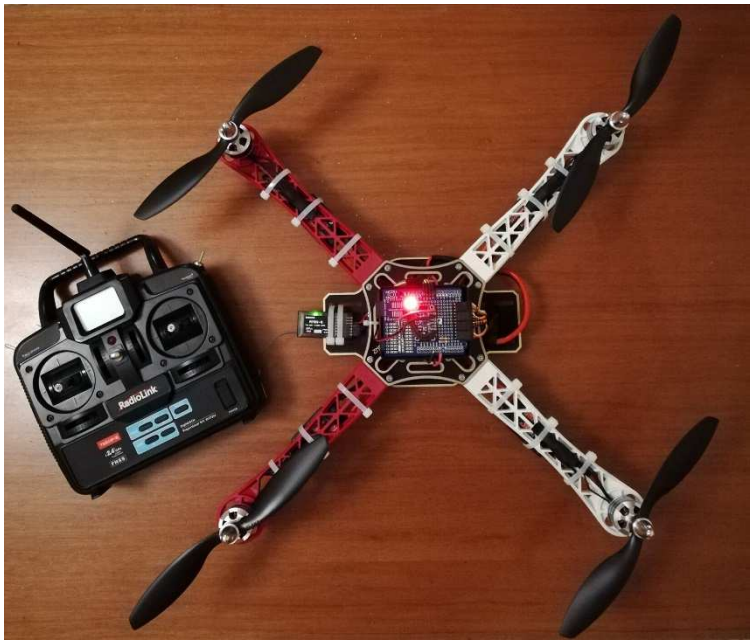
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ

ΘΕΜΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

" ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΤΕΤΡΑΚΟΠΤΕΡΟΥ ΜΕ ΚΑΙΝΟΤΟΜΙΑ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ ΤΟΥ ΕΛΕΓΚΤΗ ΠΤΗΣΗΣ"



ΟΝΟΜΑ ΦΟΙΤΗΤΗ:

ΛΟΥΡΟΣ ΑΘΑΝΑΣΙΟΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:

Δρ. ΜΙΧΑΛΗΣ ΠΑΠΟΥΤΣΙΔΑΚΗΣ

ΑΙΓΑΛΕΩ, ΙΟΥΝΙΟΣ 2018

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο / ~~Η~~ κάτωθι υπογεγραμμένος / ~~η~~ ΛΟΥΡΟΣ ΑΘΑΝΑΣΙΟΣ,

του ΙΩΑΝΝΗ, με αριθμό μητρώου 38347 φοιτητής / ~~τρια~~ του **Τμήματος Βιομηχανικής Σχεδίασης και Παραγωγής του Πανεπιστημίου Δυτικής Αττικής** πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

Ο Δηλών

Ημερομηνία

25/06/2018

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή άσκηση αφορά τον σχεδιασμό και την υλοποίηση ενός ιπτάμενου μη επανδρωμένου αεροσκάφους (quadcopter), το οποίο θα έχει τη δυνατότητα πλοήγησης με τηλεκατεύθυνση. Ειδικότερα υλοποιήθηκε μια διάταξη ελέγχου πτήσης με τη χρήση πλακέτας Arduino και τον προγραμματισμό της με γλώσσα C++.

Στο πρώτο κεφάλαιο αναφέρομαι στα μη επανδρωμένα οχήματα, τη χρήση τους, τα πλεονεκτήματά αλλά και τα μειονεκτήματά τους. Επίσης γίνεται μια ιστορική αναδρομή από το πως ξεκίνησαν μέχρι σήμερα.

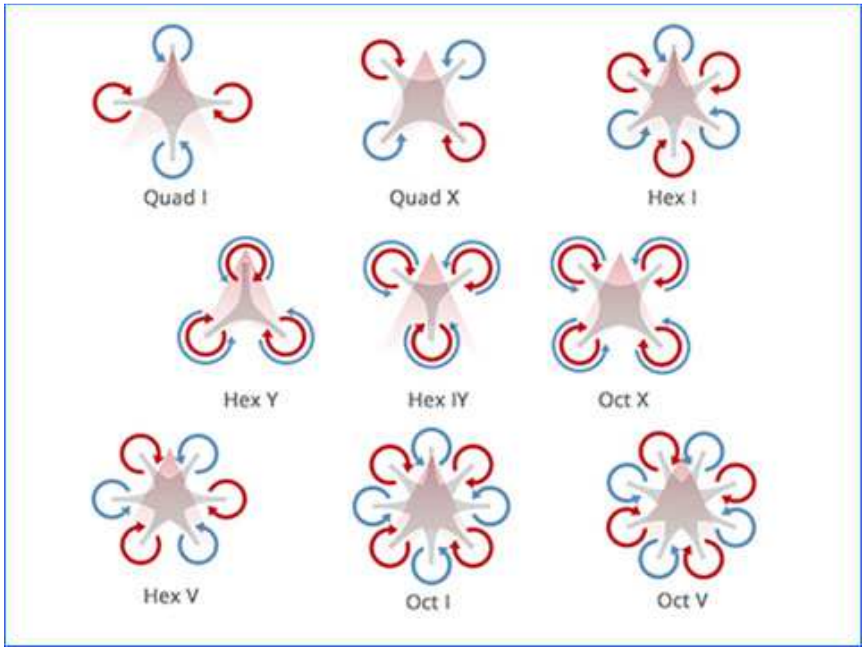
Στο δεύτερο κεφάλαιο αναφέρομαι στο κατασκευαστικό μέρος της εργασίας μου, δηλαδή στο σχεδιασμό του ηλεκτρονικού σχεδίου αλλά και στα επιμέρους εξαρτήματα που χρησιμοποίησα έτσι ώστε να υλοποιηθεί το quadcopter.

Στο τρίτο κεφάλαιο γίνεται η θεωρητική προσέγγιση της πτυχιακής εργασίας. Παρουσιάζεται ο κώδικας προγραμματισμού, ο οποίος χωρίζεται σε 3 διαφορετικά προγράμματα.

ΚΕΦΑΛΑΙΟ 1

1.1 Εισαγωγή στα Multirotors

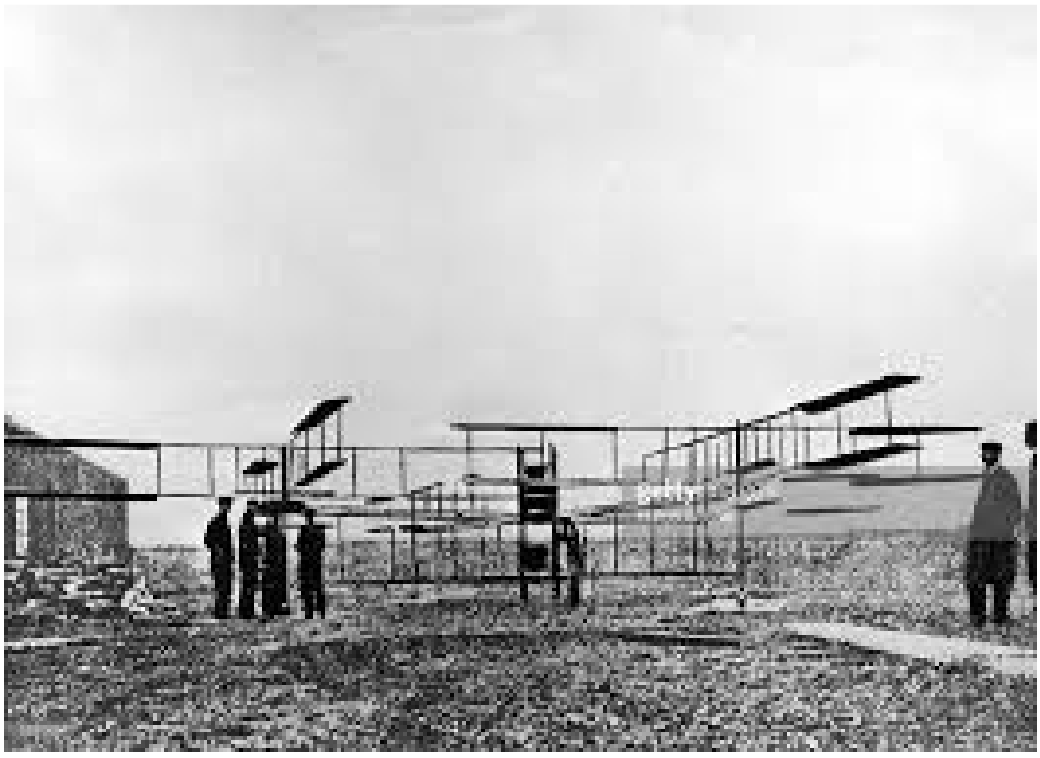
Τα εναέρια οχήματα πολλών στροφών είναι γνωστά με τη ονομασία Multirotor Aerial Vehicles (MRAV) ή με μια λέξη Multicopters. Επίσης με τη λέξη στροφείο εννοούμε ότι το όχημα είναι καθέτου απογείωσης και προσγείωσης (Vertical Take Off And Landing - VTOL).
Τα στροφεία στα οχήματα που αναφερόμαστε εδώ, είναι ουσιαστικά έλικες, όπως αυτές των αεροσκαφών, όπου το βήμα των πτερυγίων είναι σταθερό. Η γενική κατηγορία οχημάτων στη οποία ανήκει και το multirotor είναι τα rotorcrasfts (στροφειόπτερα) τα οποία είναι ιπτάμενες μηχανές βαρύτερες από τον αέρα που χρησιμοποιούν μία ή περισσότερες περιστρεφόμενες πτέρυγες, έτσι ώστε να παράγουν την απαραίτητη ώθηση για άνωση. Η επίσημη ονομασία για τέτοιου είδους οχήματα που ταυτόχρονα είναι και μη επανδρωμένα είναι RUAV (Rotorcraft Unmanned Aerial Vehicles). Τα RUAV φέρουν έστω μια μικρή αυτονομία στη πλοήγησή τους, φέρουν δηλαδή κάποιο ηλεκτρονικό κύκλωμα ελέγχου. Όταν όμως έχουν πλήρη αυτονομία και μπορούν να πλοηγηθούν χωρίς ανθρώπινη παρέμβαση ονομάζονται RUAS (Rotorcraft Unmanned Aerial System). Ως UAV αναφέρονται τα μη επανδρωμένα ιπτάμενα οχήματα σταθερής πτέρυγας (fixed-wing).
Τα Multirotors λοιπόν είναι ένα RUAV με το χαρακτηριστικό όμως ότι φέρει παραπάνω από δύο έλικες. Ο έλεγχος της κίνησής τους επιτυγχάνεται μεταβάλλοντας τη σχετική ταχύτητα περιστροφής μεταξύ των στοιχείων, ώστε να μεταβληθεί ώση και η ροπή. Επειδή όμως π έλεγχος αυτός είναι αρκετά πολύπλοκος, τον αναλαμβάνει ενσωματωμένος μικροελεγκτής στο MRAV, ο οποίος με πολλαπλές μετρήσεις από κατάλληλους αισθητήρες καταφέρνει να το σταθεροποιεί. Τα MRAV έχουν συνήθως τρείς έλικες (trirotor ή tricopter), τέσσερις έλικες (quadrotor ή quadcopter), έξι έλικες (hexarotor ή hexacopter), οκτώ έλικες (octorotor ή octocopter) ή και παραπάνω έλικες.



Σχήμα 1 : Είδη multicopter σύμφωνα με τον αριθμό ελίκων και τον σχηματισμό τους

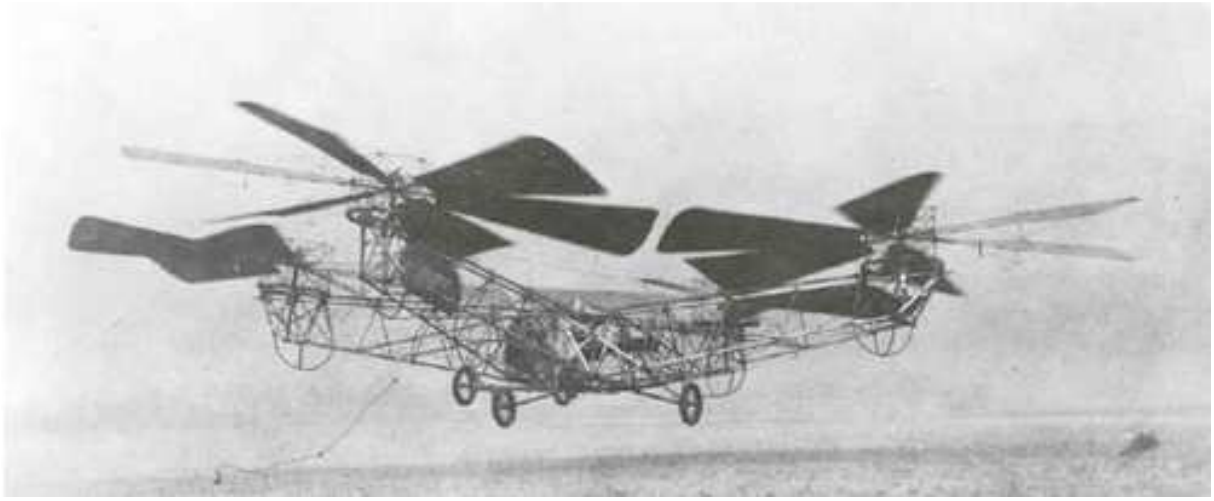
1.2 Ιστορία του Τετρακόπτερου

Τα UAV τεσσάρων ελίκων (Quadcopters) παρότι έχει λάβει ιδιαίτερο ενδιαφέρον τα τελευταία χρόνια, δεν αποτελεί καινούργια ιδέα. Τα πρώτα Quadcopters εμφανίζονται στις αρχές του 20^{ου} αιώνα, όταν μηχανικοί δοκίμασαν διάφορους τύπους μηχανών καθέτου προσγείωσης. Τα Quadcopter μάλιστα προέκυψαν χρονολογικά νωρίτερα από τα συμβατικά ελικόπτερα μιας έλικας, όπως τα γνωρίζουμε δηλαδή και σήμερα.
Το 1904 ο Γάλλος επιστήμονας Charles Richet κατασκεύασε ένα μικρό μη επανδρωμένο ελικόπτερο. Αν και το ελικόπτερο αυτό δεν είχε ιδιαίτερη επιτυχία, προκάλεσε το ενδιαφέρον ενός από τους μαθητές του, του μετέπειτα πρωτοπόρου της αεροπλοΐας, Louis Brequet. Κατά το 1906 ο Louis Brequet μαζί με τον αδερφό του Jacques, ξεκίνησαν να πειραματίζονται με το ελικόπτερο αυτό, υπό την καθοδήγηση του καθηγητή τους. Ο Louis μελέτησε προσεκτικά τα ατρακτοειδή σχήματα, εστιάζοντας την προσοχή του στο αεροπλάνο των αδελφών Wright που μόλις τρία χρόνια πριν είχαν κάνει την πρώτη επιτυχή πτήση. Το 1907, έχοντας κατανοήσει πλήρως την αεροδυναμική θεωρία του ελικοπτέρου, οι αδερφοί Brequet κατασκεύασαν το πρώτο ελικόπτερο quadrotor. Το ελικόπτερο αυτό ονομάστηκε Gyroplane No.1 και αποτελούνταν από έναν ατσαλένιο σκελετό σε σχήμα σταυρού, στις άκρες του οποίου υπήρχαν τέσσερις έλικες. Ο κάθε έλικας είχε οκτώ πτερύγια, τα οποία ήταν χωρισμένα σε δύο παράλληλες τετράδες. Τα οκτώ αυτά πτερύγια ήταν μηχανικά ενωμένα και είχαν την δυνατότητα να περιστρέφονται συγχρόνως. Οι έλικες κινούνταν από μια μηχανή 40hp που βρισκόταν δίπλα στον πιλότο. Το ελικόπτερο αυτό κατάφερε να ανυψωθεί και να πετάξει για πολύ σύντομο χρονικό διάστημα, λόγω έλλειψης ευστάθειας και κατάλληλων μέσων ελέγχου. Παρόλα αυτά το Gyroplane No.1 ήταν το πρώτο μηχανοκίνητο ελικόπτερο στην ιστορία που μπορούσε να μεταφέρει το βάρος ενός ατόμου.



