" Το να πάς στην Κούβα με την γυναίκα σου…
είναι σαν να πάς στο Άργος με το πεπόνι σου"
Τάσος Τακιάδης

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα μελέτη ολοκληρώθηκε χάρη στον επιβλέποντα καθηγητή μας κ. Γρηγόρη Νικολάου, τον οποίο και ευχαριστούμε θερμά για την καθοδήγηση, τις υποδείξεις, τις παρατηρήσεις, αλλά και για την εμπιστοσύνη του στην ομάδα μας.

Ευχαριστίες οφείλουμε στην Μαγδαληνή Βασιλειάδου για την επιμέλεια του κειμένου, τις συμβουλές και την υποστήριξή της. Αντίστοιχα ευχαριστούμε και τον Μιχάλη Καβουκλή για το σύνολο των επισημάνσεων και την επίλυση των αποριών μας σε ζητήματα προγραμματιστικής φύσεως. Η ηθική υποστήριξη της Εύης Κυριάκου και του Ματθαίου Olma συνέβαλε ουσιαστικά κατά την διάρκεια της μελέτης αυτής.

Τέλος, θα θέλαμε να ευχαριστήσουμε από καρδιάς τους γονείς μας Έμυ και Σπύρο Καβουκλή, Καλλιόπη και Αριστείδη Μαραγκό, Μαρία και Παναγιώτη Κατρανίδη για την συνεχή και αμέριστη εμπιστοσύνη και υποστήριξη, που έδειξαν καθ’ όλη την διάρκεια των σπουδών μας.

ΠΕΡΙΛΗΨΗ

Από τα τέλη του Δεύτερου Παγκοσμίου Πολέμου, ο τομέας των Μη Επανδρωμένων Αεροσκαφών (UAV) έχει απασχολήσει εκτεταμένα την επιστημονική κοινότητα. Το μεγάλο αυτό ενδιαφέρον προκαλείται από την ευελιξία και την μεγάλη ακρίβεια των αεροσκαφών αυτών για εκτέλεση αποστολών υψηλού κινδύνου. Οι πρώτες έρευνες αφορούσαν αποκλειστικά UAV με καθοδήγηση μέσω τηλεχειρισμού, ενώ με την ανάπτυξη των μικροελεγκτών είναι πλέον δυνατή η πλήρη αυτονομία τους, δεδομένου ότι μπορούν να προγραμματιστούν οι αποστολές των αεροσκαφών πριν την πτήση, αλλά και να αναπρογραμματιστούν κατά την διάρκειά της. Επιπρόσθετα μέσω ενός μικροελεγκτή δίνεται η δυνατότητα προσαρμογής του αεροσκάφους σε κάθε εξωγενή μεταβολή των συνθηκών πτήσης (καιρικά φαινόμενα, φυσικά ή τεχνητά εμπόδια).

Η παρούσα μελέτη εξετάζει τον τρόπο λειτουργίας και επικοινωνίας του μικροελεγκτή PX4 σε συνδυασμό με τις μεθόδους που ο ίδιος θέτει σε εφαρμογή για την κίνηση και τον έλεγχο ενός αεροσκάφους fixedwing. Αρχικά συλλέχθηκαν στοιχεία για το λειτουργικό σύστημα πραγματικού χρόνου τύπου UNIX του μικροελεγκτή ως το πρωταρχικό βήμα της εν λόγω έρευνας. Στη συνέχεια εξετάστηκε το περιβάλλον επικοινωνίας του με τον χειριστή, καθώς και με τα επιμέρους όργανα του αεροσκάφους. Η αναλυτική μελέτη αυτών (επιταχυνσιόμετρο, γυροσκόπιο, μαγνητόμετρο, αισθητήρας πίεσης) και ο τρόπος λειτουργίας των αισθητηρίων τους, απέδωσε την μεθοδολογία, μέσω της οποίας ο μικροελεγκτής αντιλαμβάνεται όλα τα δεδομένα πτήσης. Το αποτέλεσμα όλων των παραπάνω οδήγησε στην κατανόηση του τρόπου λήψης των αποφάσεων από τον μικροελεγκτή. Επιπρόσθετα μελετήθηκε το Φίλτρο Kalman, τμήμα του κώδικα του μικροελεγκτή, το οποίο επεξεργάζεται κάθε σήμα των οργάνων πριν φτάσει στον μικροελεγκτή. Ενδεικτικά γίνεται μια εισαγωγή στις μεθόδους στατιστικής εκτίμησης μετρούμενων μεταβλητών μέσω του Φίλτρου αυτού.

