode -> Normal. Την πρώτη φορά που εφαρμόζεται ισχύς στους κινητήρες δεν υπάρχει κάποιος χρόνος αναμονής και οι κινητήρες ανταποκρίνονται ευθύνς αμέσως με τη μετακίνηση των sticks του ραδιοπομπού, ούτε νωρίτερα, ούτε αργότερα. Εδώ η επιλογή Soft Start ενδείκνυται για ελικόπτερα, όπου ο κύριος έλικας κατά την εκκίνηση περιστρέφεται αργά εξομοιώνοντας τον έλικα ενός πραγματικού ελικοπτέρου.
- Timing -> Medium. Θα μπορούσαμε εδώ να θέσουμε και High. Αυτή η παράμετρος είναι πολύ σημαντική και εξαρτάται από πολλούς παράγοντες, όπως τον αριθμό των πόλων του κινητήρα, την ανατροφοδότηση σήματος από τους κινητήρες στα ESC κα. Υψηλότερο timing είναι πιο ισχυρό, με την έννοια ότι κάνει τους κινητήρες πιο γρήγορα αποκρινόμενους, όμως χρησιμοποιεί περισσότερη ενέργεια. Μετά από διάφορα πειράματα αρκεστήκαμε στο Medium.

- Music -> D (default). Αναφέρεται στο μουσικό τόνο που παράγουν τα ESC όταν είναι συνδεδεμένα με τους κινητήρες και εφαρμοστεί η τροφοδοσία. Αδιάφορη παράμετρος...
- Governor Mode -> Off. Χρησιμοποιείται κυρίως για τα ελικόπτερα. Έχει να κάνει με τη διατήρηση σταθερής ταχύτητας πορείας σε διάφορες απρόοπτες συνθήκες, όπως έντονες ριπές ανέμου, ή ομίχλη κλπ.

Μετά τη ρύθμιση όλων των ESC'ς έπρεπε να γίνει η βαθμονόμηση τους, δηλαδή οι ελεγκτές ταχύτητας πρέπει να αντιληφθούν το μέγιστο και ελάχιστο επίπεδο του του καναλιού Throttle (ή οποιουδήποτε άλλου), ανάλογα με τον ραδιοπομπό που χρησιμοποιείται. Για τη βαθμονόμηση ακολουθήσαμε την παρακάτω διαδικασία για κάθε ESC:

1. Συνδέουμε το ESC με τον κινητήρα. Αρχικά με οποιοδήποτε τρόπο θέλουμε. Η σωστή σύνδεση θα εξηγηθεί στην επόμενη ενότητα με τους κινητήρες.
2. Συνδέουμε το ESC με τον ραδιοδέκτη Turnigy 9x.
3. Ενεργοποιούμε τον αντίστοιχο ραδιοπομπό, στον οποίο το Throttle Stick πρέπει να είναι στο maximum – μέγιστη τιμή.
4. Συνδέουμε τη μπαταρία με το ESC. Θα ακουστούν διάφοροι μουσικοί τόνοι. Όταν η μουσική σήμανση σταματήσει, φέρουμε το Throttle μοχλό στην ελάχιστη θέση και αναμένουμε τα beeps – αποδοχής από το ESC.
5. Όταν ακουστούν, η βαθμονόμηση ESC – Κινητήρα έχει ολοκληρωθεί με επιτυχία. Εάν δεν ακούστηκε κάποια μουσική νότα, τότε συνέβη κάποιο λάθος και η διαδικασία πρέπει να επαναληφθεί από την αρχή.
6. Απενεργοποίηση ραδιοπομπού.
7. Αφαίρεση μπαταρίας από το ESC.

3.9 Ηλεκτρικοί Κινητήρες χωρίς Ψήκτρες

Οι DC (οι γνώμεις διίστανται για το αν πρέπει να αναφέρονται ως DC, ή AC) κινητήρες χωρίς ψήκτρες (Brushless DC Motors – αποκαλούνται και BLDC κινητήρες) είναι σύγχρονοι κινητήρες, οι οποίοι τροφοδοτούνται από μια DC ηλεκτρική πηγή μέσω ενός αντιστροφέα (Inverter – μετατροπέας DC σε AC), ο οποίος παράγει AC ηλεκτρικό ρεύμα για να οδηγήσει τον κινητήρα. Με τον όρο AC ρεύμα δεν εννοούμε μια ημιτονοειδής κυματομορφή, αλλά ρεύμα χωρίς περιορισμό στο είδος της κυματομορφής. Σύγχρονοι είναι οι κινητήρες, που ο ρυθμός περιστροφής τους συγχρονίζεται με τη συχνότητα του τροφοδοτούμενου ρεύματος. Επιπλέον αισθητήρες και ενεργοποιητές μπορούν να μεταβάλλουν και να προσαρμόζουν την έξοδο του αντιστροφέα και τη συχνότητα του. Ο ρότορας ενός BLDC κινητήρα συνήθως συνίσταται από έναν μόνιμο μαγνήτη.

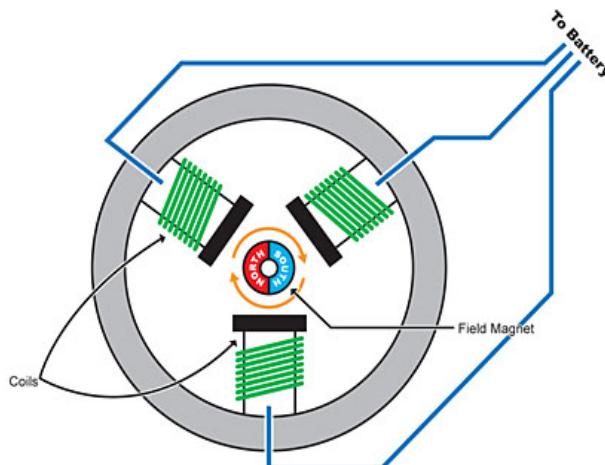
Όπως γίνεται αντιληπτό και από την ονομασία τους οι BLDC κινητήρες δε διαθέτουν βιούρτσες, ή αλλιώς ψήκτρες, σε αντίθεση με τους τυπικούς DC βηματικούς κινητήρες (stepper motor). Το γεγονός αυτό αποτελεί μεγάλο πλεονέκτημα, αφού τους καθιστά ελαφρύτερους, πιο ανθεκτικούς, μακράς διαρκείας επειδή εφαρμόζονται σε αυτούς λιγότερες δυνάμεις τριβής, ενώ διαθέτουν την ίδια ικανότητα ισχύος εξόδου, σε σχέση με τους βηματικούς κινητήρες. Επιπλέον οι ψήκτρες με την πάροδο του χρόνου κατατρίβονται και προκαλούν σπινθήρες (sparking). Γι' αυτό οι dc τυπικοί κινητήρες δεν συνίσταται να χρησιμοποιούνται σε εφαρμογές που απαιτούν μακροζωϊα και αξιοπιστία ανεξαρτήτως περιβαλλοντικών συνθηκών. Γενικώς οι BLDC προορίζονται για εμαρμογές με ομαλή κίνηση, εξομαλυμένη έξοδο, ή ταχύτητα. Επίσης κατά τη λειτουργία τους παράγουν μικρότερα επίπεδα θορύβου. Αντίθετα οι βηματικοί κινητήρες έχουν εκαποντάδες βήματα για κάθε πλήρη περιστροφή, γιατί θέλουν «βηματισμό» κατά μικρά βήματα για αυξημένη ακρίβεια. Αυτό καθιστά αναγκαία την ύπαρξη πολλών μαγνητικών πόλων. Γι' αυτό και οι βηματικοί κινητήρες

χρειάζονται περισσότερα τυλίγματα τα οποία δημιουργούν φυσικά ισχυρότερο μαγνητικό πεδίο και επομένως μεγαλύτερη ροπή για κάθε μονάδα ρεύματος. Αυτό το προτέρημα των τελευταίων όμως επιβαρύνεται από την αναπόφευκτα αυξημένη αντιηλεκτρεγερτική δύναμη (back EMF), η οποία δημιουργεί περισσότερη αντίσταση στη κίνηση του κινητήρα, μειώνοντας εντέλει την ταχύτητα περιστροφής. Αντιηλεκτρεγερτική δύναμη επάγεται φυσικά σε μικρότερο όμως βαθμό και στους BLDC κινητήρες.

Οι BLDC κινητήρες ανάλογα με τη θέση του περιστρεφόμενου στελέχους χωρίζονται σε outrunners και inrunners. Οι outrunners στριφογυρίζουν στο εξωτερικό τους (οπότε είναι φανερή η περιστροφή τους), σε αντίθεση με τους inrunner όπου ο περιστρεφόμενος πυρήνας περιέχεται μέσα στο κέλυφος του κινητήρα. Οι Outrunners μπορούν να αναπτύξουν μεγαλύτερη ροπή από αντίστοιχου μεγέθους inrunners, διότι στους outrunners οι ηλεκτρομαγνήτες απέχουν περισσότερο από το μαγνήτη και αυτό προσδίδει μεγαλύτερο χώρο μόχλευσης (leverage) στο κινητήρα[40]. Οι inrunners χρησιμοποιούνται όταν δεν είναι απαραίτητη μεγάλη ροπή, αλλά χρειάζεται υψηλή ταχύτητα περιστροφής/RPM.

Επίσης γίνεται διάκριση ανάλογα με το αν διαθέτουν βρόχο ανατροφοδότησης για επικοινωνία με τα ESC, σε sensored και sensorless BLDC κινητήρες. Η κάθε κατηγορία απαιτεί και αντίστοιχο sensored, ή sensorless ESC αντίστοιχα. Αν το ESC + BLDC είναι sensored, τότε ο κινητήρας δημιουργεί με το ESC έναν βρόχο ανάδρασης μέσω του οποίου τροφοδοτεί πίσω στο ESC ένα μικρό παλμό, ο οποίος το πληροφορεί για την ακριβή θέση του μόνιμου μαγνήτη. Αυτό επιτυγχάνεται με έναν αισθητήρα ενσωματωμένο στον sensored κινητήρα ο οποίος ανιχνεύει τη θέση του μαγνήτη. Το αισθητήριο όργανο οφείλει την αρχή λειτουργίας του στο φαινόμενο Hall (Hall detector/hall effect sensor). Βασιζόμενο σε αυτή την πληροφορία το ESC μπορεί και συγχρονίζεται καλύτερα με τον κινητήρα, επειδή γνωρίζει με ακρίβεια πότε να ενεργοποιεί την κάθε φάση. Τα καλώδια των φάσεων στους sensored κινητήρες έχουν αρίθμηση και πρέπει να συνδέονται με συγκεκριμένα καλώδια στα αντίστοιχα sensored ESC.

Στην Εικόνα 3-23 παριστάνεται το διάγραμμα ενός τυπικού outrunner BLDC κινητήρα. Ο ρότορας είναι μόνιμος μαγνήτης εξωτερικά και στο εσωτερικό ο στάτορας απαρτίζεται από πολλά ζευγάρια πηνίων/μαγνητικών πόλων (ένα ζευγάρι πηνίων αναφέρεται και ως «τύλιγμα») (στην εικόνα παίρνουμε ως παράδειγμα κινητήρα με μόνο 3 πηνία), γύρω από το στέλεχος του κινητήρα (motor shaft). Εφαρμόζοντας τάση σε ένα ζευγάρι πηνίων, προκύπτει ένας ηλεκτρομαγνήτης. Η λειτουργία/περιστροφή του συνίσταται στην αλληλεπίδραση μεταξύ του μόνιμου μαγνήτη και του



Εικόνα 3-23 Σχήμα outrunner κινητήρα χωρίς ψήκτρα[41]

ηλεκτρομαγνήτη. Όταν ο ηλεκτρομαγνήτης είναι ενεργοποιημένος, το ένα του άκρο (ένα πηνίο) γίνεται ο βόρειος πόλος και το άλλο του άκρο (το άλλο πηνίο του τυλίγματος) γίνεται ο νότιος πόλος. Επειδή ο βόρειος πόλος οποιουδήποτε μαγνήτη απωθείται από τον βόρειο πόλο ενός άλλου μαγνήτη, ο οπλισμός του κινητήρα (από υλικό μόνιμου μαγνήτη) θα περιστραφεί για να συναντήσει το νότιο πόλο του ενεργοποιημένου ηλεκτρομαγνήτη.

Καθώς ο οπλισμός περιστρέφεται για να φέρει κοντά το βόρειο και νότιο πόλο, η τροφοδοτούμενη ισχύς ενεργοποιεί άλλον ηλεκτρομαγνήτη (τον επόμενο στη σειρά) και ο οπλισμός ξαναπεριστρέφεται ώστε να συναντηθούν πάλι οι αντίθετοι πόλοι. Αυτή η διαδικασία επαναλαμβάνεται συνέχεια και πολύ γρήγορα. Όμως κάθε φορά ενεργοποιείται διαδοχικά μόνο ένα τύλιγμα και το γεγονός αυτό μειώνει δραματικά την ισχύ εξόδου/ροπή του κινητήρα. Γι' αυτό εφαρμόζονται 2, ή 3, ή γενικότερα n φάσεις με διαφορά φάσης μεταξύ τους 360/n, ενεργοποιώντας κάθε φορά κατάλληλα ζευγάρια πηνίων, πετυχαίνοντας μέγιστη ισχύ εξόδου, δηλαδή μέγιστη δυνατή ροπή και γωνιακή ταχύτητα περιστροφής. Εάν ο κινητήρας διαθέτει n τυλίγματα, τότε σε κάθε στάδιο λειτουργίας ενεργοποιούνται n / n ηλεκτρομαγνήτες.

Εμείς επιλέξαμε τους τριφασικούς outrunner, sensorless BLDC κινητήρες LD1510A-02-P Micro. Είναι 18 βημάτων, δηλαδή διαθέτουν 18 μαγνητικούς πόλους, ή 9 ηλεκτρομαγνήτες / τυλίγματα. Επομένως για κάθε φάση είναι ενεργοί 9 / 3 = 3 ηλεκτρομαγνήτες. Οι κινητήρες μαζί με τη βάση εγκατάστασης τους φαίνονται στην παρακάτω εικόνα:



Εικόνα 3-24 BLDC κινητήρες LD1510A-02-P Micro και βάση εγκατάστασης[42]

Συνδέσαμε τον κινητήρα με τη βάση του ακολουθώντας τα εξής βήματα:

- Τοποθετούμε μικρή ποσότητα από μπλέ κόλλα σπειρωμάτων Permatex σε κάθε βίδα. Η κόλλα σπειρωμάτων είναι απαραίτητη για στέρεη σύνδεση. Στην πραγματικότητα το μικρό διάκενο μεταξύ της βίδας και του περικόχλιου[43] καταλαμβάνεται ως επί το πλείστον από οξυγόνο. Το οξυγόνο με την πάροδο του χρόνου χαλαρώνει τις συνδέσεις και έπειτα από παρατέταμένη χρήση σε ένα μη επανδρωμένο αεροσκάφος οι βίδες ξελασκάρουν τελείως και ξεφεύγουν. Μετά την εφαρμογή κόλλας σπειρωμάτων το ποσοστό οξυγόνου εκμηδενίζεται.
- Σφίγγουμε τις βίδες σε criss-cross / σταυρωτό μοτίβο.
- Τοποθετούμε τον κινητήρα στην άκρη του μπράτσου του τετρακόπτερου.
- Θυμόμαστε ότι έπειτα από κάθε περίπου 10-12 πτήσεις αποσυναρμολογούμε τον κινητήρα και τοποθετούμε ειδικό λιπαντικό από λάδι σιλικόνης γύρω από τα ρουλεμάν. Αυτή η διαδικασία συμβάλει στη συντήρηση και στη μακροζωία τους.

Εφόσον τώρα έχουμε τοποθετήσει τους κινητήρες στα μπράτσα του τετρακόπτερου πρέπει να τους συνδέσουμε με τους ηλεκτρονικούς ελεγκτές ταχύτητας. Όπως είπαμε οι κινητήρες μας είναι sensorless, επομένως δεν έχει σημασία ο τρόπος σύνδεσης τους με τα ESC. Αυτό είναι εν μέρει αλήθεια. Λαμβάνουμε υπόψη και τη φορά περιστροφής του κάθε κινητήρα καθώς και το σημείο στο

οποίο πρέπει να τοποθετηθεί ο καθένας. Παρατηρώντας και στο μπλοκ διάγραμμα (**Error! Reference source not found.**) οι κινητήρες πρέπει να τοποθετηθούν σε σημείο σύμφωνα με τη διαμόρφωση που γνωρίζει ο ελεγκτής πτήσης APM 2.8. Γι' αυτό το σκοπό τοποθετούμε, όπως φαίνεται και στην Εικόνα 3-2, τους κινητήρες που περιστρέφονται με φορά αντίθετη από αυτή του ρολογιού στη θέση 1 και 2 και τους κινητήρες που περιστρέφονται με τη φορά του ρολογιού στις θέσεις 3 και 4. Αυτή είναι η μορφολογία “Quadcopter X” με την οποία έχει προγραμματισθεί ο ελεγκτής πτήσης.

Η φορά περιστροφής καθορίζεται από τον τρόπο σύνδεσης του κινητήρα με το ESC. Παίρνουμε λοιπόν δύο κινητήρες και τους συνδέουμε με τα ESC με ένα τυχαίο (τον πιο τρυφερό για τα καλώδια) τρόπο, αλλά πρέπει να ακολουθήσουμε τον ίδιο τρόπο και τους δύο συγκεκριμένους κινητήρες. Βλέπουμε τη φορά περιστροφής τους και τους τοποθετούμε στη κατάλληλη διαγώνιο. Εφόσον τους συνδέσαμε με τον ίδιο τρόπο θα πρέπει να περιστρέφονται με την ίδια φορά. [Προσοχή όταν εκτελούμε αυτή τη διαδικασία δεν πρέπει να έχουμε συνδεδεμένες τις προπέλες, διότι υπάρχει ο κίνδυνος αυτοχήματος.] Στους επόμενους δύο κινητήρες αρκεί να αντιστρέψουμε δύο καλώδια μεταξύ τους όταν τα συνδέουμε με τα ESC. Τότε θα περιστρέφονται με φορά αντίθετη με το προηγούμενο ζευγάρι κινητήρων. Τοποθετούμε και αυτό το ζευγάρι στην άλλη διαγώνιο. Τώρα πλέον γνωρίζουμε την φορά περιστροφής τους.

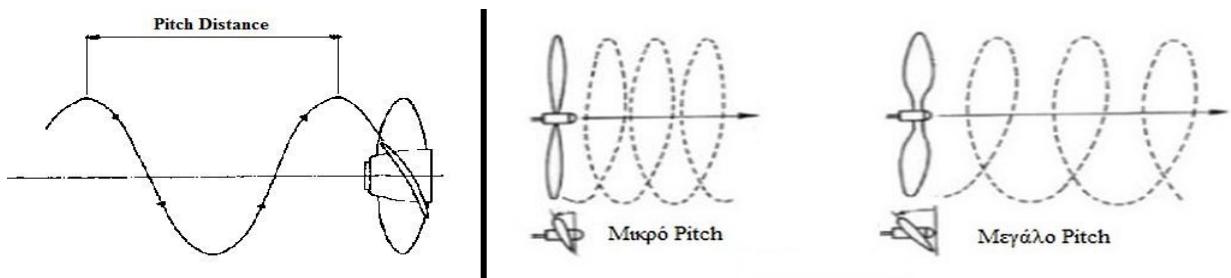
Απομένει να γίνει κάτι ακόμα πριν συνδέσουμε τα ESC με τον ελεγκτή πτήσης. Πρέπει να ισορροπίσουμε τους κινητήρες (Motor Balancing). Στο δυναμικό περιβάλλον πτήσης ενός τετρακόπτερου ακόμα και η παραμικρή ατέλεια μπορεί να προκαλέσει δονήσεις και ταλαντώσεις στο όχημα, ζητήματα τα οποία αυξάνουν την έκλυση θερμότητας και γενικά εντείνουν την ανάγκη πιο συχνής συντήρησης του. Οι κινητήρες είναι η κύρια πηγή παραγωγής δονήσεων μιας και σε αυτούς σπαταλάται το μεγαλύτερο ποσοστό ισχύος. Για να ισοσταθμίσουμε το κάθε κινητήρα κάναμε τα παρακάτω:

1. Εννοείτε ότι όλη η διαδικασία πρέπει να γίνει σε μια απολύτως στέρεα βάση. Εφόσον είχαμε τοποθετήσει τον κινητήρα στο μπράτσο του τετρακόπτερου συνδεδεμένο με το αντίστοιχο ESC, δένουμε με λαστιχάκια σε στέρεα θέση στο ίδιο μπράτσο το iPhone4 (οποιοδήποτε Smartphone αρκεί) με εγκατεστημένη μια εφαρμογή ανίχνευσης κραδασμών/δονήσεων, ή ένα σεισμόμετρο.
2. Στη συνέχεια κολλάμε απαλά ένα πολύ μικρό κομμάτι ταινίας σε ένα σημείο του κινητήρα εξωτερικά. Αυτό θα μας βοηθήσει να βρούμε την ελαφριά του άκρη.
3. Συνδέουμε την έξοδο Throttle του ραδιοδέκτη με το ESC. Ο ραδιοδέκτης μπορεί να τροφοδοτείται από το BEC του ESC.
4. Συνδέουμε και τη μπαταρία με το ESC.
5. Εφαρμόζουμε Throttle από τον ραδιοπομπό για ~10 δευτερόλεπτα. Βλέπουμε τα readings του σεισμόμετρου και παράλληλα ακούμε προσεκτικά τον ήχο του κινητήρα.
6. Επαναλαμβάνουμε τη διαδικασία τοποθετώντας κάθε φορά το κομμάτι ταινίας σε ξεχωριστό σημείο.
7. Όταν η ταινία θα βρίσκεται στο ιδανικό σημείο, ο ήχος που θα παράγεται θα είναι πιο διακριτικός και υψηλότερος και παράλληλα το πλάτος των δονήσεων στο σεισμόμετρο θα δείχνει τη μικρότερη ένδειξη. Η ισορροπία των κινητήρων είναι must στα μη επανδρωμένα αεροσκάφη!

Τέλος συνδέουμε τις εξόδους 1,2,3 και 4 του ελεγκτή πτήσης APM 2.8 με τα ESC που ελέγχουν τους κινητήρες υπό αριθμό 1,2,3 και 4 αντίστοιχα (Εικόνα 3-2).

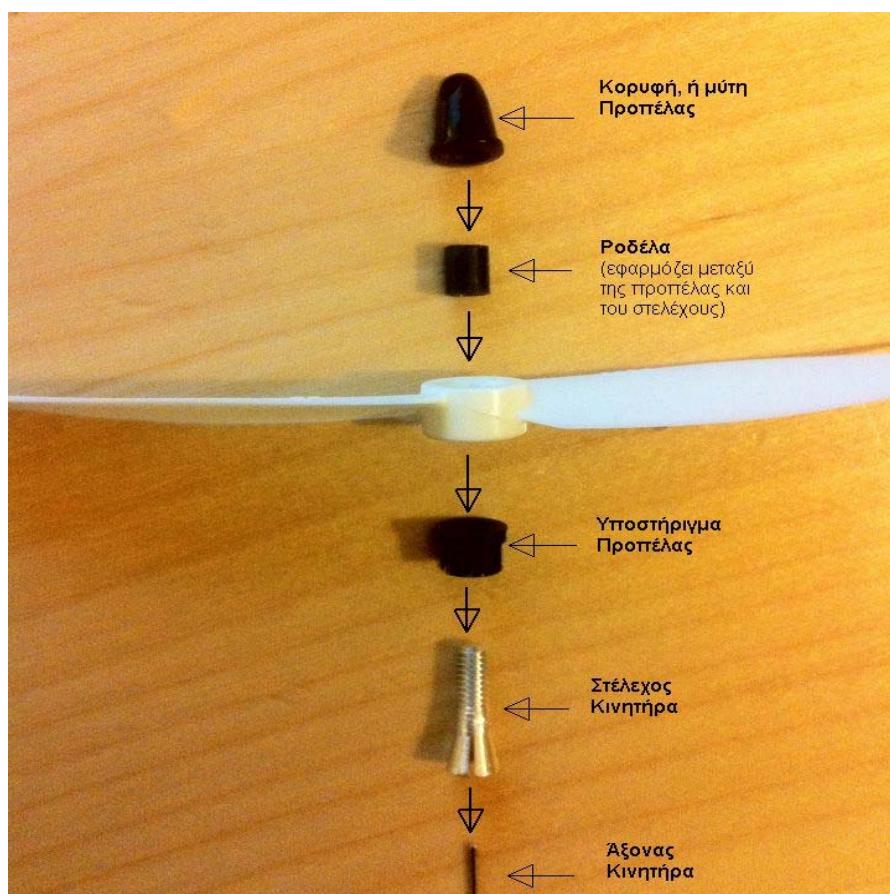
3.10 Προπέλες

Η προπέλα, ή έλιξ – έλικας μετατρέπει περιστροφική κίνηση από την μηχανή, ή τον κινητήρα σε μηχανική προωθητική και ανυψωτική δύναμη για το αεροσκάφος. Αποτελείται από έναν περιστρεφόμενο άξονα, στον οποίο προσαρτώνται ένας αριθμός πτερύγων με σχήμα αεροτομής. Έπειτα από έρευνα[44] αποφασίσαμε ότι προπέλες με δύο λεπίδες είναι κατάλληλες στο μικρής κλίμακας τετρακόπτερο που κατασκευάσαμε. Προπέλλες πολλαπλών λεπίδων έχουν την ικανότητα να μετατρέπουν ισχύ σε προωθητική δύναμη σε μικρότερο χώρο από μια προπέλα δύο λεπίδων, αλλά απαιτούν περισσότερη ισχύ (ως πιο βαριές) και παράλληλα αυξάνουν το επίπεδο δονήσεων. Σύμφωνα με τις προδιαγραφές μας επιλέξαμε τις προπέλες Gemfan 5x3. Αυτό σημαίνει ότι το μήκος τους είναι 5 ίντσες και το pitch τους 3 ίντσες. Pitch είναι το χαρακτηριστικό της προπέλας που αφορά τη διανυθείσα απόσταση σε μια πλήρης περιστροφή της (1 revolution). Η παρακάτω εικόνα προσφέρει μια πιο διαισθητική ερμηνεία.



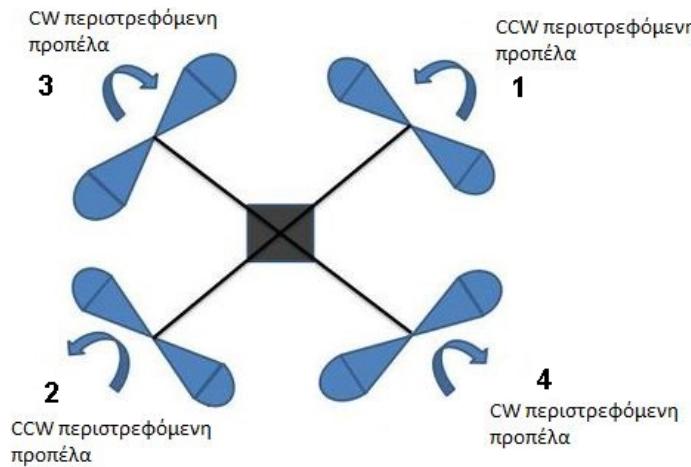
Εικόνα 3-25 Pitch προπέλας είναι η απόσταση που διανύει η προπέλα σε μια πλήρης περιστροφή της[45][46]

Η συναρμολόγησης προπέλας και η ορθή τοποθέτηση της στον κινητήρα θα γίνει καλύτερα κατανοητή αν προσέξουμε την επόμενη εικόνα. Επισημαίνεται ότι η πλευρά της προπέλας στην οποία αναφέρεται το μέγεθος της (πχ. 5030, 5030R) είναι η άνω πλευρά.



Εικόνα 3-26 Συναρμολόγηση Προπέλας

Οι προπέλες που βρίσκονται στην ίδια διαγώνιο περιστρέφονται με την ίδια φορά, τη φορά περιστροφής των κινητήρων, αντισταθμίζοντας το φαινόμενο δράσης αντίδρασης. Τότε όπως και με τους κινητήρες οι προπέλες θα εγκατασταθούν ως εξής:



Εικόνα 3-27 Διαμόρφωση Προπελών Quadcopter[47]

Οι προπέλες των UAV με φορά περιστροφής εκείνη του ρολογιού είναι γνωστές και ως pusher προπέλλες και είναι μαρκαρισμένες με ένα P ή R, μετά τη σήμανση του μεγέθους τους (πχ. 5030R, ή 5x3P). Οι προπέλες με φορά περιστροφής αντίθετη της φορά του ρολογιού είναι γνωστές ως puller propellers. Για τα χαρακτηριστικά μήκους και pitch της προπέλας, το μήκος αναγράφεται πάντοτε πρώτο και το pitch δεύτερο και η μονάδα μέτρησης τους είναι οι ίντσες, (1 ίντσα = 2.54cm), για παράδειγμα στο 5030, το 5αφορά τις ίντσες του μήκους και το 3 τις ίντσες του pitch της προπέλας.

Πριν όμως εγκαταστήσουμε την προπέλα στον κινητήρα πρέπει να την ισορροπήσουμε (και αυτή!). Σ' ένα UAV drone μια μη ισορροπημένη προπέλα δημιουργεί πολύ περισσότερο θόρυβο από μια ισορροπημένη ίδιων διαστάσεων. Επιπλέον μια μη ισορροπημένη προπέλα σπαταλάει ισχύ, αυξάνοντας τη θερμότητα της διάταξης. Η πλεονάζουσα θερμότητα παράγεται επειδή μια μη ισορροπημένη προπέλα προβάλει μια δύναμη πλαγίως του στελέχους του κινητήρα και το πιέζει ενάντια στο ρουλεμάν. Για να ισορροπήσουμε την προπέλα θα χρειαστούμε κόλλα στιγμής ή CA, έναν εξισορροπητή/ισοσταθμιστή προπελών (prop balancer) και, αν δε διαθέτει ο ίδιος, δύο υπερυψωμένα απολύτως ίδιου ύψους αντικείμενα (εμείς χρησιμοποιήσαμε ποτήρια). Ακολουθούμε την παρακάτω διαδικασία:

1. Τοποθετούμε την προπέλα, σε οριζόντια στάση, σταθερά μεταξύ του εξισορροπητή τον οποίο αναρτούμε μεταξύ των δύο υπερυψωμένων αντικειμένων, έτσι ώστε να υπάρχει αρκετός χώρος από κάτω για να περιστρέφεται η προπέλα ελεύθερη.
2. Μετακινούμε απαλά την προπέλα και παρατηρούμε τη στάση της όταν θα σταματήσει να περιστρέφεται. Το τμήμα που κρέμεται κάτω από το κέντρο είναι το βαρύ τμήμα της προπέλας.
3. Παίρνουμε ένα κομμάτι γυαλόχαρτο και τρίβουμε το βαρύ τμήμα της προπέλας από το κάτω μέρος, προσέχοντας να μην ξύσουμε τα χείλη (leading and trailing edges) της προπέλας.
4. Αφού ξύσουμε λίγο καθαρίζουμε την εναπομένουσα σκόνη και την τοποθετούμε πάλι, σε περίπου οριζόντια στάση, μεταξύ των υπερυψωμένων αντικειμένων. Αν συνεχίζει να στέκεται στην ίδια θέση τότε χρειάζεται περισσότερο τρίψιμο.
5. Επαναλαμβάνουμε τη διαδικασία μέχρι η προπέλα να παραμένει σταθερή οποτεδήποτε και αν την αφήνουμε σε περίπου οριζόντια στάση από οποιαδήποτε πλευρά.

Έπειτα από αυτό ίσως να παρατηρήσουμε ότι παρόλο που η προπέλα αναρτάται σταθερά οριζόντια, δε διατηρείται σταθερή όμως αν τη τοποθετήσουμε σε κάποια στάση διαφορετική της

οριζόντιας. Αυτό συμβαίνει επειδή παρόλο που η προπέλα είναι ισορροπημένη το κομβικό της σημείο (hub) δεν είναι! Πρέπει να ισορροπήσουμε και αυτό! Οπότε:

1. Τοποθετούμε πάλι την προπέλα στον εξισορροπητή και τον αναρτούμε προσεκτικά σε κατακόρυφη τάρα θέση.
2. Αφήνουμε από τα χέρια μας την προπέλα απαλά και βλέπουμε προς τα που θα μετακινηθεί. Την αφήνουμε μέχρι να σταματήσει να στριφογυρίζει.
3. Όταν σταματήσει, σημειώνουμε το κομβικό της σημείο που κοιτάει προς τα κάτω. Αυτό είναι το βαρύτερο τμήμα του κέντρου. Προμηθευόμαστε κόλλα στιγμής, ή κολλητικό CA και εναποθέτουμε πολύ μικρή ποσότητα στο ελαφρύτερο τμήμα του κέντρου και το αφήνουμε να στεγνώσει.
4. Η εξισορρόπηση του κομβικού σημείου μπορεί να φθείρει την ισορροπία της ίδιας της προπέλας!
5. Επαναλαμβάνουμε και τις δύο διαδικασίες μέχρις ότου η προπέλα να διατηρείται σταθερή στον εξισορροπητή οπουδήποτε και με οποιοδήποτε τρόπο την τοποθετήσουμε μεταξύ των αντικειμένων.

Τότε θα έχουμε μια τέλεια ισοσταθμισμένη προπέλα και η διαφορά κατά την πτήση είναι απολύτως εμφανής (όσο παράξενο και αν πιθανόν σας φαίνεται!). Τέλος, τοποθετούμε την προπέλα στον κινητήρα όπως δείχνει και η Εικόνα 3-26.

3.11 Τροφοδοσία – Μπαταρία Λιθίου Πολυμερούς

Μια μπαταρία λιθίου πολυμερούς (Lithium Polymer, ή LiPo για συντομία), ή για την ακρίβεια μια μπαταρία ιόντων-λιθίου πολυμερούς, είναι ένα είδος επαναφορτιζόμενης μπαταρίας τεχνολογίας ιόντων λιθίου. Διαφέροντας από τα κυλινδρικά και τα πρισματικά σχήματα κελιών, οι LiPos συναντώνται σε μια πιο ελαφριά και εύκαμπτη συσκευασία[48]. Οι κυψέλες LiPo ιστορικά αποτελούν επακόλουθο στην έρευνα ανάπτυξης μπαταριών από τις κυψέλες ιόντων-λιθίου και λιθίου-μετάλλου, οι οποίες βρίσκονταν υπό εντατική έρευνα τη δεκαετία του 80, φτάνοντας ένα βαρυσήμαντο ορόσημο με την πρώτη επιτυχή εμπορική πράξη η οποία έγινε από τη Sony το 1991. Από τότε και στο εξής οι μπαταρίες βρίσκονται σε συνεχή χρήση σε όλους τους κλάδους της βιομηχανίας και στα νοικοκυριά γενικότερα.

Ο προσδιορισμός «λιθίου πολυμερούς» έχει προκαλέσει σύγχυση μεταξύ των χρηστών, διότι χρησιμοποιείται με δύο τρόπους. Αρχικά ο όρος αναπαριστούσε μια αναπτυσσόμενη τεχνολογία που χρησιμοποιούσε ηλεκτρολύτη από πολυμερές υλικό αντί για ένα τυπικό ηλεκτρολυτικό οξύ. Το αποτέλεσμα της έρευνας (η οποία ακόμη δεν έχει πλήρως ολοκληρωθεί) είναι ένα «πλαστικό»-πολυμερές κελί μπαταρίας, το οποίο είναι πιο λεπτό και ευέλικτο, κατασκευάζεται σε ποικίλα σχήματα και μεγέθη, χωρίς κίνδυνο διαρροής του ηλεκτρολύτη.

Η δεύτερη ονομασία εμφανίστηκε όταν κάποιοι κατασκευαστές εισήγαγαν το όρο «πολυμερές» στις μπαταρίες ιόντων-λιθίου που περιέχονται σε μια εύκαμπτη συσκευασία που μοιάζει με σακίδιο. Αυτό είναι το περισσότερο δημοφιλές είδος μπαταρίας σήμερα, στο οποίο ο όρος πολυμερές δεν σημαίνει ηλεκτρολύτης από πολυμερή (μακρομόρια) αλλά αφορά την επένδυση, τη συσκευασία που είναι κατασκευασμένη από πολυμερές (πλαστικό) υλικό! Οπότε στις μπαταρίες LiPo ο ηλεκτρολύτης είναι σε υγρή μορφή ιόντων-λιθίου, ενώ η συσκευασία είναι από πολυμερές υλικό, συνήθως επίπεδη και σε σχήμα ορθογωνίου παραλληλεπιπέδου. Όμως από τεχνολογικής απόψεως οι μπαταρίες “LiPo” είναι οι ίδιες με αυτές που φέρουν την εμπορική ονομασία “Li-ion”, εφόσον η ηλεκτροχημεία της μπαταρίας είναι η ίδια. Και η δικιά μας μπαταρία λοιπόν είναι μια LiPo με την έννοια αυτής, της δεύτερης έννοιας.

Οι μπαταρίες LiPo χρησιμοποιούνται ευρέως στα ραδιοκατευθυνόμενα μοντέλα, και μάλιστα συνήθως με αρκετά κελιά τάσης 3.7V, είτε σε σειρά, είτε παράλληλα. Ο πιο γενικός όρος Li-ion χρησιμοποιείται σχεδόν οπουδήποτε άλλού, όπως κινητά τηλέφωνα, διάφορες ηλεκτρονικές συσκευές, φορητούς υπολογιστές κτλ. Θα λάβουμε ως παράδειγμα τη δική μας μπαταρία LiPo Turnigy 2700mAh 3S 20C για να κάνουμε ευδιάκριτα τα χαρακτηριστικά της.



Εικόνα 3-28 Η μπαταρία LiPo Turnigy 2700mAh 3S 20C που χρησιμοποιήσαμε[49]

Ο προσδιορισμός 3S αναφέρεται στον αριθμό των κελιών LiPo 3.7Volt που είναι σε σειρά δίνοντας αθροιστικά ονομαστική τιμή τάσης στη μπαταρίας τα $3.7 \times 3 = 11.1V$. Το κάθε κελί είναι περίπου 20% ελαφρύτερο από ισοδύναμα κυλινδρικά κελιά Li-Ion ίσης χωρητικότητας. Ένας επιπλέον σοβαρός λόγος για να χρησιμοποιηθούν στα UAV, αφού το βάρος του οχήματος πιθανόν να είναι το χαρακτηριστικό με τη πιο κρίσιμη σημασία. Το mAh στο 2700mAh ποδεικύει την χωρητικότητα της μπαταρίας σε mA×hours. Δηλαδή 2.7Ah, ή 2700mAh είναι η χωρητικότητα της μπαταρίας. Όσο μεγαλύτερη είναι η χωρητικότητα της μπαταρίας τόσο περισσότερο ρεύμα μπορεί να αποδόσει προτού εκφορτιστεί και τόσο πιο βαριά είναι.

Η διαφορά δυναμικού σε ένα κελί LiPo κυμαίνεται μεταξύ 2.7-3.0V αφότιστο και μεταξύ 4.2-4.35 πλήρως φορτισμένο. Η δική μας μπαταρία με το κάθε κελί πλήρως φορτισμένο στα 4.2V φτάνει τη συνολική τάση των $4.2 \times 3 = 12.6V$. Εκτός από το ενδεικτικό “S” μια μπαταρία LiPo μπορεί να φέρει τον χαρακτηρισμό “P” [50] στις προδιαγραφές της, που αφορά τον αριθμό των παράλληλων κελιών που διαθέτει. Για παράδειγμα μια μπαταρία με το προσδιοριστικό 4S3P διαθέτει 4 κελιά σε σειρά και το κάθε ένα διαθέτει 3 κελιά παράλληλα. Φαίνεται αυτό καλύτερα και στο επόμενο σχήμα.

→ [- 1.2Ah, 3.7V +] [- 1.2Ah, 3.7V +] [- 1.2Ah, 3.7V +] [- 1.2Ah, 3.7V +] --→
--→ → [- 1.2Ah, 3.7V +] [- 1.2Ah, 3.7V +] [- 1.2Ah, 3.7V +] [- 1.2Ah, 3.7V +] --→ --→
→ [- 1.2Ah, 3.7V +] [- 1.2Ah, 3.7V +] [- 1.2Ah, 3.7V +] [- 1.2Ah, 3.7V +] --→

Τα παράλληλα κελιά αθροίζουν την χωρητικότητα του κάθε κελιού της μπαταρίας, ενώ τα κελιά σειράς αυξάνουν τη συνολική τάση. Πχ. η μπαταρία μας με χωρητικότητα 2700mAh, ή $2.7Ah=2.7$ (Amperes × hours) μπορεί να αποδίδει σταθερά, είτε:

- 2.7A για 1 ώρα, είτε
- 5.4A για μισή ώρα, είτε
- 10.8A για 15 λεπτά, είτε
- 675mA για 4 ώρες κτλ.

Το προσδιοριστικό C στο 20C έχει να κάνει με το ρυθμό εκφόρτισης της μπαταρίας. Για παράδειγμα η δικιά μας μπαταρία με τιμές 20C και 2700mAh, σημαίνει ότι δυνητικά το μέγιστο ρεύμα που μπορεί να αποδώσει σε κανονική λειτουργία είναι $2700mAh \times 20C = 54000mAh = 54Amps$.