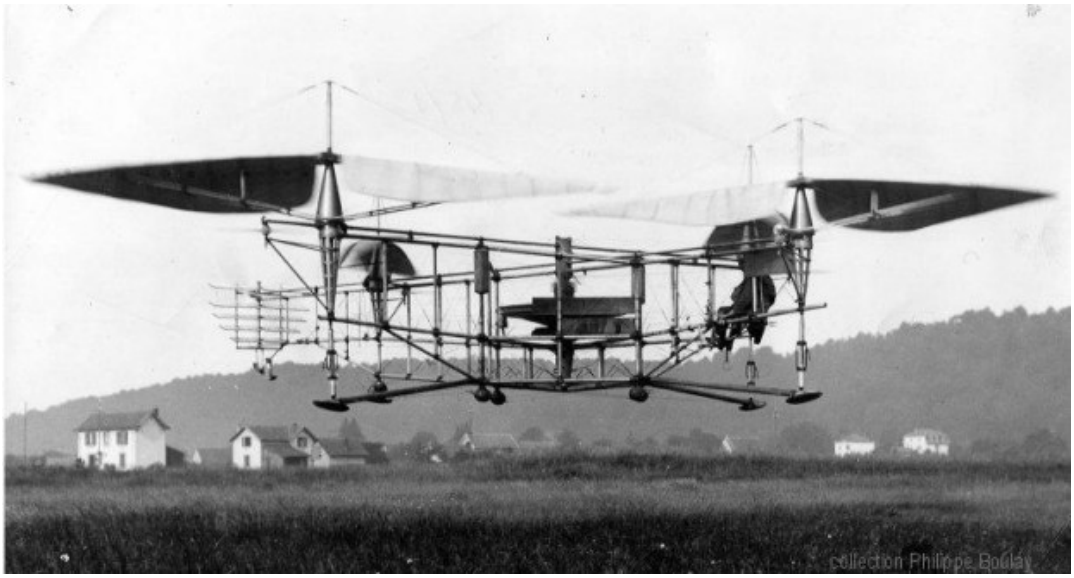
Σχήμα 2 : Το quadrotor των αδερφών Brequet (1907)

Το 1922 ένας Ρώσος μετανάστης στην Αμερική, ο Georges De Bothezat, κατόπιν συνεργασίας του με τον αμερικανικό στρατό, κατασκεύασε ένα από τα μεγαλύτερα ελικόπτερα που είχαν κατασκευαστεί εκείνη την εποχή. Ήταν ένα ελικόπτερο quadrotor με έξι πτερύγια σε κάθε έλικα, ενώ διέθετε τέσσερις βοηθητικές έλικες, ανά δύο όμοιες. Το ίδιο έτος το ελικόπτερο αυτό πέταξε, σε χαμηλό ύψος, επιτυχώς πολλές φορές. Μέχρι τα τέλη του 1923 είχε πραγματοποιήσει περίπου εκατό πτήσεις φτάνοντας σε ύψος τα 5 μέτρα.



Σχήμα 3 : Το quadrotor του Georges De Bothezat (1922)

Το 1922, παράλληλα με τον Georges De Bothezat, ο Etienne Oehmichen, κατασκεύασε ένα ελικόπτερο που αποτελούνταν από τέσσερις κυρίους έλικες και οκτώ μικρότερους βοηθητικούς. Το ελικόπτερο αυτό διέθετε μια μηχανή 120hp, η οποία στη συνέχεια αντικαταστάθηκε από μηχανή 180hp. Η ονομασία που του έδωσε ήταν Oehmichen No.2 και ήταν το πιο επιτυχημένο από τα προηγούμενα έξι μοντέλα του συγκεκριμένου μηχανικού. Πραγματοποίησε πολλές πτήσεις μέσα σε ένα χρόνο αποδεικνύοντας ότι μια μηχανή κάθετης πτήσης θα μπορούσε να έχει ευστάθεια και ευκινησία. Τον Μάιο του 1924, πραγματοποίησε μια κλειστή διαδρομή ενός χιλιομέτρου, μένοντας στον αέρα για πάνω από 7 λεπτά.



Σχήμα 4 : Το quadrotor του Etienne Oehmichen (1922)

Στα επόμενα χρόνια οι μηχανικοί έστρεψαν την προσοχή τους στα συμβατικά ελικόπτερα με μια ή δύο κύριες έλικες. Η ανάπτυξη αυτών των ελικοπτέρων ήταν ραγδαία, έτσι ώστε σιγά σιγά να παίρνουν την μορφή που έχουν και σήμερα. Παρόλη την υπεροχή των ελικοπτέρων μιας έλικας κατά καιρούς κατασκευάστηκαν ελικόπτερα quadrotor.

1.3 Χρησιμότητα των UAV

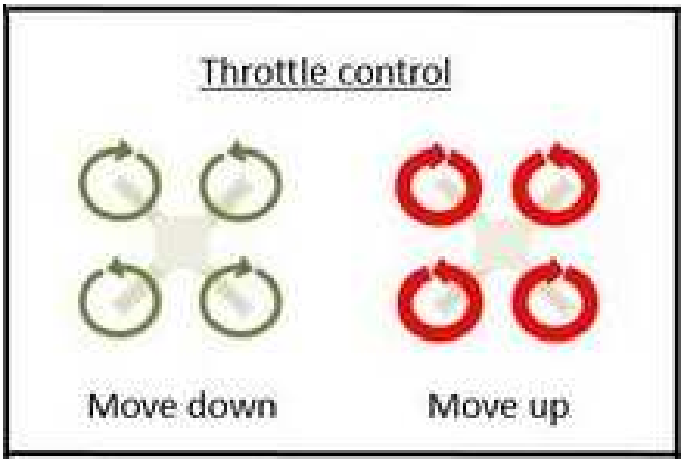
Ένα UAV (Unmanned Aerial Vehicle) είναι ένα αεροσκάφος χωρίς τη ύπαρξη άμεσου χειρισμού από κάποιο χειριστή. Οι πτήσεις που πραγματοποιούνται είναι είτε αυτόνομα είτε ελεγχόμενα μέσω μίκρο-υπολογιστών, οι οποίοι υπάρχουν μέσα στο αεροσκάφος, είτε στο χειριστήριο.
Τα UAV ελέγχονται κατά την πτήση τους από πιλότους μέσω ενός χειριστηρίου, αλλά ο αυτόνομος έλεγχος με το πέρας του χρόνου χρησιμοποιείται ολοένα και περισσότερο. Τα UAV είχαν αναπτυχθεί κατά κύριο λόγο για στρατιωτικές εφαρμογές, όμως υπάρχει μια σταδιακή αύξηση χρησιμοποίησής τους και σε αστικές εφαρμογές, όπως σε πυροσβεστικές αποστολές και σε μη στρατιωτικής ασφάλειας, για παράδειγμα στην επιτήρηση των αγωγών μιας μεγάλης εγκατάστασης. Τέλος τα UAV μπορούν να χρησιμοποιηθούν σε αποστολές έρευνας και διάσωσης, βοηθώντας ουσιαστικά στο να βρεθούν εξαφανισμένοι ή εγκλωβισμένοι άνθρωποι σε μη προσβάσιμα μέρη.
Πέρα όμως από τα αρκετά πλεονεκτήματα που μας προσφέρουν σας κατασκευές, πρέπει να ξεπεραστούν και κάποια μειονεκτήματα ως προς τη υλοποίησή τους. Το κυριότερο είναι πως όλα τα αισθητήρια παράγουν ΄΄θορύβους΄΄ με αποτέλεσμα οι μετρήσεις τους να μην είναι τόσο ακριβείς. Επίσης υπάρχει αυξημένη κατανάλωση ισχύος λόγω των πολλών μοτέρ που διαθέτουν, επομένως απαιτούνται ακριβές μπαταρίες, αλλιώς η αυτονομία τους είναι περιορισμένη. Ένα ακόμη πρόβλημα είναι η απαραίτητη χρήση κάμερας όταν τοποθετείται σε χώρους που δεν βρίσκονται στο οπτικό πεδίο του χειριστή – πιλότου, πράγμα που επιβαρύνει το χρόνο διάρκειας της πτήσης. Επιπρόσθετα, με τη χρήση κάμερας υπάρχει ένα ασαφές νομικό πλαίσιο περί ανθρωπίνων δικαιωμάτων και προσωπικού απόρρητου.

1.4 Το Quadcopter

Στο σημείο αυτό κρίνεται απαραίτητη η παρουσίαση της βασικής κατασκευής του τετρακόπτερου καθώς και του τρόπου με τον οποίο ίπταται και εκτελεί τους διάφορους ελιγμούς του.
Το quadcopter είναι ένας τύπος ελικοπτέρου που χρησιμοποιεί αποκλειστικά τέσσερις έλικες, ίσης διαμέτρου, για την ανύψωση και προώθησή του. Οι τέσσερις αυτοί έλικες τοποθετούνται συμμετρικά πάνω σε ένα σκελετό σχήματος σταυρού, ενώ στο κέντρο του σκελετού αυτού βρίσκεται το ωφέλιμο φορτίο. Οι έλικες του ελικοπτέρου περιστρέφονται ανά δύο αντίστροφα, έτσι ώστε η συνολική ροπή που εφαρμόζεται στο κέντρο του να είναι μηδενική. Συγκεκριμένα ο μπροστά αριστερά με τον πίσω δεξιά περιστρέφονται δεξιόστροφα, ενώ ο μπροστά δεξιά με τον πίσω αριστερά, αριστερόστροφα, όπως φαίνεται και στο Σχήμα.

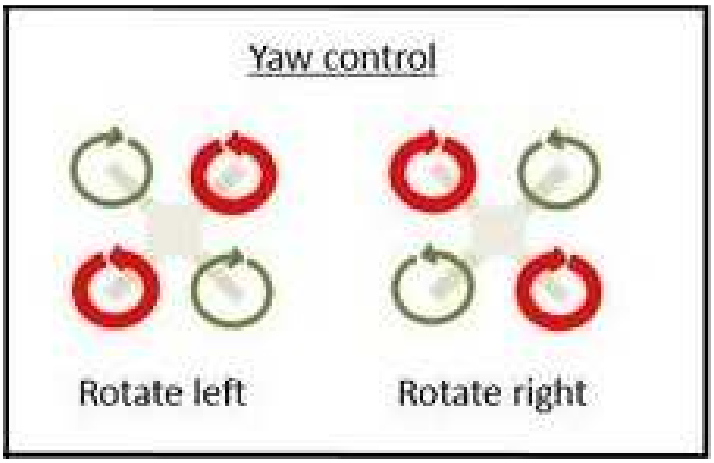
Σχήμα 5 : Οι περιστροφές στους έλικες του τετρακόπτερου.

Η κίνηση του ελικοπτέρου quadcopter ελέγχεται αποκλειστικά από τις γωνιακές ταχύτητες των τεσσάρων ελίκων του. Όταν οι τέσσερις έλικές του κινούνται με την ίδια ακριβώς γωνιακή ταχύτητα, τότε το ελικόπτερο ανυψώνεται. Συγχρόνως η κλίση του διατηρείται σταθερή, ενώ δεν περιστρέφεται γύρω από το κέντρο μάζας του, επειδή είναι εντελώς συμμετρικό (Σχήμα).



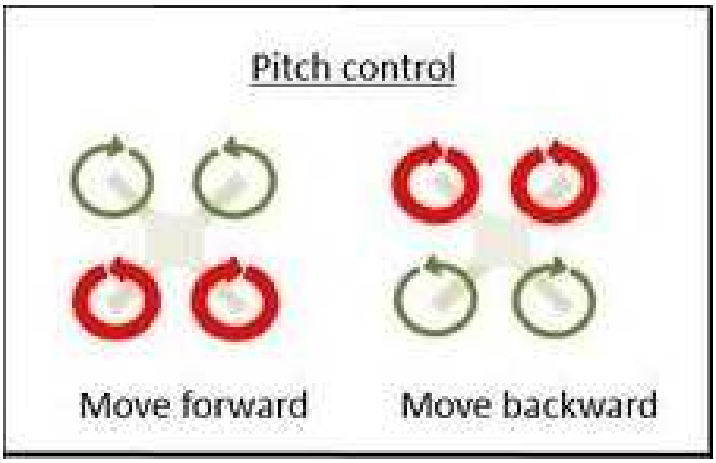
Σχήμα 6 : Ανύψωση quadcopter.

Για να επιτευχθεί η περιστροφή του ελικοπτέρου γύρω από το κέντρο μάζας του, χρειάζεται να αυξομειωθεί η ταχύτητα δύο όμοια περιστρεφόμενων κινητήρων (Σχήμα). Όταν αλλάξει η ταχύτητα ενός κινητήρα, τότε αλλάζει και η ροπή του σύμφωνα με τη χαρακτηριστική ροπής-ταχύτητας του κινητήρα-φορτίου (φορτίο θεωρούμε τον έλικα του κινητήρα). Επειδή όμως, λόγω του τρίτου νόμου του Newton, η ροπή του ρότορα του κινητήρα ισούται κάθε χρονική τιμή με τη ροπή του στάτη, η αυξομείωση της ροπής του κινητήρα ισοδυναμεί με αυξομείωση της ροπής που δέχεται η βάση από τον κινητήρα. Επομένως, η βάση τείνει να περιστραφεί γύρω από τον κάθετο άξονά της, όταν οι ροπές των τεσσάρων κινητήρων δεν είναι εξισορροπημένες. Οι ροπές αυτές, για το λόγο ότι οφείλονται στην αντίσταση του αέρα, αναφέρεται και ως ροπή αντίστασης (drag moments). Στο Σχήμα 7 φαίνεται πως επιτυγχάνεται η περιστροφή του τετρακόπτερου γύρω από τον κάθετο άξονά του. Παρουσιάζεται η δεξιόστροφη και η αριστερόστροφη περιστροφή.