Το ζήτημα της επικοινωνίας ανάμεσα στο interface του μικροελεγκτή και στο πρόγραμμα εξομοίωσης αναλύθηκε εκτενώς. Αντίστοιχα παρουσιάστηκε και αναλύθηκε

ένα τμήμα του Κώδικα που ευθύνεται για την πλοήγηση και τον έλεγχο του αεροσκάφους. Τέλος, στο πειραματικό στάδιο της εργασίας πραγματοποιήθηκαν δοκιμαστικές πτήσεις μέσω του προγράμματος εξομοίωσης και εφαρμόστηκαν δοκιμές διαφορετικών μορφών ελέγχου. Τα αποτελέσματά τους παρουσιάζονται μέσω εικόνων και γραφικών παραστάσεων, που συμπεριλαμβάνονται στο ίδιο κεφάλαιο, καθώς επίσης και ο σχολιασμός τους.

Η μελέτη ολοκληρώνεται με τις προτάσεις μιας μελλοντικής ενασχόλησης με το συγκεκριμένο θέμα, όπως διαφορετικές μορφές ελέγχου, επεξεργασία πληροφοριών μέσω κάμερας, ή σχηματισμός σμήνους UAV.

Το παρόν βιβλίο έχει την ακόλουθη δομή: αρχικά γίνεται ιστορική αναδρομή και περιγραφή των UAV, καθώς και οι βασικές αρχές ελέγχου αυτών. Παρουσιάζεται η πλατφόρμα που πραγματοποιεί τον έλεγχο του αεροσκάφους, το σύνολο των οργάνων που χρησιμοποιεί για την συλλογή πληροφοριών καθώς και το λειτουργικό σύστημα του μικροελεγκτή. Στη συνέχεια γίνεται αναφορά στον ελεγκτή PID και τον τρόπο που είναι δομημένος. Παρατίθεται και σχολιάζεται μέρος του κώδικα που ευθύνεται για την κατάσταση και την πλοήγηση του αεροσκάφους. Έπειτα αναλύεται η μεθοδολογία για την επικοινωνία του μικροελεγκτή με το πρόγραμμα εξομοίωσης, καθώς και ο τρόπος εγκατάστασης του προγράμματος με τα χαρακτηριστικά του. Τέλος παρουσιάζεται η πειραματική έρευνα, με τα αποτελέσματα, τον σχολιασμό και τα συμπεράσματα που προκύπτουν από αυτή, καθώς και οι πιθανοί τομείς για μελλοντική ενασχόληση και εξέλιξη με το αντικείμενο.

ABSTRACT

As of the ending or World War II, research in unmanned aerial vehicle (UAV) has been of high importance to the scientific community. This high interest comes mainly from the agility and precision of these airplanes when it comes to executing dangerous missions. Initial research had as goal solely remotely piloted UAV but through the evolution of microcontrollers the ability to adjust automaticaly to external triggers was provided (weather conditions, natural and artificial obstacles).

This paper studies the functionality and communication interfaces of microcontroller PX4 and its applicability guiding an fixedwing airplane. Initially the paper concentrates on the description of the unix-like real-time operating system of the microcontroller. Followingly, the communication interfaces of the microcontroller to the UAV operator as well as with the components of the airplane are analyzed. An analysis of the airplane components (accelerometer, gyroscope, magnetometer, pressure sensor) as well as a technical understanding of their functionality provided the methodology through which the microcontroller reacts to the different flight conditions. Through the latter data, the basic rational on decision making of the microcontroller are made clear. In addition, the Kalman filter is analyzed that is used in order to process the signals given by the plane components before being fed to the microcontroller.

In order to prove theory, a prototype using a fixedwing airplane and a PX4 controler was constructed. The communication between the microcontroller's interface and the simulation program, being a major issue, has been analyzed in detail. In addition a part of the microcontroller source code in charge of the guidance and airplane checks was thoroughly analysed. In the experimental part of the study real trial flights of the prototype airplane took place using the simulation software where different flight modes were used. The study is ending by stating the future work such as different flight modes, image processing for navigation or UAV group flights.

The study is structured as follows. The history of UAV evolution is provided as well as their basic control principles followed by the presentation of the platform used for the airplane check, the components and the mechanisms to collect data as well as the operating system of the microcontroller. Next the PID controller is described with details considering its structure as well as part of the code in charge of the airplane guidance. The methodology for the microcontroller and the simulation software is described as well as the installation steps. Finally the experimental study is described along with its commented results and conclusions. In the final Chapter the future work is described.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΛΙΣΤΑ ΕΙΚΟΝΩΝ