Πιθανόν να αναρωτιέστε ποιές είναι οι μονάδες που αναλύεται το μέγεθος “C” και πώς προήλθαν τα αυπέρ (Amps). Στην πραγματικότητα πρόκειται για ένα εμπειρικό μέγεθος που εισήχθηκε από κατασκευαστές των μπαταριών για εμπορικούς σκοπούς. Αυτό συνεπάγεται ότι δε διαθέτει μεγάλη ακρίβεια και επιστημονική τεκμηρίωση, παρ’όλα αυτά αποδεικνύεται ότι παριστάνει την καμπύλη εκφόρτισης μιας μπαταρίας σε ικανοποιητικό βαθμό για τις περισσότερες εφαρμογές. Αν θέλουμε περισσότερο ακριβείς μετρήσεις θα πρέπει να κοιτάξουμε την ίδια τη καμπύλη εκφόρτισης της μπαταρίας, για να δούμε την ικανότητα της να αποδεσμεύει ρεύμα όταν βρίσκεται σε διάφορα επίπεδα τάσης και χωρητικότητας φόρτισης. Γενικά όσο μειώνεται η χωρητικότητα της μπαταρίας τόσο μειώνεται η τάση και η ικανότητα απελευθέρωσης ρεύματος. Οι μπαταρίες LiPo υπερτερούν και σε αυτόν τον τομέα καθώς είναι ιδιαίτερα ανθεκτικές. Πρέπει όμως να τίθεται μεγάλη προσοχή με αυτές επειδή είναι ιδιαιτέρως πτητικές. Για σύγκριση με άλλες χημείες όπως NiCd και NiMh που σε περίπτωση δυσλειτουργίας απλώς μπορεί να χυθεί ο ηλεκτρολύτης στη συσκευασία, οι μπαταρίες LiPo δύνανται να αρπάξουν ακόμα και φωτιά! Θα μιλήσουμε αργότερα για προφυλάξεις που πρέπει να λαμβάνουμε όταν φορτίζουμε/εκφορτίζουμε και αποθηκεύουμε τις μπαταρίες LiPo.

Στις προδιαγραφές μπαταριών LiPo εκτός από τον όρο C αναγράφεται και ο μέγιστος ρυθμός C και για πόσο χρόνο ισχύει. Αυτό αφορά το μέγιστο ρυθμό εκφόρτωσης ρεύματος από τη μπαταρία, εάν χρειαστεί, και το μέγιστο χρόνο που θα μπορεί να συμβαίνει αυτό. Γενικώς, όσο μεγαλύτερη είναι η τιμή C τόσο πιο ισχυρή είναι η μπαταρία, αλλά παράλληλα μειώνεται και η διάρκεια ζωής της στις περισσότερες περιπτώσεις, δηλαδή ο αριθμός κύκλων φόρτισης – εκφόρτισης.

Τέλος, η εσωτερική αντίσταση της μπαταρίας είναι πολύ σημαντική παράμετρος, καθορίζει τις επιδόσεις και τη μακροζωΐα της. Η εσωτερική αντίσταση της μπαταρίας αυξάνεται όσο η μπαταρία εκφορτίζεται και μεγαλύτερα κελιά έχουν γενικά μικρότερες εμπεδήσεις. Όσο μικρότερη είναι η εσωτερική αντίσταση τόσο περισσότερο ρεύμα μπορεί να αποδίδει χωρίς να μειώνεται η τάση της.

Φορτίζουμε τις μπαταρίες LiPo σε έναν σταθμό φόρτισης LiPo μπαταρίων και ας έχουμε στο νού μας να μην τις φορτίζουμε περισσότερες φορές από τον αριθμό κύκλων φόρτισης (charge cycles) που συνήθως αναγράφεται στις προδιαγραφές της μπαταρίας. Εάν δεν αναγράφεται τότε για τις μπαταρίες LiPo μεσαίου μεγέθους γενικού σκοπού συνήθως είναι 300-500 φορές. Για να επιτύχουμε τις καλύτερες επιδόσεις των LiPo μπαταριών μας είναι συνετό να ακολουθούμε μερικές από τις παρακάτω οδηγίες συντήρησης:

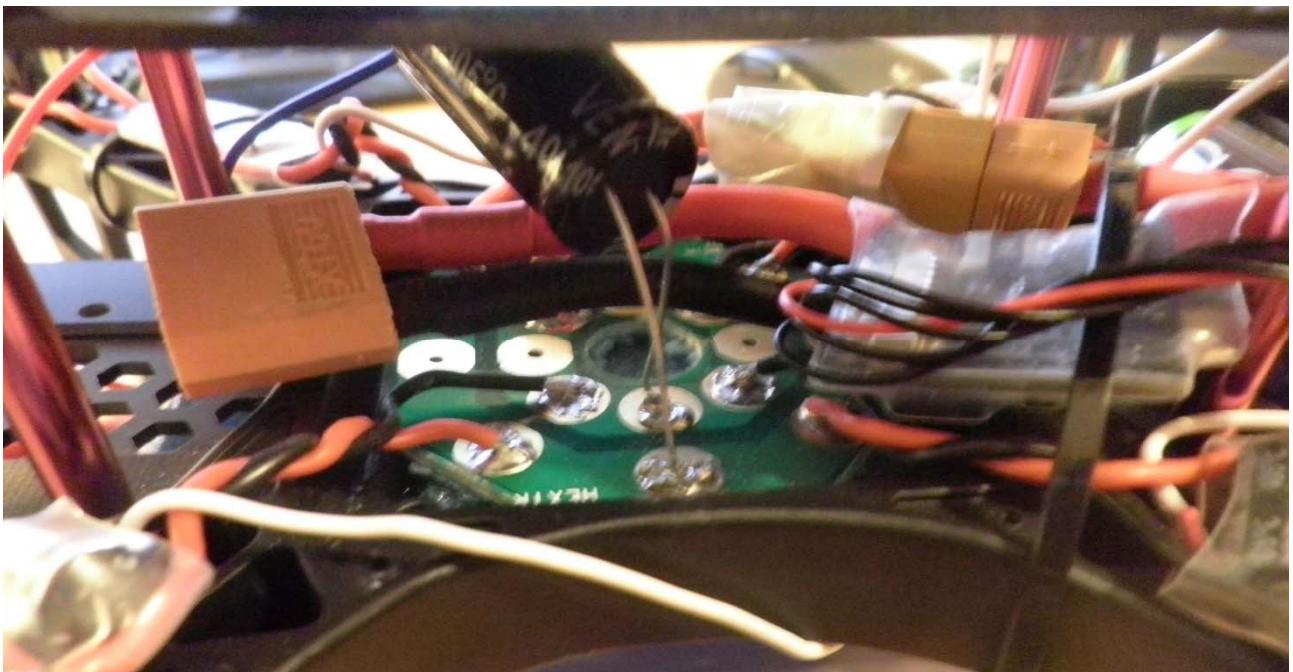
- Πρώτα συνδέουμε το τροφοδοτικό με το σταθμό φόρτισης, ύστερα το θέτουμε σε ισχύ και έπειτα συνδέουμε τη μπαταρία!
- Πάντα φορτίζουμε με ρυθμό 1C, ή λιγότερο, τη μπαταρία χρησιμοποιώντας έναν πιστοποιημένο φορτιστή LiPo. Εδώ η ένδειξη C δεν αφορά το ρυθμό εκφόρτισης, αλλά το ρυθμό φόρτισης, δηλαδή με πόσα αυπέρ να φορτίζουμε τη μπαταρία. 1C σημαίνει ότι πρέπει να ρυθμίσουμε τον φορτιστή έτσι ώστε να εισάγει τόσα αυπέρ όση είναι η χωρητικότητα της μπαταρίας σε Ah. Πχ. η μπαταρία μας που είναι 2.7Ah πρέπει να φορτίζεται με 2.7Amps/sec. Ποτέ περισσότερο από 1C ακόμη και αν αναφέρεται στις προδιαγραφές ότι είναι κάτι τέτοιο δυνατό. Υψηλότερος ρυθμός φόρτισης παράγει περισσότερη θερμότητα στο εσωτερικό του ηλεκτρολύτη που αυξάνει τις πιθανότητες ανάφλεξης του. Όπως είπαμε οι μπαταρίες Lipo είναι εξαιρετικά πτητικές και αυτό βέβαια είναι και το πλεονέκτημα τους, αλλά παράλληλα χρειάζονται επιπλέον προστασία. Η θερμότητα είναι ο χειρότερος εχθρός των μπαταριών γενικότερα!
- Η ισορροπημένη φόρτιση (balance charging) παίρνει περισσότερο χρόνο να ολοκληρωθεί, γι’ αυτό είναι καλό να έχουμε υπομονή. Η ισορροπημένη φόρτιση φορτίζει το κάθε κελί ξεχωριστά εν αντιθέσει με ολόκληρη τη μπαταρία – όλα τα κελιά μαζί. Δηλαδή φορτίζει το κάθε κελί μεταξύ του κατάλληλου ακροδέκτη (balance lead) και της γειώσης. Γι’ αυτό οι μπαταρίες LiPo που προμηθευόμαστε θα πρέπει να διαθέτουν συνδετήρες ισορροπημένης φόρτισης. Ισορροπημένη

φόρτιση είναι καλό να γίνεται κάθε περίπου 5 κύκλους φόρτισης ή και συχνότερα. Σε διαφορετική περίπτωση η τάση σε κάποιο(/α) κάθε κελί της μπαταρίας θ' αρχίσει να διαφοροποιείται έντονα. Γενικά δεν πρέπει να υπερβαίνουμε διαφορά δυναμικού πάνω από 0.5V μεταξύ του κάθε κελιού.

- Ποτέ δεν χρησιμοποιούμε τις μπαταρίες σε εφαρμογές όπου πιθανόν να χρειαστεί να αποδεσμεύσουν ρεύμα περισσότερο από αυτό που μπορούν. Για να είμαστε σίγουροι για κάτι τέτοιο προσέχουμε το μέγιστο απορροφούμενο ρεύμα της συσκευής μεγαλύτερης ισχύος που βρίσκεται ενεργή στη διάταξη/κατασκευή μας (πχ. για το τετρακόπτερο τους κινητήρες). Υψηλότερο ρεύμα σημαίνει επίσης μεγαλύτερη έκλυση θερμότητας.
- Είναι συνετό να έχουμε το νου μας καθώς φορτίζουμε τις μπαταρίες LiPo. Πάντα ενέχουν τον κίνδυνο ανάφλεξης. Επίσης καλό είναι να υπάρχει πρόχειρος εξοπλισμός κατάσβεσης πυρκαγιών, πυροσβεστήρας, ανιχνευτής καπνού. Το καλύτερο θα ήταν να τις φορτίζουμε είτε πάνω σε τσιμεντένιο δάπεδο, είτε μέσα σε πυρίμαχο (rugex) κιβώτιο, είτε μέσα σε κουτί πυρομαχικών (ammo box), είτε σε ειδική τσάντα φόρτισης LiPo.
- Όταν έχουμε σκοπό να αποθηκεύσουμε τις μπαταρίες LiPo και να μην τις χρησιμοποιήσουμε για κάποιο διάστημα, καλό θα είναι να τις εκφορτίζουμε ώστε να διατηρούν περίπου τη μισή από την πλήρη χωρητικότητα τους. Η χημεία της μπαταρίας δεν είναι τόσο ευσταθής όταν διατηρείται για παρατεταμένη χρονική διάρκεια σε μέγιστες ($>85\%$), ή ελάχιστες τιμές ($<15\%$) χωρητικότητας. Γι' αυτό το σκοπό μπορούμε να χρησιμοποιήσουμε τον ιδιαίτερο τρόπο φόρτισης που πρέπει να διατίθεται στους περισσότερους σύγχρονους φορτιστές LiPo, τον αποκαλούμενο τρόπο φόρτισης προς αποθήκευση (LiPo Storage Charge).

3.12 Πλακέτα Διανομής Ισχύος

Η πλακέτα διανομής ισχύος (Power Distribution Board) είναι αναπόσπαστο κομμάτι κάθε ηλεκτρικής συσκευής ισχύος, παρέχει ένα χώρο στον οποίο απολήγουν τα καλώδια φάσης (+) και γείωσης (GND) όλων των ηλεκτρικών συσκευών. Η κατασκευή της είναι απλή, δεν χρειάζεται εξειδικευμένα εργαλεία, αλλά εργατικώς επίπονη, διότι απαιτεί υπομονή και δεξιοτεχνία στη χρήση του κολλητηριού (Soldering Iron). Η πλακέτα διανομής ενέργειας αποτελείται από ένα αγώγιμο ηλεκτρικά υλικό συχνά κατασκευασμένο από χαλκό, το οποίο διαιρείται στα δύο, ένα μέρος στο οποίο συγκολλούνται τα καλώδια για τη γείωση (ή αρνητικό δυναμικό «-») και ένα μέρος για τα καλώδια της φάσης. Τοποθετήσαμε την πλακέτα διανομής ενέργειας στο κεντρικό τμήμα του τετρακόπτερου που έχει αρκετό χώρο και το αποτέλεσμα φαίνεται στην παρακάτω εικόνα με όλους τους ηλεκτρονικούς ελεγκτές ταχύτητας ήδη συγκολλημένους:



Εικόνα 3-29 Πλακέτα Διανομής Ισχύος

Υπάρχουν διάφορες τεχνικές λεπτομέρειες που λάβαμε υπόψη κατά τη σύνδεση όλων των κυκλωμάτων και εξαρτημάτων στο τετρακόπτερο, που δεν μπορούν να γίνουν εύκολα κατανοητές και αντιληπτές από τις εικόνες. Αυτά τα ιδιαίτερα γνωρίσματα θα παρουσιάσουμε και θα εξηγήσουμε στη συνέχεια.

Καταρχάς ίσως προσέξατε ότι στην παραπάνω εικόνα έχουμε συστρέψει τα καλώδια των ESC. Γενικώς έχουμε ακολουθήσει αυτή την τακτική σε όλα τα καλώδια ισχύος στο τετρακόπτερο. Προσπαθούμε να τα διατηρήσουμε όσο γίνεται πιο σύντομα και τα συστρέψουμε ώστε να μειωθεί το εμβαδό των βρόχων ρεύματος που δημιουργούνται μεταξύ του εκάστοτε καλωδίου φάσης και γείωσης. Πρόκειται για την κύρια πηγή μαγνητικών παρεμβολών. Το φαινόμενο αυτό εξηγείται από την εφαρμογή του νόμου των [51][52] Biot και Savart:

$$dB = \frac{\mu_0 I dL}{3\pi} \frac{R}{(z^2 + R^2)^{\frac{3}{2}}} \quad (16)$$

για τους βρόχους ρεύματος:

$$B_z = \frac{\mu_0}{4\pi} \frac{2\pi R^2 I}{(z^2 + R^2)^{\frac{3}{2}}} \quad (17)$$

Υπάρχουν τρία πράγματα που πρέπει να κρατήσουμε από την εξίσωση (17):

1. Το μαγνητικό πεδίο αυξάνεται ως συνάρτηση του περίφρακτου εμβαδού $2\pi R^2$, ακτίνας R ,
 - Όσο μεγαλύτερη η θηλιά τόσο μεγαλύτερο το επαγόμενο μαγνητικό πεδίο
 - Μπορούμε να μειώσουμε το μαγνητικό πεδίο συστρέφοντας τα καλώδια, μειώνοντας έτσι το εμβαδό των βρόχων που δημιουργούνται.
2. Το μαγνητικό πεδίο αυξάνεται ως συνάρτηση του ρεύματος I
 - Όπου υπάρχει αυτή η επιλογή, μπορούμε να προσφέρουμε την ίδια ποσότητα ισχύος και να μειώσουμε το μαγνητικό πεδίο αυξάνοντας τη τάση και μειώνοντας το ρεύμα. Από αυτή τη σκοπιά, μπαταρία με περισσότερο αριθμό κελιών είναι καλύτερη επιλογή, αλλά δε θέλαμε να αυξήσουμε περαιτέρω το βάρος του τετρακόπτερου.
3. Το μαγνητικό πεδίο μειώνεται ως συνάρτηση του κύβου της απόστασης

- Μειώνουμε την απόσταση στην οποία διέρχεται το ρεύμα, δηλαδή μειώνουμε το μήκος των καλωδίων. (Τα πράγματα γίνονται πιο περίπλοκα όσο πλησιάζουμε προς τη θηλιά, αλλά αποτελεί καλή προσέγγιση να πούμε ότι ο παρονομαστής μειώνεται με την τρίτη δύναμη.) Αυτό είναι ακόμα πιο σημαντικό για τα καλώδια μικρού AWG, όπως τα καλώδια ισχύος από την μπαταρία, γι' αυτό όπως τα δείτε είναι πολύ μικρού μήκους και τα έχουμε συστρέψει, σε σημείο που να έχουν εξαλειφθεί όλοι οι παρατηρήσιμοι βρόχοι.

Άλλες επιδιορθώσεις και «τελευταίες πινελιές» που κάναμε είναι:

- Εναίσθητα ηλεκτρονικά όπως η πυξίδα, το gps και άλλα περιφερειακά θα πρέπει να είναι όσο το δυνατό πιο μακριά από τα ESC και τα καλώδια ισχύος. Πρέπει να κατανοήσουμε ότι η κάθε επιπλέον βαθμίδα φορτίζει την κατασκευή. Κάθε κομμάτι μετάλλου που προστίθεται μπορεί να αποτελέσει έναν συνεισφέρων παράγοντα ηλεκτρομαγνητικής παρεμβολής.
- Χρησιμοποίηση μεταγωγικών και όχι γραμμικών ρυθμιστών τάσης.
- Επειδή τροφοδοτούμε τον ελεγκτή πτήσης από το APM P.M., κόψαμε τα καλώδια των ρυθμιστών τάσης BEC των ESC ακριβώς για να αποφύγουμε επιπλέον καλώδια ισχύος.
- Εφαρμόζουμε θερμοσυστελλόμενα σε οποιαδήποτε εκτεθειμένα καλώδια για να αποφύγουμε πιθανότητες βραχυκυκλώματος.
- Τα ισχυρά transient ρεύματα που ρέουν στα καλώδια ισχύος μπορούν να επιφέρουν την ανάπτυξη ρεύματος σε γειτονικούς αγωγούς, προσθέτοντας συνιστώσες θορύβου σε αυτά. Γι' αυτό το λόγο προσπαθήσαμε γενικά τα καλώδια σήματος πληροφοριών, όπως τα καλώδια σήματος που τροφοδοτούνται από την έξοδο του ελεγκτή πτήσης στα ESC'ς, να μη βρίσκονται πολύ κοντά σε καλώδια ισχύος[53].
- Τοποθετήσαμε μεγάλο ηλεκτρολυτικό πυκνωτή 1000uF, 50V χαμηλής εσωτερικής εν σειρά αντίστασης (ESR) στη πλακέτα διανομής ισχύος, για να συμπληρώνει το ρόλο φιλτραρίσματος/αποσύζευξης στους ήδη υπάρχοντες πυκνωτές στα ESC. Έτσι επιτυγχάνουμε ακόμη μεγαλύτερη εξομάλυνση της υψηλής ισχύος. Δεν πρέπει όμως να το παρακάνουμε διότι οι ηλεκτρολυτικοί πυκνωτές είναι πιο επαγωγικοί, εξαιτίας του λεπτού μεταλλικού σπειροειδούς τυλίγματος τους. Δεν είναι ότι καλύτερο για αποσύζευξη σε υψηλές συχνότητες. Υπόψιν ότι όσο μεγαλύτερη η συχνότητα, τόσο μικρότερη πρέπει να είναι η χωρητικότητα του πυκνωτή. Γι' αυτό, όπως ήδη αναφέραμε, έχουμε τοποθετήσει πολλαπλούς μικρούς πυκνωτές παράκαμψης για την τροφοδοσία του CC3200.
- Φροντίζουμε ότι το κέντρο βάρους του τετρακόπτερου είναι ακριβώς στο κέντρο και ότι γενικά υπάρχει συμμετρία μεταξύ των σκελών του.

Μεταξύ των στόχων μας μέσα από αυτές τις διορθώσεις είναι η καταστολή του θορύβου και των ηλεκτρομαγνητικών παρεμβολών που παράγονται μεταξύ των εξαρτημάτων. Ο θόρυβος, εκτός των άλλων, επηρεάζει τη ραδιοεπικοινωνία και τη τηλεμετρία και επιφέρει επιπλέον φόρτο στη λειτουργία των μηχανών. Οι βασικές πηγές θορύβου που μπορούν να επηρεάσουν τη ραδιοεπικοινωνία ή/και την τηλεμετρία είναι

1. Θόρυβος από τα ηλεκτρονικά ισχύος του αεροσκάφους, κυρίως τους κινητήρες τα ESC, αλλά και τον ελεγκτή πτήσης.
2. Θόρυβος από το ground station H/Y, ιδιαίτερα τη θύρα USB.
3. Θόρυβος από άλλους χρήστες ραδιοεπικοινωνίας ίδιας συχνότητας. Στη περίπτωση μας με πομπό δυνατοτήτων FHSS, μειώνονται οι παρεμβολές από αυτό το φαινόμενο.

4 Προγραμματισμός των Ηλεκτρονικών Συστημάτων του Μη Επανδρωμένου Αεροσκάφους

Ένα μη επανδρωμένο εναέριο όχημα οφείλει τη λειτουργία του σε πλήθος ηλεκτρονικών συστημάτων όλα προγραμματισμένα για να λειτουργούν σε αρμονία. Πέρα λοιπόν από τη διευθέτηση του υλικού και του υλικολογισμικού πρέπει να προγραμματίσουμε αυτά τα ηλεκτρονικά συστήματα. Σε αυτό το κεφάλαιο θα παρουσιάσουμε τη διαδικασία του προγραμματισμού του ελεγκτή πτήσης APM 2.8, του CC3200, όλα τα προγράμματα που χρησιμοποιήθηκαν, καθώς και τη βαθμονόμηση των αισθητήρων του ελεγκτή με το λογισμικό Mission Planner. Ο μικροελεγκτής CC3200 χρησιμοποιεί λειτουργικό σύστημα (Operating System – OS) και θα αποδειχτεί μάλλον αδύνατη η κατανόηση του κώδικα, εάν δε καταλάβουμε πρώτα κάποιες γενικές αρχές και μηχανισμούς των λειτουργικών συστημάτων. Γι' αυτό το λόγο θεωρήθηκε σκόπιμο να ξεκινήσουμε αυτό το κεφάλαιο με μια γενική, σύντομη επισκόπηση των λειτουργικών συστημάτων ενός υπολογιστικού συστήματος. Εστιάζουμε στα λειτουργικά συστήματα των ενσωματωμένων συστημάτων (Embedded Systems), τα οποία αναφέρονται και ως λειτουργικά συστήματα πραγματικού χρόνου (Real Time Operating System – RTOS). Επίσης θα αναφερθούν γενικές αρχές των υπολογιστικών συστημάτων που λαμβάνουμε συχνά υπόψη στον προγραμματισμό του CC3200. Η συγκεκριμένη ενότητα εισάγει πολλούς νέους όρους και αρκτικόλεξα που χρησιμοποιούνται συνεχώς από τους μηχανικούς λογισμικού. Για την απόδοση τους θα αναφέρεται πρώτα ο ελληνικός όρος και ύστερα σε παρένθεση ο αντίστοιχος αγγλικός. Σχεδόν πάντα όμως χρησιμοποιείται ο αγγλικός όρος και την ίδια τακτική ακολουθούμε και εμείς. Έγινε προσπάθεια για όσο το δυνατό πιο σύντομη, αλλά παραστατική απεικόνιση της ουσίας. Εκτός αυτών θα περιγράψουμε συνοπτικά τα πρωτόκολλα επικοινωνίας που χρησιμοποιούνται για το σύστημα ασύρματης τηλεμετρίας που υλοποιείται μεταξύ του CC3200 και του Ground Control Station (GCS).

4.1 Λογισμικό

Έχουμε ήδη αναφέρει μερικά από τα προγράμματα που χρησιμοποιήθηκαν, θα τα αναφέρουμε όμως και εδώ όλα μαζί. Το βασικό πρόγραμμα που χρησιμοποιήθηκε είναι το Code Composer Studio (CCS) v6.1, το οποίο είναι ένα ολοκληρωμένο περιβάλλον προγραμματισμού (IDE) σε γλώσσα C και C++, βασισμένο από την τωρινή έκδοση 6 στο EclipseIDE. Το CCS το χρησιμοποιήσαμε για να προγραμματίσουμε τον μικροελεγκτή του CC3200Launchpad, που αποτέλεσε το μεγαλύτερο μέρος της εργασία μας με το λογισμικό. Το λειτουργικό σύστημα που χρησιμοποιήθηκε στο CC3200 είναι το ιδιόκτητο TI-RTOS (Texas Instruments – Real Time Operating System).

Επιπλέον χρησιμοποιήθηκε το πρόγραμμα Mission Planner, που αναπτύχθηκε από τον Michael Oborne, για τη βαθμονόμηση των αισθητήρων και τον προγραμματισμό του ελεγκτή πτήσης APM 2.8. Για την ακρίβεια ο κώδικας του ελεγκτή πτήσης δεν άλλαξε, απλώς μεταβλήθηκαν διάφορες παράμετροι του για να ικανοποιήσουν τις ανάγκες του δικού μας τετρακόπτερου. Το Mission Planner εκτός από GCS αποτελεί και ένα GUI για τη ρύθμιση των παραμέτρων του ελεγκτή πτήσης. Κατά τη διάρκεια της πτήσης προτιμήσαμε ένα άλλο GCS, το πρόγραμμα APM Planner 2, επειδή θεωρήσαμε ότι ο φαινόμενος ορίζοντας του ανταποκρίνεται πιο ομαλά από τον αντίστοιχο του Mission Planner. Στο Mission Planner γενικά παρατηρήσαμε “lag”. Πιθανώς να φταίει η βιβλιοθήκη απόδοσης γραφικών DirectX που χρησιμοποιείται στο Mission Planner, ενώ στο APM Planner χρησιμοποιείται OpenGL. Δεν ερευνήσαμε όμως περαιτέρω.

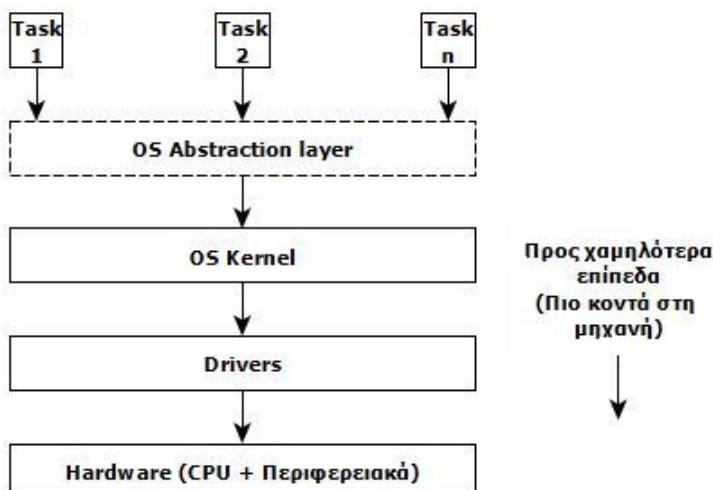
Άλλα προγράμματα είναι το u-Center της u-Blox για την ενημέρωση του firmware της μονάδας δέκτη GPS, τα προγράμματα ειριακού τερματικού εξόμοιωτή TeraTerm, Putty και Hercules, το πρόγραμμα Wireshark για την παρακολούθηση της κίνησης του δικτύου και για την εκσφαλμάτωση

διαφόρων προβλημάτων που παρουσιάστηκαν με τον προγραμματισμό των πρωτοκόλλων επικοινωνίας του CC3200. Διάφορα σχέδια και μπλοκ διαγράμματα σχεδιάστηκαν με το ελεύθερο πρόγραμμα ανοιχτού κώδικα Yed Graph Editor. Τέλος, το πρόγραμμα Mathcad της PTC χρησιμοποιήθηκε για την εκτέλεση των διάφορων μαθηματικών εξισώσεων.

4.2 Λειτουργικά Συστήματα

Το λειτουργικό σύστημα (Operating System – OS) αποτελεί λογισμικό ενός υπολογιστικού συστήματος που διευθύνει το υλικό του υπολογιστή, χρησιμοποιώντας τους διαθέσιμους υπολογιστικούς πόρους του από έναν, ή περισσότερους επεξεργαστές και προσφέρει μια μεγάλη γκάμα υπηρεσιών για τα προγράμματα των χρηστών του συστήματος. Επίσης διαχειρίζεται την μνήμη (κύρια και δευτερεύουσα) τα περιφερειακά και τις συσκευές εισόδου – εξόδου (I/O).

Το λειτουργικό σύστημα δρα ως μεσολαβητής μεταξύ των προγραμμάτων και του υλικού, παρόλο που ο κώδικας της εφαρμογής (Application Code) εκτελείται άμεσα από το υλικό και συχνά είτε κάνει κλήσεις συστήματος σε μια συνάρτηση του OS, είτε διακόπτεται από αυτό. Σημειώνεται ότι το λειτουργικό σύστημα χρησιμοποιείται σε πολύπλοκα υπολογιστικά συστήματα, σε αντίθεση με τα παρασκηνιακά, όταν απαιτείται η επεξεργασία πολλών, ιδιαίτερως περίπλοκων και συχνών λειτουργιών. Ένα OS προσφέρει μηχανισμούς οι οποίοι είναι απαραίτητοι για την υλοποίηση της κάθε λειτουργίας ανεξάρτητα από τις άλλες, παράλληλα όμως οι λειτουργίες μπορούν να «συνεργάζονται». Για παράδειγμα τα OS διαμοιρασμού χρόνου (time sharing) χρησιμοποιούνται όταν καθίσταται πολύ δύσκολη η μεταγωγή μεταξύ των πολλών διάφορων διεργασιών. Αυτή την εργασία της μεταγωγής διεργασιών (task switching) τη χειρίζεται το ίδιο το OS, χρονοπρογραμματίζοντας τη ροή τους χρησιμοποιώντας διάφορους αλγορίθμους, επιτυγχάνοντας την ομαλή εκτέλεση τους.



Εικόνα 4-1 Τα στρώματα ενός υπολογιστικού συστήματος

Μια πολύ σημαντική δυνατότητα ενός OS είναι ότι παρέχει αφαίρεση (Abstraction). Πρόκειται για μια τεχνική διαχείρισης της πολυπλοκότητας των σύγχρονων υπολογιστικών συστημάτων. Η αφαίρεση επιτρέπει τον προγραμματιστή/χρήστη/χειριστή να ασχολείται περισσότερο με το software και λιγότερο με τις συγκεκριμένες, περίπλοκες λεπτομέρειες της κάθε μηχανής – του υλικού που χρησιμοποιείται. Θα μπορούσαμε να το παρομοιάσουμε με τη δυνατότητα που δίνεται στον προγραμματιστή να εργάζεται με μια γλώσσα υψηλότερου επιπέδου πιο εύκολη και γρήγορη στην κατανόηση και διαχείριση και εκτός αυτού, μπορεί να τρέχει σε όλους τους H/Y ανεξαρτήτως υλικού (Cross-Platform). Το στρώμα της αφαίρεσης υλικού (Hardware Abstraction Layer – HAL) είναι μια λογική υποδιαιρεση του κώδικα που χρησιμεύει για την επικοινωνία μεταξύ του υλικού και

λογισμικού του Η/Υ. Προσφέρει μια διεπαφή μεταξύ του προγραμματιστή που εργάζεται στο «ανώτερο» λογισμικό και του υποκείμενου λογισμικού και υλικολογισμικού του συγκεκριμένου υπολογιστή. Λειτουργικά συστήματα βρίσκονται στις περισσότερες σύγχρονες συσκευές που εκτείνονται από κινητά τηλέφωνα, παιχνιδοκονσόλες, μηχανές server, ισχυρούς μικροελεγκτές κτλ.

4.2.1 Παρασκηνιακά Συστήματα

Μικρές, απλές υπολογιστικές μηχανές συνήθως δε διαθέτουν λειτουργικό σύστημα. Αντ' αυτού η εφαρμογή αποτελείται (προαιρετικά) από έναν ατέρμονα βρόχο ο οποίος καλεί λειτουργικές μονάδες, ή συναρτήσεις για να εκτελεστούν οι εργασίες στο «παρασκήνιο». Οι ρουτίνες εξυπηρέτησης σημάτων διακοπής (Interrupt Service Routines – ISR) ενεργοποιούνται με κάθε κατάλληλο συμβάν για να διαχειριστούν ασύγχρονα γεγονότα από ποικίλες πηγές. Τέτοια συστήματα είναι γνωστά ως παρασκηνιακά (Foreground/Background Systems).

4.2.2 Λειτουργικά Συστήματα Πραγματικού Χρόνου

Ένα λειτουργικό σύστημα πραγματικού χρόνου (Real Time Operating System – RTOS) είναι ένα OS που προορίζεται για την εξυπηρέτηση εφαρμογών πραγματικού χρόνου, δηλαδή επεξεργάζεται τα δεδομένα τη στιγμή που φτάνουν στο σύστημα χωρίς καθυστερήσεις σε διαδικασίες αποθήκευσης (buffering). Οι εφαρμογές πραγματικού χρόνου έχουν πολύ αυστηρά χρονικά περιθώρια γι' αυτό απαιτούν ένα ειδικό είδος λειτουργικού συστήματος να τις εξυπηρετήσει. Ένα RTOS πάντα θα περιλαμβάνει τα ακόλουθα χαρακτηριστικά:

- Υποστήριξη δυνατότητας εκτέλεσης πολλαπλών διεργασίων ταυτόχρονα,
- Έναν χρονοπρογραμματιστή (Scheduler) για να αποφασίσει ποιά διεργασία πρέπει να τρέξει,
- Δυνατότητα του χρονοπρογραμματιστή να διακόψει απρόοπτα (preempt) μια τρέχουσα διεργασία,
- Υποστήριξη για επικοινωνία μεταξύ των διεργασιών.

Τα σύγχρονα ενσωματωμένα συστήματα είναι πολλές φορές συστήματα πραγματικού χρόνου (Real Time System – RTS). Τα ενσωματωμένα συστήματα απαιτούν πολυδιεργασία (Multitasking). Η πολυδιεργασία είναι η ικανότητα εκτέλεσης περισσοτέρων του ενός διεργασιών, την ίδια (σχεδόν) χρονική στιγμή χρησιμοποιώντας τον ίδιο επεξεργαστή. Η CPU εναλλάσσει τις διεργασίες που τρέχουν, από τη μια στην άλλη τόσο γρήγορα, που δίνεται η εντύπωση εκτέλεσης όλων των διεργασιών την ίδια χρονική στιγμή. Όταν οι διεργασίες αυξάνονται και οι λειτουργίες τους περιπλέκονται η εναλλαγή και ο συγχρονισμός των διεργασιών αποτελεί πολύ δύσκολο έργο για τον προγραμματιστή. Σε αυτό το σημείο εισάγεται ο ρόλος του λειτουργικού συστήματος. Προσοχή να μην παρεξηγηθεί ο όρος με την πολυεπεξεργασία (Multiprocessing), στην οποία περισσότερες κεντρικές μονάδες επεξεργασίας, μοιράζονται την εργασία –κυριολεκτικά ταυτόχρονα–, με την κατανομή διαφορετικών διεργασιών σε ξεχωριστή CPU.

Το πιο διαδεδομένο RTOS αυτή τη στιγμή που χρησιμοποιείται στα ενσωματωμένα συστήματα είναι το FREE-RTOS[54], που είναι ελεύθερο και ανοικτού κώδικα. Θα κάνουμε χρήση μερικών συναρτήσεων του FREE-RTOS και εμείς στον προγραμματισμό του CC3200.

4.2.2.1 Χαρακτηριστικά Λειτουργικών Συστημάτων Πραγματικού Χρόνου

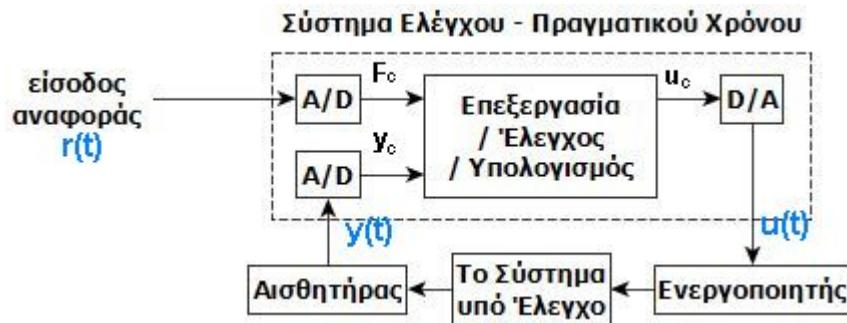
Ποιά είναι τα χαρακτηριστικά των λειτουργικών συστημάτων πραγματικού χρόνου;

- Η εκτέλεση ορίζεται από το γεγονός (event driven). Ανταποκρίνεται άμεσα σε εξωτερικά ερεθίσματα,
- Υψηλό, πολλές φορές καταστροφικό το κόστος σφάλματος,

- Πολυπρογραμματισμός (Multiprogramming), παράλληλη εκτέλεση διεργασιών (multitasking, ή multiprocessing),
- Ικανά για αξιόπιστη εκτέλεση και αυτάρκεια για μεγάλο χρονικό διάστημα,
- Αξιοπιστία. Πρόνοια για έλεγχο και αυτόματη εκσφαλμάτωση λαθών/εξαιρέσεων λογισμικού,
- Ντετερμινιστική συμπεριφορά.

4.2.2.2 Παράδειγμα Συστήματος Πραγματικού Χρόνου

Πολλά συστήματα πραγματικού χρόνου είναι συστήματα ελέγχου, δηλαδή ακολουθούν το μοντέλο ενός αισθητήρα – ενός ενεργοποιητή (One-Sensor, One-Actuator).



Εικόνα 4-2 Σύστημα Πραγματικού Χρόνου

Βασική Λειτουργία:

```

>> set timer to interrupt periodically with period T;
>> at each time interrupt do{
>>   do   analog-to-digital conversion to get y;
>>   compute control output u;
>>   output u and do digital-to-analog conversion;
>>end do;
    
```

Παρατηρήσεις

- Η μονάδα T είναι η περίοδος δειγματοληψίας και αποτελεί παράμετρο σχεδίασης κλειδί.
- Τυπικό εύρος για το T : δευτερόλεπτα, ή μιλιδευτερόλεπτα.
- Η προηγούμενη ρουτίνα πρέπει να ικανοποιεί το χρονικό περιθώριο που καθορίζεται από το T .

4.2.2.3 Πιο Περίπλοκα Συστήματα Πραγματικού Χρόνου

Multi-Rate Συστήματα Ελέγχου

- Είναι πιο περίπλοκα συστήματα ελέγχου και διαθέτουν πολλαπλούς αισθητήρες και ενεργοποιητές. Πρέπει να υποστηρίζουν βρόχους ελέγχου ποικίλλων ρυθμών. Παράδειγμα αποτελεί ο ελεγκτής πτήσης ελικοπτέρου.
- Πολλαπλές μεταβλητές που ελέγχουν λειτουργίες I/O σε διάφορες συχνότητες / ρυθμούς.

Ιεραρχικά Συστήματα Ελέγχου

- Κάθε μεταβλητή είναι βασισμένη στην επόμενη.
- Μπορεί να είναι και Multi-Rate σύστημα.

4.2.2.4 Παράμετροι Διεργασιών Λειτουργικών Συστημάτων Πραγματικού χρόνου

- Χρόνος Απελευθέρωσης (Release Time) r_i διεργασίας. Η χρονική στιγμή στην οποία η διεργασία γίνεται έτοιμη για εκτέλεση.

- Χρόνος Αποπεράτωσης (Completion Time) διεργασίας: Η χρονική στιγμή στην οποία η διεργασία ολοκληρώνει την εκτέλεση της.
- Απόλυτη Προθεσμία (Absolute Deadline) d_i διεργασίας: Η χρονική στιγμή μέχρι την οποία η διεργασία πρέπει να έχει ολοκληρώσει την εκτέλεση της.
- Σχετική Προθεσμία (Relative Deadline) D_i διεργασίας:

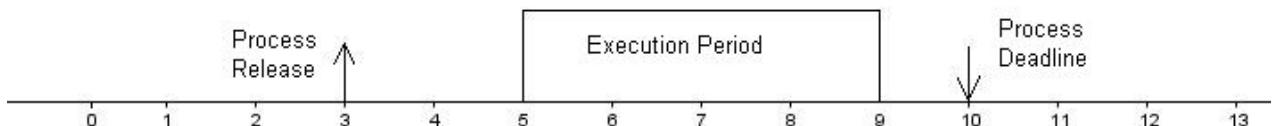
$$\text{Rel. Deadline} = \text{Abs. Deadline} - \text{Release Time} \quad (18)$$

- Χρόνος απόκρισης (Response Time) διεργασίας:

$$\text{Response Time} = \text{Completion Time} - \text{Release time} \quad (19)$$

Κάθε διεργασία χαρακτηρίζεται από το χρόνο απελευθέρωσης της r_i , την απόλυτη προθεσμία της d_i , τη σχετική προθεσμία της D_i και το χρόνο εκτέλεσης της e_i .

Εφαρμογή



Εικόνα 4-3 Εφαρμογή Διεργασίας Πραγματικού Χρόνου

Από το σχήμα παρατηρούμε ότι:

- Χρόνος Απελευθέρωσης διεργασίας: 3,
- Η απόλυτη προθεσμία της διεργασίας είναι τη χρονική στιγμή 10,
- Η σχετική προθεσμία της διεργασίας είναι 7,
- Ο χρόνος απόκρισης είναι 6,
- Ο χρόνος εκτέλεσης είναι 4.

4.2.2.5 Είδη Διεργασιών Λειτουργικών Συστημάτων Πραγματικού Χρόνου

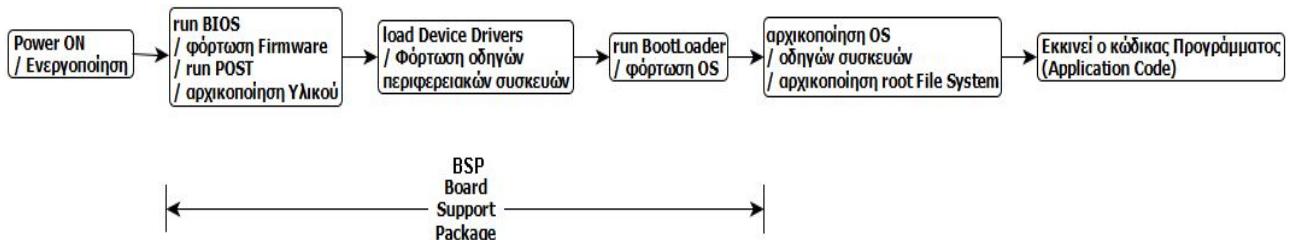
Οι διεργασίες των RTOS διακρίνονται σε περιοδικές, σποραδικές και απεριοδικές.

- Περιοδική Διεργασία:
 - Συνδέουμε κάθε διεργασία p_i με μια περίοδο P_i
 - P_i είναι το χρονικό διάστημα μεταξύ διαδοχικών απελευθερώσεων της διεργασίας.
- Σποραδική και Απεριοδική Διεργασία:
 - Σποραδική: Διαθέτει σκληρή προθεσμία (hard deadline).
 - Απεριοδική: Δεν έχει προθεσμία, ή έχει χαλαρή προθεσμία (soft deadline).