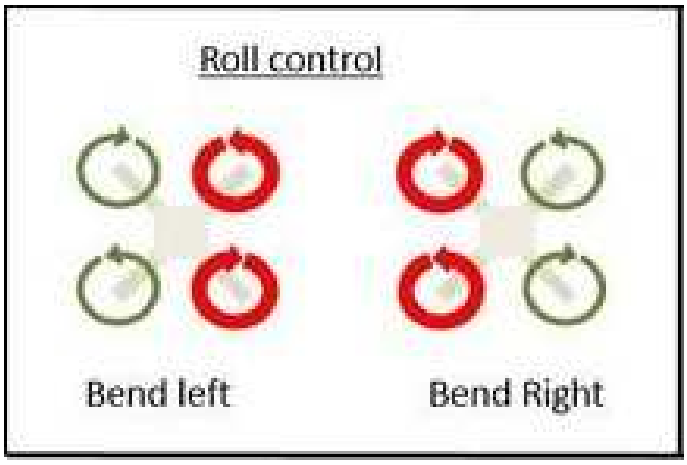
Σχήμα 7 : Περιστροφή quadcopter γύρω από τον εαυτό του.

Η παράλληλη προώθηση του ελικοπτέρου, επιτυγχάνεται όταν αυτό αποκτήσει κάποια κλίση ως προς το έδαφος. Αυτό συμβαίνει όταν δύο κινητήρες, της ίδιας πλευράς, έχουν μικρότερη ταχύτητα από τους άλλους δύο. Η κίνηση στην οριζόντια συνιστώσα φαίνεται στο Σχήμα 8.



Σχήμα 8 : Οριζόντια μετατόπιση του quadcopter.

Ενώ η κίνηση στον κάθετο άξονα φαίνεται στο Σχήμα 9.



Σχήμα 9 : Κάθετη μετατόπιση του quadcopter.

ΚΕΦΑΛΑΙΟ 2

2.1 Εξαρτήματα quadcopter

Για την κατασκευή του τετρακόπτερου χρησιμοποιήθηκαν τυποποιημένα στοιχεία που αγοράστηκαν από μαγαζιά αερομοντελισμού και ηλεκτρονικών. Η τοποθέτηση τους δεν ήταν απλή καθώς έπρεπε να γίνει με τέτοιο τρόπο ώστε το quadcopter να είναι πλήρων συμμετρικό ως προς τον οριζόντιο και κάθετο άξονα, ενώ το κέντρο βάρους του έπρεπε να είναι ακριβώς στη τομή των αξόνων αυτών.
Τα βασικά στοιχεία λοιπόν που αποτελείται το drone αυτό είναι :

- Σκελετός μοντέλου F450 (Σχήμα 10)

Σχήμα 10 : Σκελετός F450

Η βάση αυτού του σκελετού αποτελείται από αλουμίνιο χαμηλού βάρους και έχει δύο μέρη, το κάτω μέρος με διαστάσεις 18cm x 11,8cm και το επάνω με διαστάσεις 10,5cm x 10,5cm. Το κάθε ένα από τα τέσσερα πόδια αποτελείται από επεξεργασμένο πλαστικό υψηλής μηχανικής αντοχής με διαστάσεις 21,5cm. Τα πόδια βιδώνονται μεταξύ τους με αλενόβιδες M3 x 6mm στο κάτω και στο πάνω μέρος της βάσης.

- Κινητήρες EMAX XA2212 των 980KV (Σχήμα 11)

Σχήμα 11 : Κινητήρας EMAX XA2212

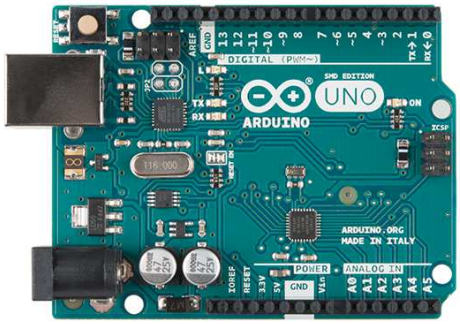
Για την κίνηση των ελίκων χρησιμοποιούνται αποκλειστικά brushless outrunner DC κινητήρες, για αυξημένη απόδοση. Έχουν διαστάσεις 4,3cm x 2,7cm . Αποτελούνται από δεκατέσσερις πόλους και περιλαμβάνουν μόνιμους μαγνήτες νεοδυμίου για υψηλή απόδοση. Η ονομασία τους outrunner δηλώνει ότι ο δρομέας, δηλαδή το κινούμενο μέρος του κινητήρα, είναι το εξωτερικό μέρος και όχι το εσωτερικό του.

- Ηλεκτρικούς Ελεγκτές Ταχύτητας (Electric Speed Controllers) EMAX Simonk των 30A (Σχήμα 12)

Σχήμα 11 : Ηλεκτρικός Ελεγκτής Ταχύτητας EMAX Simonk

Είναι απαραίτητοι για την μετατροπή του συνεχούς ρεύματος της μπαταρίας σε εναλλασσόμενο ρεύμα, για την τροφοδοσία των κινητήρων. Περιέχουν έναν μικροελεγκτή καθώς και ένα σύστημα ασφαλείας για υψηλές τιμές ρεύματος. Οι ελεγκτές που επιλέχτηκαν είναι σχεδιασμένοι για μέγιστο ρεύμα 30A .

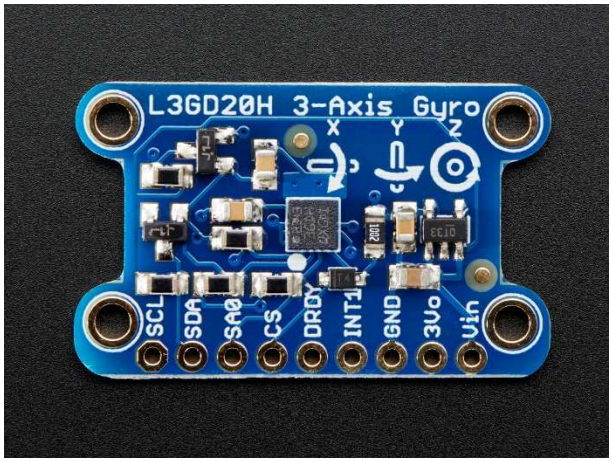
- Πλακέτα μικροελεγκτή Arduino Uno (Σχήμα 13)



Σχήμα 13 : Arduino Uno

Η πλακέτα Arduino Uno περιλαμβάνει τον μικροελεγκτή του τετρακόπτερου, τον ATMega328P. Ο μικροελεγκτής αυτός λειτουργεί στα 16 MHz, έχει 32KB Flash Memory για τον προγραμματισμό του. Οι 31 ακίδες εισόδου εξόδου που περιλαμβάνει μπορούν να χρησιμοποιηθούν είτε σαν θύρες εισόδου εξόδου γενικού σκοπού, είτε για την επικοινωνία των περιφερειακών συστημάτων με το εξωτερικό περιβάλλον. Επίσης έχει τοποθετηθεί πάνω ακριβώς από το Arduino Uno και μια Arduino Shield, με σκοπό την μείωση των καλωδιακών συνδέσεων που έχουν πραγματοποιηθεί.

- Αισθητήρας γωνιακής ταχύτητας τριών αξόνων L3GD20H (Σχήμα 14)



Σχήμα 14 : Gyro L3GD20H

Πρόκειται για μια ηλεκτρονική συσκευή που αντιλαμβάνεται την γωνιακή ταχύτητα του τετρακόπτερου. Αυτό γίνεται με αισθητήρες δόνησης, οι οποίοι χρησιμοποιούνται για την ανίχνευση της γωνιακής ταχύτητας και εφαρμόζονται σε ένα δονούμενο στοιχείο. Αυτοί οι αισθητήρες παρέχουν σταθερότητα ανιχνεύοντας την κίνηση που προκαλείται από τους διάφορους κραδασμούς.

- Μπαταρία Λιθίου 2200MAh στα 11.1Volts (Σχήμα 15)

Σχήμα 15 : Μπαταρία 2200mAh στα 11.1Volts

Η επαναφορτιζόμενη μπαταρία πολυμερούς λιθίου παρέχει υψηλότερη ενέργεια από άλλους τύπους μπαταριών λιθίου και χρησιμοποιούνται σε εφαρμογές όπου το βάρος είναι ένα κρίσιμο χαρακτηριστικό. Η συγκεκριμένη μπαταρία έχει 3 κελιά (cells) με χωρητικότητα 2200mAh και πλήρως φορτισμένη παρέχει 11,1Volts.

- Έλικες (4 τεμάχια) των 8x4.5 (Σχήμα 16)

Σχήμα 16 : Έλικες 8x4.5

Οι έλικες κατηγοριοποιούνται πάντα ανάλογα με το μήκος και το βήμα. Όταν μιλάμε για βήμα εννοούμε την απόσταση που διανύεται από ένα drone σε μία μόνο περιστροφή της έλικας. Συγκεκριμένα, εδώ χρησιμοποιούνται έλικες 8x4.5, αυτό σημαίνει ότι το μήκος είναι 8 ίντσες ενώ το βήμα είναι 4,5.

- Δέκτης σημάτων RadioLink Receiver R7EH-S (Σχήμα) και Πομπός-Χειριστήριο RadioLink T6EHP-E (Σχήμα 17)

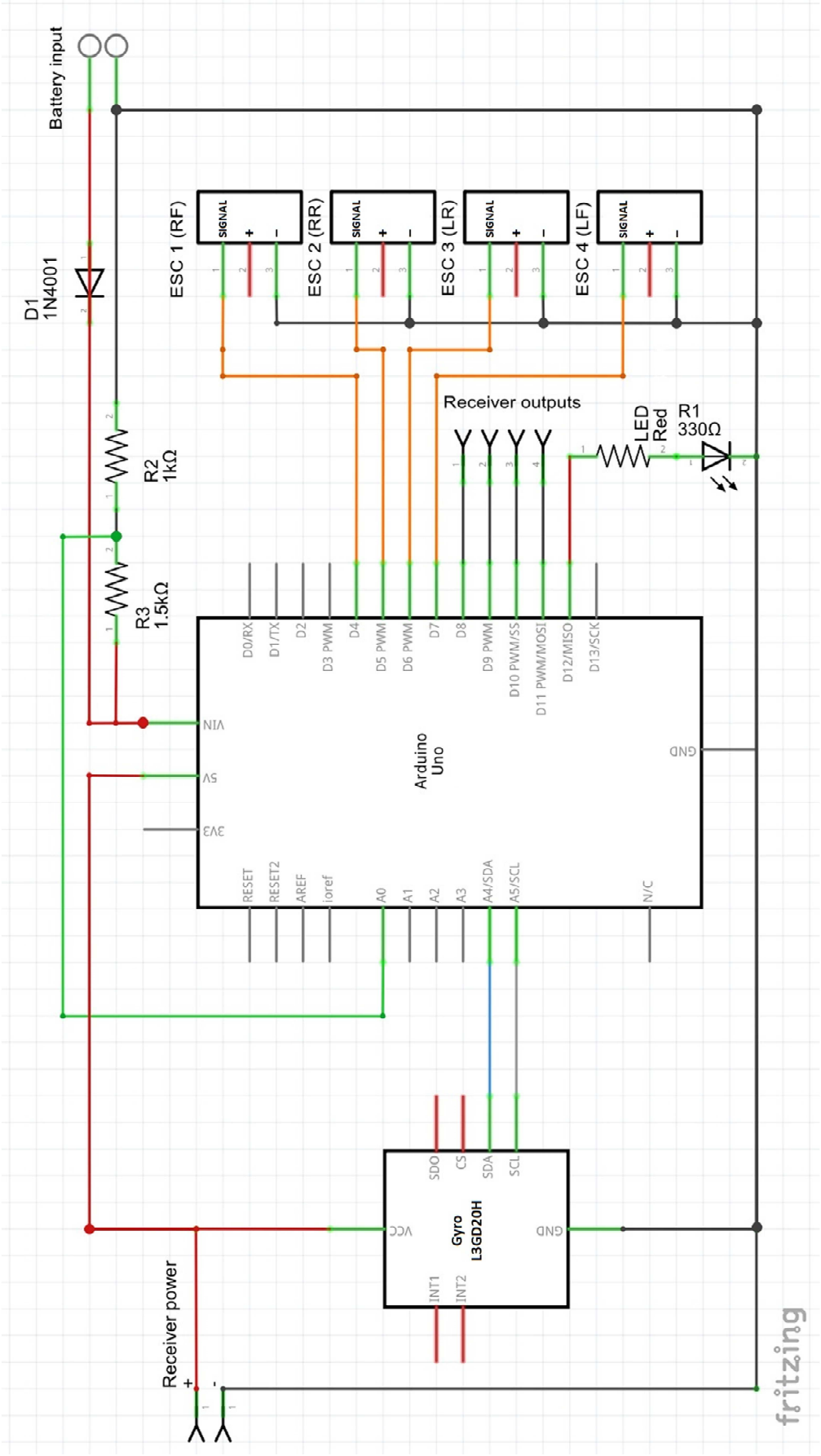
Σχήμα 17 : Χειριστήριο-Δέκτης και Πομπός RadioLink

Πρόκειται για τα βασικά στοιχεία του εξοπλισμού επικοινωνίας μας με το τετρακόπτερο. Ο δέκτης μας έχει επτά κανάλια επικοινωνίας (channels), με μέγιστο ρεύμα τροφοδοσίας τα 6Volts και ρεύμα λειτουργίας τα 19 – 25mA. Από την άλλη το χειριστήριο που χρησιμοποιούμε διαθέτει έξι κανάλια (channels) των 2,4GHz.

- Λοιπά εξαρτήματα

Εκτός από τα βασικά εξαρτήματα χρησιμοποιούνται, πέρα από τα διάφορα καλώδια, δύο αντιστάσεις των 1ΚΩ και 1.5ΚΩ, μια δίοδος 1N4001 και ένα λαμπάκι LED 330Ω.