Εισαγωγικό σημείωμα

Η εντατική ενασχόλησή της ομάδας μελέτης με το συγκεκριμένο αντικείμενο αποκάλυψε σταδιακά ζητήματα και προβλήματα που δεν γίνονται εύκολα κατανοητά ακόμη και από ειδικευμένο κοινό. Για το λόγο αυτό κρίθηκε σκόπιμο η μελέτη αυτή να έχει τη μορφή ενός βασικού εγχειριδίου, στο οποίο αναλύονται όλα τα επιμέρους στοιχεία που αποτελούν το θέμα της παρούσας έρευνας. Οι δυσκολίες που αντιμετωπίσαμε και εμείς, ως ειδικευμένο ανθρώπινο δυναμικό, κατά τη διάρκεια της συλλογής δεδομένων, μας οδήγησε στην συγγραφή ενός κειμένου, κατανοητού και εύκολα προσβάσιμου ως προς τις έννοιες και την ανάλυσή τους. Θεωρήθηκε απαραίτητη η λεπτομερής ανάλυση ορισμένων θεμάτων, που αποτελούν το υπόβαθρο για την λειτουργία του μικροελεγκτή αλλά και για τις πειραματικές εφαρμογές στις οποίες τον υποβάλαμε.

Στο πλαίσιο άλλωστε της πτυχιακής μας εργασίας θέσαμε τις βάσεις για μελλοντική έρευνα και ενασχόληση με το εν λόγω αντικείμενο, δεδομένου ότι διαπιστώσαμε την εφαρμογή λειτουργίας του σε πολλούς επαγγελματικούς τομείς, σε πρακτικό και επιστημονικό επίπεδο.

STATUS QUO

Σε αυτό το κεφάλαιο γίνεται μια σύντομη ιστορική αναδρομή της έρευνας έως σήμερα και παρουσιάζονται οι σύγχρονες τεχνικές κατασκευής και λειτουργίας μη επανδρωμένων αεροσκαφών UAV (Unmanned Aerial Vehicle).

Αεροπλάνα

Το αεροπλάνο είναι μια πτητική συσκευή, που παρά το γεγονός ότι είναι βαρύτερη από τον αέρα, με την χρήση ακίνητων πτερυγίων και την ανάπτυξη μεγάλης ταχύτητας κατορθώνει να αναπτύξει στα πτερύγια της ικανή δύναμη άνωσης, ώστε να ανυψώνεται και να διατηρείται στον αέρα. Το αεροπλάνο είναι το αποτέλεσμα τριών διαφορετικών μερών. Το σκάφος, που είναι και το κύριο σώμα του, το σύστημα προώθησης και τέλος τον μηχανικό εξοπλισμό.

Το σκάφος, το κύριο δηλαδή σώμα, αποτελείται από την άτρακτο, πάνω στην οποία είναι σταθερά συνδεδεμένες οι πτέρυγες, βασικές υπεύθυνες για την δυνατότητα ανύψωσης. Σε πολλές περιπτώσεις, σε ορισμένους τύπους αεροπλάνων, παρουσιάζεται αστάθεια, οπότε και παρατηρούνται επιπλέον πτερύγια στο τέλος της ουράς, τοποθετημένα οριζόντια και κάθετα, για την αντιμετώπιση αυτού του προβλήματος. Τα σταθερά πτερύγια διαχωρίζουν τα αεροπλάνα από τα ελικόπτερα, δεδομένου ότι στα δεύτερα συναντάμε κινούμενα πτερύγια.

Το σύστημα προώθησης των αεροπλάνων ξεκίνησε με την χρήση της έλικας, που σύντομα αντικαταστάθηκε από τον κινητήρα τζετ. Η αλλαγή αυτή προκλήθηκε από την ανάγκη ανάπτυξης μεγαλύτερης ταχύτητας, όπου η έλικα έχει πεπερασμένες δυνατότητες. Σήμερα συναντάμε και αεροπλάνα χωρίς σύστημα προώθησης, στον τύπο δηλαδή του ανεμοπλάνου. Η απογείωση επιτυγχάνεται με την βοήθεια ενός άλλου μέσου (π.χ. αεροπλάνου), ενώ για την συντήρηση του ύψους του χρησιμοποιείται η μετακίνηση των αέριων μαζών, με αποτέλεσμα την κάλυψη αρκετά μεγάλων αποστάσεων.

Τέλος ο μηχανικός εξοπλισμός αποτελείται από το σύνολο των εξαρτημάτων, των οργάνων και των εγκαταστάσεων που βρίσκονται εντός του αεροπλάνου, με σκοπό την σωστή λειτουργία του, την εκτέλεση των επιθυμητών ενεργειών και την επιβίωση του ανθρώπινου επιβατικού δυναμικού.