4.2.3 Πώς Εκκινεί ένα Υπολογιστικό Σύστημα

- 1) Αρχικώς δέχεται αίτημα επαγρύπνησης (Wake Up Call / Power ON),
- 2) Εκτέλεση οδηγιών του προγράμματος BIOS (Basic Input Output System). Το πρόγραμμα BIOS βρίσκεται σε τμήμα μητ-πιπερικής μνήμης συχνά ROM, συνήθως επί της μητρικής κάρτας του Η/Υ. Μια σημαντική διαδικασία που επικαλείται το BIOS είναι το POST (Power On Self Test), στο οποίο ελέγχονται ο επεξεργαστής ή CPU (ή οι CPU's για πολλαπλούς πυρήνες, ή επεξεργαστές), η μνήμη, οι περιφερειακές συσκευές για σφάλματα και αποθηκεύει τα αποτελέσματα σε μια ειδική θέση μνήμης. Έπειτα ο έλεγχος επιστρέφει πίσω (στη συνάρτηση του) στο BIOS το οποίο, αν δεν υπήρξαν σφάλματα με τη διαδικασία POST, προσαρμόζει το σύστημα χρονισμού (Clock setup) και θέτει παραμέτρους σχετικά με τη μνήμη Cache κα. Γενικότερα το BIOS αρχικοποιεί το υλικό.

- 3) Φορτώνονται οι οδηγοί περιφερειακών συσκευών, για να τεθούν έπειτα σε χρήση από το εκάστοτε OS.
- 4) Τρέχει το πρόγραμμα Bootloader, ή Bootstrap loader, το οποίο φορτώνει το λειτουργικό σύστημα.
- 5) Αρχικοποιείται το OS, ο πυρήνας του OS (Kernel), οι οδηγοί συσκευών και το πρωταρχικό σύστημα αρχείων (Root File System – RFS). Το σύστημα αρχείων χρησιμοποιείται για τον έλεγχο της αποθήκευσης, της μορφοποίησης στα αποθηκευτικά μέσα και της ανάκτησης των δεδομένων. Το RFS είναι αυτό στο οποίο επικάθονται όλα τα άλλα συστήματα αρχείων και μπορεί να φανεί στον αρχικό κατάλογο (Root directory) σε συστήματα τύπου UNIX.
- 6) Από εκεί και έπειτα ο χρήστης επιλέγει τι θα συμβεί. Ο κώδικας ενός προγράμματος τρέχει όταν ο χρήστης εκκινεί κάποιο πρόγραμμα. Παράλληλα με το χρήστη τρέχει ο πυρήνας του λειτουργικού συστήματος για να τον επικουρεί κατά τη λειτουργία του υπολογιστή.



Εικόνα 4-4 Αλληλουχία ενεργειών μετά την ενεργοποίηση Υπολογιστικού Συστήματος

Τα βήματα από το 2 έως το 4 μπορεί να συγκαταλέγονται στη λεγόμενη διαδικασία Πλατφόρμα Υποστήριξης Συσκευασίας (Board Support Package – BSP), τα όρια της οποίας δεν είναι αυστηρά καθορισμένα. Το BSP είναι ένα συστατικό του H/Y αρμόδιο για να παράσχει στο λειτουργικό σύστημα τις ιδιαίτερες λεπτομέρειες για την χρησιμοποιούμενη πλατφόρμα υλικού. Το OS θα χρησιμοποιήσει αυτές τις πληροφορίες για να προσφέρει μηχανισμούς αφαιρέσης υλικού (hardware abstractions) στις διεργασίες που κάνουν χρήση των υπηρεσιών του. Το BSP είναι μοναδικό τόσο για την πλατφόρμα υλικού, όσο και για το OS στο οποίο χρησιμοποιείται. Αναφερόμαστε σε αυτό αντί για όλα τα ξεχωριστά ζηματικά από το 2) έως το 4). Μετά την ολοκλήρωση του BSP αρχικοποιείται το OS και ο έλεγχος περνά στον πυρήνα του λειτουργικού συστήματος (Kernel).

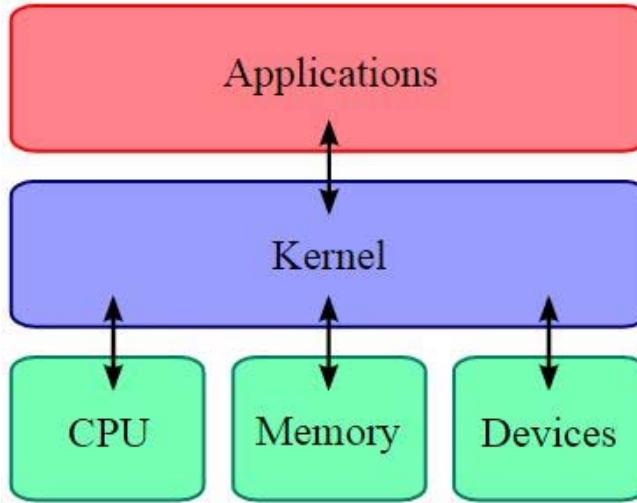
4.2.4 Πυρήνας του Λειτουργικού Συστήματος

Το Kernel είναι ένα πρόγραμμα υπολογιστή που συγκροτεί τον κεντρικό πυρήνα του OS. Διαθέτει απόλυτο έλεγχο για οτιδήποτε συμβαίνει στον H/Y. Είναι το πρώτο πρόγραμμα λογισμικού (software program) που φορτώνεται κατά την εκκίνηση και είναι υπεύθυνο για το υπόλοιπο κομμάτι της έναρξης. Επίσης διευθύνει τη μνήμη, τους διαύλους διεύθυνσης και δεδομένων και την επικοινωνία μεταξύ περιφερειακών συσκευών. Το Kernel αποτελεί θεμελιώδες συστατικό ενός σύγχρονου λειτουργικού συστήματος. Επιπλέον είναι η πιο συχνά χρησιμοποιούμενη μερίδα του OS, εδρεύει μονίμως στη κύρια μνήμη και τρέχει πάντα με προνομιούχο τρόπο (privileged mode). Το kernel απαντά στις αιτήσεις των διεργασιών για Είσοδο/Εξόδο (I/O) ή επεξεργασία, που αναφέρονται ως σήματα διακοπής του λογισμικού (Software Interrupts – SWI), και τις μεταφράζει σε οδηγίες επεξεργασίας για τη CPU, καθώς και στα σήματα διακοπής από τις περιφερειακές συσκευές – σήματα διακοπής του υλικού (Hardware Interrupts – HWI).

Ποιές είναι οι αρμοδιότητες του Kernel; Εν συντομίᾳ το kernel

- Διοικεί τις διεργασίες,
- Πραγματοποιεί αλλαγή πλαισίου περιεχομένου (Context Switch), δηλαδή εναλλάσσεται καταλλήλως μεταξύ των διεργασιών, σημάτων διακοπής και άλλων κλήσεων.

- Χρονοπρογραμματισμός (Scheduling), αποφασίζει ποια διεργασία είναι η επόμενη που πρέπει να τρέξει. Υπάρχουν πολλοί πιθανοί αλγόριθμοι χρονοπρογραμματισμού που εφαρμόζονται.
- Ελέγχει τους κρίσιμους τομείς (Critical Sections). Ο κρίσιμος τομέας είναι το τμήμα ενός προγράμματος το οποίο δε μπορεί να εκτελεστεί από περισσότερες από μία διεργασίες ταυτόχρονα. Προσφέρει επαρκής προστασία της μνήμης (memory protection) όταν πολλαπλές διεργασίες/κλήσεις αιτήσουν πρόσβαση σε έναν κρίσιμο τομέα. Γενικά το kernel προσφέρει διάφορες επιλογές για την εργασία με τους κρίσιμους τομείς.



Εικόνα 4-5 Το Kernel συνδέει τις εφαρμογές με το υλικό του Η/Υ[55]

4.2.5 Προγράμματα, Εφαρμογές, Διεργασίες

Ένα υπολογιστικό πρόγραμμα (Computer Program) είναι μια παθητική συλλογή οδηγιών που συλλογικά εκτελούν ένα συγκεκριμένο έργο όταν το συγκεκριμένο πρόγραμμα εκτελεστεί. Ένας Η/Υ προϋποθέτει την ύπαρξη λειτουργικών προγραμμάτων και τυπικά οι οδηγίες του προγράμματος εκτελούνται από τη CPU. Μια εφαρμογή (Application/App) είναι όρος ταυτόσημος με το πρόγραμμα. Και οι δύο όροι χρησιμοποιούνται εναλλακτικά. Υπάρχει όμως μια διαφορά τεχνικής φύσεως. Ένα πρόγραμμα, ή ομάδα προγραμμάτων αναφέρεται με τον όρο εφαρμογή, όταν αυτή προορίζεται για τον τελικό χρήστη (End user), δηλαδή τον χρήστη που θα χρησιμοποιήσει το πρόγραμμα (εν αντιθέσει με αυτούς που το προγραμματίζουν, το συντηρούν, ή το υποστηρίζουν). Επίσης η εφαρμογή συνήθως απαιτεί ένα λειτουργικό σύστημα για να τρέξει, διότι ο τελικός χρήστης συνήθως δεν εμβαθύνει με γνώσεις προγραμματισμού Η/Υ. Κάθε εφαρμογή είναι ένα πρόγραμμα, αλλά κάθε πρόγραμμα δεν είναι εφαρμογή. Ένα πρόγραμμα (ή μια εφαρμογή) βρίσκεται αποθηκευμένη στη δευτερεύουσα μνήμη του συστήματος, δηλαδή στο σκληρό δίσκο, ή σε άλλα αποθηκευτικά μέσα. Τα συνήθη προγράμματα είναι προγράμματα λογισμικού. Υπάρχουν όμως και προγράμματα υλικού (Hardware Programs), σε σπάνιες πλέον περιπτώσεις, τα οποία χρησιμοποιούνται για να ελέγξουν κατευθείαν το υλικό του υπολογιστή, παρακάμπτωντας κάθε στρώμα αφαίρεσης υλικού.

Η διεργασία είναι ένα μοναδικό στιγμιότυπο που δημιουργείται όταν το πρόγραμμα εκτελείται. Με άλλα λόγια όταν ένα πρόγραμμα, ή συλλογή οδηγιών, εκτελείται τότε αναφέρεται ως διεργασία. Όταν ένα πρόγραμμα εκτελείται περισσότερες από μία φορές ταυτόχρονα τότε έχουμε πολλαπλά στιγμιότυπα του προγράμματος, ή πολλαπλές διεργασίες (του ίδιου προγράμματος). Για παράδειγμα όταν ανοίγουμε πολλαπλά παράθυρα επεξεργαστή κειμένου (Text Editor). Τότε αναφερόμαστε σε multi-process program και το OS ελέγχει για ακούσια αίτηση πρόσβασης σε κρίσιμους τομείς από περισσότερες της μιας διεργασίες του προγράμματος.

Για κάθε διεργασία αποδίδεται από το OS μια μοναδική Δομή Ελέγχου Διεργασίας (Process Control Block – PCB). Το PCB είναι μια δομή δεδομένων που ενυπάρχει στο kernel του OS η οποία συγκρατεί τις πληροφορίες σχετικές με τη διεργασία. Το PCB είναι ο τρόπος με τον οποίο εκδηλώνεται η διεργασία στο kernel του OS και διαδραματίζει πρωτεύοντα ρόλο στη διοίκηση της διεργασίας. Οι πληροφορίες αυτές που διαθέτει ποικίλουν ανάλογα με το OS εν χρήση, αλλά σε γενικές γραμμές είναι οι επόμενες:

Pointer	Κατάσταση Δ.
	Ταυτότητα Διεργασίας
	Μετρητής Προγράμματος
	Καταχωρητές
	Κατάλογος ανοιχτών αρχείων
	Προνόμια Διεργασίας
	I/O πληροφορίες
	...

Εικόνα 4-6 Οι πληροφορίες που περιέχει μια Δομή Ελέγχου Διεργασίας (PCB). Το πλήθος τους εξαρτάται από το λειτουργικό σύστημα σε χρήση

- Κατάσταση διεργασίας (Process State), δυνατές τιμές που μπορεί να λάβει: New, Ready, Running, Waiting, Halted, Terminated κλπ.
- Ταυτότητα διεργασίας (Process Identification – PID): ένας μοναδικός αναγνωριστικός αριθμός που αποδίδεται σε κάθε διεργασία από το OS.
- Πληροφορίες χρονοπρογραμματισμού (Scheduling Information): Παρέχει πληροφορίες χρονοπρογραμματισμού (δείτε αργότερα), όπως την προτεραιότητα της, το χρόνο που έχει παρέλθει από την τελευταία φορά που μπλοκαρίστηκε, ή έτρεξε κα.
- Μετρητής Προγράμματος (Program Counter) είναι ένας καταχωρητής της CPU που υποδεικνύει τα περιεχόμενα της CPU σε κάποια χρονική στιγμή, δηλαδή δείχνει πάντα στην παρούσα οδηγία του προγράμματος που εκτελείται.
- Καταχωρητές της CPU (CPU Registers) συλλαμβάνουν την παρούσα κατάσταση της διεργασίας. Ο καθένας χρησιμοποιείται για την αποθήκευση τιμών σημαντικών μεταβλητών της διεργασίας.
- Πληροφορίες σχετικά με τη μνήμη (Memory Management Information). Δείχνει είτε σε σελιδοποιημένους (paged) είτε σε τμηματοποιημένους (segmented) πίνακες μνήμης (memory tables).
- Λογιστικές πληροφορίες (Accounting Information), όπως memory limits, πραγματικός χρόνος που τρέχει η διεργασία, ποσοστό απασχόλησης της CPU από την διεργασία κα.
- Πληροφορίες Εισόδου / Εξόδου, όπως με ποιές περιφερειακές συσκευές I/O η συγκεκριμένη διεργασία έχει συνεργαστεί (αν υπάρχει κάποια τέτοια συσκευή), κατάλογο ανοιχτών αρχείων κα. Γενικά αυτό το τμήμα καταγράφει πληροφορίες σχετικά με τη χρήση υπολογιστικών πόρων από την διεργασία.
- Προνόμια/Δικαιώματα διεργασίας (Process Privileges), που περιέχει πληροφορίες σχετικά με τις δυνατότητες ελευθερίας της διεργασίας, δηλαδή τους υπολογιστικούς πόρους στους οποίους μπορεί να έχει πρόσβαση.

4.2.6 Πολυνημάτωση

Τα σύγχρονα λειτουργικά συστήματα (και τα RTOS) υποστηρίζουν την εκτέλεση συνήθως μεγαλύτερων προγραμμάτων σε πολλαπλά «νήματα» (Threads) μέσα σε μια διεργασία. Αυτός ο μηχανισμός ονομάζεται πολυνημάτωση (Multithreading) και θα πρέπει να υποστηρίζεται και από τον επεξεργαστή και από το OS. Ένα εκτελέσιμο νήμα περιλαμβάνει ένα μέρος οδηγιών από το πρόγραμμα για να εκτελεστεί. Η εκτέλεση σε διαφορετικό νήμα ορίζεται από τον προγραμματιστή για λόγους λειτουργικότητας και οργάνωσης. Μια διεργασία διαιρείται σε πολλαπλά νήματα, όπου σε κάθε ένα ξεχωριστό νήμα συνήθως τοποθετείται μια σημαντική λειτουργία του προγράμματος και μεταχειρίζεται από το OS με διαφορετικό, αλλά παρόμοιο τρόπο με την εκτέλεση μιας διεργασίας. Τα νήματα τρέχουν παράλληλα μοιραζόμενα τους πόρους του υπολογιστικού συστήματος (σύμφωνα με τον μηχανισμό Multitasking – για (ψευδο-)παράλληλη επεξεργασία πολλαπλών διεργασιών). Όπως και κάθε διεργασία, κάθε νήμα συνοδεύεται από μια Δομή Ελέγχου Νήματος (Thread Control Block – TCB), που περιλαμβάνει πληροφορίες χειριζόμενες από τον πυρήνα, όπως το Δείκτη μνήμης Stack (Stack Pointer), το Μετρητή Προγράμματος (Program Counter), την κατάσταση του νήματος (Thread State), τους απασχολούμενους καταχωρητές και μια μεταβλητή δείκτη στο PCB της διεργασίας που ανήκει.

4.2.7 Λειτουργικά Συστήματα και Διαχείριση Μνήμης

Υπάρχουν δύο φυσικά είδη μνήμης. Η κύρια μνήμη και η δευτερεύουσα μνήμη. Η κύρια μνήμη, είναι μια παραπλήσια, ταυτόσημη ονομασία με τη Μνήμη Τυχαίας Προσπέλασης (Random Access Memory – RAM). Η δευτερεύουσα μνήμη είναι ο σκληρός δίσκος και τα διάφορα εξωτερικά αποθηκευτικά μέσα – CD, DVD, άλλοι δίσκοι και κασέτες. Τα προγράμματα διαμένουν στη δευτερεύουσα μνήμη και όταν εκτελούνται περνούν στην κύρια μνήμη για να μπορούν να εκτελεστούν πολύ πιο γρήγορα. Ο χρόνος αναμονής που υπάρχει από τη στιγμή της εκτέλεσης ενός προγράμματος μέχρι τη χρονική στιγμή που αυτό τελικά θα ανοίξει, οφείλεται στο overhead που προκαλείται από το χρόνο μεταφοράς των τμημάτων κώδικα του προγράμματος από την δευτερεύουσα στην κύρια μνήμη.

Το λειτουργικό σύστημα διαχειρίζεται τη λεγόμενη εικονική μνήμη (Virtual Memory). Η εικονική μνήμη είναι συνοπτικά το άθροισμα της κύριας και της δευτερεύουσας μνήμης που φανομενικά διατίθεται ως κύρια μνήμη, την οποία μπορούν να χρησιμοποιήσουν οι διεργασίες που τρέχουν. Μεταφορά μεταξύ των δύο αυτών χώρων στη μνήμη συμβαίνει αυτομάτως όποτε είναι απαραίτητο. Πιο συγκεκριμένα η εικονική μνήμη είναι ένας μηχανισμός του OS που παρέχει δύο ιδιότητες:

1. Χρησιμοποιεί σελίδες μνήμης (pages) σκληρού δίσκου (Disk Pages) για να επεκτείνει τη ποσότητα φυσικής μνήμης. Αυτή η διαδικασία ονομάζεται σελιδοποίηση (Paging).
2. Ένα στρώμα αφαίρεσης για να δώσει την ψευδαίσθηση σε κάθε διεργασία ότι έχει πρόσβαση σε ένα συνεχόμενο χώρο διευθύνσεων και δρά ανεξάρτητα από κάθε άλλη. Αυτό αποτελεί πλεονέκτημα για τη διεργασία απλοποιώντας την εκτέλεση της και το χειρισμό της από το OS.

Ακόμα το λειτουργικό σύστημα εφαρμόζει τη μέθοδο της ανταλλαγής μνήμης (Swap Memory), που είναι ποσότητα δευτερεύουσας μνήμης που χρησιμοποιείται όταν η μνήμη RAM γεμίσει. Διεργασίες που εκτελούνται σε swap μνήμη καθυστερούν φυσικά πολύ περισσότερο.

Κατά τη λειτουργία μιας διεργασίας, στη μνήμη υπάρχουν δύο τμήματα κώδικα αφοσιωμένα για την εξυπηρέτηση της. Αυτά είναι το τμήμα του χρήστη (User Space) και το τμήμα του πυρήνα (Kernel Space). Για κάθε διεργασία κατανέμεται ποσότητα μνήμης και από τα δύο κομμάτια. Αυτός ο διαχωρισμός, περιορίζει τις δυνατότητες των προγραμμάτων να προσβάλλουν και να διαφθείρουν με κακόβουλο λογισμικό, ή άθελα τους τον H/Y, ή τις άλλες διεργασίες, ή το ίδιο το OS. Κάθε

διεργασία του χρήστη έχει πρόσβαση σε ένα μικρό μόνο μέρος του kernel διαμέσου της διεπαφής που εκτίθεται από τον ίδιο τον πυρήνα – τις Κλήσεις Συστήματος (System Calls). Όταν μια διεργασία επικαλεστεί μια κλήση συστήματος, τότε ένα σήμα διακοπής του λογισμικού ενημερώνει το kernel, το οποίο έπειτα επισπεύδει (kernel dispatch) την κατάλληλη ρουτίνα εξυπηρέτησης του σήματος διακοπής για να το εξυπηρετήσει. Από την άλλη μεριά το kernel έχει κανονικά πλήρης πρόσβαση σε όλη τη διαθέσιμη μνήμη. Το μεγαλύτερο μέρος του kernel space βρίσκεται κατανεμημένο στη μνήμη από την έναρξη του υπολογιστή και το μέγεθος του είναι ανεξάρτητο από το πόσες και ποιες εφαρμογές χρήστη τρέχουν. Κατά σύμβαση το kernel space γράφεται σε υψηλότερες διευθύνσεις μνήμης από το user space.

4.2.7.1 Νήματα και Διαχείριση Μνήμης

Όταν ένα εκτελέσιμο νήμα εκκινείται, κατανέμεται από το OS χώρος στη μνήμη Stack (που είναι ένας χώρος μνήμης που λειτουργεί με τη λογική Last In, First Out – LIFO, Τελευταίος Μέσα Πρώτος Έξω) για κάθε νήμα. Αυτός ο χώρος του συγκεκριμένου νήματος δε μοιράζεται ποτέ με το χώρο ενός άλλου νήματος. Εκτός από αυτόν τον αφοσιωμένο χώρο μνήμης για κάθε νήμα το OS αποδίδει μια περιοχή μνήμης για όλα τα νήματα, ή για ολόκληρη τη διεργασία που λέγεται Heap. Η μνήμη Heap αποτελεί γενική κατηγορία μνήμης η οποία κατανέμεται δυναμικά και τυχαία, πάντα κατά την εκκίνηση του προγράμματος.

4.2.8 Χρονοπρογραμματισμός Διεργασιών

Ο χρονοπρογραμματιστής (Scheduler) είναι ένα μέρος του πυρήνα του λειτουργικού συστήματος που αποφασίζει και διευθύνει την εναλλαγή των διεργασιών, ποιά διεργασία πρέπει να τρέξει και πότε. Ο μηχανισμός αυτός αναφέρεται με τον όρο χρονοπρογραμματισμός διεργασιών (Task Scheduling). Ο χρονοπρογραμματιστής διεργασιών χρησιμοποιεί έναν αλγόριθμο χρονοπρογραμματισμού, ο οποίος επιβάλει μια πολιτική μεταγωγής διεργασιών, βασισμένος σε διάφορα κριτήρια. Τα κριτήρια αυτά ποικίλουν, μερικά σημαντικά αξίζει να τα αναφέρουμε:

- Απασχόληση CPU: να διατηρεί τη CPU όσο το δυνατόν λιγότερο απασχολημένη,
- Διακίνηση δεδομένων (throughput): μεγιστοποίηση του αριθμού των διεργασιών που ολοκληρώνονται στη μονάδα του χρόνου
- Χρόνος εξόδου (turnout time): Ελαχιστοποίηση του χρόνου αναμονής της διεργασίας, δηλαδή το χρόνο που απαιτείται μεταξύ της παραίτησης και του τερματισμού της διεργασίας.
- Χρόνος απόκρισης (response time): Ελαχιστοποίηση του χρόνου απόκρισης, που χρησιμοποιείται σε αληλεπιδραστικές διεργασίες που δέχονται είσοδο από το χρήστη.
- Πραγματικός Χρόνος (Real Time): Όταν το σύστημα πρέπει να ανταποκρίνεται εντός κάποιου αυστηρού χρονικού περιθωρίου, για να αποτρέψει την πρόκληση καταστροφικών συμβάντων. Πολιτικές βασισμένες σε αυτό το κριτήριο χρησιμοποιούνται κατά κόρον στα λειτουργικά συστήματα πραγματικού χρόνου.

4.2.8.1 Ιδιότητες Διεργασιών

Υπάρχει ένας χρονιστής (Timer) ο οποίος τρέχει πάντα και είναι αυτός που, σε συνθήκες κανονικής λειτουργίας, ενημερώνει για το πότε πρέπει να γίνει η εναλλαγή των διεργασιών, καλώντας την ISR: “OS System Clock / Tick”. Όταν έρθει ή ώρα να γίνει αυτό θέτει ένα bit (Task Assertion Flag) στο τμήμα χρονοπρογραμματισμού, του PCB της διεργασίας.

Στα RTOS οι ιδιότητες που χαρακτηρίζουν κάθε διεργασία (για την απόδοση της λέξης «διεργασία», στα αγγλικά παρατήρησα ότι ο όρος “Task” συνηθίζεται για τα RTOS, ενώ το “Process” για τα τυπικά OS) είναι κυρίως τρείς:

- I. Διεύθυνση (Address): Η διεύθυνση στη μνήμη στην οποία αποθηκεύεται η διεργασία όταν τρέξει. Αυτό το πεδίο χρησιμοποιείται συνεχώς στον χρονοπρογραμματισμό διεργασιών, για τη μεταγωγή πλαισίου περιεχομένου.
- II. Προτεραιότητα (Priority): Όσο μεγαλύτερη είναι η προτεραιότητα της διεργασίας, με τόσο μεγαλύτερη συχνότητα θα την ελέγχει ο χρονοπρογραμματιστής. Διεργασίες υψηλότερης προτεραιότητας εκτελούνται πριν από διεργασίες μικρότερης προτεραιότητας και μπορούν να διακόψουν (Preempt) διεργασίες χαμηλότερης προτεραιότητας. Κατά αυξουσα σειρά προτεραιότητας στο TI-RTOS που χρησιμοποιούμε τα είδη νημάτων ενός προγράμματος είναι:
1. Αδρανές Νήμα (Idle Task) με προτεραιότητα 0,
 2. Νήματα (Tasks) με προτεραιότητες από 1 έως 16,
 3. Σήματα Διακοπής Λογισμικού (SWI) με προτεραιότητες από 0 έως 15 και
 4. Σήματα Διακοπής Υλικού (HWI) με προτεραιότητες από 0 έως 7.
- Υπογραμμίζεται το γεγονός ότι το RTOS τα θεωρεί όλα νήματα με ξεχωριστές προτεραιότητες. Αυτές είναι οι προτεραιότητες που κατανέμονται από προεπιλογή στο TI-RTOS, υπάρχει όμως η δυνατότητα διεύρυνσης τους με το διπλασιασμό τους. Το αδρανές νήμα εισάγεται αυτόματα από τον πυρήνα κάθε φορά που δεν τρέχει κάποιο νήμα και διαθέτει τη μικρότερη προτεραιότητα. Επίσης το TI-RTOS δίνει τη δυνατότητα κάποιο νήμα να λάβει αρνητική προτεραιότητα -1 και τότε απενεργοποιείται (τοποθετείται στην κατάσταση Inactive State).
- III. Πληροφορίες για τη μνήμη Stack: Η μνήμη Stack είναι ένας τύπος δεδομένων που εξυπηρετεί την τάχυστη συλλογή αντικειμένων, με δύο βασικές λειτουργίες, Push: όταν προσαρτούνται αντικείμενα στη Stack και Pop: όταν λαμβάνονται αντικείμενα από τη Stack. Όταν γίνεται μεταγωγή περιεχομένου από νήμα σε άλλο νήμα, στη Stack αποθηκεύονται και λαμβάνονται κάθε φορά οι τοπικές μεταβλητές του νήματος και οι τιμές των καταχωρητών της CPU που απασχολούσε. Όταν ο έλεγχος επιστρέψει στο αρχικό νήμα οι τιμές αυτές γίνονται POP πίσω στο νήμα για να συνεχίσει το έργο του.



Εικόνα 4-7 Ιδιότητες Νημάτων σε ένα RTOS

Παρόμοια με τα νήματα του RTOS, οι διεργασίες των OS ακολουθούν ίδια λογική και μπορεί να διαθέτουν ακόμη περισσότερες ιδιότητες.

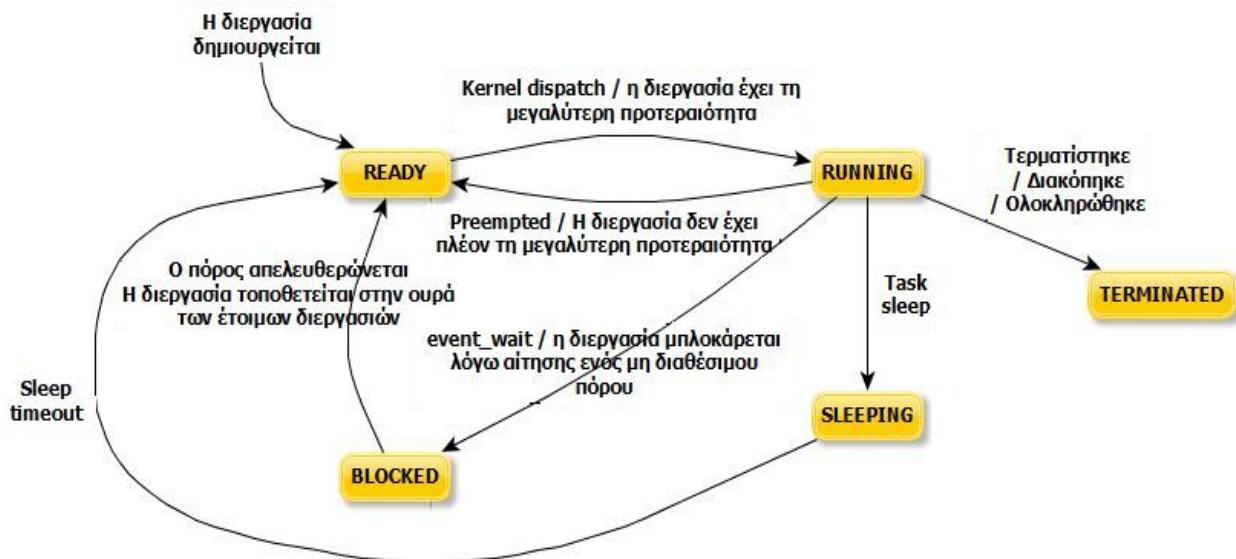
Επιπλέον υπάρχουν οι ακόλουθες ιδιότητες που αφορούν συγκεκριμένα το χρονοπρογραμματισμό των νημάτων λειτουργικών συστημάτων πραγματικού χρόνου:

- Ικανότητα χρονοπρογραμματισμού (Schedulability): Είναι η ικανότητα των διεργασιών να ικανοποιούν όλα τα σκληρά αιτήματα χρονικής προθεσμίας,
- Χρόνος αναμονής (latency) χαρακτηρίζει τη χειρότερη περίπτωση συστήματος για να ανταποκριθεί στο συμβάν.
- Δυνατότητα Ευστάθειας σε υπερφόρτωση (Stability in Overload) σημαίνει ότι το σύστημα ικανοποιεί τις αυστηρές χρονικές προθεσμίες (hard deadlines), ακόμα και αν όλες οι χρονικές προθεσμίες δεν μπορούν να προληφθούν.

4.2.8.2 Καταστάσεις Διεργασιών

Σε ένα υπολογιστικό σύστημα που υποστηρίζει πολυδιεργασία, οι διεργασίες και τα νήματα μπορούν να καταλάβουν μια ποικιλία καταστάσεων. Οι διάφορες καταστάσεις είναι σημαντικές για

την κατανόηση των λειτουργιών των διεργασιών. Το παρακάτω διάγραμμα πεπερασμένων καταστάσεων κάνει σαφή τη διάκριση μεταξύ τους.



Εικόνα 4-8 Μηχανή πεπερασμένων καταστάσεων των καταστάσεων διεργασιών

Για περισσότερη ανάλυση:

- Δημιουργήθηκε, ή Νέα (Created, ή New): Αυτή η ετικέτα χαρακτηρίζει τη διεργασία όταν πρωτοδημιουργείται. Σε αυτή τη φάση η διεργασία αναμένει άδεια για να προχωρήσει στην έτοιμη κατάσταση. Δεν αποτελεί κατάσταση διεργασίας, απλώς ένα γεγονός.
- Έτοιμη και Σε αναμονή (Ready and Waiting / Ready / Waiting): Η διεργασία έχει φορτωθεί στην κύρια μνήμη και αναμένει εκτέλεση από τη CPU. Κάθε έτοιμη διεργασία τοποθετείται σε ουρά ανάλογα με την προτεραιότητα της.
- Τρεχούμενη (Running): Μια διεργασία μεταπίπτει στην τρεχούμενη κατάσταση όταν επιλεγεί για εκτέλεση. Οι εντολές της διεργασίας εκτελούνται από τη CPU.
 - Τρόπος Πυρήνα (Kernel Mode)
 - Οι διεργασίες που εκτελούνται σε τρόπο πυρήνα έχουν πρόσβαση σε τμήματα μνήμης και του χρήστη και το πυρήνα.
 - Ελεύθερη πρόσβαση στο υλικό του υπολογιστή.
 - Διάφορες εντολές μιας διεργασίας, όπως εντολές I/O και halt, απαιτούν προνομιούχο τρόπο εκτέλεσης για να λειτουργήσουν.
 - Μια κλήση συστήματος (system call) προκαλεί το περιεχόμενο μιας διεργασίας που τρέχει σε τρόπο χρήστη, να αλλάξει σε τρόπο πυρήνα.
 - Τρόπος Χρήστη (User Mode)
 - Διεργασίες στον τρόπο χρήστη δεν έχουν πρόσβαση σε διευθύνσεις μνήμης αφοσιωμένες στον πυρήνα.
 - Όποτε ο χρήστης εκτελεί μια διεργασία αυτή εκκινεί και κανονικά βρίσκεται σε τρόπο χρήστη. Μεταπίπτει σε τρόπο πυρήνα μόνο εάν εκτελέσει μια κλήση συστήματος, σε περίπτωση που επιθυμεί να κάνει χρήση λειτουργιών I/O, ή άλλων προνομιούχων διαδικασιών.
- Παρεμποδισμένη (Blocked): Η διεργασία μπλοκαρίστηκε λόγω αιτήματος που το σύστημα δεν είναι έτοιμο να ανταποκριθεί. Η διεργασία μπλοκάρεται για να μην ξοδεύει άσκοπα χρόνο από τη CPU, ενώ το Kernel φέρνει στο προσκήνιο άλλη διεργασία για να εκτελέστει.

- Απενεργοποιημένη για προκαθορισμένο χρονικό διάστημα (Sleeping, ή Inactive). Αυτή δεν θεωρείται ξεχωριστή κατάσταση διεργασίας σε πολλά OS. Θεωρείται όμως στο TI-RTOS και FREE-RTOS.
- Τερματισμένη (Terminated): Η διεργασία είτε ολοκληρώνει την εκτέλεση της από το Running State, είτε τερματίζεται ρητά (Killed) για κάποιο λόγο. Αν πρόκειται για νήμα, ή για διεργασία που διαθέτει γονέα (Parent process), τότε παραμένει καταχωρημένη στον πίνακα διεργασιών ως ζόμπι διεργασία (Zombie process), μέχρι ο γονέας της να τερματιστεί, ή να συλλάβει το γεγονός τερματισμού της για να την τερματίσει ο ίδιος, αφαιρώντας τελικά τη καταχώριση απ' τον πίνακα.

Σε ένα λειτουργικό σύστημα πραγματικού χρόνου μια διεργασία δεν πρέπει τυπικά να τερματίζεται ποτέ. Αρκεί να εκτελεί περιοδικά κάποιο έργο. Γι' αυτό το λόγο σ'ένα RTOS θα μπορούσαμε να απλοποιήσουμε τις καταστάσεις από τέσσερις σε τρείς: Running, Ready και Blocked. Ένα νήμα ενός RTOS τυπικά ακολουθεί την επόμενη δομή:

```
void xTaskName ( void * pvParameters){  
    while (1) { // εφόσον δε θέλουμε να ολοκληρωθεί ποτέ  
        if ( condition)  
            do something;  
        else  
            do something else;  
    }  
}
```

4.2.8.3 Μεταγωγή Πλαισίου Περιεχομένου

Η Μεταγωγή Πλαισίου Περιεχομένου (Context Switch) είναι η διαδικασία που εκτελεί ο πυρήνας του OS για να μεταβάλλει την διεργασία που εκτελείται και να εισάγει μια άλλη. Η RUNNING διεργασία αντικαθίσταται με μια καινούργια. Το kernel αποθηκεύει το περιεχόμενο της «εξερχόμενης» διεργασίας στη Stack και επαναφέρει το περιεχόμενο της νέας διεργασίας. Υπενθυμίζουμε ότι κάθε διεργασία διαθέτει το δικό της χώρο στη stack και δεν υπάρχουν ποτέ αλληλοεπικαλύψεις. Με ακρίβεια οι ενέργειες που συμβαίνουν κατά τη μεταγωγή είναι οι ακόλουθες:

- Η παρούσα διεργασία διακόπτεται,
- Οι καταχωρητές της CPU για τη διεργασία αποθηκεύονται στη Stack στον αφιερωμένο για τη διεργασία πίνακα διεργασίας.
- Η διεργασία τοποθετείται στην ουρά των Ready – έτοιμων για εκτέλεση διεργασιών και αναμένει την επόμενη χρονοθυρίδα της.
- Στο PCB της διεργασίας αποθηκεύονται η χρήση μνήμης, επίπεδο προτεραιότητας κλπ.
- Η νέα διεργασία μεταπίπτει στην Running κατάσταση και οι εντολές της φορτώνονται στη CPU.
- Η νέα διεργασία τρέχει. Η εκτέλεση συνεχίζει κανονικά, δηλαδή:
 - Μεταβάλλεται κατάλληλα ο Stack Pointer, ο Program Counter και ο καταχωρητής κατάστασης του προγράμματος (Program Status Register) της νέας διεργασίας.

Είναι επόμενο ότι η διαδικασία μεταγωγής υπονοεί την ύπαρξη overhead που καθυστερεί την κανονική εκτέλεση των λειτουργιών του H/Y. Η μεταγωγή περιεχομένου δεν πρέπει να διακόπτεται από ένα σήμα διακοπής. Σε περίπτωση που διακοπεί τότε θα προκύψει μια ασυνέπεια (inconsistency), που πιθανόν να διαφθείρει αρχεία. Σε έναν τυπικό επεξεργαστή θα πρέπει να απενεργοποιούνται τα

σήματα διακοπής όταν συμβαίνει η διαδικασία της μεταγωγής πλαισίου περιεχομένου και να επαναενεργοποιούνται μόλις ολοκληρωθεί.

Πότε συμβαίνει μια αλλαγή πλαισίου περιεχομένου; Για κάθε διεργασία αφιερώνεται μια χρονοθυρίδα (Time Slice) που προσδιορίζει τη χρονική περίοδο που μια διεργασία πρέπει να τρέξει πριν συμβεί το context switch. Η έναρξη και λήξη της χρονοθυρίδας οδηγείται από σήματα διακοπής του υλικού (HWI) από έναν υψηλής ακριβείας χρονιστή του υπολογιστικού συστήματος. Όταν το kernel ειδοποιείται για το σήμα διακοπής αυτού του μετρητή, ο χρονοπρογραμματιστής πρέπει να αποφασίσει για το αν πρέπει να συμβεί μια μεταγωγή πλαισίου περιεχομένου. Αυτό σημαίνει ότι δε συμβαίνει context switch σε κάθε περιστατικό σήματος διακοπής!

Υπάρχει περίπτωση μια διεργασία που εκτελείται (βρίσκεται στη κατάσταση Running) μπορεί να διακοπεί απρόοπτα από μια άλλη διεργασία υψηλότερης προτεραιότητας. Τότε λέμε ότι η διεργασία διεκόπη πριν τη λήξη της χρονοθυρίδας της (Preemption). Τα λειτουργικά συστήματα πραγματικού χρόνου TI-RTOS και FREE-RTOS που χρησιμοποιούμε καθώς και τα περισσότερα σύγχρονα OS διαθέτουν Πυρήνα με Χρονοπρογραμματιστή Διακοπής Προτεραιότητας (Priority Preemptive Scheduler), έτσι νήματα υψηλότερης προτεραιότητας έχουν τη δύναμη να διακόπτουν νήματα μικρότερης προτεραιότητας. Αυτό το είδος εκτέλεσης αναφέρεται με τον όρο διακοπτική πολυδιεργασία (Preemptive Multitasking). Δε λειτουργούν με αυτό τον τρόπο όλοι οι χρονοπρογραμματιστές, αλλά στα πλαίσια της συγκεκριμένης εργασίας δε θα εξετάσουμε άλλους.