2.2 Ηλεκτρικό σχέδιο quadcopter



Σχήμα 18 : Ηλεκτρικό κύκλωμα συνδεσμολογίας

ΚΕΦΑΛΑΙΟ 3

3.1 Πρόγραμμα SETUP

Αφού συνδέσουμε όλα τα εξαρτήματα, ήρθε η στιγμή για τον προγραμματισμό του ελεγκτή πτήσης μας. Αυτό θα γίνει σε 3 διαφορετικά προγράμματα τα οποία πρέπει να γίνουν upload στη πλακέτα μας σε συγκεκριμένη σειρά.
Στην αρχή γίνεται το upload του προγράμματος SETUP, μέσω καλωδίου USB 2.0. Για λόγους ασφαλείας πρέπει να αφαιρεθούν οι έλικες και να ΜΗΝ είναι συνδεδεμένη η μπαταρία μας. Στη συνέχεια μέσω του προγράμματος Arduino.exe ανοίγουμε την σειριακή οθόνη(serial monitor), το ρυθμίσουμε στα 57600bps και εν συνεχεία ακολουθούμε τις οδηγίες που αναγράφονται. Οι οδηγίες αυτές έχουν να κάνουν με την ρύθμιση των παραμέτρων επικοινωνίας του χειριστηρίου μας με το drone, καθώς επίσης και με το γυροσκόπιο που έχει εγκατασταθεί. Οι πληροφορίες αυτές στο τέλος του προγράμματος αποθηκεύονται στην βιβλιοθήκη EEPROM για μεταγενέστερη χρήση στα επόμενα προγράμματα. Σε περίπτωση κάποιου σφάλματος το LED θα αναβοσβήσει και το πρόγραμμα SETUP θα τερματιστεί.
Αν όλα πάνε καλώς, χωρίς δηλαδή κανένα απολύτως πρόβλημα στη διαδικασία εγκατάστασης, τότε πρέπει να γίνει upload στο δεύτερο πρόγραμμά μας, με όνομα ESC_CALIBRATION. Ο κώδικας που χρησιμοποιείται είναι ο εξής:

```
#include <Wire.h>                 //Include the Wire.h library so we can communicate with the gyro
#include <EEPROM.h>               //Include the EEPROM.h library so we can store information into the EEPROM

//Declaring Global Variables
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
byte lowByte, highByte, type, gyro_address, error;
byte channel_1_assign, channel_2_assign, channel_3_assign, channel_4_assign;
byte roll_axis, pitch_axis, yaw_axis;
byte receiver_check_byte, gyro_check_byte;
volatile int receiver_input_channel_1, receiver_input_channel_2, receiver_input_channel_3, receiver_input_channel_4;
int center_channel_1, center_channel_2, center_channel_3, center_channel_4;
int high_channel_1, high_channel_2, high_channel_3, high_channel_4;
int low_channel_1, low_channel_2, low_channel_3, low_channel_4;
int address, cal_int;
unsigned long timer, timer_1, timer_2, timer_3, timer_4, current_time;
float gyro_pitch, gyro_roll, gyro_yaw;
float gyro_roll_cal, gyro_pitch_cal, gyro_yaw_cal;


//SETUP ROUTINE
void setup(){
  pinMode(12, OUTPUT);
  //Arduino pins default to inputs, so they don't need to be explicitly declared as inputs
  PCICR |= (1 << PCIE0);        // set PCIE0 to enable PCMSK0 scan
```

```
  PCMSK0 |= (1 << PCINT0);  // set PCINT0 (digital input 8) to trigger an interrupt on state change
  PCMSK0 |= (1 << PCINT1);  // set PCINT1 (digital input 9)to trigger an interrupt on state change
  PCMSK0 |= (1 << PCINT2);  // set PCINT2 (digital input 10)to trigger an interrupt on state change
  PCMSK0 |= (1 << PCINT3);  // set PCINT3 (digital input 11)to trigger an interrupt on state change
  Wire.begin();               //Start the I2C as master
  Serial.begin(57600);      //Start the serial connection @ 57600bps
  delay(250);                 //Give the gyro time to start
}

void loop(){              //MAIN PROGRAM

//TRANSMITTER SETUP
  if(error == 0){         //Quit the program in case of an error
    Serial.println(F("======================================================"));
    Serial.println(F("1 ) TRANSMITTER SETUP"));
    Serial.println(F("======================================================"));
    delay(1000);
    Serial.print(F("Checking for receiver signals."));
    //Wait 10 seconds until all receiver inputs are valid
    wait_for_receiver();
    Serial.println(F(""));
  }

  if(error == 0){          //Quit the program in case of an error
    delay(2000);
    Serial.println(F("Place all sticks in the center position within 10 seconds."));
    for(int i = 9;i > 0;i--){
      delay(1000);
      Serial.print(i);
      Serial.print(" ");
    }
    Serial.println(" ");
//Store the central stick positions
    center_channel_1 = receiver_input_channel_1;
    center_channel_2 = receiver_input_channel_2;
    center_channel_3 = receiver_input_channel_3;
    center_channel_4 = receiver_input_channel_4;
    Serial.println(F(""));
    Serial.println(F("Center positions stored."));
    Serial.print(F("Digital input 08 = "));
    Serial.println(receiver_input_channel_1);
    Serial.print(F("Digital input 09 = "));
    Serial.println(receiver_input_channel_2);
    Serial.print(F("Digital input 10 = "));
    Serial.println(receiver_input_channel_3);
    Serial.print(F("Digital input 11 = "));
    Serial.println(receiver_input_channel_4);
    Serial.println(F(""));
  }


  if(error == 0){             //Quit the program in case of an error
    Serial.println(F("Move the throttle stick to full throttle and back to center"));
```

```
//Check for throttle movement
    check_receiver_inputs(1);
    Serial.print(F("Throttle is connected to digital input "));
    Serial.println((channel_3_assign & 0b00000111) + 7);
    if(channel_3_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));
    else Serial.println(F("Channel inverted = no"));
    wait_sticks_zero();


    Serial.println(F(""));
    Serial.println(F("Move the roll stick to simulate left wing up and back to center"));
//Check for throttle movement
    check_receiver_inputs(2);
    Serial.print(F("Roll is connected to digital input "));
    Serial.println((channel_1_assign & 0b00000111) + 7);
    if(channel_1_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));
    else Serial.println(F("Channel inverted = no"));
    wait_sticks_zero();
  }
  if(error == 0){
    Serial.println(F(""));
    Serial.println(F("Move the pitch stick to simulate nose up and back to center"));
//Check for throttle movement
    check_receiver_inputs(3);
    Serial.print(F("Pitch is connected to digital input "));
    Serial.println((channel_2_assign & 0b00000111) + 7);
    if(channel_2_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));
    else Serial.println(F("Channel inverted = no"));
    wait_sticks_zero();
  }
  if(error == 0){
    Serial.println(F(""));
    Serial.println(F("Move the yaw stick to simulate nose right and back to center"));
//Check for throttle movement
    check_receiver_inputs(4);
    Serial.print(F("Yaw is connected to digital input "));
    Serial.println((channel_4_assign & 0b00000111) + 7);
    if(channel_4_assign & 0b10000000)Serial.println(F("Channel inverted = yes"));
    else Serial.println(F("Channel inverted = no"));
    wait_sticks_zero();
  }
  if(error == 0){
    Serial.println(F(""));
    Serial.println(F("Gently move all the sticks simultaneously to their extends"));
//Register the minimum and maximum values of the receiver channels
    register_min_max();
    Serial.println(F(""));
    Serial.print(F("Digital input 08 values:"));
    Serial.print(low_channel_1);
    Serial.print(F(" to "));
    Serial.print(center_channel_1);
    Serial.print(F(" to "));
    Serial.println(high_channel_1);
```

```
  Serial.print(F("Digital input 09 values:"));
  Serial.print(low_channel_2);
  Serial.print(F(" to "));
  Serial.print(center_channel_2);
  Serial.print(F(" to "));
  Serial.println(high_channel_2);
  Serial.print(F("Digital input 10 values:"));
  Serial.print(low_channel_3);
  Serial.print(F(" to "));
  Serial.print(center_channel_3);
  Serial.print(F(" to "));
  Serial.println(high_channel_3);
  Serial.print(F("Digital input 11 values:"));
  Serial.print(low_channel_4);
  Serial.print(F(" to "));
  Serial.print(center_channel_4);
  Serial.print(F(" to "));
  Serial.println(high_channel_4);
  Serial.println(F("Move stick 'nose up' and back to center to continue"));
  check_to_continue();
 }



 if(error == 0){
//Communication with gyro
  Serial.println(F(""));
  Serial.println(F("================================================="));
  Serial.println(F("Gyro search"));
  Serial.println(F("================================================="));
  delay(2000);


  if(type == 0){
    Serial.println(F("Searching for L3GD20H on address 0x6A/106"));
    delay(1000);
    if(search_gyro(0x6A, 0x0F) == 0xD7){
      Serial.println(F("L3GD20H found on address 0x6A"));
      type = 1;
      gyro_address = 0x6A;
    }
  }

  if(type == 0){
   Serial.println(F("Searching for L3GD20H on address 0x6B/107"));
    delay(1000);
    if(search_gyro(0x6B, 0x0F) == 0xD7){
      Serial.println(F("L3GD20H found on address 0x6B"));
      type = 1;
      gyro_address = 0x6B;
    }
  }
```

```
    if(type == 0){
      Serial.println(F("No gyro device found!"));
      error = 1;
    }

    else{
      delay(3000);
      Serial.println(F(""));
      Serial.println(F("=================================================="));
      Serial.println(F("Gyro register settings"));
      Serial.println(F("=================================================="));
      start_gyro();      //Setup the gyro for further use
    }
  }

//If the gyro is found we can setup the correct gyro axes.
  if(error == 0){
    delay(3000);
    Serial.println(F(""));
    Serial.println(F("=================================================="));
    Serial.println(F("Gyro calibration"));
    Serial.println(F("=================================================="));
    Serial.println(F("Don't move the quadcopter!! Calibration starts in 3 seconds"));
    delay(3000);
    Serial.println(F("Calibrating the gyro, this will take +/- 8 seconds"));
    Serial.print(F("Please wait"));
//Take multiple gyro data samples so we can determine the average gyro offset (calibration).
    for (cal_int = 0; cal_int < 2000 ; cal_int ++){              //Take 2000 readings for calibration.
      if(cal_int % 100 == 0)Serial.print(F("."));                //Print dot to indicate calibration.
      gyro_signalen();                                           //Read the gyro output.
      gyro_roll_cal += gyro_roll;                                //Add roll value to gyro_roll_cal.
      gyro_pitch_cal += gyro_pitch;                              //Add pitch value to gyro_pitch_cal.
      gyro_yaw_cal += gyro_yaw;                                  //Add yaw value to gyro_yaw_cal.
      delay(4);                                                  //Wait 3 milliseconds before the next loop.
    }
//We have 2000 measures. We need to devide by 2000 to get the average gyro offset.
    gyro_roll_cal /= 2000;                                       //Divide the roll total by 2000.
    gyro_pitch_cal /= 2000;                                      //Divide the pitch total by 2000.
    gyro_yaw_cal /= 2000;                                        //Divide the yaw total by 2000.

//Show the calibration results
    Serial.println(F(""));
    Serial.print(F("Axis 1 offset="));
    Serial.println(gyro_roll_cal);
    Serial.print(F("Axis 2 offset="));
    Serial.println(gyro_pitch_cal);
    Serial.print(F("Axis 3 offset="));
    Serial.println(gyro_yaw_cal);
    Serial.println(F(""));

    Serial.println(F("=================================================="));
    Serial.println(F("Gyro axes configuration"));
    Serial.println(F("=================================================="));
```

```
//Detect the left wing up movement
   Serial.println(F("Lift the left side of the quadcopter to a 45 degree angle within 10 seconds"));
//Check axis movement
   check_gyro_axes(1);
   if(error == 0){
     Serial.println(F("OK!"));
     Serial.print(F("Angle detection = "));
     Serial.println(roll_axis & 0b00000011);
     if(roll_axis & 0b10000000)Serial.println(F("Axis inverted = yes"));
     else Serial.println(F("Axis inverted = no"));
     Serial.println(F("Put the quadcopter back in its original position"));
     Serial.println(F("Move stick 'nose up' and back to center to continue"));
     check_to_continue();

//Detect the nose up movement
     Serial.println(F(""));
     Serial.println(F(""));
     Serial.println(F("Lift the nose of the quadcopter to a 45 degree angle within 10 seconds"));
//Check axis movement
     check_gyro_axes(2);
   }
   if(error == 0){
     Serial.println(F("OK!"));
     Serial.print(F("Angle detection = "));
     Serial.println(pitch_axis & 0b00000011);
     if(pitch_axis & 0b10000000)Serial.println(F("Axis inverted = yes"));
     else Serial.println(F("Axis inverted = no"));
     Serial.println(F("Put the quadcopter back in its original position"));
     Serial.println(F("Move stick 'nose up' and back to center to continue"));
     check_to_continue();

//Detect the nose right movement
     Serial.println(F(""));
     Serial.println(F(""));
     Serial.println(F("Rotate the nose of the quadcopter 45 degree to the right within 10 seconds"));
//Check axis movement
     check_gyro_axes(3);
   }
   if(error == 0){
     Serial.println(F("OK!"));
     Serial.print(F("Angle detection = "));
     Serial.println(yaw_axis & 0b00000011);
     if(yaw_axis & 0b10000000)Serial.println(F("Axis inverted = yes"));
     else Serial.println(F("Axis inverted = no"));
     Serial.println(F("Put the quadcopter back in its original position"));
     Serial.println(F("Move stick 'nose up' and back to center to continue"));
     check_to_continue();
   }
 }
 if(error == 0){
   Serial.println(F(""));
   Serial.println(F("================================================="));
```

```
    Serial.println(F("LED test"));
    Serial.println(F("==================================================="));
    digitalWrite(12, HIGH);
    Serial.println(F("The LED should now be lit"));