Μη επανδρωμένα αεροσκάφη

Ο όρος μη επανδρωμένα αεροσκάφη, επονομαζόμενα και ως UAV (Unmanned Aerial Vehicle), αναφέρεται σε αεροσκάφη που χαρακτηρίζονται από την απουσία πιλότου ή ανθρώπινου δυναμικού στο εσωτερικό τους και ποικίλουν ανάλογα το μέγεθος, το σχήμα και τα χαρακτηριστικά τους. Η πλοήγηση τους μπορεί να γίνεται είτε με τηλεχειρισμό, είτε με προγραμματισμό μικροελεγκτών.

Η μελέτη τους ξεκίνησε περί τις αρχές του $20_{ου}$ αιώνα, από τους Αμερικάνους, κατά την διάρκεια του Α' Παγκοσμίου Πολέμου, αλλά λόγω της εκεχειρίας του, σταμάτησε σύντομα λόγω έλλειψης ενδιαφέροντος. Στα μέσα του $20_{ου}$ αιώνα και κατά την διάρκεια του Β' Παγκοσμίου Πολέμου, όταν η ανάγκη δημιουργίας ενός αεροσκάφους που θα μπορούσε να εκτελεί αποστολές υψηλής επικινδυνότητας (για πολεμικούς και κατασκοπευτικούς λόγους), με την χρήση απομακρυσμένου χειρισμού, έγινε επιτακτική, προκλήθηκε η κατασκευή των μη επανδρωμένων αεροσκαφών, εξασφαλίζοντας με τον τρόπο αυτό την επιβίωση του πιλότου και του υπόλοιπου εν πτήση στρατιωτικού δυναμικού. Για τα γερμανικά στρατεύματα των Ναζί κατασκευάστηκε και χρησιμοποιήθηκε για πρώτη φορά ένα UAV. Η δυσκολία κατάρριψης του, αφύπνισε το ενδιαφέρον και προκάλεσε εκ νέου την συνέχεια των σχετικών μελετών και ερευνών στην Αμερική. Το αποτέλεσμα τους διαπιστώθηκε στον πόλεμο του Βιετνάμ, όπου τα UAV έπαιξαν καθοριστικό ρόλο ως κατασκοπευτικά και ως μαχητικά αεροσκάφη, καθώς υπήρξε μεγάλη ανάπτυξη και εξέλιξη στον τομέα του τηλεχειρισμού. Στα τέλη του ίδιου αιώνα η Ισραηλινή Πολεμική Αεροπορία προχώρησε σε τόσο μεγάλη παραγωγή πολεμικών αεροσκαφών, με αποτέλεσμα την δημιουργία

στόλου UAV, ικανού για την προμήθεια πολλών χωρών, συμπεριλαμβανομένης και της Αμερικής.

Από τις αρχές του 21ου αιώνα, μέχρι και σήμερα, τα UAV κατέχουν σταθερή και άκρως σημαντική θέση στην πολεμική αεροπορία, σε όλο τον κόσμο. Παράλληλα σημαντικός αριθμός αυτών των αεροσκαφών χρησιμοποιείται για ειρηνικούς σκοπούς, όπως στην περιβαλλοντολογία, καθώς και σε επιχειρήσεις ανθρώπινης διάσωσης.

Χρήσεις

Η τεχνολογική εξέλιξη και ανάπτυξη των UAV γίνεται σήμερα συστηματικά, ενώ το πεδίο χρήσης τους καθορίζει και τα επιμέρους χαρακτηριστικά τους. Σύμφωνα με την χρηστικότητά τους χωρίζονται σε δύο βασικά πεδία δράσης, το αμυντικό/στρατιωτικό και το κοινωνικό/ανθρωπολογικό. Το πρώτο πεδίο αφορά σε στρατιωτικές επιχειρήσεις (χαρτογράφηση, αναγνώριση και παρακολούθηση περιοχών, καθώς και επίθεση σε συγκεκριμένους στόχους με εξαιρετική ακρίβεια), ενώ το δεύτερο σε δράσεις κοινωνικού και ανθρωπολογικού χαρακτήρα (διάσωση, εποπτεία δασικών πυρκαγιών, μεταφορές, εναέρια διαφήμιση, εγχώρια αστυνόμευση, επιστημονική έρευνα).

Αεροσκάφη Μινιατούρες

Ο όρος αεροσκάφη μινιατούρες, στο εξής ως MAV (Miniature Air Vehicle), αναφέρεται σε αεροσκάφη, με ακίνητα πτερύγια, που το συνολικό μήκος του ανοίγματος των φτερών είναι μικρότερο ή ίσο του 1,5 μέτρου. Η κινητήρια δύναμη αυτών συνήθως προέρχεται από μπαταρίες τροφοδότησης για τις αναγκαίες λειτουργίες τους κατά την πτήση αλλά και την συντήρηση του κάθε αεροσκάφους. Το μειωμένο βάρος, καθώς και το μικρό ποσοστό αδράνειας, απαλλάσσει τα MAVs από τη χρήση διαδρόμου απογείωσης και προσγείωσης. Κατά συνέπεια η απογείωση πραγματοποιείται από τον

ίδιο τον χειριστή, ενώ η προσγείωση γίνεται με το κάτω μέρος του αεροσκάφους (κοιλιά).