4.2.9 Αντιπαραβολή υλοποίησεων με RTOS και χωρίς RTOS

Ένα RTOS δεν είναι απολύτως απαραίτητο για να προγραμματίσουμε αποτελεσματικά το λογισμικό ενός ενσωματωμένου συστήματος. Καθώς όμως το μέγεθος και η πολυπλοκότητα του προγράμματος μεγαλώνει, θα έρθει η στιγμή που ο εύστοχος χειρισμός της εφαρμογής θα αποβεί όλο και πιο δύσκολο έργο. Οι υπηρεσίες που προσφέρει ένα RTOS θα αποβούν ευεργετικές για πολλούς λόγους όπως:

- ✓ Προσφέρει αφαίρεση (Abstraction) σε περίπλοκες λειτουργίες χρονισμού μεταξύ των διεργασιών, τις οποίες πλέον χειρίζεται ο χρονοπρογραμματιστής,
- ✓ Επεκτασιμότητα / Συντήρηση του κώδικα. Το λογισμικό αποκτά μεγαλύτερη ανεξαρτησία από το υποκείμενο υλικό. Γίνεται λιγότερο επιδεκτικό σε πιθανές μεταβολές του υλικού,
- ✓ Η εφαρμογή μπορεί να διαχειριστεί πιο αποτελεσματικά και διαισθητικά, όταν η κάθε διεργασία ανήκει σε ένα νήμα (task). Κάθε νήμα είναι μια οργανωμένη συλλογή εντολών που εκτελούνται σειριακά για να επιτελέσουν ένα συγκεκριμένο έργο. Κάθε νήμα έχει μια προκαθορισμένη προτεραιότητα που αντανακλά άμεσα την σπουδαιότητα του στην εφαρμογή,
- ✓ Αποδοτικός χρόνος αδράνειας. Οποτεδήποτε τρέχει το αδρανές νήμα γνωρίζουμε ότι ο επεξεργαστής δεν έχει κάποια άλλη εργασία να κάνει. Επίσης το idle task προσφέρει έναν πολύ αποδοτικό μηχανισμό τοποθετώντας τη CPU σε λειτουργία χαμηλής ισχύος, εξοικονομώντας ενέργεια.
- ✓ Έτοιμο, δοκιμασμένο, cross-platform (συνήθως) λογισμικό, στη διαθεσιμότητα του προγραμματιστή,
- ✓ Κάθε φορά που συναντάται μια blocking εντολή (δείτε και->Blocking) η υλοποίηση με RTOS θα διακόψει την εκτέλεση για να τρέξει ένα άλλο νήμα, ή για να εισέλθει σε λειτουργία χαμηλής ισχύος μέχρι το αίτημα που απαιτεί η εντολή να γίνει διαθέσιμο. Χωρίς RTOS, η εκτέλεση μπλοκάρεται (η CPU απασχολείται συνεχώς..) μέχρι να γίνει διαθέσιμη μια απάντηση στο αίτημα. Θα μπορούσαμε να παρομοιάσουμε τις δύο υλοποίησεις ως εξής:
 1. Προγραμματισμός χωρίς RTOS
 - >>while (not event){
 - >> wait for the event; [AKA poll]

- >>}
 - >>else
 - >> DO something;
2. Προγραμματισμός με RTOS
- >>do {
 - >> if (event)
 - >> DO something;
 - >> else
 - >> DO something else; (eg. εισαγωγή σε Low Power Mode)
 - >>}

✓ Προσφέρει πολλούς μηχανισμούς που μπορούν να διευκολύνουν σημαντικά συγκεκριμένες λειτουργίες που εξαρτώνται από την υλοποίηση, όπως

1. ουρές (Queues) για άμεση, ντετερμινιστική επικοινωνία (πχ. μεταφορά μηνυμάτων) μεταξύ νημάτων και διεργασιών,
2. μεταβλητές σηματοφόρου (Semaphore variables) για συγχρονισμό τμημάτων κώδικα,
3. μεταβλητές “Mutex” (Mutual - Excusion) που επιτρέπουν αμοιβαίο αποκλεισμό, δηλαδή η πρόσβαση σε έναν πολύτιμο, ή επικίνδυνο υπολογιστικό πόρο επιτρέπεται μόνο σε ένα νήμα (ή μέρος κώδικα) κάθε φορά,
4. Προηγμένες λειτουργίες διαχείρισης μνήμης.

Γίνεται προφανές ότι όλοι αυτοί οι λόγοι είναι δελεαστικοί για έναν προγραμματιστή όταν χρειάζεται να αναπτύξει μια μεγάλη εφαρμογή. Κάθε συγκεκριμένο OS σίγουρα δύναται να παρέχει επιπλέον δυνατότητες και καινοτομίες.

4.3 Πρωτόκολλα Επικοινωνίας

Τα πρωτόκολλα επικοινωνίας που χρησιμοποιούμε για τη ζεύξη τηλεμετρίας είναι τρία. Το πρωτόκολλο UDP, ήτο πρωτόκολλο TCP για τη μεταφορά των δεδομένων ασύρματα μέσω του CC3200 με το GCS και το πρωτόκολλο Mavlink για τη μεταφορά των αρχικών δεδομένων τηλεμετρίας από τον ελεγκτή πτήσης στο CC3200. Ένα πρωτόκολλο επικοινωνίας καθορίζει τους κανόνες της γλώσσας στην οποία πρέπει να υπακούουν οι δύο (ή περισσότερες) δικτυομένες οντότητες για να επικοινωνήσουν.

4.3.1 Το πρωτόκολλο UDP

Το πρωτόκολλο UDP (User Datagram Protocol) είναι ένα πρωτόκολλο επιπέδου εφαρμογής ορισμένο για χρήση με το πρωτόκολλο IP του επιπέδου δικτύου σύμφωνα με το RFC 768. Προσφέρει μια υπηρεσία καλύτερης προσπάθειας (best-effort) σε ένα τερματικό σύστημα (IP host). Οι υπηρεσίες επικοινωνίας που προσφέρει είναι μη-αξιόπιστες, διότι δεν παρέχει εγγύηση και ειδοποίηση για την διανομή των δεδομένων. Η απλότητα του UDP μειώνει το overhead που παράγεται από τη χρησιμοποίηση του πρωτοκόλλου και οι υπηρεσίες που παρέχει μπορεί σε αρκετές περιπτώσεις να είναι επαρκείς.

Το UDP[56] παρέχει ένα ελάχιστο, αναξιόπιστο, βασισμένο στην καλύτερη προσπάθεια μέσο μεταφοράς των μηνυμάτων του επιπέδου εφαρμογής. Επίσης δεν εγκαθιδρύει τη σύνδεση μεταξύ των δύο επικοινωνούντων κόμβων, δεν παρέχει τη λεγόμενη «χειραγία» ανάμεσα στις οντότητες αποστολής και λήψης επιπέδου μεταφοράς πριν σταλεί ένα πακέτο. Γι' αυτό το UDP λέγεται ασυνδεσμικό. Αναφέρεται ότι παρέχει μια υπηρεσία καλύτερης προσπάθειας επειδή απλώς εκπέμπονται τα δεδομένα και αυτά ελπίζουμε να φτάσουν στο στόχο τους! Παρ' όλα αυτά διατηρεί

μια πολύ αποδοτική επικοινωνία σε μερικές εφαρμογές, εάν δεν είναι κρίσιμης σημασίας να χαθεί ένα μέρος των προς μεταφορά δεδομένων.

Μια όλο και αυξανόμενη χρήση του UDP είναι ως πρωτόκολλο σήραγγας (Tunneling Protocol), σύμφωνα με το οποίο ένας κόμβος ενθυλακώνει τα πακέτα ενός άλλου είδους πρωτοκόλλου σε δεδομένο γράμματα UDP και τα αποστέλλει σε κάποιο άλλο κόμβο στο τούνελ, ο οποίος αφαιρεί την κεφαλίδα UDP και λαμβάνει τα αρχικά πακέτα που περιλαμβάνουν το φορτίο με τα μηνύματα. Τα τούνελ δημιουργούν εικονικές ζεύξεις που συνδέουν τοποθεσίες μακρινές μεταξύ τους στη φυσική τοπολογία του Internet, αλλά μπορούν να χρησιμοποιηθούν με αυτό το τρόπο για τη δημιουργία Εικονικών Ιδιοτικών Δικτύων (Virtual Private Network – VPN).

Ακόμη το UDP δεν προσφέρει ασφάλεια στην επικοινωνία. Οι εφαρμογές που πρέπει να προστατέψουν τις επικοινωνίες τους από υποκλοπές, αλλοιώσεις, ή πλαστογραφία θα πρέπει να παράσχουν ξεχωριστά υπηρεσίες ασφάλειας χρησιμοποιώντας επιπλέον μηχανισμούς. Για τις ανάγκες τις εφαρμογής μας δεν χρειαζόμαστε ασφάλεια και αξιοπιστία στη μετάδοση της πληροφορίας και παράλληλα το κόστος επιπλέον επεξεργασίας είναι ακριβό στα ενσωματωμένα συστήματα, οπότε το πρωτόκολλο UDP αποτελεί μια έξοχη επιλογή.

4.3.2 Το πρωτόκολλο TCP

Το πρωτόκολλο TCP (Transmission Control Protocol) είναι ένα βασικό πρωτόκολλο του πρωτοκόλλου διαδικτύου (Internet Protocol – IP). Δημιουργήθηκε μαζί με το IP και συμπλήρωνε τη λειτουργία του και γι' αυτό έγινε γνωστό ως TCP/IP. [57] Το TCP παρέχει ένα μοντέλο αξιόπιστου καναλιού. Μ' ένα αξιόπιστο κανάλι, τα μεταφερόμενα bit δεδομένων δεν αλλοιώνονται (δεν αλλάζουν από 0 σε 1, ή το αντίστροφο, δεν χάνονται και παραδίδονται όλα με τη σειρά στην οποία εκπέμπονται. Αυτό είναι ακριβώς το μοντέλο υπηρεσίας που προσφέρει το TCP. Με άλλα λόγια όταν στέλνονται δεδομένα χρησιμοποιώντας το πρωτόκολλο TCP, και έχει ήδη εγκαθιδρυθεί η σύνδεση από άκρο σε άκρο, τότε μπορούμε να είμαστε πραγματικά σίγουροι ότι τα πακέτα θα ληφθούν από τον παραλήπτη. Τονίζεται όμως ότι το TCP είναι ένα πρωτόκολλο αξιόπιστης μεταφοράς δεδομένων, το οποίο υλοποιείται επάνω σ' ένα αναξιόπιστο (IP) επίπεδο δικτύου.

Το TCP λέγεται ότι είναι συνδεσμικό (Connection-Oriented protocol), επειδή πριν μπορέσει να αρχίσει μία διεργασία εφαρμογής να στέλνει δεδομένα σε μία άλλη, οι δύο διεργασίες πρέπει πρώτα να κάνουν «τρίδρομη χειραψία» μεταξύ τους, δηλαδή πρέπει να στείλουν κάποια προκαταρκτικά μηνύματα η μια στην άλλη για να καθορίσουν τις παραμέτρους της επικείμενης μεταφοράς δεδομένων. Μια διεργασία δε χρειάζεται να γνωρίζει τους ακριβείς μηχανισμούς για αποστολή δεδομένων μέσω μιας ζεύξης σε έναν άλλο κόμβο, όπως ο κατακερματισμός και η ενθυλάκωση των πακέτων στο μέσο μεταφοράς. Στο επίπεδο μεταφοράς, το πρωτόκολλο TCP διαχειρίζεται τις χειραψίες και τις λεπτομέρειες μετάδοσης και προσφέρει ένα επίπεδο αφαίρεσης (Abstraction) της σύνδεσης για το επίπεδο της εφαρμογής.

Στα χαμηλότερα επίπεδα της στοίβας πρωτοκόλλων OSI (Open Systems Interconnect), λόγω της συμφόρησης του δικτύου, μπορεί να συμβεί υπερφόρτωση των πακετών. Οπότε εξαιτίας της κίνησης, ή κάποιας άλλης απροσδόκητης συμπεριφοράς, τα πακέτα IP μπορεί να χαθούν, να αντιγραφούν ή να παραδοθούν με λάθος σειρά. Το TCP ανιχνεύει όλα αυτά τα προβλήματα διότι αριθμεί το κάθε πακέτο δεδομένων, εισάγοντας στην κεφαλίδα TCP ένα νέο πεδίο, με τίτλο «Αριθμός Ακολουθίας» (Sequence Number). Ο παραλήπτης TCP αρκεί λοιπόν να ελέγχει αυτόν τον αριθμό ακολουθίας για να καθορίσει εάν το ληφθέν πακέτο είναι μια αναμετάδοση. Επίσης για κάθε πακέτο που λαμβάνεται από τον παραλήπτη στέλνεται ένα πακέτο αποδοχής προς τον αποστολέα (για το τελευταίο παραδοθέν πακέτο) που τον ενημερώνει για την επιτυχή παράδοση. Το πακέτο αποδοχής λέγεται ACK (Acknowledged – παρελήφθη). Εάν περάσει κάποιο χρονικό διάστημα και ο

παραλήπτης δεν λάβει κάποιο επόμενο στη σειρά πακέτο που έπρεπε (ενώ μπορεί να λαμβάνει επόμενα) στέλνει μήνυμα NAK (Not Acknowledged) στον αποστολέα για να τον ενημερώσει ότι δεν έλαβε το πακέτο και ο αποστολέας το ξαναστέλνει.

Επίσης, το TCP δύναται να ανακάμψει από απώλεια πακέτου όσο το δυνατόν συντομότερα. Η προσέγγιση που συνήθως υιοθετείται είναι ο αποστολέας να επιλέγει κάθε φορά μια χρονική τιμή, που είναι πιθανή για απώλεια πακέτου στο συγκεκριμένο κανάλι, για να αποφασίσει πότε θα πρέπει να ξαναστείλει ένα πακέτο. Εφόσον δε λάβει κάποιο μήνυμα αποδοχής εντός του χρονικού περιθωρίου που τέθηκε τότε το πακέτο επαναμεταδίδεται. Υπάρχει πιθανότητα όμως, μέχρι να παρέλθει αυτός ο χρόνος να χαθεί κρίσιμη πληροφορία. Επίσης υπάρχει περίπτωση ένα πακέτο να υποστεί μία ιδιαίτερα μεγάλη καθυστέρηση. Και στις δύο περιπτώσεις ο αποστολέας ίσως να επαναμεταδώσει το πακέτο αυθαιρέτως, ακόμη και εάν ούτε το πακέτο, ούτε το ACK του έχουν χαθεί, απλώς έχουν καθυστερήσει πολύ. Αυτό εισάγει την πιθανότητα διπλότυπων πακέτων δεδομένων που ανιχνεύονται με ειδικά διπλότυπα μηνύματα αποδοχής (Duplicate Acknowledge packets) στο κανάλι αποδοχέα – παράληπτη.

Επιπλέον το πρωτόκολλο TCP διαθέτει πολλούς και περίπλοκους μηχανισμούς ανίχνευσης σφαλμάτων στο κανάλι και βεβαίωσης για την αποστολή των δεδομένων, όπως έλεγχο ροής (flow-control service), έλεγχο συμφόρησης (congestion control), έλεγχο παραθύρου λήψης (receive window control) κα. τα οποία όμως δε θα αναπτυχθούν περισσότερο σ' αυτήν την εργασία. Εμείς χρησιμοποιούμε το πρωτόκολλο TCP για την ενθυλάκωση των πακέτων στην ασύρματη επικοινωνία τηλεμετρίας που έχουμε εγκαταστήσει, ως εναλλακτική επιλογή στο UDP, παρόλο που δεν έχουμε ανάγκη τόσο καλή ποιότητα στην επικοινωνία. Η πλατφόρμα CC3200 όμως θα μπορούσε να χρησιμοποιηθεί σε άλλες εφαρμογές που απαιτούν αξιόπιστη, συνδεσμική επικοινωνία και θα μπορούσε να χρησιμοποιηθεί κανονικά το ίδιο πρόγραμμα, με πολύ μικρές αλλαγές πιθανόν μόνο στο μέγεθος των buffers που χρησιμοποιούνται για να αντανακλούν το μέγεθος των νέων εισερχόμενων πακέτων.

4.3.3 Το πρωτόκολλο MAVLINK

Τα δεδομένα τηλεμετρίας που εκπέμπονται από τον ελεγκτή πτήσης APM2.8 συμμορφώνονται με το πρωτόκολλο MavLink (Micro Aerial Vehicle LINK). Έπρεπε να εξετάσουμε τη δομή και ιδιαίτερα το μέγεθος (length) κάθε πακέτου για να προσαρμόσουμε αντίστοιχα τους buffers μας στον προγραμματισμό της επικοινωνίας. Το πρωτόκολλο MAVLINK εκτός από το APM 2.8 ακολουθείται και από τα προγράμματα GCS Mission Planner και APM Planner. Κάθε πακέτο MavLink έχει μήκος 17 byte και η δομή του είναι[58]:

- Μέγεθος μηνύματος = 17Bytes = 6B κεφαλίδα + 9B ωφέλιμο φορτίο + 2B άθροισμα ελέγχου (checksum)

Η κεφαλίδα αναλυτικότερα διακρίνεται στα εξής πεδία:

- Byte 0: κεφαλίδα μηνύματος (message header), που παίρνει πάντα την τιμή 0xFE,
- Byte 1: μέγεθος μηνύματος (message length),
- Byte 2: αριθμός ακολουθίας (sequence number), αρχίζει από το 0, φτάνει στο 255 και μετά επαναλαμβάνει τον κύκλο, ξεκινώντας πάλι από το 0 κτλ.,
- Byte 3: Ταυτότητα συστήματος αποστολής (System ID), δηλαδή περιέχει ένα αναγνωριστικό του υπολογιστικού συστήματος που στέλνει το μήνυμα, πχ. τη διεύθυνση IP,
- Byte 4: Ταυτότητα Συστατικού συστήματος (Component ID): Ποιό συστατικό/υπομονάδα του συστήματος στέλνει το μήνυμα. Αυτό το πεδίο δε χρησιμοποιείται,
- Byte 5: Ταυτότητα Μηνύματος (Message ID). Εδώ βρίσκεται το μήνυμα Heartbeat, που πιθανόν αποτελεί το πιο σημαντικό στοιχείο. Στέλνεται μια φορά το δευτερόλεπτο από το GCS, το οποίο

έπειτα προσδοκεί σε ένα “Heartbeat ACK” από τον ελεγκτή πτήσης. Εάν το GCS δεν το λάβει μετά τη λήξη του δευτερολέπτου η σύνδεση διακόπτεται και ξαναεπιχειρεί να στείλει το Heartbeat μήνυμα. Η σύνδεση θα επανέλθει όταν λάβει “Heartbeat ACK”, εντός δευτερολέπτου μετά την αποστολή του “Heartbeat”.

Το ωφέλιμο φορτίο είναι τα δεδομένα τα οποία πραγματικά θέλουμε. Το άθροισμα ελέγχου ανιχνεύει τυχόν σφάλματα. Εάν ένα πακέτο περιέχει σφάλματα το απορρίπτει. Δεν ειδοποιείται ο αποστολέας ούτε συμβαίνει κάποιο άλλο έξυπνο τρικ όπως σε πρωτόκολλα τύπου TCP. Το πεδίο checksum υπάρχει και στην κεφαλίδα του UDP. Δηλαδή εφόσον ενθυλακώνουμε, όπως θα δείτε αργότερα στον προγραμματισμό, κάθε πακέτο MavLink σε ένα πακέτο UDP τότε ελέγχονται για κάθε πακέτο δύο αθροίσματα ελέγχου, που ίσως αποτελεί πλεονασμό.

4.4 Προγραμματισμός του Ardupilot Mega με το Mission Planner

Χρησιμοποιήσαμε το λογισμικό Mission Planner για τον προγραμματισμό του ελεγκτή πτήσης APM 2.8 και για τη βαθμονόμηση των αισθητήρων του. Η έκδοση Mission Planner που χρησιμοποιήσαμε είναι η 1.3.33. Είναι μια διαδικασία 9 υποχρεωτικών βημάτων (και κάποιων επιπλέον για βελτιστοποίηση της απόκρισης και των επιδόσεων του σκάφους) για τη σωστή λειτουργία του ελεγκτή πτήσης και παρουσιάζεται παρακάτω.

4.4.1 Βασικές Ρυθμίσεις

Πριν συνδέσουμε το APM 2.8 στον H/Y πάμε CONFIG/TUNING >> Planner και πρέπει το κουτί "Reset APM on USB connect" να μην είναι επιλεγμένο. Αυτό θα μας αποβεί χρήσιμο ύστερα στην τηλεμετρία. Όταν έχουμε συνδεδεμένο το APM και με USB και με τη μονάδα τηλεμετρίας, τα δεδομένα τηλεμετρίας απορρίπτονται, προς προτίμηση της σύνδεσης με USB.

4.4.1.1 Εγκατάσταση FirmWare

Αρχικά εκτελούμε το πρόγραμμα Mission Planner. Πάμε INITIAL SETUP >> Install Firmware και επιλέγουμε το υλικολογισμικό για το Arducopter “APM:Copter V3.3 Quad”. Κατεβάζουμε το firmware και όταν είναι έτοιμο συνδέουμε τον ελεγκτή πτήσης APM 2.8 στον H/Y. Επιλέγουμε “Load Firmware” και μόλις ολοκληρωθεί, το APM κατέχει πλέον το firmware τετρακόπτερου. Αρκεί τώρα να το ρυθμίσουμε κατάλληλα.

4.4.1.2 Προσαρμογή Σκελετού Τετρακόπτερου

Επιλέγουμε INITIAL SETUP >> Frame Type και στην συνέχεια X διαμόρφωση, εφόσον ο σκελετός του δικού μας τετρακόπτερου σχηματίζει ένα X, και μετά πατάμε κλικ στο Load Params (Parameters). Ο ελεγκτής πτήσης αντιμετωπίζει διαφορετικά τα τετρακόπτερα διαφορετικών τύπων σκελετών.

4.4.1.3 Βαθμονόμηση Επιταχυνσιομέτρου

Πάμε Accel Calibration. Εδώ θα βαθμονομήσουμε το επιταχυνσιόμετρο, ώστε να μας παρέχει έγκυρες τιμές πορείας και προσανατολισμού στους τρείς άξονες, τις οποίες τιμές μπορούμε έπειτα να τις δούμε από τον κατάλογο FLIGHT DATA ως a_x , a_y και a_z . Τοποθετούμε τον ελεγκτή πτήσης σε ένα απόλυτα σταθερό και επίπεδο δάπεδο και πατάμε “Calibrate Level”. Τώρα το επιταχυνσιόμετρο γνωρίζει ποιο είναι το «επίπεδο». Επειτα πατάμε “Calibrate Accel” και ακολουθούμε τις οδηγίες που θα εμφανιστούν με ακρίβεια. Η ακριβολογία μας θα φανεί ύστερα στις μετρήσεις που θα παίρνουμε.

4.4.1.4 Βαθμονόμηση Μαγνητόμετρου

Πάμε “Compass”. Εδώ θα βαθμονομήσουμε την πυξίδα/μαγνητόμετρο. Εμείς επιλέγουμε την εξωτερική πυξίδα. Γι' αυτό αρχικά κάνουμε κλικ στο κουμπί “APM and External Compass”. Έπειτα φροντίζουμε να υπάρχει ένα «τικ» στα κουτιά “Enable compasses”, “Obtain declination automatically”, “Automatically learn offsets”, “Use this compass” και “Externally mounted”. Επίσης από το dropdown κατάλογο επιλέγουμε την παράμετρο ROTATION_ROLL_180. Αυτό αντιστρέφει την τιμή ROLL στην πυξίδα για να παρουσιάζεται όπως πρέπει. (Το υλικολογισμικό του Arducopter ολισθαίνει την τιμή της κατά 180°.) Τέλος πατάμε Live Calibration για να βαθμονομήσουμε την πυξίδα με όλες τις επιλεγμένες ρυμίσεις. Προσπαθούμε να πάρουμε όσο το δυνατόν περισσότερες μετρήσεις που να καλύπτουν την σφαίρα που τείνει να σχηματιστεί. Κατά συνέπεια περιστρέφουμε το APM κατά μήκος και των τριών του αξόνων x, y, z.

4.4.1.5 Βαθμονόμηση Ραδιοπομπού

Επιλέγουμε “Radio Calibration”. Εδώ θα γίνει η βαθμονόμηση της ραδιοεπικοινωνίας, ειδικότερα της απόκρισης των μοχλών του ραδιοπομπού Turnigy 9x. Πριν κάνουμε τη βαθμονόμηση απομένει μια κρίσιμη σημασίας λεπτομέρεια (που μας στοίχισε την καταστροφή μιας προπέλας). Το firmware του Copter αντιστρέφει την είσοδο ελέγχου για τη λειτουργία Pitch. Οπότε πρέπει να την αντιστρέψουμε πρώτα από τον ραδιοπομπό. Για να το κάνουμε αυτό με τον ραδιοπομπό Turnigy 9x:

- Πατάμε MENU για 2 δευτερόλεπτα περίπου και ερχόμαστε στον κεντρικό κατάλογο.
- Πατάμε FUNC SETTINGS -> REVERSE -> ELE(vator) και το θέτουμε ως REV, με τα κουμπιά UP/DN.
- Πατάμε MENU για να σώσουμε τις προτιμήσεις μας.

Τώρα μπορούμε να συνεχίσουμε με την βαθμονόμηση. Πατάμε από το Mission Planner το κουμπί “Calibrate Radio”. Μετακινούμε όλους τους μοχλούς σε όλα τα ακραία σημεία που μπορούν να βρεθούν. Το επαναλαμβάνουμε 2, 3 φορές για επικάλυψη τυχών σφαλμάτων. Μόλις είμαστε έτοιμοι ολοκληρώνουμε τη διαδικασία πατώντας στο Mission Planner το αντίστοιχο κουμπί. Τέλος απενεργοποιούμε τον πομπό.

4.4.1.6 Ρύθμιση Αντισφαλματικής Προστασίας

Επιλέγουμε “FailSafe”. Στο συγκεκριμένο πεδίο μπορούμε να ρυθμίσουμε διάφορα είδη αντισφαλματικής προστασίας και να επιλέξουμε μια καθορισμένη ενέργεια την οποία θα εκτελεί το μη επανδρωμένο όχημα σε περίπτωση που συμβεί κάποιο από αυτά. Υπάρχουν διάφορα είδη όπως προσγείωση, ή επιστροφή στη βάση / στο σημείο απογείωσης (Return to Launch), σε περίπτωση που χαθεί το σήμα ραδιοεπικοινωνίας, ή η τάση της μπαταρίας μειωθεί κάτω από το κατώφλι: “Low Battery” κα. Κάθε πηγή FailSafe πρέπει να διαθέτει τη μονάδα δέκτη GPS για να λειτουργήσει σωστά. Υπάρχουν τουλάχιστον 6 είδη διαφορετικών παραμέτρων που μπορούν να προκαλέσουν την ενέργεια του FailSafe. Όταν ένα γεγονός διεγείρει μηχανισμό (triggers) FailSafe χάνεται η δυνατότητα ελέγχου του οχήματος, γι' αυτό είναι καλό στην αρχή να απενεργοποιήσουμε όλες τις πηγές που μπορούν να προκαλέσουν FailSafe, διότι μπορεί να μας μπερδέψουν (σε εμάς συνέβησαν δύο ατυχήματα με αυτό, ευτυχώς όχι πολύ σοβαρά). Καλό είναι να το ενεργοποιήσουμε όταν θα έχουμε εκπαιδευτεί να χειρίζόμαστε καλά σε κοντινές αποστάσεις το μη επανδρωμένο όχημα.

4.4.1.7 Ρύθμιση Τρόπων Πτήσης

Επιλέγουμε “Flight Modes”. Αναφέραμε στον Πίνακας 1 τα διάφορα κανάλια που έχουν αποδοθεί στον ελεγκτή πτήσης. Υπάρχει μια μικρή λεπτομέρεια εδώ που πρέπει να σημειώσουμε. Η λίστα με τα κανάλια που αποδίδονται στο πίνακα αυτό είναι τα κανάλια που για τις εισόδους του ελεγκτή πτήσης APM. Δεν είναι ανάγκη όμως αυτά να είναι τα ίδια με τα κανάλια που

ενεργοποιούνται με τους μοχλούς του ραδιοπομπού. Όπως αναφέραμε το υλικολογισμικό του πομπού Turnigy 9x είναι πλήρως παραμετροποιήσιμο. Το ίδιο ισχύει και για την κατανομή των καναλιών του. Εμείς (όπως μπορείτε να δείτε και στο μπλόκ διάγραμμα στην **Error! Reference source not found.**) συνδέσαμε την έξοδο του καναλιού 6 του ραδιοδέκτη στην είσοδο 5 του ελεγκτή πτήσης. Για να ρυθμίσουμε τους τρόπους πτήσης συνδέουμε τις εξόδους του ραδιοδέκτη με τις εισόδους του ελεγκτή πτήσης. Συνδέουμε ως εξής

Έξοδος 1 Δέκτη ->	Είσοδος 1 ελεγκτή πτήσης
Έξοδος 2 Δέκτη ->	Είσοδος 2 ελεγκτή πτήσης
Έξοδος 3 Δέκτη ->	Είσοδος 3 ελεγκτή πτήσης
Έξοδος 4 Δέκτη ->	Είσοδος 4 ελεγκτή πτήσης
Έξοδος 6 Δέκτη ->	Είσοδος 5 ελεγκτή πτήσης

Πίνακας 2 Αντιστοίχηση Καναλιών ραδιοδέκτη με τις εισόδους ελεγκτή πτήσης APM2.8

Ο ελεγκτής πτήσης απαιτεί τα συγκεκριμένα πέντε κανάλια για την ραδιοεπικοινωνία. Επιπλέον κανάλια είναι προαιρετικά. Συγκεκριμένα εμείς θέσαμε

- Έξοδος 7 Δέκτη -> Είσοδος 7 ελεγκτή πτήσης,

για μια επιπλέον βοηθητική λειτουργία. Για ενεργοποίηση των επικουρικών καναλιών πρέπει να ενεργοποιήσουμε πρώτα το εκάστοτε κανάλι στον ραδιοπομπό μας.

Τώρα θα ρυθμίσουμε τους τρόπους πτήσης. Πιο συγκεκριμένα έχουμε δημιουργήσει 3 «μίξεις», χρησιμοποιώντας δύο μοχλούς του ραδιοπομπού για να παράγουμε έξοδο σε ένα μόνο κανάλι, το κανάλι 5 (τον ελεγκτή πτήσης), το οποίο ενεργοποιείται με το κανάλι 6 της ραδιοεπικοινωνίας. Πρόκειται ουσιαστικά για μια διαδικασία πολυπλεξίας ενός μοχλού 3 θέσεων και ενός μοχλού δύο θέσεων για να σχηματιστεί ένας εικονικός μοχλός των $2 \times 3 = 6$ συνολικών θέσεων. Αυτό το κάναμε για να μπορούμε να επιλέγουμε και τους 6 συνολικά τρόπους πτήσης που υποστηρίζονται χρησιμοποιώντας μόνο ένα κανάλι και όχι 3! (Εναλλακτικά θα μπορούσαμε να φτιάξουμε έναν μοχλό με 6 θέσεις.) Γι' αυτή τη διαδικασία κάναμε τα εξής:

- Έχουμε συνδεδεμένο φυσικά το APM με το Mission Planner και βλέπουμε τη στήλη Flight Modes. Έπειτα ενεργοποιούμε και τον ραδιοπομπό,
- Πατάμε MENU για 2 δευτερόλεπτα περίπου και ερχόμαστε στον κεντρικό κατάλογο,
- Βεβαιωνόμαστε ότι έχουμε επιλέξει Acro mode από τις ρυθμίσεις συστήματος. Για να το κάνουμε αυτό πάμε -> SYSTEM SETTINGS -> TYPE SELECT -> ACRO MODE Επιλέγουμε ACRO MODE πατώντας το πλήκτρο MENU μία φορά.
- Μετά από τον κεντρικό κατάλογο πάμε:
 - FUNC SETTINGS ->AUX-CH -> CH 5 ->GEAR (gyro stick) επιλογή (ή οποιοδήποτε άλλο transmitter stick που μας βολεύει)
 - και CH6 -> FLP(flaperon stick) επιλογή και MENU για αποθήκευση.
 - Μετά:FUNC SETTINGS-> PROG.MIX και φτιάχνουμε 3 mixes. Κάθε μίξη αντιστοιχεί σε μια διαφορετική θέση του διακόπτη και μπορεί να επιλέξει μέχρι και 2 τρόπους πτήσης. Σε κάθε μίξη θέτουμε τις ακόλουθες παραμέτρους: MIX #: STATE: ACTIVE, MASTER: GYR, SLAVE: FLPκαι
 - για MIX 1: , SW(ITCH): NOR(MAL) που αντιστοιχεί στο F.MODE διακόπτη στην ανώτερη(κοιτάει προς τα πάνω)θέση.Έχουμε τον διακόπτη FLP (πάνω δεξιά από το διακόπτη F.MODE) στην ανώτερη θέση (να κοιτάει προς τα πίσω). Έπειτα προσαρμόζουμε την τιμή DN RATE ώστε με τη συγκεκριμένη θέση των

δύο μοχλών να βρεθούμε στο τρόπο πτήσης 1. Παρατηρούμε στο Mission Planner πότε θα πρασινίσει η επιλογή του Flight Mode 1 καθώς μεταβάλλουμε το DNRATE, τότε θα βρισκόμαστε στον πρώτο τρόπο πτήσης και η ρύθμιση του DNRATE είναι η σωστή. Μετά πρέπει να ρυθμίσουμε και το UPRATE για να επιλέξουμε τον δεύτερο τρόπο πτήσης. Οπότε αλλάζουμε το φυσικό διακόπτη FLP στην άλλη θέση (κοιτάει προς τα μπροστά) και μεταβάλλουμε το UPRATE μέχρι να πρασινίσει το Flight Mode 2 στην οθόνη του Mission Planner (καλό θα είναι να αποδεχτούμε τις τιμές DNRATE και UPRATE που αντιστοιχούν στις όσο το δυνατόν πιο κεντρικές τιμές PWM για το κάθε τρόπο πτήσης, και να αποφεύγουμε τις ακραίες τιμές). Πατάμε το κουμπί MENU για να αποδεχτούμε τις ρυθμίσεις.

- Για MIX 2: Θέτουμε SW(ITCH): ID1 που αντιστοιχεί στο F.MODE διακόπτη στην κεντρική θέση, οπότε τον θέτουμε σε αυτή. Θέτουμε πάλι τις τιμές DNRATE για ανταπόκριση στο Flight Mode 3 με τον διακόπτη FLP προς τα πάνω και UPRATE για ρύθμιση του Flight Mode 4 με τον διακόπτη FLP προς τα κάτω.
- Για MIX 3: Θέτουμε SW(ITCH): ID2 που αντιστοιχεί στο F.MODE διακόπτη στην κατώτερη (κοιτάει προς τα κάτω) θέση. Θέτουμε πάλι τις τιμές DNRATE για ανταπόκριση στο Flight Mode 5 με τον διακόπτη FLP προς τα πάνω και UPRATE για ρύθμιση του Flight Mode 6 με τον διακόπτη FLP προς τα κάτω.

Τώρα πλέον έχουμε ρυθμίσει τη ραδιοεπικοινωνία και τους 6 συνολικά τρόπους πτήσης. Μπορούμε να επιλέξουμε τους τρόπους πτήσης της αρεσκείας μας[59]. Εμείς επιλέξαμε

Flight Mode 1 :	Stabilize
Flight Mode 2 :	AltHold
Flight Mode 3 :	Stabilize Simple Mode
Flight Mode 4 :	Loiter
Flight Mode 5 :	Stabilize
Flight Mode 6 :	Drift

Πίνακας 3 Τρόποι Πτήσης Τετρακόπτερου

Η επιλογή “Simple” στον τρόπο πτήσης επιτρέπει την τηλεκατεύθυνση του αεροσκάφους με τον προσανατολισμό αναφοράς να είναι πάντα αυτός στον οποίο ξεκίνησε το όχημα, που αναφέρεται ως HOME ή LAUNCH. Απαιτεί μόνο καλή ερμηνεία των τιμών της πυξίδας (m_x, m_y, m_z). Η επιλογή “Super Simple” στον τρόπο πτήσης επιτρέπει την τηλεκατεύθυνση του αεροσκάφους με αναφορά μόνο την κατεύθυνση από HOME. Αυτή η επιλογή απαιτεί καλή λήψη GPS (HDOP < 2.5). Μόλις επιλέξουμε τους τρόπους πτήσης πατάμε Save Modes. Γενικώς πριν την απογείωση είναι συνετό να στεκόμαστε σε μια ασφαλής απόσταση (>2m) πίσω από το τετρακόπτερο και η μύτη του να κοιτάει ακριβώς αντίθετα από εμάς. Κατά την πτήση καλό είναι να μην απομακρυνθούμε από τη θέση μας όταν έχουν επιλέξει simple, ή super simple mode για να μη χάσουμε τον προσανατολισμό μας.

4.4.1.8 Συλλογική Βαθμονόμηση Ηλεκτρονικών Ελεγκτών Ταχύτητας

Πάμε “ESC Calibration”. Σε αυτό το σημείο θα πρέπει να έχει ήδη γίνει η βαθμονόμηση του κάθε ESC ξεχωριστά, όπως περιγράφαμε στο εδάφιο «Ηλεκτρονικοί Ελεγκτές Ταχύτητας» του προηγούμενου κεφαλαίου. Εδώ θα ελέγξουμε ότι όλα τα ESC είναι συγχρονισμένα, δηλαδή όλοι οι κινητήρες εκκινούν την ίδια χρονική στιγμή. Επίσης σε αυτό το σημείο πρέπει να είμαστε έτοιμοι να τοποθετήσουμε όλα τα κομμάτια μαζί ώστε να λειτουργούν αρμονικά στο σύνολο τους. Δείτε το μπλοκ διάγραμμα στην **Error! Reference source not found.**, όπως και τον πίνακα 2.

Γι' αυτό το σκοπό συνδέουμε όλα τα ESC με τους κινητήρες (που κανονικά πρέπει να έχουμε N. Λαζαρίδης 97

Γι' αυτό το σκοπό συνδέουμε όλα τα ESC με τους κινητήρες (που κανονικά πρέπει να έχουμε ήδη συνδεδεμένα από προηγούμενες ενέργειες) και συνδέουμε την είσοδο σήματος των ESC με τις εξόδους 1, 2, 3, 4 του APM 2.8, όπως περιγράψαμε. Συνδέουμε το APM2.8 με το τροφοδοτικό του (APM Power Module), την είσοδο του τροφοδοτικού με την μπαταρία και την έξοδο ισχύος του με την πλακέτα διανομής ισχύος. Σε αυτό το προχωρημένο στάδιο θα πρέπει να έχουμε ήδη κολλήσει τις εισόδους των ESC με την πλακέτα διανομής ισχύος. Για αποφυγή τραυματισμών καλό είναι να μην έχουμε συνδεδεμένες τις προπέλες. Μετά ακολουθούμε όλες τις υπόλοιπες οδηγίες από την “ESC Calibration” όπως φαίνεται στο Mission Planner. Όταν ολοκληρώσουμε ελέγχουμε ότι όλοι οι κινητήρες ξεκινούν σε ομοφωνία εφαρμόζοντας Throttle από τον ραδιοπομπό.

Τέλος θέσαμε την παράμετρο MOT_SPIN_ARMED σε 0, για να μην εκκινούν οι κινητήρες αυτόματα μόλις οπλίζεται το τετρακόπτερο. Για μεταβολή των παραμέτρων πάμε στο Mission Planner CONFIG/TUNING >> Full Parameter List και στο πεδίο έρευνας γράφουμε την επιθυμητή παράμετρο.

4.4.1.9 Ευτρεπισμός Μοχλών του Ραδιοπομπού

Την πρώτη φορά που πετάξαμε το αεροσκάφος αντιμετωπίσαμε πολλά προβλήματα στον ευσταθή έλεγχο του, με αποτέλεσμα να εφαρμόζουμε συνεχώς τα κουμπιά ευτρεπισμού (trims) για το κάθε κανάλι. Αυτή η διαδικασία δεν ενδείκνυται για την ευστάθεια του τετρακόπτερου κάθε φορά. Το καλύτερο που μπορούμε να κάνουμε σε αυτή την περίπτωση είναι η εφαρμογή της λειτουργίας “Autotrim”. Για να την δοκιμάσουμε περιμένουμε για μια μέρα με ήπιο καιρό χωρίς ανέμους για καλύτερα αποτελέσματα. Θα οπλίσουμε το τετρακόπτερο σε τρόπο λειτουργίας Stabilize, αλλά αντί να έχουμε στραμμένο τον μοχλό του throttle προς τα δεξιά για 5 δευτερόλεπτα όπως κάνουμε κανονικά, θα τον κρατούμε στραμμένο για ~15 δευτερόλεπτα. Τότε η απογείωση του αεροσκάφους θα γίνει σε λειτουργία Autotrim. Κατά τη διάρκεια αυτής προσπαθούμε να διατηρήσουμε σταθερό το αεροσκάφος σε ύψος τουλάχιστον 2m (για αποφυγή δονήσεων και αναταράξεων με το έδαφος) για τουλάχιστον 30 δευτερόλεπτα (όσο περισσότερο, τόσο το καλύτερο), εφαρμόζοντας διάφορα trims από τον ραδιοπομπό. Όταν ολοκληρώσουμε προσγειώνουμε το όχημα και αναμένουμε 5 δευτερόλεπτα για να αποθηκευτούν οι ρυθμίσεις. Ο ελεγκτής πτήσης έχει προσαρμόσει όλα τα trims στις διάφορες παραμέτρους πτήσης του στις σωστές αναλογίες. Οπότε εμείς επανατοποθετούμε τις ρυθμίσεις trim στον ραδιοπομπό μας και ελέγχουμε τα αποτελέσματα του autotrim απογειώνοντας ξανά το όχημα σε τρόπο πτήσης Stabilize. Εάν οι διορθώσεις του Autotrim δεν μας έχουν αφήσει ικανοποιημένους επαναλαμβάνουμε τη διαδικασία κ.ο.κ.

4.4.2 Προχωρημένες Ρυθμίσεις

Αυτές οι ρυθμίσεις δεν είναι ακριβώς προχωρημένες, απλώς δεν είναι απαραίτητες. Αυτές που παρουσιάζονται εδώ όμως ήταν σημαντικό να γίνουν, καθώς παρέχουν σημαντική ευστάθεια αξιοπιστία στην πτήση του μη επανδρωμένου εναέριου οχήματος, αλλά και στις ζεύξεις τηλεμετρίας και ραδιοεπικοινωνίας. Σε αυτό το σημείο είχαν ήδη γίνει μερικές πτήσεις και είχε αποκτηθεί μια προκαταρκτική εξοικείωση με την πτήση του τετρακόπτερου. Ιδιαίτερη προσοχή τέθηκε στις παρακάτω διαδικασίες διότι πρέπει να συνδέουμε και τις προπέλες με τους κινητήρες, για να δούμε πόσο πραγματικά ρεύμα απορροφούν οι κινητήρες υπό φορτίο.