    Serial.println(F("Move stick 'nose up' and back to center to continue"));
    check_to_continue();
    digitalWrite(12, LOW);
  }

  Serial.println(F(""));

  if(error == 0){
    Serial.println(F("==================================================="));
    Serial.println(F("Final setup check"));
    Serial.println(F("==================================================="));
    delay(1000);
    if(receiver_check_byte == 0b00001111){
      Serial.println(F("Receiver channels ok"));
    }
    else{
      Serial.println(F("Receiver channel verification failed!"));
      error = 1;
    }
    delay(1000);
    if(gyro_check_byte == 0b00000111){
      Serial.println(F("Gyro axes ok"));
    }
    else{
      Serial.println(F("Gyro exes verification failed!"));
      error = 1;
    }
  }

if(error == 0){       //If all is good, store the information in the EEPROM

    Serial.println(F(""));
    Serial.println(F("==================================================="));
    Serial.println(F("Storing EEPROM information"));
    Serial.println(F("==================================================="));
    Serial.println(F("Writing EEPROM"));
    delay(1000);
    Serial.println(F("Done!"));
    EEPROM.write(0, center_channel_1 & 0b11111111);
    EEPROM.write(1, center_channel_1 >> 8);
    EEPROM.write(2, center_channel_2 & 0b11111111);
    EEPROM.write(3, center_channel_2 >> 8);
    EEPROM.write(4, center_channel_3 & 0b11111111);
    EEPROM.write(5, center_channel_3 >> 8);
    EEPROM.write(6, center_channel_4 & 0b11111111);
    EEPROM.write(7, center_channel_4 >> 8);
    EEPROM.write(8, high_channel_1 & 0b11111111);
    EEPROM.write(9, high_channel_1 >> 8);
    EEPROM.write(10, high_channel_2 & 0b11111111);
```

```
    EEPROM.write(11, high_channel_2 >> 8);
    EEPROM.write(12, high_channel_3 & 0b11111111);
    EEPROM.write(13, high_channel_3 >> 8);
    EEPROM.write(14, high_channel_4 & 0b11111111);
    EEPROM.write(15, high_channel_4 >> 8);
    EEPROM.write(16, low_channel_1 & 0b11111111);
    EEPROM.write(17, low_channel_1 >> 8);
    EEPROM.write(18, low_channel_2 & 0b11111111);
    EEPROM.write(19, low_channel_2 >> 8);
    EEPROM.write(20, low_channel_3 & 0b11111111);
    EEPROM.write(21, low_channel_3 >> 8);
    EEPROM.write(22, low_channel_4 & 0b11111111);
    EEPROM.write(23, low_channel_4 >> 8);
    EEPROM.write(24, channel_1_assign);
    EEPROM.write(25, channel_2_assign);
    EEPROM.write(26, channel_3_assign);
    EEPROM.write(27, channel_4_assign);
    EEPROM.write(28, roll_axis);
    EEPROM.write(29, pitch_axis);
    EEPROM.write(30, yaw_axis);
    EEPROM.write(31, type);
    EEPROM.write(32, gyro_address);
  //Write the EEPROM signature
    EEPROM.write(33, 'L');
    EEPROM.write(34, 'O');
    EEPROM.write(35, 'U');


  //To make sure evrything is ok, verify the EEPROM data.
    Serial.println(F("Verify EEPROM data"));
    delay(1000);
    if(center_channel_1 != ((EEPROM.read(1) << 8) | EEPROM.read(0)))error = 1;
    if(center_channel_2 != ((EEPROM.read(3) << 8) | EEPROM.read(2)))error = 1;
    if(center_channel_3 != ((EEPROM.read(5) << 8) | EEPROM.read(4)))error = 1;
    if(center_channel_4 != ((EEPROM.read(7) << 8) | EEPROM.read(6)))error = 1;

    if(high_channel_1 != ((EEPROM.read(9) << 8) | EEPROM.read(8)))error = 1;
    if(high_channel_2 != ((EEPROM.read(11) << 8) | EEPROM.read(10)))error = 1;
    if(high_channel_3 != ((EEPROM.read(13) << 8) | EEPROM.read(12)))error = 1;
    if(high_channel_4 != ((EEPROM.read(15) << 8) | EEPROM.read(14)))error = 1;

    if(low_channel_1 != ((EEPROM.read(17) << 8) | EEPROM.read(16)))error = 1;
    if(low_channel_2 != ((EEPROM.read(19) << 8) | EEPROM.read(18)))error = 1;
    if(low_channel_3 != ((EEPROM.read(21) << 8) | EEPROM.read(20)))error = 1;
    if(low_channel_4 != ((EEPROM.read(23) << 8) | EEPROM.read(22)))error = 1;

    if(channel_1_assign != EEPROM.read(24))error = 1;
    if(channel_2_assign != EEPROM.read(25))error = 1;
    if(channel_3_assign != EEPROM.read(26))error = 1;
    if(channel_4_assign != EEPROM.read(27))error = 1;

    if(roll_axis != EEPROM.read(28))error = 1;
    if(pitch_axis != EEPROM.read(29))error = 1;
```

```
    if(yaw_axis != EEPROM.read(30))error = 1;
    if(type != EEPROM.read(31))error = 1;
    if(gyro_address != EEPROM.read(32))error = 1;

    if('L' != EEPROM.read(33))error = 1;
    if('O' != EEPROM.read(34))error = 1;
    if('U' != EEPROM.read(35))error = 1;

    if(error == 1)Serial.println(F("EEPROM verification failed!"));
    else Serial.println(F("Verification done"));
   }


  if(error == 0){
    Serial.println(F("Setup is finished."));
    Serial.println(F("You can now calibrate the esc's by uploading the next code"));
  }
  else{
   Serial.println(F("The setup is aborted due to an error."));
  }
  while(1);
}

//Search for the gyro and check the Who_am_I register
byte search_gyro(int gyro_address, int who_am_i){
  Wire.beginTransmission(gyro_address);
  Wire.write(who_am_i);
  Wire.endTransmission();
  Wire.requestFrom(gyro_address, 1);
  timer = millis() + 100;
  while(Wire.available() < 1 && timer > millis());
  lowByte = Wire.read();
  address = gyro_address;
  return lowByte;
}

void start_gyro(){
  //Setup the L3GD20H
    Wire.beginTransmission(address);     //Start communication with the gyro with the address found during search
    Wire.write(0x20);                    //We want to write to register 1 (20 hex)
    Wire.write(0x0F);                    //Set the register bits as 00001111 (Turn on the gyro and enable all axis)
    Wire.endTransmission();              //End the transmission with the gyro

    Wire.beginTransmission(address);           //Start communication with the gyro (adress 1101001)
    Wire.write(0x20);                          //Start reading @ register 28h and auto increment with every read
    Wire.endTransmission();                    //End the transmission
    Wire.requestFrom(address, 1);              //Request 6 bytes from the gyro
    while(Wire.available() < 1);               //Wait until the 1 byte is received
    Serial.print(F("Register 0x20 is set to:"));
    Serial.println(Wire.read(),BIN);

    Wire.beginTransmission(address);     //Start communication with the gyro  with the address found during search
    Wire.write(0x23);                    //We want to write to register 4 (23 hex)
```

```
  Wire.write(0x90);               //Set the register bits as 10010000 (Block Data Update active & 500dps full scale)
  Wire.endTransmission();       //End the transmission with the gyro

  Wire.beginTransmission(address);              //Start communication with the gyro (adress 1101001)
  Wire.write(0x23);                             //Start reading @ register 28h and auto increment with every read
  Wire.endTransmission();                       //End the transmission
  Wire.requestFrom(address, 1);                 //Request 6 bytes from the gyro
  while(Wire.available() < 1);                  //Wait until the 1 byte is received
  Serial.print(F("Register 0x23 is set to:"));
  Serial.println(Wire.read(),BIN);

}

void gyro_signalen(){
  if (type == 1) {
  Wire.beginTransmission(address);              //Start communication with the gyro
  Wire.write(168);                              //Start reading @ register 28h and auto increment with every read
  Wire.endTransmission();                       //End the transmission
  Wire.requestFrom(address, 6);                 //Request 6 bytes from the gyro
  while(Wire.available() < 6);                  //Wait until the 6 bytes are received
  lowByte = Wire.read();                        //First received byte is the low part of the angular data
  highByte = Wire.read();                       //Second received byte is the high part of the angular data
  gyro_roll = ((highByte<<8)|lowByte);          //Multiply highByte by 256 (shift left by 8) and add lowByte
  if(cal_int == 2000)gyro_roll -= gyro_roll_cal;       //Only compensate after the calibration
  lowByte = Wire.read();                                //First received byte is the low part of the angular data
  highByte = Wire.read();                               //Second received byte is the high part of the angular data
  gyro_pitch = ((highByte<<8)|lowByte);                 //Multiply highByte by 256 (shift left by 8) and add lowByte
  if(cal_int == 2000)gyro_pitch -= gyro_pitch_cal;         //Only compensate after the calibration
  lowByte = Wire.read();                                //First received byte is the low part of the angular data
  highByte = Wire.read();                               //Second received byte is the high part of the angular data
  gyro_yaw = ((highByte<<8)|lowByte);                   //Multiply highByte by 256 (shift left by 8) and ad lowByte
  if(cal_int == 2000)gyro_yaw -= gyro_yaw_cal;             //Only compensate after the calibration
  }
}

//Check if a receiver input value is changing within 30 seconds
void check_receiver_inputs(byte movement){
 byte trigger = 0;
 int pulse_length;
 timer = millis() + 30000;
 while(timer > millis() && trigger == 0){
  delay(250);
  if(receiver_input_channel_1 > 1750 || receiver_input_channel_1 < 1250){
    trigger = 1;
    receiver_check_byte |= 0b00000001;
    pulse_length = receiver_input_channel_1;
  }
  if(receiver_input_channel_2 > 1750 || receiver_input_channel_2 < 1250){
    trigger = 2;
    receiver_check_byte |= 0b00000010;
    pulse_length = receiver_input_channel_2;
  }
  if(receiver_input_channel_3 > 1750 || receiver_input_channel_3 < 1250){
```

```
      trigger = 3;
      receiver_check_byte |= 0b00000100;
      pulse_length = receiver_input_channel_3;
     }
     if(receiver_input_channel_4 > 1750 || receiver_input_channel_4 < 1250){
      trigger = 4;
      receiver_check_byte |= 0b00001000;
      pulse_length = receiver_input_channel_4;
     }
   }
   if(trigger == 0){
     error = 1;
     Serial.println(F("No stick movement detected in the last 30 seconds!"));
   }
//Assign the stick to the function.
   else{
     if(movement == 1){
       channel_3_assign = trigger;
       if(pulse_length < 1250)channel_3_assign += 0b10000000;
     }
     if(movement == 2){
       channel_1_assign = trigger;
       if(pulse_length < 1250)channel_1_assign += 0b10000000;
     }
     if(movement == 3){
       channel_2_assign = trigger;
       if(pulse_length < 1250)channel_2_assign += 0b10000000;
     }
     if(movement == 4){
       channel_4_assign = trigger;
       if(pulse_length < 1250)channel_4_assign += 0b10000000;
     }
   }
}

void check_to_continue(){
   byte continue_byte = 0;
   while(continue_byte == 0){
     if(channel_2_assign == 0b00000001 && receiver_input_channel_1 > center_channel_1 + 150)continue_byte
= 1;
     if(channel_2_assign == 0b10000001 && receiver_input_channel_1 < center_channel_1 - 150)continue_byte =
1;
     if(channel_2_assign == 0b00000010 && receiver_input_channel_2 > center_channel_2 + 150)continue_byte
= 1;
     if(channel_2_assign == 0b10000010 && receiver_input_channel_2 < center_channel_2 - 150)continue_byte =
1;
     if(channel_2_assign == 0b00000011 && receiver_input_channel_3 > center_channel_3 + 150)continue_byte
= 1;
     if(channel_2_assign == 0b10000011 && receiver_input_channel_3 < center_channel_3 - 150)continue_byte =
1;
     if(channel_2_assign == 0b00000100 && receiver_input_channel_4 > center_channel_4 + 150)continue_byte
= 1;
```

```
    if(channel_2_assign == 0b10000100 && receiver_input_channel_4 < center_channel_4 - 150)continue_byte =
1;
    delay(100);
  }
  wait_sticks_zero();
}

//Check if the transmitter sticks are in the neutral position
void wait_sticks_zero(){
  byte zero = 0;
  while(zero < 15){
    if(receiver_input_channel_1 < center_channel_1 + 20 && receiver_input_channel_1 > center_channel_1 -
20)zero |= 0b00000001;
    if(receiver_input_channel_2 < center_channel_2 + 20 && receiver_input_channel_2 > center_channel_2 -
20)zero |= 0b00000010;
    if(receiver_input_channel_3 < center_channel_3 + 20 && receiver_input_channel_3 > center_channel_3 -
20)zero |= 0b00000100;
    if(receiver_input_channel_4 < center_channel_4 + 20 && receiver_input_channel_4 > center_channel_4 -
20)zero |= 0b00001000;
    delay(100);
  }
}

//Checck if the receiver values are valid within 10 seconds
void wait_for_receiver(){
  byte zero = 0;
  timer = millis() + 10000;
  while(timer > millis() && zero < 15){
    if(receiver_input_channel_1 < 2100 && receiver_input_channel_1 > 900)zero |= 0b00000001;
    if(receiver_input_channel_2 < 2100 && receiver_input_channel_2 > 900)zero |= 0b00000010;
    if(receiver_input_channel_3 < 2100 && receiver_input_channel_3 > 900)zero |= 0b00000100;
    if(receiver_input_channel_4 < 2100 && receiver_input_channel_4 > 900)zero |= 0b00001000;
    delay(500);
    Serial.print(F("."));
  }
  if(zero == 0){
    error = 1;
    Serial.println(F("."));
    Serial.println(F("No valid receiver signals found!"));
  }
  else Serial.println(F(" OK"));
}

//Register the minimum and maximum receiver values and exit when the sticks are back in the neutral position
void register_min_max(){
  byte zero = 0;
  low_channel_1 = receiver_input_channel_1;
  low_channel_2 = receiver_input_channel_2;
  low_channel_3 = receiver_input_channel_3;
  low_channel_4 = receiver_input_channel_4;
  while(receiver_input_channel_1 < center_channel_1 + 20 && receiver_input_channel_1 > center_channel_1 -
20)delay(250);
  Serial.println(F("Measuring endpoints...."));
```