Δυσκολίες κατασκευής

Η τεχνική κατασκευής των MAVs αντιμετωπίζει μεγάλη εξειδίκευση, δεδομένου ότι ο βασικός στόχος είναι η εξασφάλιση της μεγαλύτερης δυνατής αυτονομίας τους. Για τον περιορισμό των αδυναμιών κατασκευής των MAVs, της μικρής ισχύος τους κυρίως, βασικό μέλημα του κατασκευαστή είναι το μέγεθος και το βάρος του σκάφους και των εξαρτημάτων πλοήγησης.

Η συνεχώς αυξανόμενη ζήτηση για μικρότερα σε μέγεθος MAVs αλλά με περισσότερες δυνατότητες απαιτούν νέες τεχνολογικές κατασκευαστικές και λειτουργικές μεθόδους. Η σύγχρονη έρευνα επιδιώκει την αναζήτηση νέων τεχνικών μέσων για να είναι δυνατή: η εξοικονόμηση περισσότερης αυτονομίας ως προς τις ώρες πτήσης, η μεγαλύτερη ευελιξία, η καλύτερη επικοινωνία μεταξύ MAV και σταθμού εδάφους, η ασφαλέστερη και ταχύτερη προσέγγιση του στόχου από το κατευθυνόμενο MAV με τη χρήση των αλγόριθμων πλοήγησης.

Τρέχουσες τεχνολογίες

Προς αυτήν την κατεύθυνση κινούνται σήμερα πολλές κατασκευαστικές εταιρίες, που χρησιμοποιούν εξαιρετικά εξελιγμένους τρόπους κατασκευής και λειτουργίας των UAV, λόγω της τεράστιας τεχνολογικής ανάπτυξης στους επιστημονικούς χώρους. Η δυνατότητα χρήσης των UAVs σε ολοένα και περισσότερα πεδία δράσης, καθώς και οι διαφορετικές συνθήκες, που ο εξοπλισμός των UAVs θα πρέπει να αντιμετωπίσει, πολλαπλασιάζουν τις μεθόδους κατασκευής.

Σε διεθνές πανεπιστημιακό επίπεδο κυριαρχούν τρία μεγάλα ερευνητικά κέντρα, το ETH (Zurich), το PRINCETON (New Jersey) και το ENAC (Toulouse), ενώ

παράλληλα δραστηριοποιούνται και πολλές ιδιωτικές εταιρίες που προωθούν και αυτές στη συνεχή ανάπτυξη και εξέλιξη των UAVs.

Η παρούσα μελέτη επικεντρώνεται στον μικροελεγκτή PX4, όπως αυτός αναπτύχθηκε και κατασκευάστηκε από ερευνητική ομάδα, του πανεπιστήμιο του ETH, (https://pixhawk.ethz.ch/px4/modules/px4fmu), δεδομένου ότι προσφέρει ένα περιβάλλον open-source και open-hardware, με δυνατότητα συνεχούς εξέλιξης και βελτιστοποίησης του ίδιου του μικροελεγκτή, αλλά και των περιφερειακών του συνολικά.

ΜΙΚΡΟΕΛΕΓΚΤΗΣ

Στο κεφάλαιο παρουσιάζεται το βασικό εξάρτημα λειτουργίας των UAV, ο μικροελεγκτής, ο "εγκέφαλος" του UAV ως κέντρο λειτουργίας του, καθώς και το μέσο επικοινωνίας του με τον άνθρωπο.