4.4.2.1 Βαθμονόμηση Τροφοδοτικού Ελεγκτή Πτήσης APM

Αρχικώς θα πρέπει να βαθμονομήσουμε τον αισθητήρας τάσης και ρεύματος που περιλαμβάνεται στο τροφοδοτικό ισχύος του ελεγκτή πτήσης APM. Για να γίνει αυτό θα χρειαστούμε ένα πολύμετρο και έναν μετρητή/αναλυτή ισχύος, ο οποίος να είναι εφοδιασμένος με συνδέτες XT-60 που χρησιμοποιούν τα υπόλοιπα ηλεκτρονικά. Χρησιμοποιήσαμε το πρακτικό μετρητή ισχύος και αναλυτή τάσης HK-010 της Hobbyking, που μπορεί να μετρήσει μέχρι και 6kW. Πρώτα

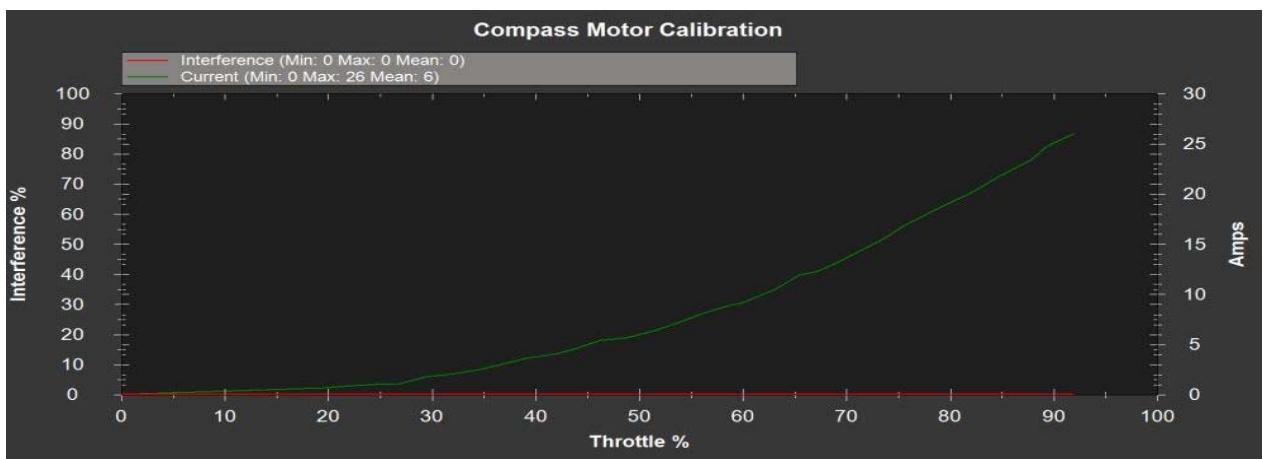
βαθμονομούμε την τάση. Μετράμε με το βιολτόμετρο την τάση της μπαταρίας. Μετά συνδέουμε τη μπαταρία με το τροφοδοτικό του APM. Συνδέουμε το APM με το Mission Planner και πάμε INITIAL SETUP >> Optional Hardware >> Battery Monitor και βλέπουμε τι τάση αναφέρει. Το πιο πιθανό είναι η τιμή που αναφέρει να αποκλίνει, οπότε τη διορθώνουμε σε αυτή που μόλις διαβάσαμε χρησιμοποιώντας το πολύμετρο.

Έπειτα βαθμονομούμε τον αισθητήρα ρεύματος του τροφοδοτικού. Συνδέουμε τον αναλυτή ισχύος με τη μπαταρία και τον ελεγκτή πτήσης APM. Ο αναλυτής ισχύος θα αναφέρει το ποσό ρεύματος που διέρχεται από αυτό και αποτελεί ένας ξειρετικό όργανο μέτρησης ρεύματος. Θα πρέπει να προσαρμόσουμε το τροφοδοτικό του APM, ώστε να υποδεικνύει τιμές πολύ κοντά σε αυτές του αναλυτή. Πάμε πάλι στη στήλη του Battery Monitor. Βλέπουμε ότι στο ρεύμα θα αναφέρει 0. Είναι αλήθεια ότι με μικρά ρεύματα (<2A) δεν είναι ακριβές το αισθητήριο όργανο ρεύματος του τροφοδοτικού APM. Οπότε πρέπει να αυξήσουμε Throttle (με τις προπέλες συνδεδεμένες αντίστροφα για να μην απογειωθεί το τετρακόπτερο) μέχρι στον αναλυτή ισχύος να αναφέρεται μια τιμή ρεύματος περίπου στα 10Amp. Διατηρούμε το Throttle ως έχει και μετά κοιτάμε στο Mission Planner να δούμε την τιμή που αναφέρεται εκεί. Το πιθανότερο είναι να αναφέρει περισσότερο ρεύμα από αυτό του αναλυτή, οπότε το διορθώνουμε μέχρι οι δύο τιμές να εξισωθούν σύμφωνα πάντα με την ένδειξη του αναλυτή ισχύος.

4.4.2.2 Βαθμονόμηση Πυξίδας & Κινητήρων

Είναι κρίσιμης σημασίας οι τιμές του μαγνητόμετρου m_x, m_y, m_z να είναι όσο το δυνατό πιο ακριβείς, αφού πολλές λειτουργίες του ελεγκτή πτήσης βασίζονται σε αυτές. Έχει αποδειχθεί ότι το πλήθος των παρεμβολών είναι ανάλογο της ποσότητας ρεύματος που απορροφάται από την πηγή ισχύος. Στην αρχή χρησιμοποιούσαμε την εσωτερική πυξίδα, δηλαδή αυτή που είναι συγκολλημένη στην πλατφόρμα του ελεγκτή πτήσης. Πήγαμε στο πρόγραμμα Mission Planner >> INITIAL SETUP >> Optional Hardware >> Compass/Motor Calib πατήσαμε Start (ακολουθούμε εδώ τις ίδιες οδηγίες προετοιμασίας που αναφέρονται στην προηγούμενη υποενότητα) για να δούμε την ποσότητα παρεμβολών και προβλήθηκε ένα γράφημα με πολύ υψηλά ποσοστά ηλ/κών παρεμβολών τα οποία άντως αντανακλούσαν και δικαιολογούνται από την όντως κακή ποιότητα πτήσης που είχαμε αρχικώς.

Γι' αυτό χρησιμοποιήσαμε την εξωτερική πυξίδα που βρίσκεται σε υπερυψωμένο σημείο (περίπου 14cm από το κοντινότερο καλώδιο ισχύος) μαζί με την μονάδα GPS και εκτελέσαμε πάλι τη διαδικασία. Από τον ραδιοπομπό αυξάναμε το Throttle σταδιακά μέχρι περίπου το 90%. Το καινούριο αποτέλεσμα αποτυπώνεται στο παρακάτω γράφημα.



Εικόνα 4-9 Βαθμονόμηση Μαγνητόμετρου – Κινητήρων. Στο γράφημα παριστάνεται το ποσοστό παρεμβολών σε συνάρτηση με την καταναλισκόμενο ρεύμα και την ένταση του Throttle

Παρατηρούμε απολύτως μηδενικές τιμές παρεμβολών, που είναι το τέλειο αποτέλεσμα στο οποίο ευελπιστούσαμε.

Επίσης λάβαμε και τις παρακάτω χρήσιμες πληροφορίες για την κατανάλωση ρεύματος από το τετρακόπτερο που πιθανόν να χρησιμεύσουν.

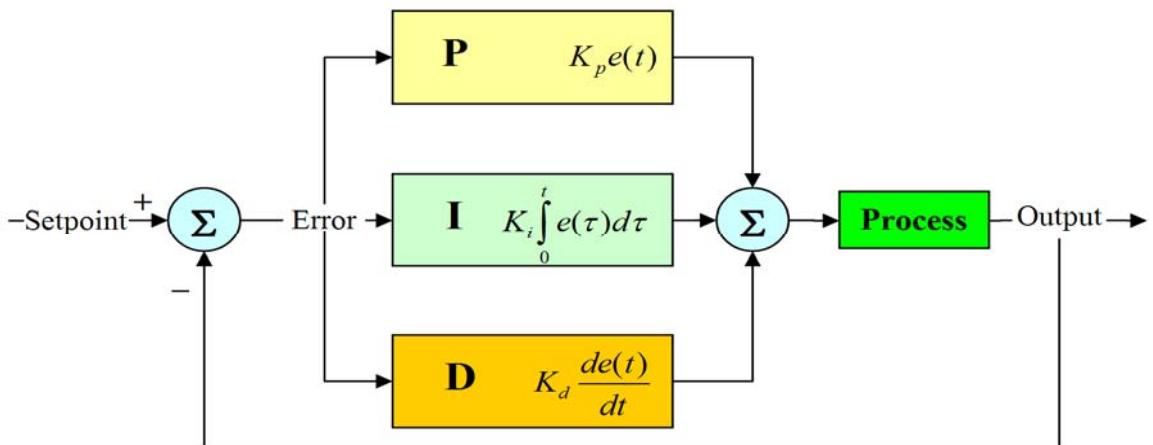
Throttle (%)	Κατανάλωση Ρεύματος (Amp)
16.666	3.6
33.333	6.9
50	11.5
66.666	16.3
83.333	21.5
100	22

Πίνακας 4 Αναγνώσεις κατανάλωσης ρεύματος τετρακόπτερου σε συνάρτηση με το εφαρμοζόμενο Throttle

[!Ιδιαίτερη **προσοχή** διότι οι δύο τελευταίες βαθμονομήσεις δεν είναι τόσο εύκολες και ασφαλείς όσο ίσως φαίνεται από την παρουσίαση !]

4.4.2.3 Ρύθμιση Ελεγκτών PID

Οι ελεγκτές PID (Proportional – Integral – Derivative) έχουν σχεδιαστεί για συστήματα ελέγχου, με απότερο σκοπό να εξουδετερώνουν/ελαττώνουν την ανάγκη συνεχούς προσοχής και συντήρησης τους από τον διαχειριστή του συστήματος. Ο ελεγκτής PID συνεχώς μετρά μια τιμή σφάλματος ως τη διαφορά μεταξύ μιας επιθυμητής τιμής και μιας μετρούμενης μεταβλητής. Έπειτα επιχειρεί να ελαχιστοποιήσει την τιμή σφάλματος, η οποία μπορεί να είναι η θέση μιας βαλβίδας, η ποσότητα ισχύος που παρέχεται σε μια θερμάστρα, ή η σταθεροποίηση του τρόπου λειτουργίας pitch που εφαρμόζεται από τον αυτόματο πιλότο APM! Ακολουθεί το γενικό διάγραμμα ενός ελεγκτή PID.



Εικόνα 4-10 Μπλοκ Διάγραμμα PID ελεγκτή[60]

Η έξοδος υπολογίζεται από το άθροισμα των συνιστώσων των τριών ελεγκτών. Αφού ο PID ελεγκτής βασίζεται στη μετρούμενη μεταβλητή της διαδικασίας και όχι στη γνώση των ιδιοτήτων της υποκείμενης διαδικασίας είναι ευρέως εφαρμόσιμος. Βρίσκοντας την κατάλληλη τιμή για τον κάθε ελεγκτή αποτελεί μια πολύ δύσκολη διαδικασία, γι' αυτό και δεν ασχοληθήκαμε ιδιαίτερα με τον συντονισμό των PID ελεγκτών του τετρακόπτερου. Από γενικότερες γνώσεις και μελέτες μας πάντως [61] σχετικά με τους PID ελεγκτές και πλέον στην εφαρμογή τους στο τετρακόπτερο,

καταλήξαμε στο συμπέρασμα ότι ο συντονισμός των PID μεταβλητών είναι μια συνάρτηση που εξαρτάται από πραγματικά πάρα πολλούς παράγοντες (στο hardware και στο software) τους οποίους και προσπαθήσαμε να βελτιώσουμε με ποικίλες μικροπαρεμβάσεις σε όλη τη διάρκεια του έργου μας. Τελικά οι τιμές των PID ελεγκτών που θέσαμε φαίνονται παρακάτω:



Εικόνα 4-11 Συντονισμός μεταβλητών PID στο τετρακόπτερο

Το λογισμικό του ελεγκτή πτήσης ArduCopter δίνει μια δυνατότητα αυτορρύθμισης των ελεγκτών PID με την εφαρμογή ενός νέου τρόπου πτήσης που ονομάζεται Autotune. Εμείς δεν χρησιμοποιήσαμε καθόλου αυτή τη δυνατότητα εξαιτίας δυσκολιών στην εύρεση αρκετού ελεύθερου διαθέσιμου χώρου που απαιτεί για την υλοποίηση της (ιδανικά τόσου χώρου όσο καταλαμβάνει ένα γήπεδο ποδοσφαίρου).

4.5 Προγραμματισμός του Μικροελεγκτή CC3200 με το Code Composer Studio

Ο κώδικας του CC3200 παρουσιάζεται στις επόμενες σελίδες. Επειδή είναι αρκετά μακρύς και πιθανών αρχικά να φανεί περίπλοκος, ως συμπλήρωμα των ήδη πολλών σχολίων που θα βρείτε στον κώδικα, θα προσπαθήσουμε εδώ να κάνουμε πιο κατανοητή τη ροή του, τμηματοποιώντας τον και εξηγώντας τις επιμέρους δομημένες εργασίες που επιτελεί. Θα διατεθούν όλα τα αρχεία του project στο Code Composer Studio μαζί με την εργασία.

Η επιδίωξη μας χρησιμοποιώντας το CC3200 είναι να μας παρέχει μια ασύρματη διεπαφή η οποία συνδέει τον ελεγκτή πτήσης APM 2.8 με το GCS και στέλνει τα δεδομένα τηλεμετρίας από το APM στο GCS, αλλά και λαμβάνει εντολές από το GCS που τις προωθεί στο APM. Στη συνέχεια εξηγούμε τι ακριβώς κάνει το CC3200.

Στην αρχή του προγράμματος, εκκινείται η πλατφόρμα CC3200, ενεργοποιούνται τα interrupts, ενεργοποιούνται και παραμετροποιούνται όλοι οι ακροδέκτες που θα χρησιμοποιηθούν, ενεργοποιούνται οι μονάδες UART, ο DMA controller, οι πηγές χρονισμού για Run Mode (δεν έχει

ρυθμιστεί καμιά λειτουργία χαμηλής ισχύος (Low Power Mode – LMP), προβάλλεται το banner της εφαρμογής, καταχωρούνται κάποια προκαταρκτικά interrupts τα οποία ενεργοποιούνται μετά τον χρονοπρογραμματιστή, φτιάχνονται τα Tasks για το TI-RTOS και εκκινεί ο χρονοπρογραμματιστής του. Στην αρχή τρέχει το Task που αρχικοποιεί τις ρυθμίσεις της συσκευής για την εφαρμογή μας και το οποίο δεν πρέπει να απασχοληθεί από τα άλλα tasks. Γι' αυτό στην αρχή του συγκεκριμένου Task απενεργοποιούμε τα άλλα νήματα και τα ξαναενεργοποιούμε μόλις ολοκληρωθεί.

Το CC3200 (στην πιο απλή μορφή που δεν δέχεται παραμέτρους εισόδου από H/Y) όταν εκκινεί, αναμένει 30 δευτερόλεπτα για να πατηθεί ένα από τα δύο κουμπιά SW2, ή SW3. Το πάτημα του κουμπιού SW2 θα σημάνει την επιθυμία του χειριστή να χρησιμοποιήσει το πρωτόκολλο UDP, ενώ το πάτημα του SW3 σημαίνει ότι θα χρησιμοποιηθεί το πρωτόκολλο TCP για την επικοινωνία τηλεμετρίας. Εάν πατηθεί το κουμπί SW2, τότε αναμένει άλλα 2 δευτερόλεπτα σε περίπτωση που πατηθεί ξανά το κουμπί SW2. Εάν δεν πατηθεί ξανά τότε το CC3200 θα λειτουργήσει ως κόμβος TCP client – πελάτης, επομένως το GCS πρέπει να είναι TCP server – διακομιστής. Εάν το κουμπί SW2 πατηθεί ξανά μέσα σε αυτό το χρονικό περιθώριο των δύο δευτερολέπτων τότε το CC3200 θα λειτουργήσει ως TCP server. Σε κάθε περίπτωση τα χρονικά περιθώρια ρυθμίζονται με διαφορετικούς χρονιστές γενικού σκοπού (General Purpose Timers). Εάν δεν πατηθεί κανένα κουμπί στην αρχή εντός 30 δευτερολέπτων τότε ανάβει το πορτοκαλί led και η εφαρμογή τερματίζεται. Γενικότερα, όποτε ανάβει το πορτοκαλί led, σημαίνει ότι η εφαρμογή έχει τερματιστεί και η συσκευή θα πρέπει να επανεκκινηθεί για να λειτουργήσει.

Μετά την επιλογή του πρωτοκόλλου ρυθμίζεται ο επεξεργαστής δικτύουσης του CC3200 στη προκαθορισμένη μορφή (ως Station mode και ρυθμίζονται οι πολιτικές αντιμετώπισης που ακολουθεί εντός του δικτύου), εκκινείται το CC3200 σε τρόπο (soft) AP και περιμένει τη σύνδεση ενός κόμβου. Ο κόμβος με το GCS λογισμικό συνδέεται στο CC3200 που έχει hard coded SSID: CC3200Thesis_N.L. και έπειτα το CC3200 κάνει ping στο τελευταίο. Μετά από μερικές ακόμη ρυθμίσεις όταν ολοκληρωθεί το ping σημαίνει ότι η επικοινωνία σε επίπεδο δικτύου μεταξύ των δύο έχει εγκαθιδρυθεί. Έπειτα αρχικοποιούνται και ρυθμίζονται τα sockets για το επιλεγμένο πρωτόκολλο[62][63], για τη διαμόρφωση της επικοινωνίας σε επίπεδο μεταφοράς.

- Για πρωτόκολλο UDP η full-duplex επικοινωνία και η μεταφορά των δεδομένων είναι πολύ απλή:
 1. Ανοιγμα Socket (port + IP),
 2. Bind Socket σε μια τοπική port + IP της συσκευής (σε AP τρόπο είναι 192.168.1.1) για να μπορεί να λαμβάνει δεδομένα σε μια hard coded port,
 3. Στέλνει επανειλημμένως πακέτα UDP και παράλληλα λαμβάνει UDP πακέτα.
- Για πρωτόκολλο TCP η επικοινωνία και η μεταφορά των δεδομένων έχει ως εξής:
 - Για λειτουργία ως TCP Client:
 1. Ανοιγμα Socket,
 2. Bind Socket σε τοπική port,
 3. Σύνδεση σε TCP Server, ο οποίος φυσικά πρέπει να είναι έτοιμος πριν από τον πελάτη (GCS->TCP server) (η τρίδρομη χειραγία ολοκληρώνεται),
 4. Εκπέμπει επανειλημμένως πακέτα TCP και παράλληλα λαμβάνει TCP πακέτα.
 - Για λειτουργία ως TCP Server:
 1. Ανοιγμα Socket,
 2. Bind socket,
 3. Listening mode για να ακούει για συνδέσεις από TCP clients,
 4. Accept όταν βρεί κάποιον TCP client που προσπαθεί να συνδεθεί, για να αποδεχτεί τη σύνδεση (η χειραγία ολοκληρώνεται μετά από αυτό το βήμα),
 5. Στέλνει κατ' επανάληψη πακέτα TCP και παράλληλα λαμβάνει TCP πακέτα.

Μετά τη ρύθμιση του πρωτοκόλλου επικοινωνίας ρυθμίζονται οι παράμετροι του uDMA controller, για τον οποίο θα χρησιμοποιηθούν 3 κανάλια για μεταφορά. Τα δύο θα λειτουργούν σε Ping-Pong mode[27] λαμβάνοντας τα δεδομένα που έρχονται από το APM μέσω της διεπαφής UART1 και τοποθετώντας τα σε δύο buffers διαδοχικά. Το άλλο θα λειτουργεί σε Basic Mode (απλής μεταφοράς), προωθώντας δεδομένα στο APM, τη στιγμή που τα λαμβάνει από το GCS. Το μέγεθος όλων των buffers έχει ρυθμιστεί σε 17B για να συμφωνεί με το μέγεθος του πακέτου MavLink.

Μετά έχουν πλέον ενεργοποιηθεί όλες οι μονάδες και έχουν ρυθμιστεί όλες οι παράμετροι οπότε ανάβει το κόκκινο led για να σημάνει αυτό το γεγονός. Αυτό το προκαταρκτικό νήμα τερματίζεται και ο έλεγχος δεν ξαναεπιστρέφει σ' αυτό. Από εκεί και ύστερα το νήμα του πρωτοκόλλου και το interrupt κάνουν τη δουλειά κατ' επανάληψη.

Κάθε φορά που γεμίζει ο FIFO buffer της μονάδας UART1 προκαλείται interrupt το οποίο προετοιμάζει τον DMA για τις επόμενες μεταφορές. Μόλις ολοκληρωθεί το interrupt ενεργοποιείται η μεταβλητή σηματοφόρου g_sendSignal ένας μηχανισμός – κλειδί του RTOS για το πρόγραμμα, η οποία συγχρονίζει τον γεμάτο πλέον buffer με το νήμα του επιλεγμένου πρωτοκόλλου, όπου και θα σταλούν τελικά τα περιεχόμενα του στο GCS και παράλληλα θα ληφθούν δεδομένα από το GCS (εάν υπάρχουν). Εάν λαμβάνονται δεδομένα τότε τα προωθεί αμέσως μέσω του Tx DMA καναλιού στο APM. Ο κύκλος αυτός επαναλαμβάνεται συνεχώς.

Σε περίπτωση που το CC3200 πρέπει να λειτουργήσει ως Station για να συνδεθεί σε κάποιο τοπικό δίκτυο Wlan θα πρέπει να δεχτεί είσοδο από τον υπολογιστή. Το CCS επιτρέπει τη χρήση κάποιων “Predefined Symbols” (προκαθορισμένων συμβόλων) σε επίπεδο Project τα οποία έχουν ισχύ σε κάθε αρχείο του Project. Όποτε συναντάται η οδηγία προεπεξεργαστή: #ifdef Terminal, τότε (τα περιεχόμενα μέχρι το #endif) θεωρείται ότι το CC3200 θα εκκινήσει ευρισκόμενο σε υπολογιστή και θα δεχτεί είσοδο από κάποιο πρόγραμμα serial terminal emulator. Σε αυτή τη περίπτωση λοιπόν το CC3200 προτρέπει το χρήστη:

- Για λειτουργία ως AP ή WLAN – Station mode
 - Για Wlan ο χρήστης επιλέγει SSID, password (αν υπάρχει), είδος ασφάλειας που χρησιμοποιείται (0 για open, 1 για wep, 2 για WPA/WPA2, 3 για WPA enterprise, 4 για PBC, ή 5 για PIN) και τη διεύθυνση IP του στόχου στο οποίο βρίσκεται το GCS και πρέπει να συνδεθεί.
- Για το πρωτόκολλο που θα χρησιμοποιηθεί, είτε UDP, είτε TCP
 - Για TCP ο χειριστής διερωτάται επιπλέον εάν το CC3200 θα είναι TCP client ή TCP server

Αυτή είναι συνοπτικά όλη η διαδικασία. Κάποιες επιπλέον σημαντικές παρατηρήσεις θα τονιστούν αμέσως μετά, όσον αφορά την υλοποίηση του κώδικα και την εκτέλεση της εφαρμογής.

1. Ρύθμιση κλειδί για την υλοποίηση της επικοινωνίας ήταν να θέσουμε στο socket τον τρόπο λειτουργίας ως Non-Blocking. Blocking εντολές εκτελούνται «σε σειρά» επειδή η μία θα μπλοκάρει την εκτέλεση μέχρι να ολοκληρωθεί. Για παράδειγμα το API sl_Recv από προεπιλογή είναι blocking, και θα σταματούσε την εκτέλεση τρώγοντας ανεκτίμητο χρόνο μέχρι να λάβει δεδομένα, ενώ δεν είναι η σειρά του να λάβει δεδομένα κάθε φορά που εκτελείται το νήμα που το περιέχει! Non-Blocking σημαίνει ότι οι εντολές εκτελούνται παράλληλα, όποτε η καθεμία έχει δουλειά να κάνει. Αυτό ουσιαστικά σημαίνει polling στις εντολές, κάτι κοστοβόρο, αλλά στη συγκεκριμένη περίπτωση προτιμητέο. Υπάρχει και καλύτερος τρόπος υλοποίησης με άλλα API's (sl_Select) που ελέγχουν πότε το κάθε API «έχει δουλειά να κάνει» και το επιλέγουν. Αυτό ήταν πιο δύσκολο και δεν υπήρχε χρόνος για περαιτέρω ψάξιμο.

2. Εφόσον θέλουμε να στείλουμε δεδομένα το σύστημα μας πρέπει να είναι σε θέση να μεταφέρει τα δεδομένα τουλάχιστον με τον ρυθμό με τα οποία αυτά λαμβάνονται από τη μονάδα UART1. Άλλιώς θα έχουμε υπερχείλιση (overflow) του buffer. Κάθε βαθμίδα UART έχει δύο FIFO buffers (για Tx και Rx) ο καθένας βάθους 16 bytes και μπορεί να γεννάει Interrupts σε διάφορα επίπεδα πληρότητας. Εμείς ρυθμίσαμε ώστε να προκαλείται interrupt κάθε 8 bytes πληρότητας, δηλαδή μέχρι ο FIFO να γεμίσει στο ήμισυ. Ο FIFO μειώνει το overhead, διότι έτσι δεν χρειάζεται να προκαλείται interrupt σε κάθε χαρακτήρα/byte που λαμβάνεται. Πρέπει όμως να βεβαιωνόμαστε ότι διαβάζουμε από τον FIFO αρκετά συχνά έτσι ώστε να μην υπερχειλίζει. Η τέχνη του συντονισμού της διεπαφής UART είναι να θέσουμε το interrupt αρκετά αραιά ώστε η CPU να μην υπερλειτουργεί, αλλά παράλληλα αρκετά συχνά ώστε να μην συμβαίνει buffer overflow. Η όλη ρύθμιση των παραμέτρων του UART, των buffers και του DMA για τη σωστή αποστολή και λήψη τους ήταν πολύ δύσκολη εργασία και τελικά βρήκαμε τη χρυσή τομή ύστερα από πολλά πειράματα, προσπάθειες και (βεβαίως) λάθη.
3. Ο επεξεργαστής του δικτύου επικοινωνεί με τον επεξεργαστή των εφαρμογών χρησιμοποιώντας διεπαφή SPI στα 20 MHz. Δεν είναι ένας ο επεξεργαστής για όλους τους μηχανισμούς.
4. Χρησιμοποιήσαμε το πρόγραμμα UNIFLASH[64] της TI για να φορτώσουμε το binary αρχείο με την εφαρμογή στη Serial FLASH μνήμη του CC3200. Η Serial Flash διαθέτει ιδιόκτητο σύστημα αρχείων. Ο κώδικας της εφαρμογής (application code) φορτώνεται στην εσωτερική RAM (Internal RAM – IRAM) της συσκευής από τη SFlash όταν ενεργοποιείται η συσκευή. Οι drivers (και ο simplelink host driver) εκτελούνται από την IRAM, τα API's των drivers (από τα οποία τα API's του επεξεργαστή δικτύουσης Simplelink, ανήκουν στο Middleware) εκτελούνται από τη RAM. [65]Ο bootstrap loader της συσκευής βρίσκεται σε ROM. Στη ROM επίσης ανήκουν οι βιβλιοθήκες των περιφερειακών οδηγών συσκευών (Peripheral driver libraries), με σκοπό να μειώσουν το RAM footprint της συσκευής, ουσιαστικά μειώνοντας και το μέγεθος του κτισμένου binary αρχείου. Οι βιβλιοθήκες περιέχουν μια συλλογή ρουτινών που παρέχουν αφαίρεση στον προγραμματιστή. Για να χρησιμοποιηθούν πρέπει να οριστεί η οδηγία προεπεξεργαστή: TARGET_IS_CC3200 σε επίπεδο project. Οι συγκεκριμένες ρουτίνες αναγνωρίζονται με το όνομα τους να ξεκινάει από MAP_, πχ. MAP_UtilsDelay.
5. Το πρόγραμμα μας (code+data+stack size+heap size) πρέπει να έχει μέγεθος μεταξύ 240KB. Σε κάποια στιγμή ο linker παρουσίασε σφάλμα, επειδή το code segment γέμισε. Παράλληλα το data segment ήταν στο 50% οπότε έπρεπε να προσαρμόσουμε κατάλληλα, μεγαλώνοντας το code segment και μειώνοντας το data segment. Τα προηγούμενα έγιναν στο αρχείο Linker και ο κώδικας που τροποποιήσαμε φαίνεται παρακάτω και ανήκει στο αρχείο cc3200v1p32.cmd.

```

//*****
// The starting address of the application. Normally the interrupt vectors
// must be located at the beginning of the application.
//*****

#define RAM_BASE 0x20004000
MEMORY           /* System memory map */
{
/* Application uses internal RAM for program and data */
/* Note that you can only increase the CODE memory by the amount you deduct the DATA
   memory and vice versa. The total memory is constant.*/
/*The .out file contains lot more information like debugging info, headers whereas
   .bin only contains the actual executable binary code.*/
//    SRAM_CODE (RWX) : origin = 0x20004000, length = 0x12FFF
//    SRAM_DATA (RWX) : origin = 0x20017000, length = 0x19000
    SRAM_CODE (RWX) : origin = 0x20004000, length = 0x16FFF //+0x4000+1
    SRAM_DATA (RWX) : origin = 0x2001B000, length = 0x15000 // -0x4000
}

```

6. Εκτός από την ανάθεση (όπως θα παρατηρήσετε στον κώδικα) σε κάθε νήμα 4096B αφοσιωμένη μνήμη Stack, αναθέσαμε στην εφαρμογή από το CCS, συνολική μνήμη Stack 0x2000 και συνολική μνήμη Heap 0x8000, που είναι υπεραρκετή.
7. Τα κουμπιά SW2 και SW3 σε κανονική λειτουργία είναι LOW και με το πάτημα τους «τραβιούνται» προς Vec (High).
8. Το πρόγραμμα έχει φτιαχτεί ώστε να μπορεί να χρησιμοποιηθεί είτε το FREERTOS (με μια μικρή τροποποίηση) είτε το TI-RTOS. Η οδηγία προεπεξεργαστή #ifdef USE_FREERTOS και τα περιεχόμενα μέχρι το αντίστοιχο #endif χρησιμοποιείται σε περίπτωση που κάποιος επιθυμήσει το FREERTOS και περιέχονται εκεί οι βασικές συναρτήσεις callbacks που πρέπει να ρυθμίσει εάν θέλει. Εμείς χρησιμοποιήσαμε το TI-RTOS που παρόλο που είναι πιο, ομολογουμένως, μπερδεμένο προσφέρει μια πολύ εμπεριστατωμένη μέθοδο για debugging, το RTOS Object View (ROV) που δεν είναι διαθέσιμη με το FREE-RTOS. Το ROV μας βοήθησε πολύ στο debugging. Ανα πάσα στιγμή αναφέρει το κάθε τι για την εξέλιξη του RTOS.
9. [66] Ένα OS λαμβάνει δεδομένα σε δεσμίδες μεγέθους ίσου με το μήκος σελίδας (εικονικής μνήμης). Κάθε φορά που λαμβάνει δεδομένα από σελίδες που βρίσκονται σε δευτερεύοντα μνήμη, τότε διεγείρεται το σήμα διακοπής page fault (στα Windows αναφέρεται ως hard fault από το Resource Monitor). Καλή τακτική προγραμματισμού (μάλλον απαραίτητη) είναι να οριοθετούμε τα δεδομένα που αποθηκεύονται στη μνήμη ώστε να έχουν μήκος ίσο με το μήκος σελίδας, αλλιώς επιβαρύνεται περισσότερο η CPU, προκαλώντας περισσότερα από 1 page fault για κάθε δέσμη δεδομένων. Γι' αυτό καλούμε το:

`#pragma DATA_ALIGN(gpCtlTbl, 1024)`

ώστε ο μεταγλωττιστής του CCS να οριοθετήσει κατάλληλα τα δεδομένα του πίνακα ελέγχου (control table) του μDMAcontroller, όταν αρχικοποιείται, ο οποίος χρησιμοποιείται συχνά στην εφαρμογή μας. Η ελαχιστοποίηση του αριθμού των page faults είναι μια πολύ αποτελεσματική μέθοδος για αύξηση των επιδόσεων. Αποτελεί κοινή πρακτική να χρησιμοποιείται μεγάλο μέγεθος block για τον DMA, καθώς τις περισσότερες φορές μεταφορές με μεγάλα blocks από τις περιφερειακές συσκευές πραγματοποιούνται αρκετά πιο γρήγορα.

Ο παρακάτω κώδικας αρχικώς αντιγράφηκε στο πρόγραμμα Notepad++ και ύστερα επικολλήθηκε εδώ με μορφοποίηση τύπου HTML (στο Notepad++ επιλέγουμε τον κώδικα και ύστερα πάμε: Plugins >> NppExport >> Copy HTML to clipboard).

```

*****
*                               INCLUDES - Start
*****



//Standard libraries
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

// FREE-RTOS/TI-RTOS include
#include "osi.h"

// SYS/BIOS kernel library Headers
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/family/arm/m3/Hwi.h>

//driverlib includes
#include "hw_ints.h"
#include "hw_types.h"
#include "interrupt.h"
#include "hw_memmap.h"
#include "timer.h"
#include "uart.h"
#include "prcm.h"
#include "utils.h"
#include "hw_uart.h"
#include "hw_gpio.h"
#include "pin.h"
#include "udma.h"
#include "gpio.h"
#include "rom.h"
#include "rom_map.h"

//Simplelink includes
#include "simplelink.h"

//middleware includes
#include "uart_drv.h"

//common interface includes
#include "timer_if.h"
#include "gpio_if.h"
#include "button_if.h"
#include "udma_if.h"
#include "uart_if.h"
#include "common.h"
#include "pinmux.h"

*****
*                               INCLUDES - End
*****



*****



*                               DEFINES - Start
*****



// Interrupt vector table entry, depending on the IDE in use
#if defined(ccs)
extern void (* const g_pfnVectors[])(void);
#endif

```

```

*****



*                               DEFINES - Start
*****



// Interrupt vector table entry, depending on the IDE in use
#if defined(ccs)
extern void (* const g_pfnVectors[])(void);
#endif

```

```

#if defined(gcc)
extern void (* const g_pfnVectors[])(void);
#endif
#if defined(ewarm)
extern uVectorEntry __vector_table;
#endif

#define APP_NAME           "N.L. Thesis. Telemetry for Quadcopter/Drone
project"
#define APP_VERSION        "1.0"
#define TASK_STACK_SIZE    4096

#ifndef TERMINAL
#define SSID_LEN_MAX       32
#define BSSID_LEN_MAX      6
#define SECURITY_KEY_LEN_MAX 64
#endif

// Network Configuration values
#define UDP_PORT            14550
#define TCP_PORT             5760
#define HOST_NAME            "www.ti.com" // Ping host to check Internet
access

//Values for below macros set the ping properties
#define PING_INTERVAL        1000      // In msecs
#define PING_TIMEOUT          3000      // In ms
#define PING_PKT_SIZE         20        // In Bytes(B)
#define NO_OF_ATTEMPTS        3
#define PING_FLAG              0

// Size of Transfer Buffers, UDMA_SIZE_8
// UDMA_SIZE_## # should be increased as buffer sizes increase,
// Greater UDMA_SIZE_##'s yield higher throughput especially for ping-pong type
transfers
// The MAVLINK msg len is 17B
#define UDMA_RXCH_BUF_SIZE     17        // In Bytes
#define UDMA_TXCH_BUF_SIZE     17        // (B)

// APIs parameters
#define SL_STOP_TIMEOUT        200      // in msecs

// RTOS
#define SL_SPAWN_TASK_PRIORITY   9
#define COM_PROTOCOL_TASK_PRIORITY 4
#define GREEN_LIGHT_PRIORITY     1
#define INACTIVE_STATE_PRIORITY -1

// System
#define SYSTEM_CLOCK            80000000
#define UARTA1_BAUD_RATE        57600
#define MAX_NUM_CH                64 //32*2 entries
#define CTL_TBL_SIZE               64 //32*2 entries

// For uDMA initialization. Compiler aligns gpCtlTbl control table variable as
// 1024-byte-boundary page-aligned memory addressed space
#if defined(gcc)
tDMAControlTable gpCtlTbl[CTL_TBL_SIZE] __attribute__(( aligned(1024)));
#endif
#if defined(ccs)
#pragma DATA_ALIGN(gpCtlTbl, 1024)
tDMAControlTable gpCtlTbl[CTL_TBL_SIZE];
#endif
#if defined(ewarm)
#pragma data_alignment=1024
tDMAControlTable gpCtlTbl[CTL_TBL_SIZE];
#endif

```

```

// MISC
#define GOOD_COUNT          200
#define NULL                0
#define null                0
#define SUCCESS              0
#define FAILURE             -1
#define FALSE               0
#define false               0
#define TRUE                1
#define true                1
#define FAILURE             -1
#define FOREVER             1

#define UART_PRINT           Report
#define LOOP_FOREVER() \
{ \
    while(1); \
}

//Application status/error codes
typedef enum{
    LAN_CONNECTION_FAILED = - 0x7D0,
    INTERNET_CONNECTION_FAILED = LAN_CONNECTION_FAILED - 1,
    SOCKET_CREATE_ERROR = INTERNET_CONNECTION_FAILED - 1,
    CLIENT_CONNECTION_FAILED = SOCKET_CREATE_ERROR - 1,
    DEVICE_NOT_IN_STATION_MODE = CLIENT_CONNECTION_FAILED - 1,
    RECV_ERROR = DEVICE_NOT_IN_STATION_MODE - 1,
    SEND_ERROR = RECV_ERROR - 1,
    BIND_ERROR = SEND_ERROR - 1,
    LISTEN_ERROR = BIND_ERROR - 1,
    SOCKET_OPT_ERROR = LISTEN_ERROR - 1,
    CONNECT_ERROR = SOCKET_OPT_ERROR - 1,
    ACCEPT_ERROR = CONNECT_ERROR - 1,
    INTERRUPT_ERROR = ACCEPT_ERROR - 1,
    STATUS_CODE_MAX = - 0xBBB
}e_AppStatusCodes;

//Application exit codes
enum enExitCode{
    EXIT_SUCCESS,
    EXIT_GENERIC_FAIL,
    EXIT_SEMAPHORE_FAIL,
    EXIT_DMA_FAIL,
    EXIT_SOCKET_FAIL,
    EXIT_TASK_FAIL,
    EXIT_SEC_FAIL,
    EXIT_IP_FAIL,
    EXIT_TCP_FAIL,
};

/*****************************************
*                               DEFINES - End
*****************************************/

```



```

/*****************************************
*                               GLOBAL VARIABLES - Start
*****************************************/

```



```

//SimpleLink Status - response on various async events -
volatile unsigned long g_ulStatus      = 0;

#ifndef TERMINAL // Values below shall be modified by the user according to AP
properties

```

```

unsigned long g_ulGatewayIP      = 0; // Network Gateway IP address
(to be used by Callback)
unsigned char g_ucConnectionSSID[SSID_LEN_MAX+1]; //Connection SSID
(to be used by Callback)
char g_ucNetworkSSID[SSID_LEN_MAX+1]; //Connection SSID
char g_ucSecurityKey[SECURITY_KEY_LEN_MAX];
g_ucSecurityType;
g_ucConnectionBSSID[BSSID_LEN_MAX]; //Connection BSSID
#else
volatile _Bool g_bButtonPress    = FALSE;
volatile _Bool g_bTimerA1Flag    = FALSE;
volatile unsigned int g_uiNoButtonPress = 0;
#endif

// Network/Communication global variables
unsigned long g_ulDestIp        = 0; // Connected host IP address
unsigned long g_ulPingPacketsRecv = 0;
volatile static _Bool g_bBufTurn   = TRUE;
volatile static _Bool g_bUdpOrTcp  = TRUE;
volatile static _Bool g_bApOrWlan  = TRUE;
volatile static _Bool g_bTcpClientOrServer = TRUE;
volatile int g_iDisconnected     = FALSE; // If toggled we reinitiate
the socket connection on station reconnect
int g_iUdpSockHandle           = 0; // UDP socket descriptor
(handle)
int g_iTcpSockHandle           = 0; // TCP client socket handle
int g_iNewTcpSockID            = SL_EAGAIN; // TCP server socket
handle
SlSockAddrIn_t the_connected_host
int g_iSockBlockSize           = sizeof(SlSockAddrIn_t);

// The three DMA buffers used for delivering the data-packets
char g_cUart1RxBufA[UDMA_RXCH_BUF_SIZE] = "";
char g_cUart1RxBufB[UDMA_RXCH_BUF_SIZE] = "";
char g_cUart1TxBuf[UDMA_TXCH_BUF_SIZE] = "";

// Counts number of buffer refills and transmissions, receptions
unsigned long g_ulRxABufFillCount = 0;
unsigned long g_ulRxBBufFillCount = 0;
unsigned long g_ulTxBufFillCount = 0;
unsigned long g_ulTxCount       = 0;
unsigned long g_ulRxCount       = 0;

// RTOS
OsiTaskHandle UdpHandle, TcpHandle, GreenLightHandle;
OsiSyncObj_t g_sendSignal; // Semaphore variable

// Misc
char g_cErrorBuffer[100];

/********************* GLOBAL VARIABLES - End *********************/

```

```

/********************* LOCAL FUNCTION PROTOTYPES - Start *********************/

```

```

// Event Handlers/Callbacks/Hook function definitions
void SimpleLinkHttpServerCallback(SlHttpServerEvent_t
*pHttpEvent, SlHttpServerResponse_t *pHttpResponse);
void SimpleLinkGeneralEventHandler(SlDeviceEvent_t *pDevEvent);
void SimpleLinkWlanEventHandler(SlWlanEvent_t *pWlanEvent);
void SimpleLinkNetAppEventHandler (SlNetAppEvent_t *pNetAppEvent);