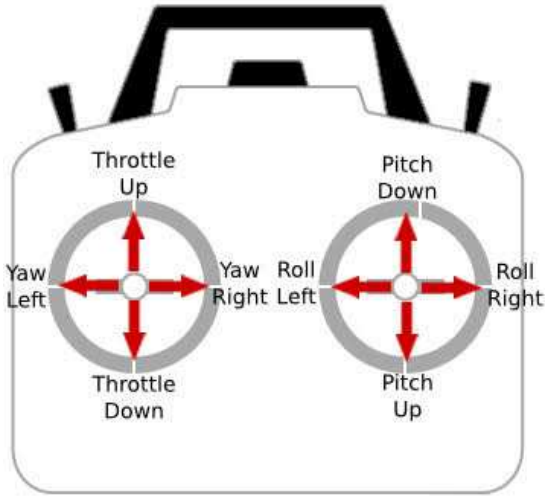
```
  while(zero < 15){
    if(receiver_input_channel_1 < center_channel_1 + 20 && receiver_input_channel_1 > center_channel_1 -
20)zero |= 0b00000001;
    if(receiver_input_channel_2 < center_channel_2 + 20 && receiver_input_channel_2 > center_channel_2 -
20)zero |= 0b00000010;
    if(receiver_input_channel_3 < center_channel_3 + 20 && receiver_input_channel_3 > center_channel_3 -
20)zero |= 0b00000100;
    if(receiver_input_channel_4 < center_channel_4 + 20 && receiver_input_channel_4 > center_channel_4 -
20)zero |= 0b00001000;
    if(receiver_input_channel_1 < low_channel_1)low_channel_1 = receiver_input_channel_1;
    if(receiver_input_channel_2 < low_channel_2)low_channel_2 = receiver_input_channel_2;
    if(receiver_input_channel_3 < low_channel_3)low_channel_3 = receiver_input_channel_3;
    if(receiver_input_channel_4 < low_channel_4)low_channel_4 = receiver_input_channel_4;
    if(receiver_input_channel_1 > high_channel_1)high_channel_1 = receiver_input_channel_1;
    if(receiver_input_channel_2 > high_channel_2)high_channel_2 = receiver_input_channel_2;
    if(receiver_input_channel_3 > high_channel_3)high_channel_3 = receiver_input_channel_3;
    if(receiver_input_channel_4 > high_channel_4)high_channel_4 = receiver_input_channel_4;
    delay(100);
  }
}

//Check if the angular position of a gyro axis is changing within 10 seconds
void check_gyro_axes(byte movement){
  byte trigger_axis = 0;
  float gyro_angle_roll, gyro_angle_pitch, gyro_angle_yaw;
  //Reset all axes
  gyro_angle_roll = 0;
  gyro_angle_pitch = 0;
  gyro_angle_yaw = 0;
  gyro_signalen();
  timer = millis() + 10000;
  while(timer > millis() && gyro_angle_roll > -30 && gyro_angle_roll < 30 && gyro_angle_pitch > -30 &&
gyro_angle_pitch < 30 && gyro_angle_yaw > -30 && gyro_angle_yaw < 30){
    gyro_signalen();
    if(type == 1){
      gyro_angle_roll += gyro_roll * 0.00007;             //0.00007 = 17.5 (md/s) / 250(Hz)
      gyro_angle_pitch += gyro_pitch * 0.00007;
      gyro_angle_yaw += gyro_yaw * 0.00007;
    }

    delayMicroseconds(3700); //Loop is running @ 250Hz. +/-300us is used for communication with the gyro
  }
//Assign the moved axis to the orresponding function (pitch, roll, yaw)
  if((gyro_angle_roll < -30 || gyro_angle_roll > 30) && gyro_angle_pitch > -30 && gyro_angle_pitch < 30 &&
gyro_angle_yaw > -30 && gyro_angle_yaw < 30){
    gyro_check_byte |= 0b00000001;
    if(gyro_angle_roll < 0)trigger_axis = 0b10000001;
    else trigger_axis = 0b00000001;
  }
  if((gyro_angle_pitch < -30 || gyro_angle_pitch > 30) && gyro_angle_roll > -30 && gyro_angle_roll < 30 &&
gyro_angle_yaw > -30 && gyro_angle_yaw < 30){
    gyro_check_byte |= 0b00000010;
    if(gyro_angle_pitch < 0)trigger_axis = 0b10000010;
```

```
    else trigger_axis = 0b00000010;
  }
  if((gyro_angle_yaw < -30 || gyro_angle_yaw > 30) && gyro_angle_roll > -30 && gyro_angle_roll < 30 &&
gyro_angle_pitch > -30 && gyro_angle_pitch < 30){
    gyro_check_byte |= 0b00000100;
    if(gyro_angle_yaw < 0)trigger_axis = 0b10000011;
    else trigger_axis = 0b00000011;
  }

  if(trigger_axis == 0){
    error = 1;
    Serial.println(F("No angular motion is detected in the last 10 seconds!!! (ERROR 4)"));
  }
  else
  if(movement == 1)roll_axis = trigger_axis;
  if(movement == 2)pitch_axis = trigger_axis;
  if(movement == 3)yaw_axis = trigger_axis;

}

//This routine is called every time input 8, 9, 10 or 11 changed state
ISR(PCINT0_vect){
  current_time = micros();
//Channel 1=========================================
  if(PINB & B00000001){                                 //Is input 8 high?
    if(last_channel_1 == 0){                            //Input 8 changed from 0 to 1
      last_channel_1 = 1;                               //Remember current input state
      timer_1 = current_time;                           //Set timer_1 to current_time
    }
  }
  else if(last_channel_1 == 1){                         //Input 8 is not high and changed from 1 to 0
    last_channel_1 = 0;                                 //Remember current input state
    receiver_input_channel_1 = current_time - timer_1;        //Channel 1 is current_time - timer_1
  }
//Channel 2=========================================
  if(PINB & B00000010 ){                                //Is input 9 high?
    if(last_channel_2 == 0){                            //Input 9 changed from 0 to 1
      last_channel_2 = 1;                               //Remember current input state
      timer_2 = current_time;                           //Set timer_2 to current_time
    }
  }
  else if(last_channel_2 == 1){                         //Input 9 is not high and changed from 1 to 0
    last_channel_2 = 0;                                 //Remember current input state
    receiver_input_channel_2 = current_time - timer_2;        //Channel 2 is current_time - timer_2
  }
//Channel 3=========================================
  if(PINB & B00000100 ){                                //Is input 10 high?
    if(last_channel_3 == 0){                            //Input 10 changed from 0 to 1
      last_channel_3 = 1;                               //Remember current input state
      timer_3 = current_time;                           //Set timer_3 to current_time
    }
  }
  else if(last_channel_3 == 1){                         //Input 10 is not high and changed from 1 to 0
```

```
    last_channel_3 = 0;                                 //Remember current input state
    receiver_input_channel_3 = current_time - timer_3;        //Channel 3 is current_time - timer_3

  }
//Channel 4=======================================
  if(PINB & B00001000 ){                              //Is input 11 high?
    if(last_channel_4 == 0){                          //Input 11 changed from 0 to 1
      last_channel_4 = 1;                             //Remember current input state
      timer_4 = current_time;                         //Set timer_4 to current_time
    }
  }
  else if(last_channel_4 == 1){                       //Input 11 is not high and changed from 1 to 0
    last_channel_4 = 0;                               //Remember current input state
    receiver_input_channel_4 = current_time - timer_4;        //Channel 4 is current_time - timer_4
  }
}
```