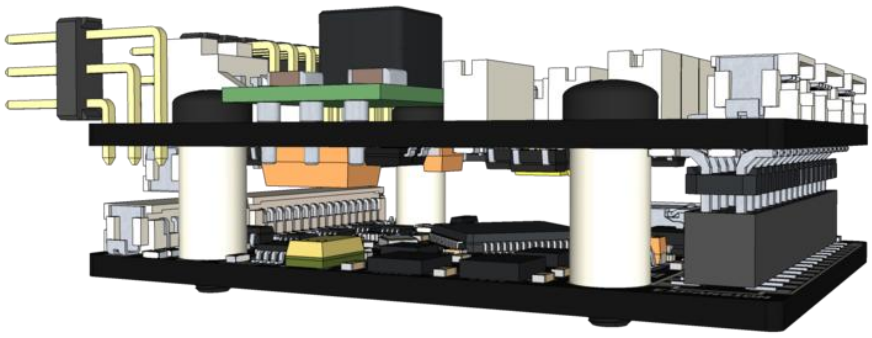
Ένα σύστημα αυτόματου πιλότου, όπως και κάθε MEMS (Microelectromechanical system), λειτουργεί μέσω ενός μικροελεγκτή-μικροϋπολογιστή που αποτελεί το κέντρο διαχείρισης-ελέγχου και επενεργητή. Ο μικροϋπολογιστής αυτός αναλαμβάνει να επεξεργαστεί όλα τα δεδομένα, που συλλέγει με τα αισθητήριά του, από το περιβάλλον, καθώς και όλα εκείνα τα στοιχεία, που του παρέχει ο χρήστης, με σκοπό να ανταποκριθεί όσο το δυνατόν γρηγορότερα και ακριβέστερα, για να μεταβεί στη ζητούμενη κατάσταση ή να ολοκληρώσει μια συγκεκριμένη ενέργεια. Στην περίπτωση της παρούσας μελέτης απαιτείται να ακολουθήσει μια σειρά από γεωγραφικά σημεία μέσω ενός μονοπατιού, να μπορέσει να παρακολουθήσει τη θέση του στο χώρο, καθώς και να παραμείνει ανεπηρέαστο από οποιαδήποτε εξωτερική παρεμβολή όπως π.χ. ριπές άνεμου.

Υπάρχουν διάφορες εκδοχές ολοκληρωμένων autopilot πλατφόρμων, από τις οποίες ενδεικτικά αναφέρεται η πλέον δημοφιλής πλατφόρμα Ardupilot με τον επεξεργαστή Atmega 2560 στα 16Mhz, καθώς και η πλατφόρμα paparazzi, η οποία χρησιμοποιεί πολλές παραλλαγές του απαιτούμενου hardware (μικροελεγκτών, αισθητηρίων, GPS) όπως Lisa/L v1.1, Lisa/M v2.0, Lisa/S, Apogee v1.00, KroozSD, Umarim v1.0, Tiny v2.11, TWOG v1.0 κ.α. με επεξεργαστές ARM Cortex M3 στα 72Mhz και ARM7TDMI-S στα 60Mhz.

Ο μικροελεγκτής που χρησιμοποιήθηκε για την ολοκλήρωση της παρούσας εργασίας είναι ο PX4 και δημιουργήθηκε από μια ομάδα μεταπτυχιακών φοιτητών του τμήματος Computer Vision and Geometry group του Πανεπιστημίου ETH της Ζυρίχης το 2009, με εμπνευστή της ιδέας τον Lorenz Meier. Υποστηρίζεται από το τμήμα Computer

Vision and Geometry group, το Autonomous Systems Lab και το Automatic Control Laboratory του Πανεπιστημίου ETH και είναι εμπορικά διαθέσιμος από την 3D Robotics. Η πλατφόρμα PX4 είναι ένα open source (hardware και software) ολοκληρωμένο σύστημα αυτόματου πιλότου κατάλληλο για UAV οχήματα fixed-wing (αεροπλάνα) και multi rotors (ελικόπτερα με έναν ή περισσότερους κινητήρες), για αυτοκίνητα, για πλοία και για κάθε είδους ρομποτικής κινούμενης πλατφόρμας. Αποτελείται από δύο (2) πλακέτες, την πλακέτα PX4fmu (flight management unit – Autopilot) και την πλακέτα PX4IO (Airplane servo and I/O module), οι οποίες ενώνονται μεταξύ τους με το PX4 expansion bus system σε μορφή sandwich.

Ο PX4 επιλέχτηκε βάση της ταχύτητάς του και των ολοκληρωμένων περιφερειακών του. Η ταχύτητά του είναι απαραίτητη για να μπορέσει να γίνει επεξεργασία των δεδομένων με την χρήση εκτεταμένων φίλτρων Kalman, για την βέλτιστη εκτίμηση των μεταβλητών που αφορούν στο σύστημα, κατά τέτοιο τρόπο ώστε το εκάστοτε σφάλμα να ελαχιστοποιείται στατιστικά.



2-1. Πλατφόρμα μικροελεγκτή PX4 .