```

```

void SimpleLinkPingReport(SlPingReport_t *pPingReport);
void SimpleLinkSockEventHandler(SlSockEvent_t *pSock);

//initialization/configuration function definitions
static void BoardInit(void);
void PinMuxConfig(void);
void InitTerm();
void ClearTerm();
static void DisplayBanner(char * AppName);
void UDMAInit();
extern void DmaSwIntHandler(void);
extern void DmaErrorIntHandler(void);
void InitialSetupAP( void * pvParameters);
static long ConfigureSimpleLinkToDefaultState();
static void InitializeVariables();
static int SetAPMode(int iMode);
static long PingTest(unsigned long ulIpAddr);
int UdpClientServerConfig(unsigned short usPort);
int TcpClientServerConfig(unsigned short usPort);
int Uart1ConfigDmaTransfer(void);
#ifndef TERMINAL
int IpAddressParser(char *cUserInputIp);
void InitialSetupWlan( void *pvParameters );
static long WlanConnect();
static long WlanPingInternetHost();
#else
void DecisionMaking(void);
int ConfigureTimeoutTimer( unsigned long ultimerBase, unsigned long
ultimerPeriph,
    unsigned long ultimerMode, unsigned long ulPrescaleVal, unsigned short
usIntPriority,
    void (*TimerInterruptHandler) (void), unsigned long ultimeoutVal, unsigned
long ultimer );
static unsigned char GetTimerInterruptNum(unsigned long ultimerBase,
    unsigned long ultimer);
int ButtonSetup ( void (*SW3ButtonInterruptHandler) (void),
    void (*SW2ButtonInterruptHandler) (void) );
#endif

//RTOS TASKS
void TransmitReceiveUdpPackets(void * pvParameters);
void SendRecvTcpStream(void * pvParameters);
void GreenLight(void * pvParameters);

//Interrupts
void Uart1IntHandler(void);
#ifndef TERMINAL
void Sw3InterruptHandler(void);
void Sw2InterruptHandler(void);
void TimerA0InterruptHandler(void);
void TimerA1InterruptHandler(void);
#endif

#ifndef NOTERM      // Check the error, handle and display it through UART
#define ERR_PRINT(x) Report("Error [%d] at line [%d] in function [%s]
\n\r",x,__LINE__,__FUNCTION__)
#define ASSERT_ON_ERROR(error_code) \
{ \
    if(error_code < 0) \
    { \
        sprintf(g_cErrorBuffer,"Error [%d] at line [%d] in " \
"function [%s]", error_code,__LINE__,__FUNCTION__); \
        UART_PRINT(g_cErrorBuffer); \
        UART_PRINT("\n\r"); \
        return error_code; \
    } \
}
#endif

```

```

/***** LOCAL FUNCTION PROTOTYPES - End *****/
***** /



/***** EVENT HANDLERS / CALLBACKS - Start *****/
***** /



/* FreeRTOS User Hook/Callback Functions enabled in FreeRTOSConfig.h */
***** /



#ifndef USE_FREERTOS
***** /
 *brief Application defined hook (or callback) function - assert
 *       Handle Asserts here
 *param[in] pcFile - Pointer to the File Name
 *param[in] ulLine - Line Number
 *return none
***** /
void vAssertCalled( const char *pcFile, unsigned long ulLine )
{
    while(1)
    {
    }
}

/*brief Application defined idle task hook
 *       Handle Idle Hook for Profiling, Power Management etc
 *param none
 *return none
***** /
void vApplicationIdleHook( void )
{
}

/*brief Application defined malloc failed hook
 *       Handle Memory Allocation Errors
 *param none
 *return none
***** /
void vApplicationMallocFailedHook()
{
    while(1)
    {
    }
}

/*brief Application defined stack overflow hook
 *       Handle FreeRTOS Stack Overflow
 *param none
***** /

```

```

        *return      none
***** */
void vApplicationStackOverflowHook(OsiTaskHandle *pxTask, signed char
*pcTaskName)
{
    while(1)
    {
    }
}
#endif //USE_FREERTOS

/*
*brief      This function handles HTTP dynamic content tokens and errors
*param[in]   pServerEvent - Contains the relevant event information
*param[in]   pServerResponse - Should be filled by the user with the
*                           relevant response information
*return     None
*warning    This function must be included or else the compiler goes nuts
***** */

void SimpleLinkHttpServerCallback(SlHttpServerEvent_t *pHttpEvent,
                                  SlHttpServerResponse_t *pHttpResponse)
{
    // Unused in this application
}

/*
*brief      This function handles General Events
*           Most of the general errors are not FATAL and are to be handled
appropriately
*           by the application
*param[in]   pDevEvent - Pointer to General Event Info
*return     None
***** */

void SimpleLinkGeneralEventHandler(SlDeviceEvent_t *pDevEvent)
{
    UART_PRINT(" [GENERAL EVENT] - ID=[%d] Sender=[%d]\n\r",
               pDevEvent->EventData.deviceEvent.status,
               pDevEvent->EventData.deviceEvent.sender);
}

/*
*brief      Callback is triggered on various Wlan Network events such as
Connect/Disconnect
*           Depending on Wlan, or AP mode function triggers connection status
bits
*           to be set, or cleared appropriately.
*param      pWlanEvent pointer indicating Event type
*return     None
*note      When the event handler is triggered, we are using event driven
programming to
*           decide what to do next. They are handled as SWIs from the OS
*Warning  Event Handlers can not block
***** */

```

```

void SimpleLinkWlanEventHandler(SlWlanEvent_t *pWlanEvent)
{
    if (g_bApOrWlan){ /*** Following network events are handled in AP mode ***/

        switch(pWlanEvent->Event) {
            case SL_WLAN_CONNECT_EVENT: { //we just need to monitor the
                g_ulStatus variable to check the status of the device
                SET_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION);
            }
            break;

            case SL_WLAN_DISCONNECT_EVENT: {
                g_iDisconnected = TRUE;
                slWlanConnectAsyncResponse_t* pEventData = NULL;

                CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION); // Status bit for
                CC3200 connections/disconnections
                CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_AQUIRED); // Status bit for
                CC3200 Ip events

                pEventData = &pWlanEvent->EventData.STAandP2PModeDisconnected;

                // If the user has initiated 'Disconnect' request,
                // 'reason_code' is SL_USER_INITIATED_DISCONNECT
                if(SL_USER_INITIATED_DISCONNECT == pEventData->reason_code)
                    UART_PRINT("[WLAN EVENT] CC3200 disconnected from the AP on
operator's request \n\r");
                else
                    UART_PRINT("[WLAN EVENT] WARNING: CC3200 disconnected from the
AP on an ERROR...!\n\r");
            }
            break;

            // The following event is expected to be triggered
            case SL_WLAN_STA_CONNECTED_EVENT:{// triggers when client connects to
device
                SET_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION); // Status bit for
                station connections/disconnections when CC3200 is in AP mode
                break;
            }

            // the client disconnecting from the device isn't meant to happen and
            // shouldn't happen in our application
            case SL_WLAN_STA_DISCONNECTED_EVENT:{// triggers when host disconnects
from CC3200-AP mode
                // g_iDisconnected = TRUE;
                CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION);
                CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_LEASED); // Status bit for
                station Ip leases/releases when CC3200 is in AP mode
                UART_PRINT("[WLAN EVENT] IP Released for Client:
IP=%d.%d.%d.%d\n\r",
                           SL_IPV4_BYT
                           g_ulDestIp, 3), SL_IPV4_BYT
                           g_ulDestIp, 2),
                           SL_IPV4_BYT
                           g_ulDestIp, 1), SL_IPV4_BYT
                           g_ulDestIp, 0));
                break;
            }

            default: {
                UART_PRINT("[WLAN EVENT] WARNING: Unexpected event [0x%x]\n\r",
pWlanEvent->Event);
            }
            break;
        } // end AP switch statement
    }
#endif TERMINAL

```

```

else { /*** Following network events are handled in WLAN - Station mode ***/

    switch (pWlanEvent->Event)      {
        case SL_WLAN_CONNECT_EVENT: {
            SET_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION);

            // Copy new connection SSID and BSSID to global parameters
            memcpy(g_ucConnectionSSID,
                   pWlanEvent->EventData.STAandP2PModeWlanConnected.ssid_name,
                   pWlanEvent->EventData.STAandP2PModeWlanConnected.ssid_len);
            memcpy(g_ucConnectionBSSID,
                   pWlanEvent->EventData.STAandP2PModeWlanConnected.bssid,
                   SL_BSSID_LENGTH);

            UART_PRINT( "[WLAN EVENT] CC3200 Connected to the AP: %s ,"
                        "BSSID: %x:%x:%x:%x:%x\n\r", g_ucConnectionSSID,
                        g_ucConnectionBSSID[0], g_ucConnectionBSSID[1],
                        g_ucConnectionBSSID[2],
                        g_ucConnectionBSSID[3], g_ucConnectionBSSID[4],
                        g_ucConnectionBSSID[5] );
        }
        break;

        case SL_WLAN_DISCONNECT_EVENT: {
            g_iDisconnected = TRUE;
            CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_CONNECTION);
            CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_AQUIRED);

            // Contains the AP's parameters. We will display them later on
terminal
            slWlanConnectAsyncResponse_t* pEventData = NULL;

            pEventData = &pWlanEvent->EventData.STAandP2PModeDisconnected;

            // In case the user disconnected voluntarily
            if(SL_USER_INITIATED_DISCONNECTION == pEventData->reason_code)
                UART_PRINT( "[WLAN EVENT] CC3200 disconnected from the AP on
operator's request \n\r");
            else
                UART_PRINT( "[WLAN EVENT] WARNING: CC3200 disconnected from the
AP on an ERROR..!!\n\r");

            memset(g_ucConnectionSSID, 0, sizeof(g_ucConnectionSSID));
            memset(g_ucConnectionBSSID, 0, sizeof(g_ucConnectionBSSID));
        }
        break;

        default: {
            g_iDisconnected = TRUE;
            UART_PRINT( "[WLAN EVENT] WARNING: Unexpected event [0x%x]\n\r",
pWlanEvent->Event);
        }
        break;
    } // end Wlan switch statement
} // endif CC3200 Network mode choise
#endif
}

```

```

*****
*brief      This function handles networking application events such as IP
acquisition,
*brief      IP leased, IP released etc.

*param[in]  pNetAppEvent - Pointer to NetApp Event Info

```

```

*return      None
***** */

void SimpleLinkNetAppEventHandler(SlNetAppEvent_t *pNetAppEvent)
{
    switch(pNetAppEvent->Event) {
        case SL_NETAPP_IPV4_IPACQUIRED_EVENT: // Handler in case of Wlan Station mode
        case SL_NETAPP_IPV6_IPACQUIRED_EVENT: {
            SET_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_AQUIRED);
#ifdef TERMINAL
            if (!g_bApOrWlan) { // Events to be handled when CC3200 is in Wlan mode
                SlIpV4AcquiredAsync_t *pEventData = NULL;

                pEventData = &pNetAppEvent->EventData.ipAcquiredV4; // Ip Acquired Event Data
                g_ulGatewayIP = pEventData->gateway; // Grab and store the Gateway IP address

                UART_PRINT( "[NETAPP EVENT] IP Acquired: IP=%d.%d.%d.%d\n\r"
                            "Gateway=%d.%d.%d.%d\n\rDNS=%d.%d.%d.%d\n\r",
                            SL_IPV4_BYT(pNetAppEvent->EventData.ipAcquiredV4.ip,3),
                            SL_IPV4_BYT(pNetAppEvent->EventData.ipAcquiredV4.ip,2),
                            SL_IPV4_BYT(pNetAppEvent->EventData.ipAcquiredV4.ip,1),
                            SL_IPV4_BYT(pNetAppEvent->EventData.ipAcquiredV4.ip,0),
                            SL_IPV4_BYT(pNetAppEvent-
                                         >EventData.ipAcquiredV4.gateway,3),
                            SL_IPV4_BYT(pNetAppEvent-
                                         >EventData.ipAcquiredV4.gateway,2),
                            SL_IPV4_BYT(pNetAppEvent-
                                         >EventData.ipAcquiredV4.gateway,1),
                            SL_IPV4_BYT(pNetAppEvent-
                                         >EventData.ipAcquiredV4.gateway,0),
                            SL_IPV4_BYT(pNetAppEvent-
                                         >EventData.ipAcquiredV4.dns,3),
                            SL_IPV4_BYT(pNetAppEvent-
                                         >EventData.ipAcquiredV4.dns,2),
                            SL_IPV4_BYT(pNetAppEvent-
                                         >EventData.ipAcquiredV4.dns,1),
                            SL_IPV4_BYT(pNetAppEvent->EventData.ipAcquiredV4.dns,0));
            };
        }
#endif
        break;
    }

    /* Following events should occur only when CC3200 operates in AP mode */
    case SL_NETAPP_IP_LEASED_EVENT: { // CC3200 as AP leases IP address to connected host
        SET_STATUS_BIT(g_ulStatus, STATUS_BIT_IPLEASED);

        // Assigns IP from the DHCP server
        g_ulDestIp = (pNetAppEvent)->EventData.ipLeased.ip_address;
        UART_PRINT( "[NETAPP EVENT] IP Leased to Client: IP=%d.%d.%d.%d\n\r",
                    SL_IPV4_BYT(g_ulDestIp, 3), SL_IPV4_BYT(g_ulDestIp, 2),
                    SL_IPV4_BYT(g_ulDestIp, 1), SL_IPV4_BYT(g_ulDestIp, 0));
        break;
    }

    case SL_NETAPP_IP_RELEASED_EVENT: { // CC3200 as AP releases IP of connected host
        CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_IPLEASED);
        UART_PRINT( "[NETAPP EVENT] IP Released for Client:
                    IP=%d.%d.%d.%d\n\r",
                    SL_IPV4_BYT(g_ulDestIp, 3), SL_IPV4_BYT(g_ulDestIp, 2),
                    SL_IPV4_BYT(g_ulDestIp, 1), SL_IPV4_BYT(g_ulDestIp, 0));
    }
}

```

```

        break;
    }

default:
    UART_PRINT( "[ NETAPP EVENT ] WARNING: Unexpected event [ 0x%x ]\n\r",
pNetAppEvent->Event);
    break;
} //end switch statement
}

//*************************************************************************
*brief      This function handles ping report events
*param[in]  pPingReport - Ping report statistics
*return     None
//*************************************************************************/

void SimpleLinkPingReport(SlPingReport_t *pPingReport)
{
    SET_STATUS_BIT(g_ulStatus, STATUS_BIT_PING_DONE);
    g_ulPingPacketsRecv = pPingReport->PacketsReceived;
}

//*************************************************************************
*brief      This function handles socket events indication. Closing,
connecting,
*           accepting socket connections etc.
*           Used for sockets that require connection protocols
*param[in]  pSock - Pointer to Socket Event Info
*return     None
//*************************************************************************/

void SimpleLinkSockEventHandler(SlSockEvent_t *pSock)
{
    switch( pSock->Event ) {
        case SL_SOCKET_ASYNC_EVENT : :
        case SL_SOCKET_TX_FAILED_EVENT : {
            switch( pSock->socketAsyncEvent.SockTxFailData.status ) {
                case SL_ECLOSE :// Triggers if remote end has closed
the socket
                    // This event can also be detected if Recv returns 0
                    UART_PRINT( "[ SOCK ERROR ] - Remote end has closed the socket
(%d) operation "
                                "failed to transmit all queued packets\n\r",
                                pSock->socketAsyncEvent.SockTxFailData.sd);
                    GPIO_IF_LedOff(MCU_ALL_LED_IND);
                    // Close the appropriate socket descriptor
                    if (g_bUdpOrTcp)
                        sl_Close(g_iTcpSockHandle);
                    else{
                        if (g_bTcpClientOrServer)
                            sl_Close(g_iTcpSockHandle);
                        else{
                            sl_Close(g_iNewTcpSockID);
                            sl_Close(g_iTcpSockHandle);
                        }
                    }
                    GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
            }
            break;
        }
    }
}

```

```

        default           :{
            g_iDisconnected = TRUE;
            // This event should occur only when a disconnectin happens
            CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_IP_LEASED);
            UART_PRINT("[SOCK ERROR] - TX FAILED : socket %d , reason
(%d)\n\r",
                        pSock->socketAsyncEvent.SockTxFailData.sd,
                        pSock->socketAsyncEvent.SockTxFailData.status);
            break; // break inner switch default statement scope
        } // end inner default statement
    } // End inner switch statement
}
break;

default:
    UART_PRINT("[SOCK EVENT] - Unexpected Event [%x0x]\n\r", pSock-
>Event);
    break;
} //end outer switch statement
}

*****
*          EVENT HANDLERS / CALLBACKS - End
*****

```

```

*****
*          SETUP FUNCTIONS - Start
*****

```

```

*****
*brief      Function initializes the application variables (housekeeping)

*param     None
*return    None
*****
static void InitializeVariables()
{
    g_ulStatus         = 0;
    g_ulPingPacketsRecv = 0;
#ifdef TERMINAL
    g_ulGatewayIP      = 0;
    memset(g_ucConnectionSSID, 0, sizeof(g_ucConnectionSSID));
    memset(g_ucConnectionBSSID, 0, sizeof(g_ucConnectionBSSID));
#endif
}


```

```

*****
*brief      This function puts the device in its default state. It cleans all
the
*          persistent settings stored in NVMEM (viz. connection profiles &
policies,
*          power policy etc). More specifically the function:
*          - Sets the mode to STATION
*          - Configures connection policy to Auto and AutoSmartConfig
*          - Deletes all the stored profiles
*          - Enables DHCP

```

```

*           - Disables Scan policy
*           - Sets Tx power to maximum
*           - Sets power policy to normal
*           - Unregister mDNS services
*           - Remove all filters

*param      none
*return     On success, zero is returned. On error, negative is returned

*CAUTION! sl_Start must be called before any other simplelink API is used,
*          or after sl_Stop is called for restarting the network processor,
and if OS
*          environment always after the simplelink task is called.
***** */

static long ConfigureSimpleLinkToDefaultState()
{
    long                                lRetVal      = -1;
    long                                lMode        = -1;
    unsigned char                        ucVal        = 1;
    unsigned char                        ucConfigOpt = 0;
    unsigned char                        ucConfigLen = 0;
    unsigned char                        ucPower      = 0;
    SlVersionFull                       ver         = {0};
    _WlanRxFilterOperationCommandBuff_t RxFilterIdMask = {0};

    // sl_Start initialize the WiFi communication interface, sets the enable
pin of the device
    lMode = sl_Start(0, 0, 0);
    ASSERT_ON_ERROR(lMode); // 0 for ROLE_STA = default mode

    // If the device is not in station-mode, try configuring it in station-mode
    if (ROLE_STA != lMode) {
        if (ROLE_AP == lMode) {
            // If the device is in AP mode, we need to wait for it to acquire an
IP
            // address before doing anything else
            while(!IS_IP_ACQUIRED(g_ulStatus)) {
#ifndef SL_PLATFORM_MULTI_THREADED
                _SlNonOsMainLoopTask(); // for NONOS environment this must be
calle in loop
#endif
            }
        }
    }

    // Switch to STA role and restart
    lRetVal = sl_WlanSetMode(ROLE_STA);
    ASSERT_ON_ERROR(lRetVal);

    // Restart network processor to apply changes
    lRetVal = sl_Stop(0xFF);
    ASSERT_ON_ERROR(lRetVal);

    lRetVal = sl_Start(0, 0, 0);
    ASSERT_ON_ERROR(lRetVal);

    // Check if the device is in station again
    if (ROLE_STA != lRetVal) {
        // We don't want to proceed if the device is not coming up in STA-
mode
        return DEVICE_NOT_IN_STATION_MODE;
    }
}

// Get the device's version-information
ucConfigOpt = SL_DEVICE_GENERAL_VERSION;
ucConfigLen = sizeof(ver);

```

```

lRetVal = sl_DevGet(SL_DEVICE_GENERAL_CONFIGURATION, &ucConfigOpt,
                     &ucConfigLen, (unsigned char *)(&ver));
ASSERT_ON_ERROR(lRetVal);

UART_PRINT("Host Driver Version: %s\n\r", SL_DRIVER_VERSION);
UART_PRINT("Build Version %d.%d.%d.%d.%d.%d.%d.%d.%d.%d\n\r",
ver.NwpVersion[0], ver.NwpVersion[1], ver.NwpVersion[2], ver.NwpVersion[3],
ver.ChipFwAndPhyVersion.FwVersion[0], ver.ChipFwAndPhyVersion.FwVersion[1],
ver.ChipFwAndPhyVersion.FwVersion[2], ver.ChipFwAndPhyVersion.FwVersion[3],
ver.ChipFwAndPhyVersion.PhysicalVersion[0], ver.ChipFwAndPhyVersion.PhysicalVersion[1],
ver.ChipFwAndPhyVersion.PhysicalVersion[2], ver.ChipFwAndPhyVersion.PhysicalVersion[3]);

ver.ChipFwAndPhyVersion.PhysicalVersion[2], ver.ChipFwAndPhyVersion.PhysicalVersion[3]);

// Set connection policy to Auto + SmartConfig (Device's default connection
policy)
lRetVal = sl_WlanPolicySet(SL_POLICY_CONNECTION,
                           SL_CONNECTION_POLICY(1, 0, 0, 0, 1), NULL, 0);
ASSERT_ON_ERROR(lRetVal);

// Deletes all stored profiles
lRetVal = sl_WlanProfileDel(0xFF);
ASSERT_ON_ERROR(lRetVal);

// Device in station-mode. Disconnect previous connection if any. The
function returns
// 0 if 'Disconnected done', negative number if already disconnected. Ignore
other return-codes
lRetVal = sl_WlanDisconnect();
if(!lRetVal) {
    while(IS_CONNECTED(g_ulStatus)) {
#ifndef SL_PLATFORM_MULTI_THREADED
        _SlNonOsMainLoopTask();
#endif
    }
}

// Enable DHCP client. Setting IP address by DHCP to FileSystem using WLAN
sta mode
// This is the system's default mode for acquiring an IP address after WLAN
connection
lRetVal = sl_NetCfgSet(SL_IPV4_STA_P2P_CL_DHCP_ENABLE, 1, 1, &ucVal);
ASSERT_ON_ERROR(lRetVal);

// Disable scan
ucConfigOpt = SL_SCAN_POLICY(0);
lRetVal = sl_WlanPolicySet(SL_POLICY_SCAN, ucConfigOpt, NULL, 0);
ASSERT_ON_ERROR(lRetVal);

// Set Tx power level for station mode
// Number between 0-15, as dB offset from max power - 0 will set max power
ucPower = 0;
lRetVal = sl_WlanSet(SL_WLAN_CFG_GENERAL_PARAM_ID,
                     WLAN_GENERAL_PARAM_OPT_STA_TX_POWER, 1, (unsigned char *)&ucPower);
ASSERT_ON_ERROR(lRetVal);

// Set PM policy to normal
lRetVal = sl_WlanPolicySet(SL_POLICY_PM, SL_NORMAL_POLICY, NULL, 0);
ASSERT_ON_ERROR(lRetVal);

// Unregister mDNS services
lRetVal = sl_NetAppMDNSUnRegisterService(0, 0);
ASSERT_ON_ERROR(lRetVal);

// Remove all 64 filters (8*8)
memset(RxFilterIdMask.FilterIdMask, 0xFF, 8);
lRetVal = sl_WlanRxFilterSet(SL_REMOVE_RX_FILTER, (_u8 *)&RxFilterIdMask,
                           sizeof(_WlanRxFilterOperationCommandBuff_t));

```

```

    ASSERT_ON_ERROR(lRetVal);

    lRetVal = sl_Stop(SL_STOP_TIMEOUT);
    ASSERT_ON_ERROR(lRetVal);

    InitializeVariables();

    return lRetVal; // Success
}

/*********************************************
 *brief      This function sets the WLAN mode as AP mode
 *param      iMode is the current mode of the device
 *return     sl_start return value(int).
********************************************/

static int SetAPMode(int iMode)
{
    long lRetVal      = -1;
    char pcSsidName[] = "CC3200Thesis_N.L."; // SSID of CC3200 AP

    UART_PRINT("SSID: CC3200Thesis_N.L.\n\r");

    lRetVal = sl_WlanSetMode(ROLE_AP); // API sets AP mode
    ASSERT_ON_ERROR(lRetVal);

    lRetVal = sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_SSID, //API sets Wlan
properties
                         strlen(pcSsidName), (unsigned char*) pcSsidName);
    ASSERT_ON_ERROR(lRetVal);

    UART_PRINT("Device is configured in AP mode\n\r");

    // Restart Network processor - Device has to be reset to apply changes
    lRetVal = sl_Stop(SL_STOP_TIMEOUT);

    // reset status bits
    CLR_STATUS_BIT_ALL(g_ulStatus);

    return sl_Start(NULL, NULL, NULL);
}

/*********************************************
 *brief      CC3200 will try to ping the machine that has just connected to it
 *param      ulIpAddr is the ip address of the station which has connected to
 *           the device
 *return     0 if ping is successful, -1 for error
********************************************/

static long PingTest(unsigned long ulIpAddr)
{
    signed long          lRetVal = -1;
    SlPingStartCommand_t PingParams;
    SlPingReport_t       PingReport; // for statistics

    CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_PING_DONE); // Clear ping status-bit

    PingParams.PingIntervalTime = PING_INTERVAL;
    PingParams.PingSize = PING_PKT_SIZE;
    PingParams.PingRequestTimeout = PING_TIMEOUT;
    PingParams.TotalNumberOfAttempts = NO_OF_ATTEMPTS;
}

```

```

PingParams.Flags = PING_FLAG;
PingParams.Ip = ulIpAddr; // Client's ip address

UART_PRINT("Pinging...!\n\r");
// Pings the network host, by sending ICMP ECHO_REQUEST
lRetVal = sl_NetAppPingStart((SlPingStartCommand_t*)&PingParams, SL_AF_INET,
                             (SlPingReport_t*)&PingReport, SimpleLinkPingReport);
ASSERT_ON_ERROR(lRetVal);

g_ulPingPacketsRecv = PingReport.PacketsReceived;

// Wait for NetApp Event
while(!IS_PING_DONE(g_ulStatus)) {
#ifndef SL_PLATFORM_MULTI_THREADED
    _SlNonOsMainLoopTask();
#endif
}

if ( (g_ulPingPacketsRecv > 0) && (g_ulPingPacketsRecv <= NO_OF_ATTEMPTS) )
{
    // LAN connection successful
    return SUCCESS;
} else // Problem with LAN connection
    ASSERT_ON_ERROR(LAN_CONNECTION_FAILED);
}

/*****************
 *brief      Start simplelink to default state. Assert AP WiFi mode, wait for the
station
 *
 *           to connect to the device, run the ping test. Configure uDMA
controller and
 *
 *           set socket parameters

*param      pparameters is the pointer to the list of parameters that can be
*           passed to the task while creating it. Not in use here.
*return     None

*note       1) AP mode must use static IP settings.
 *           2) All NWP status change functions require system restart for
changes to take effect
 *
 *           3) CC3200 AP mode supports only one client/server connected at a
time
 *
 *           4) This Task should not be disturbed from other tasks until it is
complete
*****************/
void InitialSetupAP( void *pvParameters)
{
    int          iPingResult      = 0;
    unsigned char ucDHCP;
    long         lRetVal         = -1;
    unsigned int uiUdpPortNum   = UDP_PORT;
    unsigned int uiTcpPortNum   = TCP_PORT;
    OsiReturnVal_e xSemaphoreResult;
    SlNetCfgIpV4Args_t ipV4 = {0}; // For printing Network stats on screen
    unsigned char len = sizeof(SlNetCfgIpV4Args_t);

    InitializeVariables();

    // Suspend other tasks until this one is done
    Task_setPri(UdpHandle, INACTIVE_STATE_PRIORITY);
    Task_setPri(TcpHandle, INACTIVE_STATE_PRIORITY);
    Task_setPri(GreenLightHandle, INACTIVE_STATE_PRIORITY);
#ifndef TERMINAL

```

```

        // Lets the user make decision on the communication protocol based on button
pressed
        DecisionMaking();
#endif

        // We could skip the following function if we are certain the device will
start in
        // it's default state. - I'm not.
lRetVal = ConfigureSimpleLinkToDefaultState();
if(lRetVal < 0) {
    if (DEVICE_NOT_IN_STATION_MODE == lRetVal)
        UART_PRINT("WARNING: Failed to configure the device in its default
state.\n\r");
    sl_Stop(SL_STOP_TIMEOUT);
}
UART_PRINT("Device is configured in default state \n\r");

        // If everything's alright restart the network processor
lRetVal = sl_Start(NULL,NULL,NULL);
if (lRetVal < 0) {
    UART_PRINT("WARNING: Failed to start the device \n\r");
    sl_Stop(SL_STOP_TIMEOUT);
}

        // Configures the networking mode and ssid name for AP mode
if(lRetVal != ROLE_AP) {
    if(SetAPMode(lRetVal) != ROLE_AP) {
        UART_PRINT("WARNING: Unable to set AP mode, exiting
Application...\n\r");
        sl_Stop(SL_STOP_TIMEOUT);
    }
}

while(!IS_IP_ACQUIRED(g_ulStatus)) { // Callback at work here
    //looping till ip is acquired
}

        // Get network configuration. Assigns static IP to itself = 192.168.1.1
lRetVal = sl_NetCfgGet(SL_IPV4_AP_P2P_GO_GET_INFO,&ucDHCP,&len,
                      (unsigned char *)&ipV4);
if (lRetVal < 0) {
    UART_PRINT("WARNING: Failed to get network configuration \n\r");
    sl_Stop(SL_STOP_TIMEOUT);
}
// Display CC3200 network status
UART_PRINT("CC3200 Networks status:\n\r"
DHCP: ON IP: %d.%d.%d.%d MASK: %d.%d.%d.%d GW %d.%d.%d.%d DNS:
%d.%d.%d\n\n\r",
SL_IPV4_BYT(ipV4.ipV4,3),SL_IPV4_BYT(ipV4.ipV4,2),SL_IPV4_BYT(ipV4.ipV4,1),

SL_IPV4_BYT(ipV4.ipV4,0),SL_IPV4_BYT(ipV4.ipV4Mask,3),SL_IPV4_BYT(ipV4.ipV4Ma
sk,2),
SL_IPV4_BYT(ipV4.ipV4Mask,1),SL_IPV4_BYT(ipV4.ipV4Mask,0),
SL_IPV4_BYT(ipV4.ipV4Gateway,3),SL_IPV4_BYT(ipV4.ipV4Gateway,2),
SL_IPV4_BYT(ipV4.ipV4Gateway,1),SL_IPV4_BYT(ipV4.ipV4Gateway,0),
SL_IPV4_BYT(ipV4.ipV4DnsServer,3),SL_IPV4_BYT(ipV4.ipV4DnsServer,2),
SL_IPV4_BYT(ipV4.ipV4DnsServer,1),SL_IPV4_BYT(ipV4.ipV4DnsServer,0) );

UART_PRINT("Connect a client to Device\n\r");
while(!IS_IP_LEASED(g_ulStatus)) {
    //waiting for a client to connect
}
UART_PRINT("Client is connected to Device\n\r");

```

```

iPingResult = PingTest(g_ulDestIp);
if(iPingResult < 0) {
    UART_PRINT("WARNING: Ping to client failed\n\r");
}
UART_PRINT("Ping succesfull\n\r");

// Create the Binary-Semaphore variable
xSemaphoreResult = osi_SyncObjCreate( &g_sendSignal );
if (xSemaphoreResult){
    UART_PRINT("WARNING: Semaphore creation error\n\r");
    sl_Stop(SL_STOP_TIMEOUT);
    GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
    exit(EXIT_SEMAPHORE_FAIL);
}

// Open and configure the appropriate protocol sockets and then resume other
tasks
switch (g_bUdpOrTcp){
case TRUE:
    lRetVal = UdpClientServerConfig(uiUdpPortNum);
    Task_setPri(UdpHandle, COM_PROTOCOL_TASK_PRIORITY);
    Task_setPri(GreenLightHandle, GREEN_LIGHT_PRIORITY);
    break;
case FALSE:
    lRetVal = TcpClientServerConfig(uiTcpPortNum);
    Task_setPri(TcpHandle, COM_PROTOCOL_TASK_PRIORITY);
    Task_setPri(GreenLightHandle, GREEN_LIGHT_PRIORITY);
    break;
}
if (lRetVal < 0){
    UART_PRINT("WARNING: Could not set socket parameters\n\r");
    sl_Stop(SL_STOP_TIMEOUT);
    GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
    exit(EXIT_SOCKET_FAIL);
}

// Configure the uDMA for communication with the external device
lRetVal = Uart1ConfigDmaTransfer();
if (lRetVal < 0){
    UART_PRINT("WARNING: Failed to configure uDMA controller
parameters.\n\r");
    sl_Stop(SL_STOP_TIMEOUT);
    GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
    exit(EXIT_DMA_FAIL);
}
else
    UART_PRINT("uDMA controller activated.\n\r");

// When all Setup processes are done light the red led
GPIO_IF_LedOn(MCU_RED_LED_GPIO);
}

/*****************
 *brief This function checks the internet connection by pinging an external
host, as
 *
 *resolved by Domain Name System (DNS) protocol.
 *
 *param None
 *return 0 on success, negative error-code on error
*****************/
#endif TERMINAL
static long WlanPingInternetHost()
{

```

```

long          lRetVal      = -1;
unsigned long ulIpExtHostAddr = 0;
SlPingStartCommand_t pingParams   = {0};
SlPingReport_t    pingReport     = {0};

CLR_STATUS_BIT(g_ulStatus, STATUS_BIT_PING_DONE); // Clear ping status bit
g_ulPingPacketsRecv = 0;

// Set the ping parameters
pingParams.PingIntervalTime = PING_INTERVAL;
pingParams.PingSize = PING_PKT_SIZE;
pingParams.PingRequestTimeout = PING_TIMEOUT;
pingParams.TotalNumberOfAttempts = NO_OF_ATTEMPTS;
pingParams.Flags = 0;
pingParams.Ip = g_ulGatewayIP;

// API gets IP of HOST_NAME using DNS
lRetVal = sl_NetAppDnsGetHostByName( (signed char*)HOST_NAME,
sizeof(HOST_NAME),
&ulIpExtHostAddr, SL_AF_INET);
ASSERT_ON_ERROR(lRetVal); // In case of Internet access/DNS failure

// Replace the ping address to match HOST_NAME's IP address
pingParams.Ip = ulIpExtHostAddr;

UART_PRINT("Pinging...!\n\r");
// tries to ping HOST_NAME's.. now ip address
lRetVal = sl_NetAppPingStart((SlPingStartCommand_t*)&pingParams, SL_AF_INET,
                             (SlPingReport_t*)&pingReport, SimpleLinkPingReport);
ASSERT_ON_ERROR(lRetVal);

while(!IS_PING_DONE(g_ulStatus)) { // Wait for Ping Event
#ifndef SL_PLATFORM_MULTI_THREADED // Must be called on NONOS apps
    _SlNonOsMainLoopTask(); // unused in this program
#endif
}

if ( (g_ulPingPacketsRecv > 0) && (g_ulPingPacketsRecv <= NO_OF_ATTEMPTS) )
{
    return SUCCESS; // Internet access is possible
} else // Problem with LAN connection
    ASSERT_ON_ERROR(INTERNET_CONNECTION_FAILED);
}

/*****************
 *brief This function connects to the AP with SSID name and security
parameters
 *
 *specified by the user at startup.

*param None
*return None

*warning! If the WLAN connection fails due to erroneous parameters, SSID
name, or
 *
 *inexistence of such a network execution will be stuck in this
function forever.
***** */

```

```

static long WlanConnect()
{
    SlSecParams_t secParams = {0};
    long         lRetVal   = 0;

    secParams.Key = (signed char*)g_ucSecurityKey;
    secParams.KeyLen = strlen(g_ucSecurityKey);

```

```

secParams.Type = g_ucSecurityType;

lRetVal = sl_WlanConnect((signed char*)g_ucNetworkSSID,
strlen(g_ucNetworkSSID), 0,
&secParams, 0);
ASSERT_ON_ERROR(lRetVal);

// Wait for WLAN Event acquisition
while( (!IS_CONNECTED(g_ulStatus)) || (!IS_IP_ACQUIRED(g_ulStatus)) ) {
    // Toggle LEDs to Indicate Connection Progress
    GPIO_IF_LedOff(MCU_IP_ALLOC_IND);
    MAP_UtilsDelay(800000);
    GPIO_IF_LedOn(MCU_IP_ALLOC_IND);
    MAP_UtilsDelay(800000);
}

return SUCCESS;
}

/*****************
 *brief      Start simplelink to default state. Connect to the specified AP, run
ping tests
 *
*          to verify that connection status OK. Configure uDMA controller and
set socket parameters

*param     pvParameters - Pointer to the list of parameters that can be passed
to the
*          task while creating it
*return    None

*note      1) All NWP functions require system restart for changes to take
effect
 *
*          2) This Task should not be disturbed by other tasks until it is
complete
*****************/
void InitialSetupWlan( void *pvParameters )
{
    int             iPingResult      = 0;
    long            lRetVal         = -1;
    unsigned int    uiUdpPortNum    = UDP_PORT;
    unsigned int    uiTcpPortNum    = TCP_PORT;
    OsiReturnVal_e  xSemaphoreResult;

    InitializeVariables();

    // Suspend other tasks until this one is done
    Task_setPri(UdpHandle, INACTIVE_STATE_PRIORITY);
    Task_setPri(TcpHandle, INACTIVE_STATE_PRIORITY);
    Task_setPri(GreenLightHandle, INACTIVE_STATE_PRIORITY);

    // Setup CC3200 on default state
    lRetVal = ConfigureSimpleLinkToDefaultState();
    if(lRetVal < 0) {
        if (DEVICE_NOT_IN_STATION_MODE == lRetVal)
            UART_PRINT("WARNING: Failed to configure the device in its default
state.\n\r");
        sl_Stop(SL_STOP_TIMEOUT);
    }
    UART_PRINT("Device is configured in default state \n\r");

    // If everything's alright re-start the network processor
    lRetVal = sl_Start(NULL,NULL,NULL);
    if (lRetVal < 0) {
        UART_PRINT("WARNING: Failed to start the device \n\r");
    }
}

```

```

        sl_Stop(SL_STOP_TIMEOUT);
    }
    UART_PRINT("Device started as STATION \n\r");

    // Connect to WLAN AP
    lRetVal = WlanConnect();
    if(lRetVal < 0) {
        UART_PRINT("WARNING: Failed to establish connection with an AP \n\r");
        sl_Stop(SL_STOP_TIMEOUT);
    }
    UART_PRINT("Connection established with AP and IP is aquired\n\r");

    // Checking Lan connection by pinging to the AP gateway
    iPingResult = PingTest(g_ulGatewayIP);
    if(iPingResult < 0) {
        UART_PRINT("WARNING: CC3200 could not ping the gateway \n\r");
        sl_Stop(SL_STOP_TIMEOUT);
    }
    UART_PRINT("Pinged the Gateway!\n\r");

    // Checking Network connection by pinging to the AP gateway
    iPingResult = PingTest(g_ulDestIp);
    if(iPingResult < 0) {
        UART_PRINT("WARNING: CC3200 could not ping the target host \n\r");
        sl_Stop(SL_STOP_TIMEOUT);
    }
    UART_PRINT("Pinged the Target!\n\r");

    // Checking the internet connection by pinging to external host
    iPingResult = WlanPingInternetHost();
    if(iPingResult < 0) {
        UART_PRINT("WARNING: CC3200 could not ping the external host \n\r");
        sl_Stop(SL_STOP_TIMEOUT);
    }
    UART_PRINT("Pinged the External Host!\n\rNetwork connection
established\n\r");

    // Create the Binary-Semaphore variable
    xSemaphoreResult = osi_SyncObjCreate( &g_sendSignal );
    if (xSemaphoreResult){
        UART_PRINT("WARNING: Semaphore creation error\n\r");
        sl_Stop(SL_STOP_TIMEOUT);
        GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
        exit(EXIT_SEMAPHORE_FAIL);
    }

    // Open and configure the appropriate protocol sockets and then resume other
tasks
    switch (g_bUdpOrTcp) {
        case TRUE:
            lRetVal = UdpClientServerConfig(uiUdpPortNum);
            Task_setPri(UdpHandle, COM_PROTOCOL_TASK_PRIORITY);
            Task_setPri(GreenLightHandle, GREEN_LIGHT_PRIORITY);
            break;
        case FALSE:
            lRetVal = TcpClientServerConfig(uiTcpPortNum);
            Task_setPri(TcpHandle, COM_PROTOCOL_TASK_PRIORITY);
            Task_setPri(GreenLightHandle, GREEN_LIGHT_PRIORITY);
            break;
    }
    if (lRetVal < 0) {
        UART_PRINT("WARNING: Could not set socket parameters\n\r");
        sl_Stop(SL_STOP_TIMEOUT);
        GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
        exit(EXIT_SOCKET_FAIL);
    }
}

```

```

}

// Configure the uDMA for communication with the external device
lRetVal = Uart1ConfigDmaTransfer();
if (lRetVal < 0) {
    UART_PRINT("WARNING: Failed to configure uDMA controller
parameters.\n\r");
    sl_Stop(SL_STOP_TIMEOUT);
    GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
    exit(EXIT_DMA_FAIL);
}
else
    UART_PRINT("uDMA controller activated.\n\r");

// When all Setup processes are done light the red led
GPIO_IF_LedOn(MCU_RED_LED_GPIO);
}

/*********************  

 *brief      This function will be used in case no computer is available as a  

means for
 *          the user to insert input to choose the desired protocol of  

communication.
 *          If button SW3 is pressed UDP will be the selected protocol.  

 *          If button SW2 is pressed TCP will be the selected protocol.  

 *          User will have a timelimit specified by TimerA0 timeout to make  

his choise,  

 *          or else he'll have to restart the device.  

*param[in]  none
*return     none
*****  

#else
void DecisionMaking(void)
{
    long lRetVal = -1;

    while ( !g_bButtonPress )
    {
        // Loop here until a button is pressed
    }

    if (g_bUdpOrTcp)
        UART_PRINT("UDP Selected.\n\r");
    else{
        UART_PRINT("TCP Selected.\n\r");

        // Setup and Register Timer A1
        lRetVal =
ConfigureTimeoutTimer(TIMERA1_BASE, PRCM_TIMERA1, TIMER_CFG_PERIODIC, 0, \
                    INT_PRIORITY_LVL_2, TimerA1InterruptHandler, 2000, TIMER_A);
        if (lRetVal < 0)
            exit(EXIT_TASK_FAIL);

        while ( !g_bTimerA1Flag )
        {
            // Loop for 2 seconds. If the SW2 button has been pressed again
during
                // this time period CC3200 will operate as a TCP Server. Otherwise
as TCP client
        }

        if ( g_uiNoButtonPress == 1 ){
            g_bTcpClientOrServer = TRUE;