3.2 Πρόγραμμα ESC_CALIBRATION

Το πρόγραμμα ESC_CALIBRATION είναι απαραίτητο για την ταυτόχρονη λειτουργία και των τεσσάρων ελίκων μας. Αν αυτό δεν το πρόγραμμα δεν μεταφορτωθεί στο Arduino, τα μοτέρ μας θα περιστρέφονται σε διαφορετικές ταχύτητες. Όταν τελειώσει το upload του συγκεκριμένου προγράμματος μπορούμε να βγάλουμε το καλώδιο USB και να τοποθετήσουμε τους έλικές μας.



Σχήμα 19 : Οι κινήσεις του χειριστηρίου

Όταν γίνει αυτό τοποθετούμε το THROTTLE του χειριστηρίου στη μέγιστη θέση και συνδέουμε την μπαταρία. Όταν τα ESCs (Electric Speed Controllers) θα αρχίζουν να παράγουν έναν ήχο, τότε θα τοποθετήσουμε το THROTTLE στην ελάχιστη θέση. Μετά από αυτό τα μοτέρ θα ξεκινήσουν την ίδια στιγμή. Το δεύτερο μέρος της εγκατάστασής μας ολοκληρώνεται και αμέσως μετά πρέπει να κάνουμε upload το τρίτο και τελευταίο πρόγραμμα με όνομα FLIGHT_CONTROLLER. Ο κώδικας που χρησιμοποιείται είναι ο εξής:

```
#include <EEPROM.h>          //Include the EEPROM.h library so we can store information onto the EEPROM

//Declaring global variables
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
byte eeprom_data[36];
int receiver_input_channel_1, receiver_input_channel_2, receiver_input_channel_3, receiver_input_channel_4;
int counter_channel_1, counter_channel_2, counter_channel_3, counter_channel_4, start;
int receiver_input[5];
//int temp;
unsigned long timer_channel_1, timer_channel_2, timer_channel_3, timer_channel_4, esc_timer, esc_loop_timer;
```

```
unsigned long zero_timer, timer_1, timer_2, timer_3, timer_4, current_time;

//SETUP ROUTINE
void setup(){
  //Arduino Uno pins default to inputs, so they don't need to be explicitly declared as inputs
  DDRD |= B11110000;                        //Configure digital poort 4, 5, 6 and 7 as output
  DDRB |= B00010000;                        //Configure digital poort 12 as output

  PCICR |= (1 << PCIE0);                    // set PCIE0 to enable PCMSK0 scan
  PCMSK0 |= (1 << PCINT0);                  // set PCINT0 (digital input 8) to trigger an interrupt on state change
  PCMSK0 |= (1 << PCINT1);                  // set PCINT1 (digital input 9)to trigger an interrupt on state change
  PCMSK0 |= (1 << PCINT2);                  // set PCINT2 (digital input 10)to trigger an interrupt on state change
  PCMSK0 |= (1 << PCINT3);                  // set PCINT3 (digital input 11)to trigger an interrupt on state change

  //Read EEPROM for fast access data
  for(start = 0; start <= 35; start++)eeprom_data[start] = EEPROM.read(start);

  //Check the EEPROM signature to make sure that the setup program is executed
  while(eeprom_data[33] != 'L' || eeprom_data[34] != 'O' || eeprom_data[35] != 'U'){
    delay(500);
    digitalWrite(12, !digitalRead(12));           //Change the led status to indicate error.
  }
  wait_for_receiver();                          //Wait until the receiver is active.
  zero_timer = micros();                        //Set the zero_timer for the first loop.
}

//Main program loop
void loop(){
  receiver_input_channel_3 = convert_receiver_channel(3);    //Convert the actual receiver signals for throttle to
the standard 1000 - 2000us

  while(zero_timer + 4000 > micros());                       //Start the pulse after 4000 micro seconds.
  zero_timer = micros();                                     //Reset the zero timer.
  PORTD |= B11110000;                                        //Set port 4, 5, 6 and 7 high at once
  timer_channel_1 = receiver_input_channel_3 + zero_timer;   //Calculate the time when digital port 4 is set low
  timer_channel_2 = receiver_input_channel_3 + zero_timer;   //Calculate the time when digital port 5 is set low
  timer_channel_3 = receiver_input_channel_3 + zero_timer;   //Calculate the time when digital port 6 is set low
  timer_channel_4 = receiver_input_channel_3 + zero_timer;   //Calculate the time when digital port 7 is set low

  while(PORTD >= 16){                           //Execute the loop until digital port 4 to 7 is low
    esc_loop_timer = micros();                  //Check the current time
    if(timer_channel_1 <= esc_loop_timer)PORTD &= B11101111; //When the delay time is expired, digital port
4 is set low
    if(timer_channel_2 <= esc_loop_timer)PORTD &= B11011111; //When the delay time is expired, digital port
5 is set low
    if(timer_channel_3 <= esc_loop_timer)PORTD &= B10111111; //When the delay time is expired, digital port
6 is set low
    if(timer_channel_4 <= esc_loop_timer)PORTD &= B01111111; //When the delay time is expired, digital port
7 is set low
  }

}
```

```
//This routine is called every time input 8, 9, 10 or 11 changed state
ISR(PCINT0_vect){
  current_time = micros();
  //Channel 1=========================================
  if(PINB & B00000001){                                    //Is input 8 high?
    if(last_channel_1 == 0){                               //Input 8 changed from 0 to 1
      last_channel_1 = 1;                                  //Remember current input state
      timer_1 = current_time;                              //Set timer_1 to current_time
    }
  }
  else if(last_channel_1 == 1){                            //Input 8 is not high and changed from 1 to 0
    last_channel_1 = 0;                                    //Remember current input state
    receiver_input[1] = current_time - timer_1;            //Channel 1 is current_time - timer_1
  }
  //Channel 2=========================================
  if(PINB & B00000010 ){                                   //Is input 9 high?
    if(last_channel_2 == 0){                               //Input 9 changed from 0 to 1
      last_channel_2 = 1;                                  //Remember current input state
      timer_2 = current_time;                              //Set timer_2 to current_time
    }
  }
  else if(last_channel_2 == 1){                            //Input 9 is not high and changed from 1 to 0
    last_channel_2 = 0;                                    //Remember current input state
    receiver_input[2] = current_time - timer_2;            //Channel 2 is current_time - timer_2
  }
  //Channel 3=========================================
  if(PINB & B00000100 ){                                   //Is input 10 high?
    if(last_channel_3 == 0){                               //Input 10 changed from 0 to 1
      last_channel_3 = 1;                                  //Remember current input state
      timer_3 = current_time;                              //Set timer_3 to current_time
    }
  }
  else if(last_channel_3 == 1){                            //Input 10 is not high and changed from 1 to 0
    last_channel_3 = 0;                                    //Remember current input state
    receiver_input[3] = current_time - timer_3;            //Channel 3 is current_time - timer_3
  }
  //Channel 4=========================================
  if(PINB & B00001000 ){                                   //Is input 11 high?
    if(last_channel_4 == 0){                               //Input 11 changed from 0 to 1
      last_channel_4 = 1;                                  //Remember current input state
      timer_4 = current_time;                              //Set timer_4 to current_time
    }
  }
  else if(last_channel_4 == 1){                            //Input 11 is not high and changed from 1 to 0
    last_channel_4 = 0;                                    //Remember current input state
    receiver_input[4] = current_time - timer_4;            //Channel 4 is current_time - timer_4
  }
}

//Checck if the receiver values are valid within 10 seconds
void wait_for_receiver(){
  byte zero = 0;                                           //Set all bits in the variable zero to 0
  while(zero < 15){                                        //Stay in this loop until the 4 lowest bits are set
```

```
   if(receiver_input[1] < 2100 && receiver_input[1] > 900)zero |= 0b00000001;  //Set bit 0 if the receiver pulse 1 is within the 900 - 2100 range
   if(receiver_input[2] < 2100 && receiver_input[2] > 900)zero |= 0b00000010;  //Set bit 1 if the receiver pulse 2 is within the 900 - 2100 range
   if(receiver_input[3] < 2100 && receiver_input[3] > 900)zero |= 0b00000100;  //Set bit 2 if the receiver pulse 3 is within the 900 - 2100 range
   if(receiver_input[4] < 2100 && receiver_input[4] > 900)zero |= 0b00001000;  //Set bit 3 if the receiver pulse 4 is within the 900 - 2100 range
   delay(500);                                                  //Wait 500 milliseconds
  }
}

//This part converts the actual receiver signals to a standardized 1000 – 1500 – 2000 microsecond value.
//The stored data in the EEPROM is used.
int convert_receiver_channel(byte function){
  byte channel, reverse;                                       //First we declare some local variables
  int low, center, high, actual;
  int difference;

  channel = eeprom_data[function + 23] & 0b00000111;       //What channel corresponds with the specific function
  if(eeprom_data[function + 23] & 0b10000000)reverse = 1;      //Reverse channel when most significant bit is set
  else reverse = 0;                                            //If the most significant is not set there is no reverse

  actual = receiver_input[channel];                          //Read the actual receiver value for the corresponding function
  low = (eeprom_data[channel * 2 + 15] << 8) | eeprom_data[channel * 2 + 14];  //Store the low value for the specific receiver input channel
  center = (eeprom_data[channel * 2 - 1] << 8) | eeprom_data[channel * 2 - 2]; //Store the center value for the specific receiver input channel
  high = (eeprom_data[channel * 2 + 7] << 8) | eeprom_data[channel * 2 + 6];   //Store the high value for the specific receiver input channel

  if(actual < center){                                         //The actual receiver value is lower than the center value
   if(actual < low)actual = low;                               //Limit the lowest value to the value that was detected during setup
   difference = ((long)(center - actual) * (long)500) / (center - low);      //Calculate and scale the actual value to a 1000 - 2000us value
   if(reverse == 1)return 1500 + difference;                    //If the channel is reversed
   else return 1500 - difference;                               //If the channel is not reversed
  }
  else if(actual > center){                                    //The actual receiver value is higher than the center value
   if(actual > high)actual = high;                             //Limit the lowest value to the value that was detected during setup
   difference = ((long)(actual - center) * (long)500) / (high - center);     //Calculate and scale the actual value to a 1000 - 2000us value
   if(reverse == 1)return 1500 - difference;                    //If the channel is reversed
   else return 1500 + difference;                               //If the channel is not reversed
  }
  else return 1500;
}
```