PX4fmu (flight management unit –Autopilot)

Ο PX4 ως μικροεπεξεργαστική ισχύ ήταν η δυνατότερη διαθέσιμη πλατφόρμα στην αγορά κατά τη χρονική στιγμή έναρξης της εργασίας αυτής (Μάιος 2013). Ο PX4 φέρει επεξεργαστή ARM Cortex M4 αρχιτεκτονικής ARMv7E-M συχνότητας 168 Mhz / 252 MIPS και έχει 192KB SRAM / 1024KB FLASH, υποστηρίζει λειτουργικό σύστημα πραγματικού χρόνου (RTOS), το NuttX, και έχει USB Bootloader για την αναβάθμιση του λογισμικού μέσω της micro USB θύρας που διαθέτει, από περιβάλλον Windows, Linux και Macintosh και έχει ενσωματωμένα τα απαραίτητα αισθητήρια (IMU) για να υπάρξει έλεγχος της πορείας του ελεγχόμενου συστήματος. Μερικά από τα αισθητήρια αυτά είναι το γυροσκόπιο, το επιταχυνσιόμετρο, το μαγνητόμετρο και το αισθητήριο βαρομετρικής πίεσης. Ως θύρες επικοινωνίας του μικροελεγκτή συναντάμε τις UART, 2 θύρες I2C, 1 θύρα SPI και 1 θύρα CAN. Τέλος, διαθέτει και υποδοχή κάρτας επέκτασης μνήμης microSD.

Συγκεντρωτικά τα χαρακτηριστικά του PX4fmu, όπως παρουσιάζονται στο επίσημο site (https://pixhawk.ethz.ch/px4) είναι:

- 168 MHz / 252 MIPS Cortex-M4F
- Hardware floating point unit
- POSIX-compatible RTOS
- SIMD extensions
- 192KB SRAM / 1024 KB Flash
- USB Bootloader (software updates, Windows, Linux, Mac OS supported)

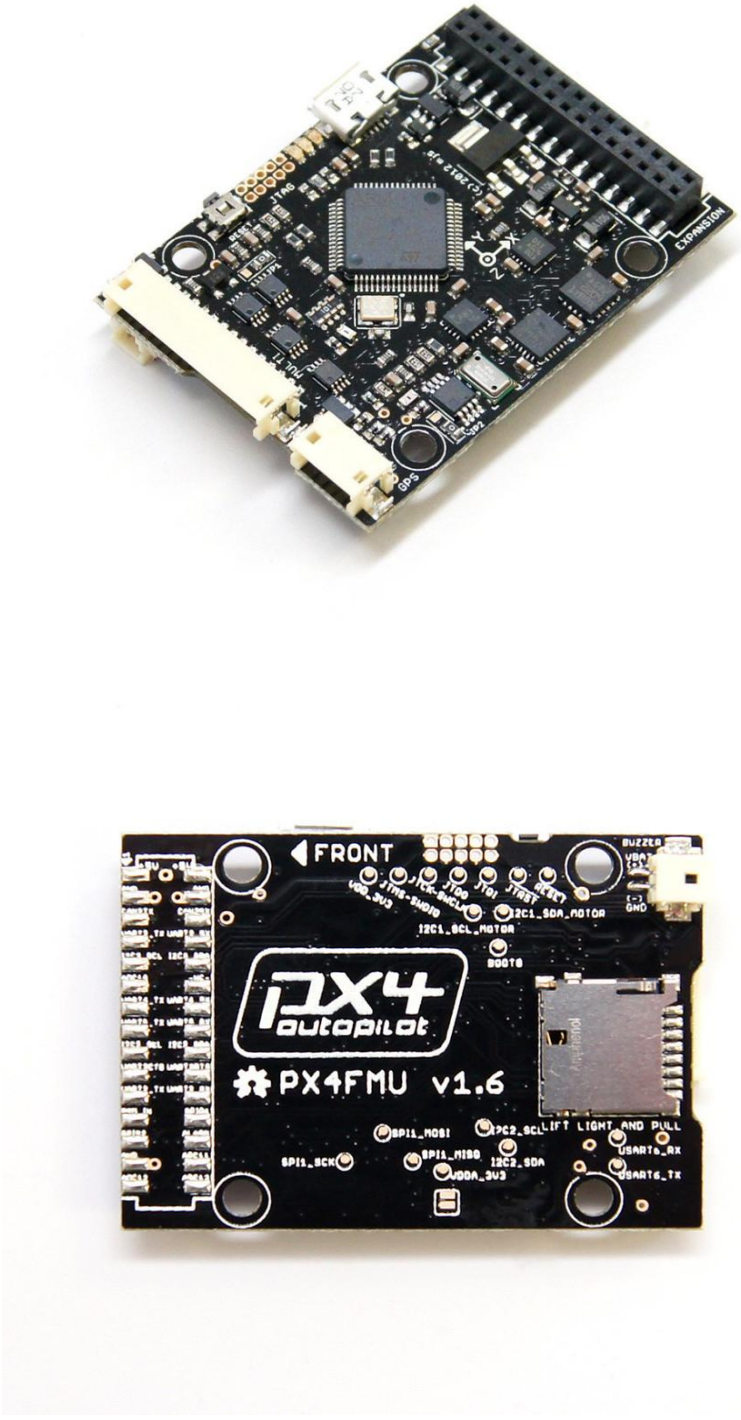
Interfaces

- 4x UART, 2x I2C, 1x SPI, 1x CAN
- External magnetometer port (I2C1 or I2C3, compatible with this board: 3DR magnetometer breakout board)
- MicroSD slot