```

```

        UART_PRINT("CC3200=TCP Client\n\n\r");
    }else if ( g_uiNoButtonPress == 2 ){
        g_bTcpClientOrServer = FALSE;
        UART_PRINT("CC3200=TCP Server\n\n\r");
    }

    // Disable and unregister TimerA1 interrupt. It's done
    MAP_TimerIntDisable(TIMERA1_BASE, TIMER_TIMA_TIMEOUT);
    MAP_IntDisable(INT_TIMERA1A);
    MAP_IntUnregister(INT_TIMERA1A);
}

// Disable and unregister TimerA0 interrupt. It's not needed anymore
MAP_TimerIntDisable(TIMERA0_BASE, TIMER_TIMA_TIMEOUT);
MAP_IntDisable(INT_TIMERA0A);
MAP_IntUnregister(INT_TIMERA0A);
//Clear, Disable and unregister the GPIO Button SW2-Interrupt. Served it's
purpose
MAP_GPIOIntDisable(GPIOA2_BASE, GPIO_INT_PIN_6);
MAP_GPIOIntClear(GPIOA2_BASE, GPIO_INT_PIN_6);
MAP_IntDisable(INT_GPIOA2);
MAP_IntUnregister(INT_GPIOA2);
//Clear, Disable and unregister the GPIO SW3-Button Interrupt. We no longer
need it
MAP_GPIOIntDisable(GPIOA1_BASE, GPIO_INT_PIN_5);
MAP_GPIOIntClear(GPIOA1_BASE, GPIO_INT_PIN_5);
MAP_IntDisable(INT_GPIOA1);
MAP_IntUnregister(INT_GPIOA1);
}

/*********************brief Initialize and setup a Timer peripheral module. Setup Interrupt
trigger source
 * as timeout interrupt. Finally enable the specified timer/counter for
operation.

 *param ulTimerBase      = Base Address for the timer peripheral
selected
 *param ulTimerPeriph   = Specified Timer Peripheral
 *param ulTimerMode      = Mode that the timer will be utilized
 *param ulPrescaleVal   = Sets the value for the prescaler, for 0 the
prescaler
 *
 *                      is not used
 *param usIntPriority   = Priority of the timer interrupt
 *param TimerInterruptHandler = Pointer to the Interrupt Handling function,
which has
 *
 *                      no arguments and returns void/nothing
 *param ulTimeoutVal    = Timeout value that upon met the interrupt
will hit
 *param ulTimer          = Timer subperipheral module in use
 *return none
*****/



int ConfigureTimeoutTimer(unsigned long ulTimerBase, unsigned long
ulTimerPeriph,
    unsigned long ulTimerMode, unsigned long ulPrescaleVal, unsigned short
usIntPriority,
    void (*TimerInterruptHandler) (void), unsigned long ulTimeoutVal, unsigned
long ulTimer )
{
    int lRetVal = -1;

    // Initializing Selected Timer
    MAP_PRCMPeripheralClkEnable(ulTimerPeriph, PRCM_RUN_MODE_CLK);
}

```

```

MAP_PRCMPeripheralReset(ulTimerPeriph);
MAP_TimerConfigure(ulTimerBase,ulTimerMode);
MAP_TimerPrescaleSet(ulTimerBase,ulTimer,ulPrescaleVal);

    // Setup the interrupts for the timer
#if defined(USE_TIRTOS) || defined(USE_FREERTOS) ||
defined(SL_PLATFORM_MULTI_THREADED)
    if(ulTimer == TIMER_BOTH) {
        lRetVal = osi_InterruptRegister(GetTimerInterruptNum(ulTimerBase,
TIMER_A),
                                         (P_OSI_INTR_ENTRY)TimerInterruptHandler, usIntPriority);
        lRetVal = osi_InterruptRegister(GetTimerInterruptNum(ulTimerBase,
TIMER_B),
                                         (P_OSI_INTR_ENTRY)TimerInterruptHandler, usIntPriority);
    }else{
        lRetVal = osi_InterruptRegister(GetTimerInterruptNum(ulTimerBase,
ulTimer),
                                         (P_OSI_INTR_ENTRY)TimerInterruptHandler, usIntPriority);
    }
#else
    MAP_IntPrioritySet(GetTimerInterruptNum(ulTimerBase, ulTimer),
usIntPriority);
    MAP_TimerIntRegister(ulTimerBase, ulTimer, TimerInterruptHandler);
#endif
    if ( lRetVal < 0 ){
        UART_PRINT("CRITICAL ERROR: Timer interrupt register failure.\n\r");
        ASSERT_ON_ERROR(INTERRUPT_ERROR);
    }

    // Enable the Timer Interrupt trigger source
    if(ulTimer == TIMER_BOTH)
        MAP_TimerIntEnable(ulTimerBase, TIMER_TIMA_TIMEOUT|TIMER_TIMB_TIMEOUT);
    else
        MAP_TimerIntEnable(ulTimerBase, ( (ulTimer == TIMER_A) ?
TIMER_TIMA_TIMEOUT :
TIMER_TIMB_TIMEOUT ) );

    // Set the timer timeout value
    MAP_TimerLoadSet(ulTimerBase,ulTimer,MILLISECONDS_TO_TICKS(ulTimeoutVal));
    // Enable the GPTimer module with all parameters specified
    MAP_TimerEnable(ulTimerBase,ulTimer);

    return SUCCESS;
}

```

```

*****
 *brief      This function selects and returns the requested Timer Interrupt
Number

*param      ulTimerBase = Base Address of the Timer Peripheral selected
*param      ulTimer      = The selected Timer module
*return
*****


static unsigned char GetTimerInterruptNum(unsigned long ulTimerBase, unsigned
long ulTimer)
{
    if (ulTimer == TIMER_A) {
        switch (ulTimerBase) {
        case TIMERA0_BASE:
            return INT_TIMERA0A;
        case TIMERA1_BASE:
            return INT_TIMERA1A;
        case TIMERA2_BASE:
            return INT_TIMERA2A;
    }
}

```

```

        case TIMERA3_BASE:
            return INT_TIMERA3A;
        default:
            return INT_TIMERA0A;
    }
} else if (ulTimer == TIMER_B) {
    switch (ulTimerBase) {
    case TIMERA0_BASE:
        return INT_TIMERA0B;
    case TIMERA1_BASE:
        return INT_TIMERA1B;
    case TIMERA2_BASE:
        return INT_TIMERA2B;
    case TIMERA3_BASE:
        return INT_TIMERA3B;
    default:
        return INT_TIMERA0B;
    }
} else
    return INT_TIMERA0A;
}

/**************************************************************************
 *brief      Initializes Push Button SW3 and registers an interrupt handler for it
 */
*param[in]  Sw2ButtonHandler : Interrupt Handler function for SW2 LP button
*param[in]  Sw3ButtonHandler : Interrupt Handler function for SW3 LP button
*return     none
/**************************************************************************/

int ButtonSetup ( void (*SW3ButtonInterruptHandler) (void),
                  void (*SW2ButtonInterruptHandler) (void) )
{
    long lRetVal = -1;

    if(SW3ButtonInterruptHandler != NULL) {
        // Set Interrupt Type for GPIO
        MAP_GPIOIntTypeSet(GPIOA1_BASE,GPIO_PIN_5,GPIO_FALLING_EDGE);

        // Register Interrupt handler
#if defined(USE_TIRTOS) || defined(USE_FREERTOS) ||
defined(SL_PLATFORM_MULTI_THREADED)
        lRetVal = osi_InterruptRegister(INT_GPIOA1,
(P_OSI_INTR_ENTRY)SW3ButtonInterruptHandler,
INT_PRIORITY_LVL_2);
        if ( lRetVal < 0 ){
            UART_PRINT("CRITICAL ERROR: Button SW3 Interrupt register failed.\n\r");
            ASSERT_ON_ERROR(INTERRUPT_ERROR);
        }
#else
        MAP_IntPrioritySet(INT_GPIOA1, INT_PRIORITY_LVL_2);
        MAP_GPIOIntRegister(GPIOA1_BASE, SW3ButtonInterruptHandler);
#endif
        // Enable Interrupt
        MAP_GPIOIntClear(GPIOA1_BASE,GPIO_INT_PIN_5);
        MAP_IntPendingClear(INT_GPIOA1);
        MAP_IntEnable(INT_GPIOA1);
        MAP_GPIOIntEnable(GPIOA1_BASE,GPIO_INT_PIN_5);
    }

    if(SW2ButtonInterruptHandler != NULL) {
        // Set Interrupt Type
        MAP_GPIOIntTypeSet(GPIOA2_BASE,GPIO_PIN_6,GPIO_FALLING_EDGE);

```

```

        // Register Interrupt handler
#if defined(USE_TIRTOS) || defined(USE_FREERTOS) ||
defined(SL_PLATFORM_MULTI_THREADED)
    lRetVal =
osi_InterruptRegister(INT_GPIOA2,(P_OSI_INTR_ENTRY)SW2ButtonInterruptHandler,
                        INT_PRIORITY_LVL_2);
    if ( lRetVal < 0 ){
        UART_PRINT("CRITICAL ERROR: Button SW2 Interrupt register
failed.\n\r");
        ASSERT_ON_ERROR(INTERRUPT_ERROR);
    }
#else
    MAP_IntPrioritySet(INT_GPIOA2, INT_PRIORITY_LVL_2);
    MAP_GPIOIntRegister(GPIOA2_BASE, SW2ButtonInterruptHandler);
#endif
    // Enable Interrupt
    MAP_GPIOIntClear(GPIOA2_BASE,GPIO_INT_PIN_6);
    MAP_IntPendingClear(INT_GPIOA2);
    MAP_IntEnable(INT_GPIOA2);
    MAP_GPIOIntEnable(GPIOA2_BASE,GPIO_INT_PIN_6);
}

return SUCCESS;
}
#endif

/*****
 *brief      UDP socket parameters configuration. Opens a UDP client side
 *           socket. Then binds the socket to local address to be used in
 *           listening mode for clients to gain access.
 *           Does not make transmissions here. The UDP server must be
 *           already connected and listening to the device on UDP_PORT.
 *
 *param[in]   port number on which the server will be listening on
 *return     0 on success, -1 on error
***** */

int UdpClientServerConfig(unsigned short usPort)
{
    int iStatus;
    SlSockAddrIn_t UdpLocalAddr; // Contains parameters for server-side
socket services
#ifdef NON_BLOCKING_SOCKET
    SlSockNonblocking_t enableOptionNB;
#endif

    // creating a (Blocking by default) UDP socket - an endpoint for
communication.
    // Blocking socket means that the commands sl_Recv/sl_RecvFrom are blocked
    // (aka, put the task to sleep, until the asynchronous event triggers) until
the
    // buffer specified is completely loaded.
    g_iUdpSockHandle = sl_Socket(SL_AF_INET,SL SOCK_DGRAM, IPPROTO_UDP);
    if( g_iUdpSockHandle < 0 ) {
        UART_PRINT("WARNING: UDP socket creation fault.\n\r");
        sl_Close(g_iUdpSockHandle);
        ASSERT_ON_ERROR(SOCKET_CREATE_ERROR);
    }
    else
        UART_PRINT("UDP Socket creation complete.\n\r");

    /* CC3200 as UDP CLIENT Setup */

```

```

        // filling the connected host's parameters address, port, protocol, name
etc.
        g_slSocketParam.sin_family = SL_AF_INET;
        g_slSocketParam.sin_port = sl_Htons( (unsigned short)usPort );
        g_slSocketParam.sin_addr.s_addr = sl_Htonl( (unsigned int)g_ulDestIp );

#endif NON_BLOCKING_SOCKET /* Force non-blocking socket implementation */
enableOptionNB.NonblockingEnabled = 1;

iStatus = sl_SetSockOpt(g_iUdpSockHandle, SL_SOL_SOCKET, SL_SO_NONBLOCKING,
(_u8 *)&enableOptionNB, sizeof(enableOptionNB));
if( iStatus < 0 ) {
    UART_PRINT( "WARNING: Error setting socket option.\n\r");
    sl_Close(g_iUdpSockHandle);
    ASSERT_ON_ERROR(SOCKET_OPT_ERROR);
}
else
    UART_PRINT( "UDP Socket set as Non-Blocking.\n\r");
#endif

/* CC3200 as UDP SERVER Setup */

// filling the UDP local socket address
UdpLocalAddr.sin_family = SL_AF_INET;
// reorder bytes from CPU order to Network order, i.e. in Big-Endian format
UdpLocalAddr.sin_port = sl_Htons((unsigned short)usPort);
UdpLocalAddr.sin_addr.s_addr = 0; // 0 indicates local host

// binding local socket to a specific UDP port
iStatus = sl_Bind(g_iUdpSockHandle, (SlSockAddr_t *)&UdpLocalAddr,
g_iSockBlockSize);
if( iStatus < 0 ) {
    UART_PRINT( "WARNING: UDP socket binding error.\n\r");
    sl_Close(g_iUdpSockHandle);
    ASSERT_ON_ERROR(BIND_ERROR);
}
else
    UART_PRINT( "UDP Socket binding complete.\n\r");

return SUCCESS;
}

/*****************
 *brief      This function opens a TCP socket client and server side socket.
 *          It sets the parameters for the TCP sockets, makes the socket a
non-blocking
 *
 *          socket to optimize performance during communication.
 *
 *          It tries to connect to a TCP client/server by putting the socket
in
 *
 *          listening mode and accepting a connection, before actually
the sending/receiving of useful payload occurs.

*param[in]  port number on which the server will be listening on
*return     0 on success, -1 on error

*note       1) This function will wait for an incoming connection until one is
established
 *
 *note       2) TCP Protocol must first handshake the two endpoints before
transferring the
 *
 *          payload.
*****************/
int TcpClientServerConfig(unsigned short usPort)

```

```

{
    int iStatus;
    SlSockAddrIn_t TcpLocalAddress;
#ifndef NON_BLOCKING_SOCKET
    SlSockNonblocking_t enableOptionNB;
#endif

    UART_PRINT("TCP Configuration initiated. A TCP host should be ready & stand
by.\n\r");
    GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
    MAP_UtilsDelay(50000000); // Press "connect" here on remote end
    GPIO_IF_LedOff(MCU_ORANGE_LED_GPIO);

    // creating a TCP socket
    g_iTcpSockHandle = sl_Socket(SL_AF_INET, SL SOCK_STREAM, IPPROTO_TCP);
    if( g_iTcpSockHandle < 0 ) {
        UART_PRINT("WARNING: TCP socket creation error.\n\r");
        sl_Close(g_iTcpSockHandle);
        ASSERT_ON_ERROR(SOCKET_CREATE_ERROR);
    }
    else
        UART_PRINT("TCP Socket creation complete.\n\r");

#ifndef NON_BLOCKING_SOCKET /* Setting socket as a Non-Blocking Socket */
    enableOptionNB.NonblockingEnabled = 1;
    iStatus = sl_SetSockOpt( g_iTcpSockHandle, SL_SOL_SOCKET,
    SL_SO_NONBLOCKING,
                    (_u8 *)&enableOptionNB, sizeof(enableOptionNB) );
    if( iStatus < 0 ) {
        UART_PRINT("WARNING: Error in modifying TCP socket option.\n\r");
        sl_Close(g_iTcpSockHandle);
        ASSERT_ON_ERROR(SOCKET_OPT_ERROR);
    }
    else
        UART_PRINT("Defining TCP Socket as Non-Blocking.\n\r");
#endif

    //filling the TCP local socket address
    TcpLocalAddress.sin_family = SL_AF_INET;
    TcpLocalAddress.sin_port = sl_Hton((unsigned short)usPort);
    TcpLocalAddress.sin_addr.s_addr = 0; // filling my local host IP

    // binding local socket to a specific TCP port
    iStatus = sl_Bind(g_iTcpSockHandle,(SlSockAddr_t *)&TcpLocalAddress,
    g_iSockBlockSize);
    if( iStatus < 0 ) {
        UART_PRINT("WARNING: TCP socket binding error.\n\r");
        sl_Close(g_iTcpSockHandle);
        ASSERT_ON_ERROR(BIND_ERROR);
    }
    else
        UART_PRINT("TCP Socket port bind complete.\n\r");

    if (g_bTcpClientOrServer) { /* CC3200 as TCP CLIENT Setup */
        // filling the TCP server socket address. Should use only when CC3200 is
        TCP client
        g_slSocketParam.sin_family = SL_AF_INET;
        g_slSocketParam.sin_port = sl_Hton((unsigned short) usPort); // reorder
        bytes from CPU order to Network byte order
        g_slSocketParam.sin_addr.s_addr = sl_Htonl((unsigned int) g_ulDestIp);

        do { // Connecting to a TCP server /**TCP Server must first be
        active to be connected!***/
            iStatus = sl_Connect(g_iTcpSockHandle,(SlSockAddr_t *)
        &g_slSocketParam, g_iSockBlockSize);
            if (iStatus < 0) {
                if (iStatus == SL_EALREADY)

```

```

        MAP_UtilsDelay(10000); //wait a bit for the server to respond
    } else {
        UART_PRINT("WARNING: TCP connection error.\n\r");
        sl_Close(g_iTcpSockHandle);
        ASSERT_ON_ERROR(CONNECT_ERROR);
    }
} else {
    UART_PRINT("Successfully connected to the TCP Server.\n\r");
    break;
}
} while (iStatus < 0);

} else { /* CC3200 as TCP SERVER Setup */
    // putting the socket in listening mode for the incoming TCP client
    connection
    iStatus = sl_Listen(g_iTcpSockHandle, 0);
    if (iStatus < 0) {
        UART_PRINT("WARNING: Setting socket in listening mode failed.\n\r");
        sl_Close(g_iTcpSockHandle);
        ASSERT_ON_ERROR(LISTEN_ERROR);
    }
    UART_PRINT("Listening for incoming connections...\n\r");

    GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
    // Below function accepts an incoming connection, if there is any, and
    creates a new file
    // descriptor to serve the now connected socket. Otherwise returns
    SL_EAGAIN, or error
    while (g_iNewTcpSockID < 0) {
        g_iNewTcpSockID = sl_Accept(g_iTcpSockHandle, (struct SlSockAddr_t
*) &g_slSocketParam,
            (SlSocklen_t*) &g_iSockBlockSize);
        if (g_iNewTcpSockID < 0) {
            if (g_iNewTcpSockID == SL_EAGAIN) // try again, but wait a bit
for the client
                MAP_UtilsDelay(10000);
            else { // error
                UART_PRINT("Unable to accept a connection from TCP
client.\n\r");
                sl_Close(g_iNewTcpSockID);
                sl_Close(g_iTcpSockHandle);
                ASSERT_ON_ERROR(ACCEPT_ERROR);
            }
        } else{
            UART_PRINT("Successfully connected with a TCP client.\n\r");
            GPIO_IF_LedOff(MCU_ORANGE_LED_GPIO);
        }
    }
} //end TCP Client or Server check

return SUCCESS;
}


```

```

*****
*brief      Configures UART1-Rx to be used in conjunction with uDMA.
*
*          Sets up UART1-Rx uDMA channel. Also enables UART1.
*
*          The UART is configured so that uDMA-RX channel will receive any
*          incoming data into a pair of buffers in ping-pong mode.

*param     None
*return    None

*note      Whenever a DMA transfer is complete uDMA controller will cause an
interrupt on

```

```

*           the uART and Uart1IntHandler ISRwill be responsible for the
continuation of
*           the DMA transfers from then onwards.
*****
```

```

int Uart1ConfigDmaTransfer(void)
{
    long lRetVal      = -1;

    // Registers the interrupt handler for serving a UART1 interrupt
#ifndef SL_PLATFORM_MULTI_THREADED || defined(TIRTOS) || defined(FREE_RTOS)
    lRetVal = osi_InterruptRegister(INT_UARTA1,(P_OSI_INTR_ENTRY)
Uart1IntHandler, \
        INT_PRIORITY_LVL_2);
    if ( lRetVal < 0 ){
        UART_PRINT("CRITICAL ERROR: UART1 Interrupt register failed.\n\r");
        ASSERT_ON_ERROR(INTERRUPT_ERROR);
    }
#else
    MAP_IntPrioritySet(INT_UARTA1,INT_PRIORITY_LVL_2);
    MAP_UARTIntRegister(UARTA1_BASE,Uart1IntHandler);
#endif

// UART1 Init
MAP_UARTConfigSetExpClk(UARTA1_BASE,
MAP_PRCMPeripheralClockGet(PRCM_UARTA1),
                UARTA1_BAUD_RATE, (UART_CONFIG_WLEN_8 |
UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
MAP_UARTFlowControlSet(UARTA1_BASE, UART_FLOWCONTROL_NONE);

// Assigns a uDMA-controller peripheral mapping for the selected DMA channel
MAP_uDMAChannelAssign(UDMA_CH10_UARTA1_RX);
MAP_uDMAChannelAssign(UDMA_CH11_UARTA1_TX);
// Selects the uDMA channel for UARTA1-Rx transfers
UDMAChannelSelect(UDMA_CH10_UARTA1_RX, NULL);
UDMAChannelSelect(UDMA_CH11_UARTA1_TX, NULL);

// Set RX trigger threshold to 4. This will be used by the uDMA controller
to
// signal when more data should be transferred.
// The uDMA RX channel will trigger the Tx and Rx UART1 interrupt when the
FIFO
// is half full, ie it holds 4 out of 8 bytes (best all round method).
// Each time 4 bytes will be transferred in a burst.
MAP_UARTFIFOLevelSet(UARTA1_BASE, UART_FIFO_RX4_8, UART_FIFO_RX4_8);

// Read the interrupt status of the UART and clear any pending interrupts
// This might be useful after warm resets
lRetVal = MAP_UARTIntStatus(UARTA1_BASE, 1);
MAP_UARTIntClear(UARTA1_BASE,lRetVal);

// Enable the UART peripheral interrupts. uDMA controller will cause an
// interrupt on the UART interrupt signal when a uDMA transfer is complete.
MAP_UARTIntEnable(UARTA1_BASE,UART_INT_DMARX);

// Setup the transfers
UDMASetupTransfer(UDMA_CH10_UARTA1_RX | UDMA_PRI_SELECT, UDMA_MODE_PINGPONG,
                sizeof(g_cUart1RxBufA), UDMA_SIZE_8, UDMA_ARB_2,
                (void*)(UARTA1_BASE + UART_O_DR), UDMA_SRC_INC_NONE,
                g_cUart1RxBufA, UDMA_DST_INC_8);

UDMASetupTransfer(UDMA_CH10_UARTA1_RX | UDMA_ALT_SELECT, UDMA_MODE_PINGPONG,
                sizeof(g_cUart1RxBufB), UDMA_SIZE_8, UDMA_ARB_2,
                (void*)(UARTA1_BASE + UART_O_DR), UDMA_SRC_INC_NONE,
                g_cUart1RxBufB, UDMA_DST_INC_8);
```

```

UDMASetupTransfer(UDMA_CH11_UARTA1_TX | UDMA_PRI_SELECT, UDMA_MODE_BASIC,
    sizeof(g_cUart1TxBuf), UDMA_SIZE_8, UDMA_ARB_2,
    g_cUart1TxBuf, UDMA_SRC_INC_8, (void*)(UARTA1_BASE + UART_O_DR),
    UDMA_DST_INC_NONE);

// Enable the UART for operation, and enable the uDMA Tx and Rx path
channels
MAP_UARTEnable(UARTA1_BASE);
MAP_UARTDMAEnable(UARTA1_BASE, UART_DMA_RX | UART_DMA_TX);

return SUCCESS;
}

/*********************  

 *brief      Parses the destination IP address given by the user to see whether  

 it's valid.  

 *          If it is, the IP is stored into g_ulDestIp  

 *param     cUserInputIp    char pointer to input string  

 *return    0 : correct IP, -1 : incorrect IP  

*****  

#ifndef TERMINAL
int IpAddressParser(char *cUserInputIp)
{
    volatile int i = 0;
    unsigned long ulUserIpAddress = 0;
    unsigned int uiUserData;
    char * cpToken;

    // Break user input into a series of tokens using dot(".")

delimiter/separator
    cpToken = strtok(cUserInputIp, "."); // Get the first token (MSB)
    uiUserData = (int)strtoul(cpToken, 0, 10);
    while(i < 4) {
        // Checks IP validity
        if( (cpToken != NULL) && (uiUserData < 256) ) {
            ulUserIpAddress |= uiUserData;
            if(i < 3)
                ulUserIpAddress <= 8;
            cpToken = strtok(NULL, ".");
            // Get the other tokens 1 Octet/Byte at a time
            uiUserData = (int)strtoul(cpToken, 0, 10); // Convert token to
integer format
            ++i;
        }
        else
            return FAILURE;
    }

    // final IP address doesn't contain dots, it contains 4 Octets, or 8 Hex
digits
    g_ulDestIp = ulUserIpAddress;
    UART_PRINT("Target IP selected: %d.%d.%d.%d\n\r",
SL_IPV4_BYTE(g_ulDestIp, 3),
    SL_IPV4_BYTE(g_ulDestIp, 2), SL_IPV4_BYTE(g_ulDestIp, 1),
    SL_IPV4_BYTE(g_ulDestIp, 0) );

    return SUCCESS;
}
#endif

*****
```

```

*brief      Initializes the DMA controller and
*           the McASP (Multi-Channel Audio Serial Port) module

*param      None
*return     None.
*****
***** */

void UDMAInit()
{
    unsigned int uiLoopCnt;

    // Enable McASP at the PRCM module (not needed in our app)
    MAP_PRCMPeripheralClkEnable(PRCM_UDMA, PRCM_RUN_MODE_CLK);
    MAP_PRCMPeripheralReset(PRCM_UDMA);

    // Register interrupt handlers
#if defined(USE_TIRTOS) || defined(USE_FREERTOS) ||
defined(SL_PLATFORM_MULTI_THREADED)
    // SL_PLATFORM_MULTI_THREADED: if app uses any OS + networking(simplelink)
    osi_InterruptRegister(INT_UDMA, DmaSwIntHandler, INT_PRIORITY_LVL_1);
    osi_InterruptRegister(INT_UDMAERR, DmaErrorIntHandler, INT_PRIORITY_LVL_1);
#else
    MAP_IntPrioritySet(INT_UDMA, INT_PRIORITY_LVL_1);
    MAP_uDMAIntRegister(UDMA_INT_SW, DmaSwIntHandler);

    MAP_IntPrioritySet(INT_UDMAERR, INT_PRIORITY_LVL_1);
    MAP_uDMAIntRegister(UDMA_INT_ERR, DmaErrorIntHandler);
#endif

    // Enable uDMA using master enable
    MAP_uDMAEnable();

    // Initialize (clear) Control Table
    memset(gpCtlTbl, 0, sizeof(tDMAControlTable)*CTL_TBL_SIZE);
    MAP_uDMAControlBaseSet(gpCtlTbl);

    // Reset App Callbacks for all DMA channel sources
    for(uiLoopCnt = 0; uiLoopCnt < MAX_NUM_CH; uiLoopCnt++)
    {
        gfpAppCallbackHndl[uiLoopCnt] = NULL;
    }
}

/*****
*brief      Prints the Application Startup banner on serial terminal

*param      none
*return     none
***** */

static void DisplayBanner(char * AppName)
{
    UART_PRINT("\n\n\n\r");
    UART_PRINT("\t\t ****\n\r");
    UART_PRINT("\t\t\t CC3200 Application \n\r");
    UART_PRINT("\t\t\t %s \n\r", APP_NAME);
    UART_PRINT("\t\t ****\n\r");
    UART_PRINT("\n\n\n\r");
}

```

```

*brief      Function clears console window

*return    none
*****
**** */

void ClearTerm()
{
    Message( "\33[2J\r" );
}

*****



/*brief      Configures the UART module to be used for Terminal communication

*param    none
*return   none
*****
**** */

void InitTerm()
{
#ifndef NOTERM

MAP_UARTConfigSetExpClk(UARTA0_BASE, MAP_PRCMPeripheralClockGet(CONSOLE_PERIPH),
                        UART_BAUD_RATE, (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                        UART_CONFIG_PAR_NONE));
    MAP_UARTFlowControlSet(UARTA0_BASE, UART_FLOWCONTROL_NONE);
#endif
    UARTEnable(UARTA0_BASE);
}

*****



/*brief      Pin assignment and first configuration, setting the clock for
peripherals used.

*param    None
*return   None
*****
**** */

void PinMuxConfig(void)
{
    // Enable Peripheral Clocks
    MAP_PRCMPeripheralClkEnable(PRCM_UARTA0, PRCM_RUN_MODE_CLK);
    MAP_PRCMPeripheralClkEnable(PRCM_UARTA1, PRCM_RUN_MODE_CLK);
    MAP_PRCMPeripheralClkEnable(PRCM_GPIOA1, PRCM_RUN_MODE_CLK);
    MAP_PRCMPeripheralClkEnable(PRCM_GPIOA2, PRCM_RUN_MODE_CLK);

    // Reset peripheral modules to default state first
    MAP_PRCMPeripheralReset(PRCM_UARTA0);
    MAP_PRCMPeripheralReset(PRCM_UARTA1);
    MAP_PRCMPeripheralReset(PRCM_GPIOA1);
    MAP_PRCMPeripheralReset(PRCM_GPIOA2);

    // Configure PIN_55 for UART0_TX
    MAP_PinTypeUART(PIN_55, PIN_MODE_3);

    // Configure PIN_57 for UART0_RX
    MAP_PinTypeUART(PIN_57, PIN_MODE_3);

    // Configure PIN_7 for UART1_TX
    MAP_PinTypeUART(PIN_07, PIN_MODE_5);

    // Configure PIN_8 for UART1_RX
    MAP_PinTypeUART(PIN_08, PIN_MODE_5);
}

```

```

    // Configure PIN_64/Red Led for GPIO Output
MAP_PinTypeGPIO(PIN_64, PIN_MODE_0, FALSE);
MAP_GPIODirModeSet(GPIOA1_BASE, 0x2, GPIO_DIR_MODE_OUT);

    // Configure PIN_02/Green Led for GPIO Output
MAP_PinTypeGPIO(PIN_02, PIN_MODE_0, FALSE);
MAP_GPIODirModeSet(GPIOA1_BASE, 0x8, GPIO_DIR_MODE_OUT);

    // Configure PIN_01/Orange Led for GPIO Output
MAP_PinTypeGPIO(PIN_01, PIN_MODE_0, FALSE);
MAP_GPIODirModeSet(GPIOA1_BASE, 0x4, GPIO_DIR_MODE_OUT);

    // Configure PIN_04/SW3-Button - for GPIO Input
MAP_PinTypeGPIO(PIN_04, PIN_MODE_0, FALSE);
MAP_GPIODirModeSet(GPIOA1_BASE, 0x20, GPIO_DIR_MODE_IN);

    // Configure PIN_15/SW2-Button - for GPIO Input
MAP_PinTypeGPIO(PIN_15, PIN_MODE_0, false);
MAP_GPIODirModeSet(GPIOA2_BASE, 0x40, GPIO_DIR_MODE_IN);
}

```

```

/*********************************************
*brief  Board Initialization & Configuration
*param  None
*return None
*****************************************/

```

```

static void BoardInit(void)
{
// In case of TI-RTOS vector table is initialize by OS itself
#ifndef USE_TIRRTOS
    // Set vector table base
#if defined(ccs) || defined(gcc)
    MAP_IntVTableBaseSet((unsigned long)&g_pfnVectors[0]);
#endif
#if defined(ewarm)
    MAP_IntVTableBaseSet((unsigned long)&__vector_table);
#endif
#endif

    // Enables Processor and the master interrupt knob
    MAP_IntMasterEnable();
    MAP_IntEnable(Fault_Systick);

    // This should be the first call in main() for proper initialization of the
device
    PRCMCC3200MCUInit();
}

```

```

/*********************************************
*                               SETUP FUNCTIONS - End
*****************************************/

```

```

/*********************************************
*                               RTOS TASKS - Start
*****************************************/

```

```

/*
 *brief This thread transmits the buffers with the data it received from
 *       the external device via the UART1 interface to the remote host. It
also
*       receives the UDP packets from the connected station and flushes them
to the
*       external device via the uDMA Tx Channel.
*
*       The Task runs after every UART1-ISR completion. They are synchronized
using a
*       semaphore variable.

*param None
*return None

*note For UDP protocol initial handshaking is not required (connection-less
socket)
***** */

void TransmitReceiveUdpPackets(void * pvParameters)
{
    int iRetVal = -1;

    while(1){
        if ( !osi_SyncObjWait( &g_sendSignal , OSI_WAIT_FOREVER ) ){ // Waiting
on the semaphore
            /* Transmitting */
            if ( g_bBufTurn ){// The Rx A buffer of the primary control
structure
                // will send data
                iRetVal = sl_SendTo(g_iUdpSockHandle, g_cUart1RxBufA,
UDMA_RXCH_BUF_SIZE, 0,
                                (SlSockAddr_t *)&g_slSocketParam,
g_iSockBlockSize);
            }else if ( !g_bBufTurn ){// The Rx B buffer of the alternate control
                // structure will send data
                iRetVal = sl_SendTo(g_iUdpSockHandle, g_cUart1RxBufB,
UDMA_RXCH_BUF_SIZE, 0,
                                (SlSockAddr_t *)&g_slSocketParam,
g_iSockBlockSize);
            }
            if (iRetVal > 0)
                ++g_ultTxCount; // used to count the No of transmissions

            /* Receiving */
            iRetVal = sl_RecvFrom(g_iUdpSockHandle, g_cUart1TxBuf,
UDMA_TXCH_BUF_SIZE, 0,
                                ( SlSockAddr_t *)&g_slSocketParam,
(SlSocklen_t *)&g_iSockBlockSize );
            if (iRetVal > 0){
                ++g_ulRxCount; // Counts the No of receptions

                // Leads the received data to the external device.
                UDMASetupTransfer(UDMA_CH11_UARTA1_TX| UDMA_PRI_SELECT,
UDMA_MODE_BASIC,
                                sizeof(g_cUart1TxBuf), UDMA_SIZE_8, UDMA_ARB_2,
g_cUart1TxBuf, UDMA_SRC_INC_8, (void *)(UARTA1_BASE +
UART_O_DR),
                                UDMA_DST_INC_NONE);
                ++g_ultBufFillCount;
            }
        }// end semaphore case
        osi_Sleep(1);
    }
}

```

```

/*
 *brief      Task is responsible for the actual transfers between the two network
hosts,
 *
receives
 *
from the connected host. The received data will also be sent to the
device
 *
via the DMA UART1-Tx channel.

*param    None
*return   None
***** */

void SendRecvTcpStream(void * pvParameters)
{
    int iRetVal = -1;

    while(1){
        if ( !osi_SyncObjWait( &g_sendSignal , OSI_WAIT_FOREVER) ){ // Waiting
on the semaphore
            if (g_iDisconnected == false){ // This should occur in normal
uninterrupted operation
                /* Sending */
                if (g_bTcpClientOrServer) { /* TCP Client Sends */
                    if (g_bBufTurn) // The Rx A buffer will send data
                        iRetVal = sl_Send(g_iTcpSockHandle, g_cUart1RxBufA,
UDMA_RXCH_BUF_SIZE, 0);
                    else if (!g_bBufTurn) // The Rx B buffer will send data
                        iRetVal = sl_Send(g_iTcpSockHandle, g_cUart1RxBufB,
UDMA_RXCH_BUF_SIZE, 0);
                    if (iRetVal > 0)
                        ++g_ulTxCount; // Counts the No of succesful
transmissions

                } else { /* TCP Server Sends */
                    if (g_bBufTurn) // The Rx A buffer will send data
                        iRetVal = sl_Send(g_iNewTcpSockID, g_cUart1RxBufA,
UDMA_RXCH_BUF_SIZE, 0);
                    else if (!g_bBufTurn) // The Rx B buffer will send data
                        iRetVal = sl_Send(g_iNewTcpSockID, g_cUart1RxBufB,
UDMA_RXCH_BUF_SIZE, 0);
                    if (iRetVal > 0)
                        ++g_ulTxCount; // Counts the No of succesful
transmissions
                }

                /* Receiving */
                if (g_bTcpClientOrServer) { /* TCP Client Receives */
                    iRetVal = sl_Recv(g_iTcpSockHandle, g_cUart1TxBuf,
UDMA_TXCH_BUF_SIZE, 0);
                } else { /* TCP Server Receives */
                    iRetVal = sl_Recv(g_iNewTcpSockID, g_cUart1TxBuf,
UDMA_TXCH_BUF_SIZE, 0);
                }
                if (iRetVal > 0) {
                    ++g_ulRxCount; //increment a counter to signal that data has
been received

                    // Leads the received data to the external device.
                    UDMASetupTransfer(UDMA_CH11_UARTA1_TX | UDMA_PRI_SELECT,
UDMA_MODE_BASIC,
                                         sizeof(g_cUart1TxBuf), UDMA_SIZE_8, UDMA_ARB_2,
                                         g_cUart1TxBuf, UDMA_SRC_INC_8, (void
*) (UARTA1_BASE + UART_O_DR),
                                         UDMA_DST_INC_NONE);
                    ++g.ulTxBuffFillCount;
                }
            }
        }
    }
}

```

```

        }

    } else { /* In case of a disconnection (for more than ~5s) the
socket is closed
        automatically by the kernel. We have to recreate the socket and
reconnect */
        unsigned int      uiTcpPortNum = TCP_PORT;

        // A small delay to signal reconnection-ready /***TIMING is
everything!***/
        UART_PRINT("Connection imminent..\n\r");
        GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
        while (!IS_IP_LEASED(g_ulStatus)) {
            // waiting on the client to reconnect
        }
        GPIO_IF_LedOff(MCU_ORANGE_LED_GPIO);

        // recreating the socket fd
        g_iTcpSockHandle = sl_Socket(SL_AF_INET, SL SOCK_STREAM,
IPPROTO_TCP);
#ifdef NON_BLOCKING_SOCKET
        SlSockNonblocking_t enableOptionNB;
        enableOptionNB.NonblockingEnabled = 1;
        sl_SetSockOpt(g_iTcpSockHandle, SL_SOL_SOCKET,
SL_SO_NONBLOCKING, (_u8 *) &enableOptionNB, sizeof(enableOptionNB) );
#endif
        if (g_bTcpClientOrServer) { /* TCP Client Reconnection */
            // filling the TCP server socket address
            g_slSocketParam.sin_family = SL_AF_INET;
            g_slSocketParam.sin_port = sl_Htons((unsigned short)
uiTcpPortNum); // reorder bytes from CPU order to Network byte order
            g_slSocketParam.sin_addr.s_addr = sl_Htonl((unsigned int)
g_ulDestIp);

            do {
                iRetVal = sl_Connect(g_iTcpSockHandle, (SlSockAddr_t *)
&g_slSocketParam, g_iSockBlockSize);
                if (iRetVal < 0)
                    MAP_UtilsDelay(10000);
                else {
                    g_iDisconnected = FALSE;
                    break;
                }
            } while (iRetVal < 0); //end sl_Connect attempt
        }//end reconnection case
    }//end disconnected event driven case handler
}//end semaphore case
osi_Sleep(1);
}

}

/*
 *brief    Lights the green led to indicate that the device is working
properly.
 *      It's transmitting and receiving data as planned.

 *param    None
 *return   None
*/
void GreenLight(void * pvParameters)
{
    for(;;) {
        if ( (g_ulTxCount > GOOD_COUNT) && (g_ulRxCount > GOOD_COUNT) &&
(g_ulRxABuffFillCount > GOOD_COUNT)

```

```

        && (g_ulRxBBufFillCount > GOOD_COUNT) && (g_ulTxBBufFillCount >
GOOD_COUNT) ) {
    GPIO_IF_LedOn(MCU_GREEN_LED_GPIO);
    Task_setPri(GreenLightHandle, INACTIVE_STATE_PRIORITY); // we are
done with this task
}
else // we don't want this task to cause overhead so we check it every
10 secs
    osi_Sleep(10000);
}
}

```

```

/*
*                               RTOS TASKS - End
*/

```

```

/*
*                               INTERRUPTS - Start
*/

```

```

/*
*brief      The interrupt handler for UART1 (ISR/HWI). This interrupt will
occur
*
*      when a DMA transfer is complete using the UART1 uDMA channel.
*      This interrupt handler will switch between Rx ping-pong buffers A
and B
*
*      and the Tx "basic-mode-DMA" buffer.
*
*      It will also restart a uDMA transfer if the prior one is complete.
*      This will keep the UART running continuously sending data back and
forth.