3.3 Πρόγραμμα FLIGHT_CONTROLLER

Για την εγκατάσταση αυτού του προγράμματος, απλά χρειάζεται να γίνει upload το πρόγραμμα FLIGHT_CONTROLLER στη πλακέτα Arduino, αφού αφαιρέσουμε πρώτα την μπαταρία μας. Μετά από αυτό μπορούμε να ξεκινήσουμε την πρώτη μας πτήση σε χαμηλό ύψος,για λόγους ασφαλείας, για να δούμε αν το τετρακόπτερο ανταποκρίνεται στις εντολές που του δίνουμε. Ο κώδικας είναι ο εξής:

```
#include <Wire.h>                    //Include the Wire.h library so we can communicate with the gyro.
#include <EEPROM.h>                  //Include the EEPROM.h library so we can store information onto the EEPROM

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//PID gain and limit settings
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
float pid_p_gain_roll = 1.2;              //Gain setting for the roll P-controller (1.2)
float pid_i_gain_roll = 0.05;             //Gain setting for the roll I-controller (0.05)
float pid_d_gain_roll = 15;               //Gain setting for the roll D-controller (15)
int pid_max_roll = 400;                   //Maximum output of the PID-controller (+/-)

float pid_p_gain_pitch = pid_p_gain_roll;  //Gain setting for the pitch P-controller.
float pid_i_gain_pitch = pid_i_gain_roll;  //Gain setting for the pitch I-controller.
float pid_d_gain_pitch = pid_d_gain_roll;  //Gain setting for the pitch D-controller.
int pid_max_pitch = pid_max_roll;          //Maximum output of the PID-controller (+/-)

float pid_p_gain_yaw = 4.0;               //Gain setting for the pitch P-controller. //4.0
float pid_i_gain_yaw = 0.02;              //Gain setting for the pitch I-controller. //0.02
float pid_d_gain_yaw = 0.0;               //Gain setting for the pitch D-controller.
int pid_max_yaw = 400;                    //Maximum output of the PID-controller (+/-)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Declaring global variables
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
byte eeprom_data[36];
byte highByte, lowByte;
int receiver_input_channel_1, receiver_input_channel_2, receiver_input_channel_3, receiver_input_channel_4;
int counter_channel_1, counter_channel_2, counter_channel_3, counter_channel_4, loop_counter;
int esc_1, esc_2, esc_3, esc_4;
int throttle, battery_voltage;
int cal_int, start, gyro_address;
int receiver_input[5];
unsigned long timer_channel_1, timer_channel_2, timer_channel_3, timer_channel_4, esc_timer, esc_loop_timer;
unsigned long timer_1, timer_2, timer_3, timer_4, current_time;
unsigned long loop_timer;
```

```
double gyro_pitch, gyro_roll, gyro_yaw;
double gyro_axis[4], gyro_axis_cal[4];
float pid_error_temp;
float pid_i_mem_roll, pid_roll_setpoint, gyro_roll_input, pid_output_roll, pid_last_roll_d_error;
float pid_i_mem_pitch, pid_pitch_setpoint, gyro_pitch_input, pid_output_pitch, pid_last_pitch_d_error;
float pid_i_mem_yaw, pid_yaw_setpoint, gyro_yaw_input, pid_output_yaw, pid_last_yaw_d_error;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Setup routine
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void setup(){
  //Serial.begin(57600);
  //Read EEPROM for fast access data.
  for(start = 0; start <= 35; start++)eeprom_data[start] = EEPROM.read(start);
  gyro_address = eeprom_data[32];                     //Store the gyro address in the variable.

  Wire.begin();                                       //Start the I2C as master.

  //Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs.
  DDRD |= B11110000;                                  //Configure digital poort 4, 5, 6 and 7 as output.
  DDRB |= B00110000;                                  //Configure digital poort 12 and 13 as output.

  //Use the led on the Arduino for startup indication.
  digitalWrite(12,HIGH);                              //Turn on the warning led.

  //Check the EEPROM signature to make sure that the setup program is executed
  while(eeprom_data[33] != 'L' || eeprom_data[34] != 'O' || eeprom_data[35] != 'U')delay(10);

  set_gyro_registers();                               //Set the specific gyro registers.

  for (cal_int = 0; cal_int < 1250 ; cal_int ++){     //Wait 5 seconds before continuing.
    PORTD |= B11110000;                               //Set digital poort 4, 5, 6 and 7 high.
    delayMicroseconds(1000);                          //Wait 1000us.
    PORTD &= B00001111;                               //Set digital poort 4, 5, 6 and 7 low.
    delayMicroseconds(3000);                          //Wait 3000us.
  }

  //Let's take multiple gyro data samples so we can determine the average gyro offset (calibration).
  for (cal_int = 0; cal_int < 2000 ; cal_int ++){     //Take 2000 readings for calibration.
    if(cal_int % 15 == 0)digitalWrite(12, !digitalRead(12));  //Change the led status to indicate calibration.
    gyro_signalen();                                  //Read the gyro output.
    gyro_axis_cal[1] += gyro_axis[1];                 //Add roll value to gyro_roll_cal.
    gyro_axis_cal[2] += gyro_axis[2];                 //Add pitch value to gyro_pitch_cal.
    gyro_axis_cal[3] += gyro_axis[3];                 //Add yaw value to gyro_yaw_cal.
    //We don't want the esc's to be beeping annoyingly. So let's give them a 1000us puls while calibrating the gyro.
    PORTD |= B11110000;                               //Set digital poort 4, 5, 6 and 7 high.
    delayMicroseconds(1000);                          //Wait 1000us.
    PORTD &= B00001111;                               //Set digital poort 4, 5, 6 and 7 low.
    delay(3);                                         //Wait 3 milliseconds before the next loop.
  }
  //Now that we have 2000 measures, we need to devide by 2000 to get the average gyro offset.
  gyro_axis_cal[1] /= 2000;                           //Divide the roll total by 2000.
  gyro_axis_cal[2] /= 2000;                           //Divide the pitch total by 2000.
```

```
  gyro_axis_cal[3] /= 2000;                          //Divide the yaw total by 2000.

  PCICR |= (1 << PCIE0);                             //Set PCIE0 to enable PCMSK0 scan.
  PCMSK0 |= (1 << PCINT0);                           //Set PCINT0 (digital input 8) to trigger an interrupt on state change.
  PCMSK0 |= (1 << PCINT1);                           //Set PCINT1 (digital input 9)to trigger an interrupt on state change.
  PCMSK0 |= (1 << PCINT2);                           //Set PCINT2 (digital input 10)to trigger an interrupt on state change.
  PCMSK0 |= (1 << PCINT3);                           //Set PCINT3 (digital input 11)to trigger an interrupt on state change.

  //Wait until the receiver is active and the throtle is set to the lower position.
  while(receiver_input_channel_3 < 990 || receiver_input_channel_3 > 1020 || receiver_input_channel_4 < 1400){
    receiver_input_channel_3 = convert_receiver_channel(3);    //Convert the actual receiver signals for throttle to the standard 1000 - 2000us
    receiver_input_channel_4 = convert_receiver_channel(4);    //Convert the actual receiver signals for yaw to the standard 1000 - 2000us
    start ++;                                                  //While waiting increment start whith every loop.
    //We don't want the esc's to be beeping annoyingly. So let's give them a 1000us puls while waiting for the receiver inputs.
    PORTD |= B11110000;                                        //Set digital poort 4, 5, 6 and 7 high.
    delayMicroseconds(1000);                                   //Wait 1000us.
    PORTD &= B00001111;                                        //Set digital poort 4, 5, 6 and 7 low.
    delay(3);                                                  //Wait 3 milliseconds before the next loop.
    if(start == 125){                                          //Every 125 loops (500ms).
      digitalWrite(12, !digitalRead(12));                      //Change the led status.
      start = 0;                                               //Start again at 0.
    }
  }
  start = 0;                                                   //Set start back to 0.

  //Load the battery voltage to the battery_voltage variable.
  //65 is the voltage compensation for the diode.
  //12.6V equals ~5V @ Analog 0.
  //12.6V equals 1023 analogRead(0).
  //1260 / 1023 = 1.2317.
  //The variable battery_voltage holds 1050 if the battery voltage is 10.5V.
  battery_voltage = (analogRead(0) + 65) * 1.2317;

  //When everything is done, turn off the led.
  digitalWrite(12,LOW);                                        //Turn off the warning led.
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Main program loop
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void loop(){

  receiver_input_channel_1 = convert_receiver_channel(1);    //Convert the actual receiver signals for pitch to the standard 1000 - 2000us.
  receiver_input_channel_2 = convert_receiver_channel(2);    //Convert the actual receiver signals for roll to the standard 1000 - 2000us.
  receiver_input_channel_3 = convert_receiver_channel(3);    //Convert the actual receiver signals for throttle to the standard 1000 - 2000us.
  receiver_input_channel_4 = convert_receiver_channel(4);    //Convert the actual receiver signals for yaw to the standard 1000 - 2000us.
```

```
//Let's get the current gyro data and scale it to degrees per second for the pid calculations.
gyro_signalen();

gyro_roll_input = (gyro_roll_input * 0.8) + ((gyro_roll / 57.14286) * 0.2);          //Gyro pid input is deg/sec.
gyro_pitch_input = (gyro_pitch_input * 0.8) + ((gyro_pitch / 57.14286) * 0.2);       //Gyro pid input is deg/sec.
gyro_yaw_input = (gyro_yaw_input * 0.8) + ((gyro_yaw / 57.14286) * 0.2);             //Gyro pid input is deg/sec.

//For starting the motors: throttle low and yaw left (step 1).
if(receiver_input_channel_3 < 1050 && receiver_input_channel_4 < 1050)start = 1;
//When yaw stick is back in the center position start the motors (step 2).
if(start == 1 && receiver_input_channel_3 < 1050 && receiver_input_channel_4 > 1450){
  start = 2;
  //Reset the pid controllers for a bumpless start.
  pid_i_mem_roll = 0;
  pid_last_roll_d_error = 0;
  pid_i_mem_pitch = 0;
  pid_last_pitch_d_error = 0;
  pid_i_mem_yaw = 0;
  pid_last_yaw_d_error = 0;
}
//Stopping the motors: throttle low and yaw right.
if(start == 2 && receiver_input_channel_3 < 1050 && receiver_input_channel_4 > 1950)start = 0;

//The PID set point in degrees per second is determined by the roll receiver input.
//In the case of deviding by 3 the max roll rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
pid_roll_setpoint = 0;
//We need a little dead band of 16us for better results.
if(receiver_input_channel_1 > 1508)pid_roll_setpoint = (receiver_input_channel_1 - 1508)/3.0;
else if(receiver_input_channel_1 < 1492)pid_roll_setpoint = (receiver_input_channel_1 - 1492)/3.0;

//The PID set point in degrees per second is determined by the pitch receiver input.
//In the case of deviding by 3 the max pitch rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
pid_pitch_setpoint = 0;
//We need a little dead band of 16us for better results.
if(receiver_input_channel_2 > 1508)pid_pitch_setpoint = (receiver_input_channel_2 - 1508)/3.0;
else if(receiver_input_channel_2 < 1492)pid_pitch_setpoint = (receiver_input_channel_2 - 1492)/3.0;

//The PID set point in degrees per second is determined by the yaw receiver input.
//In the case of deviding by 3 the max yaw rate is aprox 164 degrees per second ( (500-8)/3 = 164d/s ).
pid_yaw_setpoint = 0;
//We need a little dead band of 16us for better results.
if(receiver_input_channel_3 > 1050){ //Do not yaw when turning off the motors.
  if(receiver_input_channel_4 > 1508)pid_yaw_setpoint = (receiver_input_channel_4 - 1508)/3.0;
  else if(receiver_input_channel_4 < 1492)pid_yaw_setpoint = (receiver_input_channel_4 - 1492)/3.0;
}
//PID inputs are known. So we can calculate the pid output.
calculate_pid();

//The battery voltage is needed for compensation.
//A complementary filter is used to reduce noise.
//0.09853 = 0.08 * 1.2317.
battery_voltage = battery_voltage * 0.92 + (analogRead(0) + 65) * 0.09853;
```

```
  //Turn on the led if battery voltage is to low.
  if(battery_voltage < 1030 && battery_voltage > 600)digitalWrite(12, HIGH);

  throttle = receiver_input_channel_3;                                  //We need the throttle signal as a base signal.

  if (start == 2){                                                      //The motors are started.
    if (throttle > 1800) throttle = 1800;                               //We need some room to keep full control at full throttle.
    esc_1 = throttle - pid_output_pitch + pid_output_roll - pid_output_yaw; //Calculate the pulse for esc 1 (front-right - CCW)
    esc_2 = throttle + pid_output_pitch + pid_output_roll + pid_output_yaw; //Calculate the pulse for esc 2 (rear-right - CW)
    esc_3 = throttle + pid_output_pitch - pid_output_roll - pid_output_yaw; //Calculate the pulse for esc 3 (rear-left - CCW)
    esc_4 = throttle - pid_output_pitch - pid_output_roll + pid_output_yaw; //Calculate the pulse for esc 4 (front-left - CW)

    if (battery_voltage < 1240 && battery_voltage > 800){               //Is the battery connected?
      esc_1 += esc_1 * ((1240 - battery_voltage)/(float)3500);          //Compensate the esc-1 pulse for voltage drop.
      esc_2 += esc_2 * ((1240 - battery_voltage)/(float)3500);          //Compensate the esc-2 pulse for voltage drop.
      esc_3 += esc_3 * ((1240 - battery_voltage)/(float)3500);          //Compensate the esc-3 pulse for voltage drop.
      esc_4 += esc_4 * ((1240 - battery_voltage)/(float)3500);          //Compensate the esc-4 pulse for voltage drop.
    }

    if (esc_1 < 1200) esc_1 = 1200;                                     //Keep the motors running.
    if (esc_2 < 1200) esc_2 = 1200;                                     //Keep the motors running.
    if (esc_3 < 1200) esc_3 = 1200;                                     //Keep the motors running.
    if (esc_4 < 1200) esc_4 = 1200;                                     //Keep the motors running.

    if(esc_1 > 2000)esc_1 = 2000;                                       //Limit the esc-1 pulse to 2000us.
    if(esc_2 > 2000)esc_2 = 2000;                                       //Limit the esc-2 pulse to 2000us.
    if(esc_3 > 2000)esc_3 = 2000;                                       //Limit the esc-3 pulse to 2000us.
    if(esc_4 > 2000)esc_4 = 2000;                                       //Limit the esc-4 pulse to 2000us.
  }

  else{
    esc_1 = 1000;                                                       //If start is not 2 keep a 1000us pulse for ess-1.
    esc_2 = 1000;                                                       //If start is not 2 keep a 1000us pulse for ess-2.
    esc_3 = 1000;                                                       //If start is not 2 keep a 1000us pulse for ess-3.
    esc_4 = 1000;                                                       //If start is not 2 keep a 1000us pulse for ess-4.
  }

  //All the information for controlling the motor's is available.
  //The refresh rate is 250Hz. That means the esc's need there pulse every 4ms.
  while(micros() - loop_timer < 4000);                                  //We wait until 4000us are passed.
  loop_timer = micros();                                                //Set the timer for the next loop.

  PORTD |= B11110000;                                                   //Set digital outputs 4,5,6 and 7 high.
  timer_channel_1 = esc_1 + loop_timer;                                 //Calculate the time of the faling edge of the esc-1 pulse.
  timer_channel_2 = esc_2 + loop_timer;                                 //Calculate the time of the faling edge of the esc-2 pulse.
  timer_channel_3 = esc_3 + loop_timer;                                 //Calculate the time of the faling edge of the esc-3 pulse.
  timer_channel_4 = esc_4 + loop_timer;                                 //Calculate the time of the faling edge of the esc-4 pulse.

  while(PORTD >= 16){                                                   //Stay in this loop until output 4,5,6 and 7 are low.
```

```
   esc_loop_timer = micros();                                  //Read the current time.
   if(timer_channel_1 <= esc_loop_timer)PORTD &= B11101111;        //Set digital output 4 to low if the time is
expired.
   if(timer_channel_2 <= esc_loop_timer)PORTD &= B11011111;        //Set digital output 5 to low if the time is
expired.
   if(timer_channel_3 <= esc_loop_timer)PORTD &= B10111111;        //Set digital output 6 to low if the time is
expired.
   if(timer_channel_4 <= esc_loop_timer)PORTD &= B01111111;        //Set digital output 7 to low if the time is
expired.
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//This routine is called every time input 8, 9, 10 or 11 changed state
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
ISR(PCINT0_vect){
  current_time = micros();
  //Channel 1=========================================
  if(PINB & B00000001){                                       //Is input 8 high?
    if(last_channel_1 == 0){                                  //Input 8 changed from 0 to 1
      last_channel_1 = 1;                                     //Remember current input state
      timer_1 = current_time;                                 //Set timer_1 to current_time
    }
  }
  else if(last_channel_1 == 1){                               //Input 8 is not high and changed from 1 to 0
    last_channel_1 = 0;                                       //Remember current input state
    receiver_input[1] = current_time - timer_1;               //Channel 1 is current_time - timer_1
  }
  //Channel 2=========================================
  if(PINB & B00000010 ){                                      //Is input 9 high?
    if(last_channel_2 == 0){                                  //Input 9 changed from 0 to 1
      last_channel_2 = 1;                                     //Remember current input state
      timer_2 = current_time;                                 //Set timer_2 to current_time
    }
  }
  else if(last_channel_2 == 1){                               //Input 9 is not high and changed from 1 to 0
    last_channel_2 = 0;                                       //Remember current input state
    receiver_input[2] = current_time - timer_2;               //Channel 2 is current_time - timer_2
  }
  //Channel 3=========================================
  if(PINB & B00000100 ){                                      //Is input 10 high?
    if(last_channel_3 == 0){                                  //Input 10 changed from 0 to 1
      last_channel_3 = 1;                                     //Remember current input state
      timer_3 = current_time;                                 //Set timer_3 to current_time
    }
  }
  else if(last_channel_3 == 1){                               //Input 10 is not high and changed from 1 to 0
    last_channel_3 = 0;                                       //Remember current input state
    receiver_input[3] = current_time - timer_3;               //Channel 3 is current_time - timer_3

  }
  //Channel 4=========================================
  if(PINB & B00001000 ){                                      //Is input 11 high?
```

```
    if(last_channel_4 == 0){                    //Input 11 changed from 0 to 1
      last_channel_4 = 1;                       //Remember current input state
      timer_4 = current_time;                   //Set timer_4 to current_time
    }
  }
  else if(last_channel_4 == 1){                 //Input 11 is not high and changed from 1 to 0
    last_channel_4 = 0;                         //Remember current input state
    receiver_input[4] = current_time - timer_4; //Channel 4 is current_time - timer_4
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Subroutine for reading the gyro
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void gyro_signalen(){
  //Read the L3GD20H
  if(eeprom_data[31] == 3){
    Wire.beginTransmission(gyro_address);           //Start communication with the gyro (adress 1101001)
    Wire.write(168);                                //Start reading @ register 28h and auto increment with every read
    Wire.endTransmission();                         //End the transmission
    Wire.requestFrom(gyro_address, 6);              //Request 6 bytes from the gyro
    while(Wire.available() < 6);                    //Wait until the 6 bytes are received
    lowByte = Wire.read();                          //First received byte is the low part of the angular data
    highByte = Wire.read();                         //Second received byte is the high part of the angular data
    gyro_axis[1] = ((highByte<<8)|lowByte);         //Multiply highByte by 256 (shift left by 8) and ad lowByte
    lowByte = Wire.read();                          //First received byte is the low part of the angular data
    highByte = Wire.read();                         //Second received byte is the high part of the angular data
    gyro_axis[2] = ((highByte<<8)|lowByte);         //Multiply highByte by 256 (shift left by 8) and ad lowByte
    lowByte = Wire.read();                          //First received byte is the low part of the angular data
    highByte = Wire.read();                         //Second received byte is the high part of the angular data
    gyro_axis[3] = ((highByte<<8)|lowByte);         //Multiply highByte by 256 (shift left by 8) and ad lowByte
  }

  if(cal_int == 2000){
    gyro_axis[1] -= gyro_axis_cal[1];               //Only compensate after the calibration
    gyro_axis[2] -= gyro_axis_cal[2];               //Only compensate after the calibration
    gyro_axis[3] -= gyro_axis_cal[3];               //Only compensate after the calibration
  }
  gyro_roll = gyro_axis[eeprom_data[28] & 0b00000011];
  if(eeprom_data[28] & 0b10000000)gyro_roll *= -1;
  gyro_pitch = gyro_axis[eeprom_data[29] & 0b00000011];
  if(eeprom_data[29] & 0b10000000)gyro_pitch *= -1;
  gyro_yaw = gyro_axis[eeprom_data[30] & 0b00000011];
  if(eeprom_data[30] & 0b10000000)gyro_yaw *= -1;

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Subroutine for calculating pid outputs
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void calculate_pid(){
  //Roll calculations
```

```
  pid_error_temp = gyro_roll_input - pid_roll_setpoint;
  pid_i_mem_roll += pid_i_gain_roll * pid_error_temp;
  if(pid_i_mem_roll > pid_max_roll)pid_i_mem_roll = pid_max_roll;
  else if(pid_i_mem_roll < pid_max_roll * -1)pid_i_mem_roll = pid_max_roll * -1;

  pid_output_roll = pid_p_gain_roll * pid_error_temp + pid_i_mem_roll + pid_d_gain_roll * (pid_error_temp - pid_last_roll_d_error);
  if(pid_output_roll > pid_max_roll)pid_output_roll = pid_max_roll;
  else if(pid_output_roll < pid_max_roll * -1)pid_output_roll = pid_max_roll * -1;

  pid_last_roll_d_error = pid_error_temp;

  //Pitch calculations
  pid_error_temp = gyro_pitch_input - pid_pitch_setpoint;
  pid_i_mem_pitch += pid_i_gain_pitch * pid_error_temp;
  if(pid_i_mem_pitch > pid_max_pitch)pid_i_mem_pitch = pid_max_pitch;
  else if(pid_i_mem_pitch < pid_max_pitch * -1)pid_i_mem_pitch = pid_max_pitch * -1;

  pid_output_pitch = pid_p_gain_pitch * pid_error_temp + pid_i_mem_pitch + pid_d_gain_pitch * (pid_error_temp - pid_last_pitch_d_error);
  if(pid_output_pitch > pid_max_pitch)pid_output_pitch = pid_max_pitch;
  else if(pid_output_pitch < pid_max_pitch * -1)pid_output_pitch = pid_max_pitch * -1;

  pid_last_pitch_d_error = pid_error_temp;

  //Yaw calculations
  pid_error_temp = gyro_yaw_input - pid_yaw_setpoint;
  pid_i_mem_yaw += pid_i_gain_yaw * pid_error_temp;
  if(pid_i_mem_yaw > pid_max_yaw)pid_i_mem_yaw = pid_max_yaw;
  else if(pid_i_mem_yaw < pid_max_yaw * -1)pid_i_mem_yaw = pid_max_yaw * -1;

  pid_output_yaw = pid_p_gain_yaw * pid_error_temp + pid_i_mem_yaw + pid_d_gain_yaw * (pid_error_temp - pid_last_yaw_d_error);
  if(pid_output_yaw > pid_max_yaw)pid_output_yaw = pid_max_yaw;
  else if(pid_output_yaw < pid_max_yaw * -1)pid_output_yaw = pid_max_yaw * -1;

  pid_last_yaw_d_error = pid_error_temp;
}

//This part converts the actual receiver signals to a standardized 1000 – 1500 – 2000 microsecond value.
//The stored data in the EEPROM is used.
int convert_receiver_channel(byte function){
  byte channel, reverse;                                            //First we declare some local variables
  int low, center, high, actual;
  int difference;

  channel = eeprom_data[function + 23] & 0b00000111;         //What channel corresponds with the specific function
  if(eeprom_data[function + 23] & 0b10000000)reverse = 1;       //Reverse channel when most significant bit is set
  else reverse = 0;                                                 //If the most significant is not set there is no reverse

  actual = receiver_input[channel];                                 //Read the actual receiver value for the corresponding function
```

```
  low = (eeprom_data[channel * 2 + 15] << 8) | eeprom_data[channel * 2 + 14];  //Store the low value for the specific receiver input channel
  center = (eeprom_data[channel * 2 - 1] << 8) | eeprom_data[channel * 2 - 2]; //Store the center value for the specific receiver input channel
  high = (eeprom_data[channel * 2 + 7] << 8) | eeprom_data[channel * 2 + 6];   //Store the high value for the specific receiver input channel

  if(actual < center){                                  //The actual receiver value is lower than the center value
    if(actual < low)actual = low;                       //Limit the lowest value to the value that was detected during setup
    difference = ((long)(center - actual) * (long)500) / (center - low);      //Calculate and scale the actual value to a 1000 - 2000us value
    if(reverse == 1)return 1500 + difference;                   //If the channel is reversed
    else return 1500 - difference;                              //If the channel is not reversed
  }
  else if(actual > center){                             //The actual receiver value is higher than the center value
    if(actual > high)actual = high;                     //Limit the lowest value to the value that was detected during setup
    difference = ((long)(actual - center) * (long)500) / (high - center);     //Calculate and scale the actual value to a 1000 - 2000us value
    if(reverse == 1)return 1500 - difference;                   //If the channel is reversed
    else return 1500 + difference;                              //If the channel is not reversed
  }
  else return 1500;
}

void set_gyro_registers(){

  //Setup the L3GD20H
  if(eeprom_data[31] == 3){
    Wire.beginTransmission(gyro_address);               //Start communicationwith the address found during search.
    Wire.write(0x20);                                   //We want to write to register 1 (20 hex).
    Wire.write(0x0F);                           //Set the register bits as 00001111 (Turn on the gyro and enable all axis).
    Wire.endTransmission();                             //End the transmission with the gyro.

    Wire.beginTransmission(gyro_address);               //Start communication with the address found during search.
    Wire.write(0x23);                                   //We want to write to register 4 (23 hex).
    Wire.write(0x90);               //Set the register bits as 10010000 (Block Data Update active & 500dps full scale).
    Wire.endTransmission();                             //End the transmission with the gyro.

    //Let's perform a random register check to see if the values are written correct
    Wire.beginTransmission(gyro_address);               //Start communication with the address found during search
    Wire.write(0x23);                                   //Start reading @ register 0x23
    Wire.endTransmission();                             //End the transmission
    Wire.requestFrom(gyro_address, 1);                  //Request 1 bytes from the gyro
    while(Wire.available() < 1);                        //Wait until the 6 bytes are received
    if(Wire.read() != 0x90){                            //Check if the value is 0x90
      digitalWrite(12,HIGH);                            //Turn on the warning led
      while(1)delay(10);                                //Stay in this loop for ever
    }
  }
}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

1) «The UAV» Available: http://www.theuav.com/.
2) «RobotShop» RobotShop Inc. Available: http://www.robotshop.com/en/brushless-motor-kit-ardrone.html.
3) P.Bupe, Available: http://diydrones.com/.
   Available: http://diydrones.com/profiles/blogs/autonomous-uav-clustering-network
4) Clearpath Robotics, «Robohub.org,» Available: http://robohub.org/ros-101-intro-to-the-robot-operating-system/.
5) G.M. Van der Zalm, Tuning of PID-Type Controllers: Literature Overview, 2004
6) Τζαφέστας,Πομποτική Ανάλυση Έλεγχος Σχεδιασμός Προγραμματισμός Αίσθηση, Αθήνα, 2003
7) Arduino, Available: http:/www.arduino.cc
8) Microkopter, Available: http:/www.mikrokopter.de
9) ArduIMU Project, Available: http://code.google.com/p/ardu-imu/
10) Terp Connect, http://terpconnect.umd.edu/~leishman/Aero/history.html