```

```

*param     None
*return    None

*Note      We should clear the individual interrupt bit-flags early to keep
the interrupt handler from executing immediately after exiting.

*Warning   Generally the Interrupt Handler Routine should complete as fast as
possible (no delays in it etc.), or else unexpected events may
occur. For
*
*      this purpose it should implement only non-blocking calls
*/

```

```

void Uart1IntHandler(void)
{
    long lMode    = -1;
    long lStatus = -1;

    // Read the interrupt status. The int source flag is returned.
    lStatus = MAP_UARTIntStatus(UARTA1_BASE, 1);
    // Clear the interrupt assertion flag. Ready for the next interrupt after
    exiting this one
    MAP_UARTIntClear(UARTA1_BASE, lStatus);

    // Check the DMA control table to see if the ping-pong "A" transfer is
    complete.
    lMode = MAP_uDMAChannelModeGet(UDMA_CH10_UARTA1_RX | UDMA_PRI_SELECT);
}

```

```

    // If the primary DMA structure indicates stop, then the "A" Rx buf-transfer
is done
    if(lMode == UDMA_MODE_STOP)      {
        // Set up the next transfer for the "A" buffer
        UDMASetupTransfer(UDMA_CH10_UARTA1_RX | UDMA_PRI_SELECT,
UDMA_MODE_PINGPONG,
                           sizeof(g_cUart1RxBufA), UDMA_SIZE_8, UDMA_ARB_2,
                           (void *)(UARTA1_BASE + UART_O_DR), UDMA_SRC_INC_NONE,
                           g_cUart1RxBufA, UDMA_DST_INC_8);
        ++g_ulRxABufFillCount;
        g_bBufTurn = TRUE;
    }

    // Check the DMA control table to see if the ping-pong "B" transfer is
complete.
    lMode = MAP_uDMAChannelModeGet(UDMA_CH10_UARTA1_RX | UDMA_ALT_SELECT);

    // If the alternate DMA structure indicates stop, then the "B" Rx buf-
transfer is done
    if(lMode == UDMA_MODE_STOP)      {
        // Set up the next transfer for the "B" buffer
        UDMASetupTransfer(UDMA_CH10_UARTA1_RX | UDMA_ALT_SELECT,
UDMA_MODE_PINGPONG,
                           sizeof(g_cUart1RxBufB), UDMA_SIZE_8,
                           UDMA_ARB_2, (void *)(UARTA1_BASE + UART_O_DR),
                           UDMA_SRC_INC_NONE, g_cUart1RxBufB, UDMA_DST_INC_8);
        ++g_ulRxBBuffFillCount;
        g_bBufTurn = FALSE;
    }

    osi_SyncObjSignalFromISR( &g_sendSignal );
}

//*****************************************************************************
*brief      GPIO Interrupt Handler for SW3 button press
*           If user presses this button within the timelimit of Timer's A0
timeout value
*           UDP protocol will be selected

*param      None
*return     None
*****
#ifndef TERMINAL
void Sw3InterruptHandler(void)
{
    unsigned long ulIntStatus;

    // Clear SW3 Interrupt flag
    ulIntStatus = MAP_GPIOIntStatus(GPIOA1_BASE, 1);
    MAP_GPIOIntClear(GPIOA1_BASE, ulIntStatus);

    g_bUdpOrTcp      = TRUE; //UDP Select
    g_bButtonPress   = TRUE;
}
*****
```

```

//*****************************************************************************
*brief      GPIO Interrupt Handler for SW2 button press
*           If user presses this button within the timelimit of Timer's A0
timeout value
*           TCP protocol will be selected for communication

*param      None
*return     None
```

```

*****
void Sw2InterruptHandler(void)
{
    unsigned long ulIntStatus;

    ++g_uiNoButtonPress;

    // Clear SW2 Interrupt flag
    ulIntStatus = MAP_GPIOIntStatus(GPIOA2_BASE, 1);
    MAP_GPIOIntClear(GPIOA2_BASE, ulIntStatus);

    g_bUdpOrTcp      = FALSE; //TCP Select
}

/*****
 *brief      The interrupt handler for the timer A0. Will be triggered 30 seconds
after
 *
 *           startup to quit program execution if user hasn't pressed a button.

 *param      None
 *return     none
*****
```

```

void TimerA0InterruptHandler(void)
{
    // Whenever Orange LED turns on, either the system is blocking or a fatal
exception
    // error has occurred which called system abort. When the latter happens the
device is
    // inoperable and has to be restarted to be used
    GPIO_IF_LedOff(MCU_ALL_LED_IND);
    GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
    BIOS_exit(100); // Quit Application
}
```

```

*****
```

```

 *brief      The interrupt handler for the timer A1. It will be used in case
CC3200 will
 *
 *           operate as a TCP server. It is registered after SW2 1st button press
and
 *
 *           activated with SW2 button 2nd button press, if it occurs within less
than 3
 *
 *           seconds after the first SW2 button press, unless 30 seconds from
startup pass
 *
 *           first. In the latter case Timer A0 is triggered first which
deactivates all
 *
 *           button and timer interrupts.

 *param      None
 *return     none
*****
```

```

void TimerA1InterruptHandler(void)
{
    unsigned long ulIntStatus;

    ulIntStatus = MAP_TimerIntStatus(TIMERA1_BASE, 1);
    MAP_TimerIntClear(TIMERA1_BASE, ulIntStatus);

    g_bTimerA1Flag = TRUE;
}
#endif
```

```

/*
*                                     INTERRUPTS - End
*/
*****



/*****
*                                     MAIN FUNCTION - Start
*****/



int main()
{
    long          lRetVal           = -1;
#ifndef TERMINAL
    short         sInvalidInputTries = 5;
    static _Bool   bInputCheck      = FALSE;
    static _Bool   bSsidCheck       = FALSE;
    static _Bool   bPswdCheck       = FALSE;
    static _Bool   bPswdTypeCheck   = FALSE;
    static _Bool   bIpCheck        = FALSE;
    char          cCmdBuf[64];
    short         sInput           = -1;
    short         sSecType         = -1;
#endif

    // Board Initialization
    BoardInit();

    // Configure the pinmux settings for the peripherals exercised
    PinMuxConfig();

    // Initializing Console UART0
    InitTerm();
    ClearTerm();

    // Displays Application Banner
    DisplayBanner(APP_NAME);

    // uDMA controller Initialization
    UDMAInit();

    // Setup the Leds
    GPIO_IF_LedConfigure(LED1|LED2|LED3);
    GPIO_IF_LedOff(MCU_ALL_LED_IND);

    /*
     * All tasks and interrupts must be registered and enabled prior to starting
     RTOS.
     * Also, no interrupt or task can be executed before starting the RTOS
     scheduler.
     */
#ifndef TERMINAL
    // Setup and Register Timer A0 Interrupt
    lRetVal =
ConfigureTimeoutTimer(TIMERA0_BASE, PRCM_TIMERA0, TIMER_CFG_PERIODIC, 0, \
                      INT_PRIORITY_LVL_1, TimerA0InterruptHandler, 30000, TIMER_A);
    if (lRetVal < 0){
        UART_PRINT("CRITICAL ERROR: Failed to Initialize Timer-A0.\n\r");
        exit(EXIT_TASK_FAIL);
    }

```

```

// Setup buttons for the user to choose UDP (SW3), or TCP(SW2)
lRetVal = ButtonSetup (Sw3InterruptHandler, Sw2InterruptHandler);
if (lRetVal < 0) {
    UART_PRINT("WARNING: Failed to configure the Buttons.\n\r");
    exit(EXIT_TASK_FAIL);
}

//Enable the SimpleLink Host driver to handle async network events
lRetVal = VStartSimpleLinkSpawnTask(SL_SPAWN_TASK_PRIORITY);
if (lRetVal < 0) {
    ERR_PRINT(lRetVal);
    UART_PRINT("CRITICAL ERROR: Failed to create SimpleLink Task.\n\r");
    exit(EXIT_TASK_FAIL);
}

// Create the InitialSetupAP task
lRetVal = osi_TaskCreate(InitialSetupAP,(const signed char*) "Configur-
ing",
                        TASK_STACK_SIZE, NULL, 7, NULL);
if (lRetVal < 0) {
    ERR_PRINT(lRetVal);
    UART_PRINT("WARNING: Failed to create task for WLAN connection.\n\r");
    exit(EXIT_TASK_FAIL);
}

// Creates task for UDP communication with the connected host.
lRetVal = osi_TaskCreate(TransmitReceiveUdpPackets,(const signed char*) "UDP
Packet Transmission",
                        TASK_STACK_SIZE, NULL, COM_PROTOCOL_TASK_PRIORITY,&UdpHandle);
if (lRetVal < 0) {
    ERR_PRINT(lRetVal);
    UART_PRINT("WARNING: Failed to create task for UDP communication.\n\r");
    exit(EXIT_TASK_FAIL);
}

// Creates task for TCP communication with the remote host
lRetVal = osi_TaskCreate(SendRecvTcpStream,(const signed char*) "TCP Stream
Export",
                        TASK_STACK_SIZE, NULL, COM_PROTOCOL_TASK_PRIORITY,&TcpHandle);
if (lRetVal < 0) {
    ERR_PRINT(lRetVal);
    UART_PRINT("WARNING: Failed to create task for TCP communication.\n\r");
    exit(EXIT_TASK_FAIL);
}

// Task to light the green led as a sign that the device is working as
expected
lRetVal = osi_TaskCreate(GreenLight,(const signed char*) "GREEN LIGHT",
                        TASK_STACK_SIZE, NULL, GREEN_LIGHT_PRIORITY,&GreenLightHandle);
if (lRetVal < 0) {
    ERR_PRINT(lRetVal);
    UART_PRINT("WARNING: Failed to create task for Green-LED.\n\r");
    exit(EXIT_TASK_FAIL);
}

// Once the task scheduler is called control doesn't return in the function
that
// called it ever again. Only tasks and ints will be executed from now on
and all
// those tasks and interrupts must have already been registered and enabled
osi_start();

#else      /***** User Input Preferences *****/
do{ // Selecting either AP-Mode or Wlan-Station mode
    UART_PRINT("Options:\n\rFor operation as an AP press 1\r\nTo connect to
an AP \

```

```

press 0\r\nTo exit press 8\r\n");
GetCmd(cCmdBuf, sizeof(cCmdBuf));
sInput = (short)strtoul(cCmdBuf, 0, 10); // Convert to int No

switch (sInput){
case 1:
    UART_PRINT("AP-Mode Selected.\n\n\r");
    bInputCheck = FALSE;
    break;

case 0:
    UART_PRINT("Wlan Station-Mode Selected.\n\n\r");
    g_bApOrWlan = FALSE;
    bInputCheck = FALSE;
    break;

case 8:
    UART_PRINT("Exiting Application... \n\n\r");
    GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
    exit(EXIT_SUCCESS);
    break;

default:
    if (!sInvalidInputTries) {
        UART_PRINT("That's wrong.\n\n\r");
        exit(EXIT_GENERIC_FAIL);
    } else {
        --sInvalidInputTries;
        UART_PRINT("Wrong Input. Try Again.\n\rYou have ");
        if (sInvalidInputTries == 1)
            UART_PRINT("1 more try.\n\r");
        else
            UART_PRINT("%i tries remaining.\n\r", sInvalidInputTries);
    }
    bInputCheck = TRUE;
    break;
}

}while ( bInputCheck );

// In case Wlan Station mode has been selected prompt operator to enter AP
parameters
if (!g_bApOrWlan){
    sInput = 1; // initing..
    sInvalidInputTries = 5;
    do {
        if ( sInput != 1 || sInput != 2 || sInput != 3 || sInput != 4 )

            UART_PRINT("Options:\n\rTo enter the Access Point SSID press 1\r\nTo
enter "
press 3\n\r"
            "the AP password press 2\r\nTo enter the AP security type
            "To enter the IP of the target host press 4\n\rTo exit press
8\r\n");
        GetCmd(cCmdBuf, sizeof(cCmdBuf));
        sInput = (short) strtoul(cCmdBuf, 0, 10);

        switch (sInput) {
        case 1:
            UART_PRINT("Enter AP's SSID\n\r");
            GetCmd(cCmdBuf, sizeof(cCmdBuf));
            strcpy( g_ucNetworkSSID, cCmdBuf );
            bSsidCheck = TRUE; // SSID input complete
            break;
        }
    }
}

```

```

case 2:
    UART_PRINT("Enter the AP's password.\n\rFor no password press
carriage return(enter)\n\r");
    GetCmd(cCmdBuf, sizeof(cCmdBuf));
    if ( cCmdBuf == "CR" || cCmdBuf == "LF" || cCmdBuf[0] == null ||
cCmdBuf == "@" || cCmdBuf == "NULL"){ // cross platform thinking
        UART_PRINT("Acknowledged\n\r");
        strcpy(g_ucSecurityKey, NULL);
    }
    else
        strcpy(g_ucSecurityKey, cCmdBuf);
    bPswdCheck = TRUE; // Password input set
break;

case 3:
    UART_PRINT("Enter the security type used by the AP.\n\r");
    UART_PRINT("For no Security Press 0.\n\rFor WEP press 1\n\rFor
WPA/WPA2 press 2\n\r \
    "For WPA Enterprise press 3\n\rFor WPS-PBC press 4\n\rFor WPS-PIN press
5\n\r");
    GetCmd(cCmdBuf, sizeof(cCmdBuf));
    sSecType = (short) strtoul(cCmdBuf, 0, 10); // convert char
pointer to int

    // Check to determine the kind of security type
    switch ( sSecType ){
        case 0:
            g_ucSecurityType = SL_SEC_TYPE_OPEN;
            break;
        case 1:
            g_ucSecurityType = SL_SEC_TYPE_WEP;
            break;
        case 2:
            g_ucSecurityType = SL_SEC_TYPE_WPA_WPA2;
            break;
        case 3:
            g_ucSecurityType = SL_SEC_TYPE_WPA_ENT;
            break;
        case 4:
            g_ucSecurityType = SL_SEC_TYPE_WPS_PBC;
            break;
        case 5:
            g_ucSecurityType = SL_SEC_TYPE_WPS_PIN;
            break;
        default:
            UART_PRINT("This security type is not supported\n\r");
            exit(EXIT_SEC_FAIL);
            break;
    }
    bPswdTypeCheck = TRUE; // Password type input complete
break;

case 4:
    { // When there are jump(s) to other an context/function inside a
case statement we must add curly brackets to confine the scope of case
statement. Otherwise unexpected event may occur
    do {
        UART_PRINT("Enter target IP\n\r");
        lRetVal = GetCmd(cCmdBuf, sizeof(cCmdBuf));

        if (!lRetVal) { // No input given
            UART_PRINT("\n\rEnter Valid Input.\n\r");
            bIpCheck = FALSE;
            --sInvalidInputTries;
        } else {
            if (IpAddressParser(cCmdBuf) < 0) {
                --sInvalidInputTries;
            }
        }
    }
}

```

```

        switch ( sInvalidInputTries ){
            case 1:
                UART_PRINT( "Wrong Input.\n\rYou have 1 more
try.\n\r" );
                break;
            case 0:
                UART_PRINT( "That is wrong. Abort\n\r" );
                exit(EXIT_IP_FAIL);
                break;
            default:
                UART_PRINT( "Wrong Input.\n\rYou have %i tries
remaining.\n\r" , sInvalidInputTries );
                break;
        }
    } else
        bIpCheck = TRUE;
}
} while(!bIpCheck); // repeat until a valid IP is given or may
mistakes have been made
break;
}

case 8:
UART_PRINT( "Exiting Application...\n\r" );
GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
exit(EXIT_SUCCESS);
break;

default:
if (!sInvalidInputTries) {
    UART_PRINT( "That's wrong. Aborting..\n\r" );
    exit(EXIT_GENERIC_FAIL);
} else{
    --sInvalidInputTries;
    UART_PRINT( "Wrong Input. Try Again.\n\rYou have " );
    if (sInvalidInputTries == 1)
        UART_PRINT( "1 more try.\n\r" );
    else
        UART_PRINT( "%i tries remaining.\n\r" ,
sInvalidInputTries );
}
break;
}

// end input switch

}while ( (!bSsidCheck) || (!bPswdCheck) || (!bPswdTypeCheck) ||
(!bIpCheck) ); // When ALL are TRUE, exit loop
}// endif Wlan case

if ( sInput != 8 ){
sInvalidInputTries = 5; // Initializing
do{ // Prompts the User to select either TCP or UDP for communication
    UART_PRINT("Options:\n\rFor UDP press 1\r\nFor TCP press 0\r\nTo
exit press 8\r\n" );
    GetCmd(cCmdBuf, sizeof(cCmdBuf));
    sInput = (short)strtoul(cCmdBuf,0,10);

    if ( (sInput == 1) || (sInput == 0) ){
        // Starts the SimpleLink Host to create tasks and add them to a
queue
        // The task handles the communication with simple link chipset,
        // for the host controller driver communication to handle sync &
lRetVal = VStartSimpleLinkSpawnTask(SL_SPAWN_TASK_PRIORITY);
}
}
}

```

```

        if(lRetVal < 0) {
            ERR_PRINT(lRetVal);
            UART_PRINT("CRITICAL ERROR: Failed to create SimpleLink
Task.\n\r");
            exit(EXIT_TASK_FAIL);
        }

        if (g_bApOrWlan){ // Create the InitialSetupAP task
            lRetVal = osi_TaskCreate( InitialSetupAP, (const signed
char*)"Configur-at-ing AP",
                                TASK_STACK_SIZE, NULL, 7, NULL );
            if(lRetVal < 0) {
                ERR_PRINT(lRetVal);
                UART_PRINT("WARNING: Failed to create task for AP
implementation.\n\r");
                exit(EXIT_TASK_FAIL);
            }
            bInputCheck = FALSE;
        }else{ // Create the InitialSetupWlan task, to access a Wlan
station
            lRetVal = osi_TaskCreate( InitialSetupWlan, (const signed
char*)"Configur-at-ing Wlan",
                                TASK_STACK_SIZE, NULL, 7, NULL );
            if(lRetVal < 0) {
                ERR_PRINT(lRetVal);
                UART_PRINT("WARNING: Failed to create task for WLAN
connection.\n\r");
                exit(EXIT_TASK_FAIL);
            }
            bInputCheck = FALSE;
        }
    }

    switch ( sInput ) {
    case 1:           /*** UDP-Enable ***/
        UART_PRINT("UDP Selected.\n\r");
        bInputCheck = FALSE;
        // Creates task for UDP communication with the connected station
        lRetVal = osi_TaskCreate( TransmitReceiveUdpPackets, (const
signed char*) "UDP Packet Transmission",
                                TASK_STACK_SIZE, NULL, COM_PROTOCOL_TASK_PRIORITY,
&UdpHandle);
        if (lRetVal < 0) {
            ERR_PRINT(lRetVal);
            UART_PRINT("WARNING: Failed to create task for UDP
communication.\n\r");
            exit(EXIT_TASK_FAIL);
        }
        break;

    case 0:           /*** TCP-Enable ***/
        UART_PRINT("TCP Selected.\n\r");
        g_bUdpOrTcp = FALSE;
        bInputCheck = FALSE;
        // Creates task for TCP communication with the remote host
        lRetVal = osi_TaskCreate( SendRecvTcpStream, (const signed
char*) "TCP Stream Export",
                                TASK_STACK_SIZE, NULL, COM_PROTOCOL_TASK_PRIORITY,
&TcpHandle);
        if (lRetVal < 0) {
            ERR_PRINT(lRetVal);
            UART_PRINT("WARNING: Failed to create task TCP
communication.\n\r");
            exit(EXIT_TASK_FAIL);
        }
    }

    do { // prompt for TCP server or TCP client

```

```

        UART_PRINT("For CC3200 as TCP client press 1\n\rFor CC3200
as "
            "TCP server press 0\n\rFor default setting press
3\n\r");
    GetCmd(cCmdBuf, sizeof(cCmdBuf));
    lRetVal = (long)strtoul(cCmdBuf, 0, 10);

    if( lRetVal == 1 ){ /* Device as TCP Client selected */
        UART_PRINT("TCP Client Selected\n\r");
        g_bTcpClientOrServer = TRUE;
        break;
    } else if( lRetVal == 0 ){ /* Device as TCP Server selected */
/*
        UART_PRINT("TCP Server Selected\n\r");
        g_bTcpClientOrServer = FALSE;
        break;
    } else if( lRetVal == 3 ){ // TCP client is default. Option
exists in case user is in doubt
        UART_PRINT("Default is TCP Client.\n\r");
        g_bTcpClientOrServer = TRUE;
        break;
    } else {
        --sInvalidInputTries;
        switch ( sInvalidInputTries ){
        case 1:
            UART_PRINT("Wrong Input.\n\rYou have 1 more
try.\n\r");
            break;
        case 0:
            UART_PRINT("Wrong input! Abort\n\r");
            exit(EXIT_TCP_FAIL);
            break;
        default:
            UART_PRINT("Wrong Input.\n\rYou have %i tries
remaining.\n\r", sInvalidInputTries);
            break;
        }
    }
} while( FOREVER );
break;

case 8:
    UART_PRINT("Exiting Application...\n\r");
    GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
    exit(EXIT_SUCCESS);
break;

default:
    if (!sInvalidInputTries) {
        UART_PRINT("That's just wrong.\n\r");
        exit(EXIT_GENERIC_FAIL);
    } else {
        --sInvalidInputTries;
        UART_PRINT("Wrong Input. Try Again.\n\rYou have ");
        if (sInvalidInputTries == 1)
            UART_PRINT("1 more try.\n\r");
        else
            UART_PRINT("%i tries remaining.\n\r",
sInvalidInputTries);
    }
    bInputCheck = TRUE;
    break;
}

}while( bInputCheck );
//endif (sInput != 8);

```

```

    // Task to light the green led as a mark that SimpleLink program works as
expected
    if ( (sInput == 0) || (sInput == 1) ) {
        lRetVal = osi_TaskCreate( GreenLight, (const signed char*) "GREEN
LIGHT",
                                TASK_STACK_SIZE, NULL, GREEN_LIGHT_PRIORITY, &GreenLightHandle);
        if (lRetVal < 0) {
            ERR_PRINT(lRetVal);
            UART_PRINT("WARNING: Failed to create task for Green-LED.\n\r");
            exit(EXIT_TASK_FAIL);
        }
    }

    // Invokes the task scheduler
    osi_start();
#endif

    return EXIT_SUCCESS;
}

//*****************************************************************************
*          MAIN FUNCTION - End
*****

```

5 Επίλογος – Συμπεράσματα

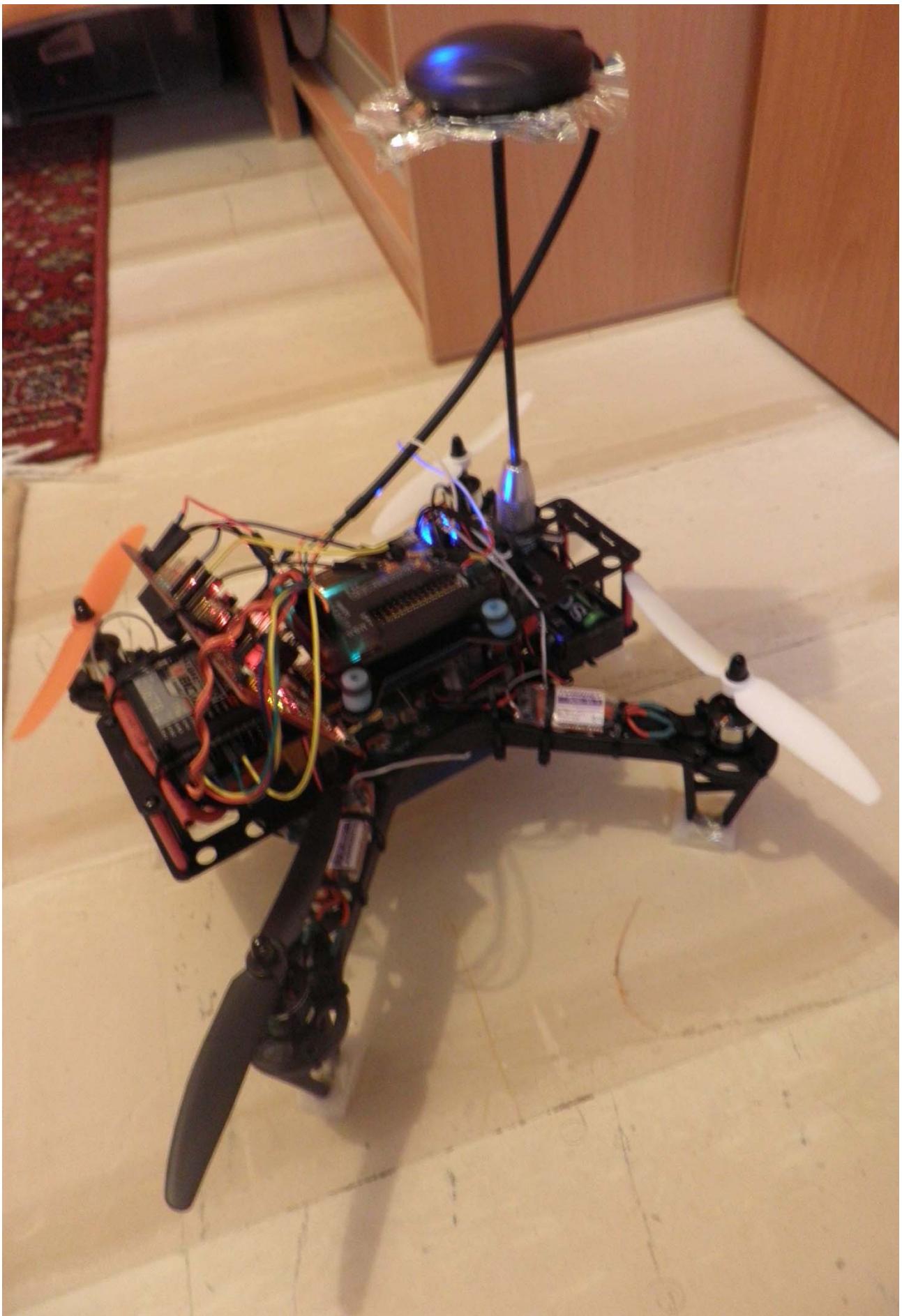
Το μη επανδρωμένο εναέριο όχημα τεσσάρων ελίκων που σχεδιάστηκε, συναρμολογήθηκε και προγραμματίσθηκε φαίνεται πανέτοιμο στην εικόνα 5-1. Οι τελικές πτήσεις μας με το τετρακόπτερο ήταν απολαυστικές και είμαι πολύ ικανοποιημένος και περήφανος που ένα project διάρκειας ολίγων μηνών, έληξε με τεράστια επιτυχία.

Πριν ολοκληρώσουμε θα ήθελα να αναφέρω κάποιες προσθήκες που θα μπορούσαν να γίνουν. Είναι καλό να γίνουν γνωστές σε όποιον επιχειρήσει να κατασκευάσει το δικό του UAVdrone και χρησιμοποιήσει αυτή την εργασία ως οδηγό/αναφορά:

- Θωράκιση με αλουμινόχαρτο μπορεί να βοηθήσει στη καταστολή ηλεκτρομαγνητικών παρεμβολών που προέρχονται από AC ηλεκτρικό ρεύμα, που ρέει από τα ESC στους κινητήρες.
- Εάν τοποθετήσουμε δαχτυλίδια από φερρίτη (ferrite rings) στα καλώδια του κάθε ESC, σε συνδυασμό με τη συστροφή τους, θα μειωθούν σημαντικά οι επαγώμενες ηλ/κές παρεμβολές.
- Λεπτά φύλλα από το υλικό MuMetal (δυσεύρετο, πωλείται στο eBay) μπορούν να θωρακίσουν από κάθε πηγή παραγωγής θορύβου και ηλ/κών παρεμβολών μεταξύ των ηλεκτρονικών κυκλωμάτων.
- Θα μπορούσε να γίνει ένα τελικό στάδιο μέτρησης και μετριασμού των δονήσεων και ταλαντώσεων που προέρχονται από τη συνολική, έτοιμη πλέον κατασκευή, όταν βρίσκεται σε λειτουργία.
- Η τοποθέτηση καλυμμάτων γύρω από τις προπέλες (Propeller Shroud, Ducted Propeller), έχει αποδειχτεί πειραματικά και μαθηματικά ότι αυξάνει σημαντικά την αποδοτικότητα της προπέλας, [67][68] αυξάνοντας το χρόνο πτήσης και παράλληλα ελαττώνει ενεργά το επίπεδο θορύβου που παράγεται από τις προπέλες που σε γενικές γραμμές είναι αρκετά έντονος. Τα καλύμματα για ένα UAV είναι σχετικά ελαφριά (πχ. για ένα 250mm QuadCopter όπως το δικό μας, ζυγίζει ~8g το ένα).
- Αντί για την εγκατάσταση ή υπ' αριθμό ESC στο n-κόπτερο, είναι προτιμότερη η χρήση All-In-One-ESC, από πλευράς βάρους, κόστους, ηλ/κών παρεμβολών, όγκου και κόπου όσον αφορά τη σύνδεση και κόλληση του κάθε ESC στη πλακέτα διανομής ισχύος. Η τελευταία ενσωματώνεται

στη βαθμίδα All-In-One-ESC! Τα All-In-One ESC, δεν είναι ακόμη ιδιαιτέρως δημοφιλή, αλλά τελευταία παρατηρείται η χρησιμότητα τους και σίγουρα στο μέλλον θα αντικαταστήσουν πλήρως τα κοινά ESC των UAV drones.

- Τοποθέτηση αισθητήρα(-ων) υπερύχων για ανίχνευση και αποφυγή αντικειμένων.



Εικόνα 5-1 Το τετρακόπτερο έτοιμο για πτήση

6 Κατάλογος Εικόνων:

Εικόνα 1-1 Δομικό Διάγραμμα των στοιχειωδών υποσυστημάτων ενός UAV drone.....	11
Εικόνα 2-1 Σχέδιο ιπτάμενης μηχανής του Λεονάρντο Ντα Βίντσι[1]	15
Εικόνα 2-2 Αεροτομή (Airfoil) - Πλευρική Άποψη. Απεικονίζονται τα βασικά χαρακτηριστικά της και οι δυνάμεις που εφαρμόζονται[2]	18
Εικόνα 2-3 Απεικόνιση της αρχής του Bernoulli σε υγρό εντός σωλήνα[3].....	19
Εικόνα 2-4 Η Αρχή του Bernoulli σε μια αεροτομή	19
Εικόνα 2-5 Αναπαράσταση της Δύναμης Magnus[4].....	20
Εικόνα 2-6 Κατεύθυνση εκτροπής της κίνησης αντικειμένων στο Βόρειο και Νότιο Ήμισφαίριο της γης, εξαιτίας του φαινομένου Coriolis[5]	21
Εικόνα 2-7 Εκδήλωση φαινομένου γυροσκοπίου σε προπέλα ελικοπτέρου[6].....	22
Εικόνα 2-8 Παρουσιάζονται τα ρεύματα αέρα στο ελικόπτερο σε κανονικές συνθήκες λειτουργίας (πάνω) και σε συνθήκες λειτουργίας αυτοπεριστροφής (κάτω)[7]	23
Εικόνα 2-9 Οι δυνάμεις που εφαρμόζονται σε ένα αεροπλάνο. Εφάμιλλο είναι το διάγραμμα για στροφιόπτερα. Θεωρούμε ότι οι δυνάμεις ασκούνται στο κέντρο βάρους του αεροσκάφους[8]	24
Εικόνα 2-10 Μηχανισμοί ελέγχου Αεροπλάνου[9]	27
Εικόνα 2-11 Παρουσιάζονται τα πηδάλια ελέγχου πτήσης των ελικοπτέρων. Αντίστοιχοι μηχανισμοί υπάρχουν στα αεροπλάνα και λοιπά αεροσκάφη[10]	28
Εικόνα 2-12 Κατεύθυνση ροπής της κύριας προπέλας και κατεύθυνση της πρωθητικής δύναμης της οπίσθιας προπέλας αντιτιθέμενη στην αντίδραση ροπής που προκαλείται[12]	29
Εικόνα 3-1 Μπλόκ Διάγραμμα διαδικασίας περιστροφής στροφέων.....	31
Εικόνα 3-2 Μποκ διάγραμμα ενός τετρακόπτερου που παριστάνει το Σύστημα Ελέγχου Πτήσης και τους κινητήρες.....	31
Εικόνα 3-3 Δυνάμεις προώθησης που παράγονται από μια προπέλα διαμέτρου dp [13].....	34
Εικόνα 3-4 Σχέση αποδοτικότητας προπέλας μικρού, μεσαίου και μεγάλου pitch με το λόγο προπόρευσης(Advance Ratio). Ο λόγος προπόρευσης στα UAV λαμβάνει υψηλές τιμές[14]	35
Εικόνα 3-5 Αναλυτικό Μπλοκ Διάγραμμα Quadcopter + GCS + R/C Tx	40
Εικόνα 3-6 Απλοποιημένο Μπλοκ Διάγραμμα Quadcopter + GCS + R/C Tx	41
Εικόνα 3-7 Μπλοκ Διάγραμμα Λειτουργίας Ελεγκτή Πτήσης.....	42
Εικόνα 3-8 Στην εικόνα φαίνεται η πλατφόρμα ελεγκτή πτήσης APM 2.8	43
Εικόνα 3-9 APM Power Module. Τροφοδοτικό του APM 2.8.....	44
Εικόνα 3-10 Δέκτης Turnigy 9x	46
Εικόνα 3-11 Πομπός Turnigy 9x V2	47
Εικόνα 3-12 Τρόποι ανάθεσης λειτουργιών πτήσης με τον R/C πομπό	49
Εικόνα 3-13 Μπλοκ Διάγραμμα συστήματος ασύρματης τηλεμετρίας.....	50

Εικόνα 3-14 Πρόσοψη του CC3200 αναπτυξιακού με τους τρόπους χρήσης όλων των ακροδεκτών του[29]	54
Εικόνα 3-15 Μηχανικό γυροσκόπιο σε περιστροφή[30]	56
Εικόνα 3-16 Τυπικό Βαρόμετρο[31].....	57
Εικόνα 3-17 Τα βασικότερα χαρακτηριστικά ενός αισθητήρα υπερήχων[32]	58
Εικόνα 3-18 Μονάδα GPS δέκτη +πυξίδα υπερυψωμένη με ειδικό στύλο στο τετρακόπτερο	60
Εικόνα 3-19 Σύνδεση μονάδας GPS Neo-6M με FTDI. Προβάλλονται τα pinouts της κάθε συσκευής[36].....	60
Εικόνα 3-20 Παλμοί διαμορφωμένοι κατά Εύρος (τεχνική PWM)[37]	61
Εικόνα 3-21 Αρχή λειτουργίας ESC. Τρόπος που ένα ESC τροφοδοτεί έναν τριφασικό brushless κινητήρα	62
Εικόνα 3-22 ESC Turnigy Plush-12A-E & κάρτα προγραμματισμού Turnigy BESC Programming Card[39].....	64
Εικόνα 3-23 Σχήμα outrunner κινητήρα χωρίς ψήκτρα[41]	66
Εικόνα 3-24 BLDC κινητήρες LD1510A-02-P Micro και βάση εγκατάστασης[42]	67
Εικόνα 3-25 Pitch προπέλας είναι η απόσταση που διανύει η προπέλα σε μια πλήρης περιστροφή της[45][46]	69
Εικόνα 3-26 Συναρμολόγηση Προπέλας	69
Εικόνα 3-27 Διαμόρφωση Προπελών Quadcopter[47].....	70
Εικόνα 3-28 Η μπαταρία LiPo Turnigy 2700mAh 3S 20C που χρησιμοποιήσαμε[49]	72
Εικόνα 3-29 Πλακέτα Διανομής Ισχύος	75
Εικόνα 4-1 Τα στρώματα ενός υπολογιστικού συστήματος	78
Εικόνα 4-2 Σύστημα Πραγματικού Χρόνου	80
Εικόνα 4-3 Εφαρμογή Διεργασίας Πραγματικού Χρόνου.....	81
Εικόνα 4-4 Αλληλουχία ενεργειών μετά την ενεργοποίηση Υπολογιστικού Συστήματος.....	82
Εικόνα 4-5 Το Kernel συνδέει τις εφαρμογές με το υλικό του H/Y[55]	83
Εικόνα 4-6 Οι πληροφορίες που περιέχει μια Δομή Ελέγχου Διεργασίας (PCB). Το πλήθος τους εξαρτάται από το λειτουργικό σύστημα σε χρήση	84
Εικόνα 4-7 Ιδιότητες Νημάτων σε ένα RTOS	87
Εικόνα 4-8 Μηχανή πεπερασμένων καταστάσεων των καταστάσεων διεργασιών.....	88
Εικόνα 4-9 Βαθμονόμηση Μαγνητόμετρου – Κινητήρων. Στο γράφημα παριστάνεται το ποσοστό παρεμβολών σε συνάρτηση με το καταναλισκόμενο ρεύμα και την ένταση του Throttle	99
Εικόνα 4-10 Μπλοκ Διάγραμμα PID ελεγκτή[60].....	100
Εικόνα 4-11 Συντονισμός μεταβλητών PID στο τετρακόπτερο	101
Εικόνα 5-1 Το τετρακόπτερο έτοιμο για πτήση.....	155

7 Κατάλογος Πινάκων:

Πίνακας 1 Κανάλια Ελέγχου Ελεγκτή Πτήσης.....	48
Πίνακας 2 Αντιστοίχηση Καναλιών ραδιοδέκτη με τις εισόδους ελεγκτή πτήσης APM2.8...	96
Πίνακας 3 Τρόποι Πτήσης Τετρακόπτερου	97
Πίνακας 4 Αναγνώσεις κατανάλωσης ρεύματος τετρακόπτερου σε συνάρτηση με το εφαρμοζόμενο Throttle.....	100

8 Αναφορές / Links

- [1] <https://en.wikipedia.org/wiki/Helicopter>
- [2] <https://commons.wikimedia.org/wiki/File:Airfoil.svg>
- [3] <https://www.emaze.com/@AIQRQFIL/Bernoulli's-Principle-Project>
- [4] https://www.youtube.com/watch?v=8kVuKAqy_2k
- [5] <http://oceanservice.noaa.gov/education/kits/currents/05currents1.html>
- [6] <http://www.copters.com/aero/gyro.html>
- [7] <https://en.wikipedia.org/wiki/Autorotation>
- [8] <https://www.grc.nasa.gov/www/k-12/airplane/smotion.html>
- [9] <http://blog.oscarliang.net/inverse-kinematics-implementation-hexapod-robots/>
- [10] <http://www.rc-airplane-world.com/how-helicopters-fly.html>
- [11] <http://www.wisegeek.com/what-is-torque-reaction.htm>
- [12] http://www.thaitechnics.com/helicopter/heli_principle.html
- [13] <http://aviation.stackexchange.com/questions/8819/is-there-any-equation-to-bind-velocity-thrust-and-power/8822#8822>
- [14] <http://www.rcex.cz/?p=3593>
- [15] http://multicopter.forestblue.nl/lipo_need_calculator.html
- [16] http://www.banggood.com/APM-Flight-Controller-Set-APM-2_8-6MH-GPS-OSD-Radio-Telemetry-etc-p-960341.html
- [17] <http://diydrones.com/profiles/blogs/pixhawk-and-apm-power-consumption>
- [18] <http://ardupilot.org/copter/docs/common-apm-board-leds.html>
- [19] https://en.wikipedia.org/wiki/Dead_reckoning
- [20] <http://www.personal-drones.net/wp-content/uploads/2013/08/Turnigy-9x.pdf>
- [21] Ασύρματες Επικοινωνίες και Δίκτυα, William Stallings, Τζιόλα, 2007, σελ. 230
- [22] http://www.rcuniverse.com/magazine/newproduct.cfm?product_id=4522
- [23] <http://www.rcgroups.com/forums/showthread.php?t=1261306>
- [24] <http://www.tech-faq.com/pulse-position-modulation.html>
- [25] http://www.vims.edu/cbnerr/_docs/monitoring_docs/chpt6.pdf
- [26] <http://www.wireless-technology-advisor.com/wireless-telemetry-systems.html>
- [27] <http://www.ti.com/lit/ug/swru367c/swru367c.pdf>
- [28] <http://www.ti.com.cn/lit/ug/swru372b/swru372b.pdf>
- [29] <https://www.element14.com/community/servlet/JiveServlet/showImage/38-17443-207268/pinout.png>
- [30] <https://www.oreablog.com/2012/12/just-hownimble-are-your-decisions/gyroscope/>
- [31] <http://www.physicalgeography.net/fundamentals/7d.html>

- [32] <https://www.youtube.com/watch?v=Ie3C9-VmR2g>
- [33] Μέθοδοι και Συστήματα Εντοπισμού Θέσης – Πλοϊγηση σε Εσωτερικούς Χώρους, Πτυχιακή Εργασία - Κατσούλη Αναστασία, 2008, σελ.16
- [34] <http://copter.ardupilot.com/wiki/common-ublox-gps/>
- [35] [https://github.com/jlnaudin/x-drone/wiki/How-to-setup-the-GPS-Ublox-NEO-6M-\(Crius\)-CN-06-V2](https://github.com/jlnaudin/x-drone/wiki/How-to-setup-the-GPS-Ublox-NEO-6M-(Crius)-CN-06-V2)
- [36] <http://arduino.stackexchange.com/questions/12740/find-number-of-rx-and-tx-pin-on-arduimu>
- [37] <https://hekilledmywire.wordpress.com/2011/08/03/introduction-to-pwm-part-7/>
- [38] <https://www.youtube.com/watch?v=OZNxbxL7cdc>
- [39] http://www.hobbyking.com/hobbyking/store/_2161_TURNIGY_Plush_12amp_Speed_Controller_w_BEC.html
- [40] <http://www.rcgroups.com/forums/showthread.php?t=1490915>
- [41] <http://hpiracing.world/en/brushless>
- [42] http://www.hobbyking.com/hobbyking/store/_45113_LD1510A_02_P_Micro_Brushless_Outrunner_Motor_16_5g_.html
- [43] <http://www.permatech.com/resources/product-literature>
- [44] <http://hartzellprop.com/faq/technical-questions/>
- [45] http://www.pilotfriend.com/training/flight_training/fxd_wing/props.htm
- [46] http://www.fastonline.org/CD3WD_40/CD3WD/FISH/T0024E/EN/B988_5.HTM
- [47] http://www.socialledge.com/sjsu/index.php?title=S14:_Quadcopter
- [48] https://en.wikipedia.org/wiki/Lithium_polymer_battery
- [49] http://www.hobbyking.com/hobbyking/store/_56095_Turnigy_2700mAh_3S_20C_Lipo_Pack_Suitable_for_Quanum_Nova_Phantom_QR_X350_.html
- [50] <http://www.rcuniverse.com/forum/e-flight-power-sources-126/2523300-lipos-parallel.html>
- [51] <http://hyperphysics.phy-astr.gsu.edu/%E2%80%8Chbase/magnetic/curloo.html#c2>
- [52] <http://ardupilot.org/plane/docs/common-magnetic-interference.html>
- [53] <http://diy.stackexchange.com/questions/20168/is-dont-mix-data-and-power-cables-and-dont-loop-power-cables-still-valid-n>
- [54] <http://www.freertos.org/>
- [55] [https://en.wikipedia.org/wiki/Kernel_\(operating_system\)](https://en.wikipedia.org/wiki/Kernel_(operating_system))
- [56] Δικτύωση Υπολογιστών, Kurose | Ross, M. Γκιούρδας, 2013, σελ. 198
- [57] Δικτύωση Υπολογιστών, Kurose | Ross, M. Γκιούρδας, 2013, σελ. 230
- [58] http://dev.ardupilot.com/wp-content/uploads/sites/6/2015/05/MAVLINK_FOR_DUMMIESPart1_v.1.1.pdf
- [59] <http://ardupilot.org/copter/docs/flight-modes.html>
- [60] <https://en.wikipedia.org/wiki/File:PID-feedback-loop-v1.png>
- [61] <http://diydrones.com/forum/topics/arducopter-tuning-guide>

- [62] http://beej.us/guide/bgnet/output/print/bgnet_A4_2.pdf
- [63] http://wiki.trekk.com/Introduction_to_BSD_Sockets
- [64] http://processors.wiki.ti.com/index.php/CC31xx_%26_CC32xx_UniFlash_Quick_Start_Guide
- [65] Section 6: <http://www.ti.com.cn/lit/ug/swru369c/swru369c.pdf>
- [66] <http://stackoverflow.com/questions/13621386/what-is-pragma-align-in-c>
- [67] <http://diydrones.com/profiles/blogs/propeller-shroud-project>
- [68] https://en.wikipedia.org/wiki/Ducted_propeller