- PPM / RC control input (sum signal format, many compatible receivers, all channels on one connector)
- Up to 8 GPIOs, 2 25mA high power, up to 4 PWM (servo out)
- Battery sense (1-18V), Buzzer (up to 1.0 A, VBAT driven)
- Reverse polarity protection on all power inputs
- Buzzer (PWM) output
- JTAG / SWD (ARM-Mini 10 pos / 0.05” connector)

Sensors

- MPU-6000 (3D ACC / Gyro)
- L3GD20 (3D Gyro)
- HMC5883L (3D Mag)
- MS5611 (barometric pressure)



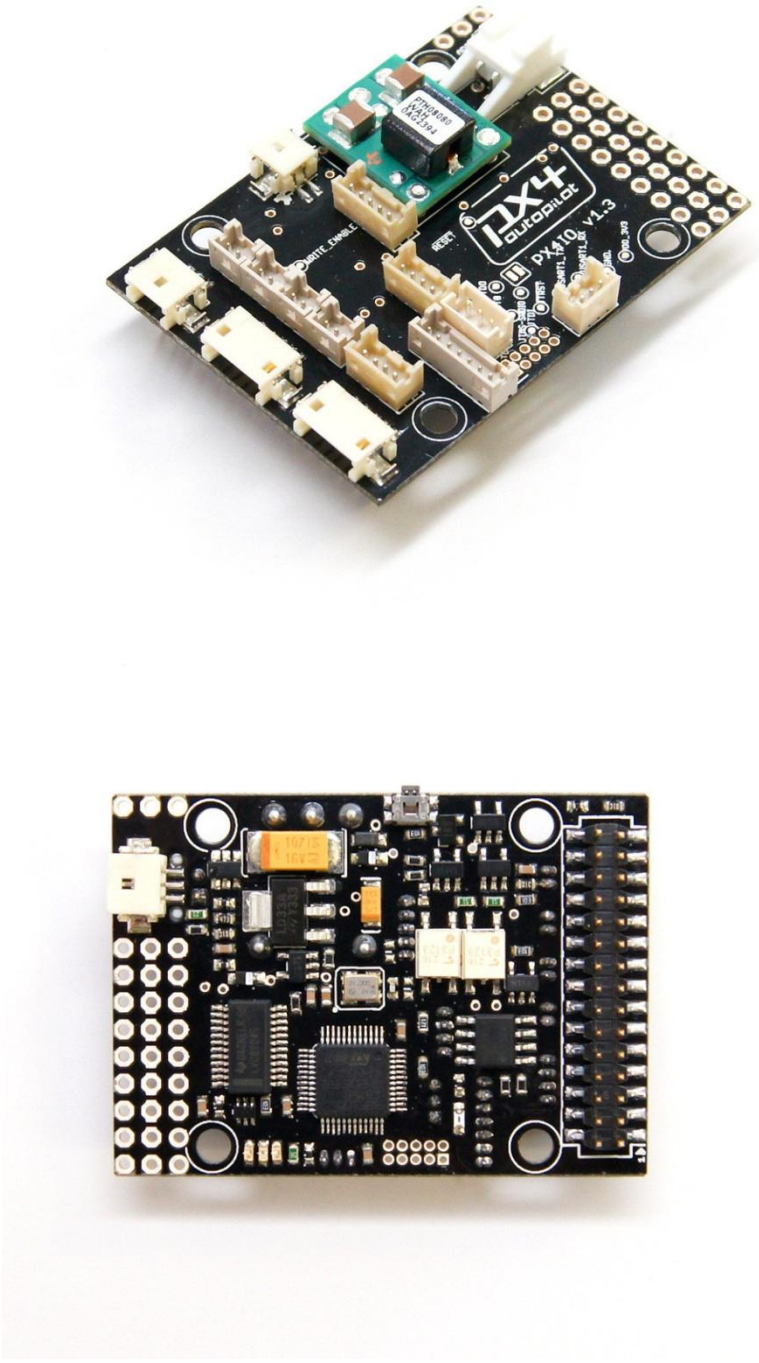
2-2. Οι δύο όψεις μιας πλακέτας μικροελεγκτή PX4fmu.

PX4IO (Airplane Servo and I/O Module)

Η PX4IO είναι η πλακέτα με τους ακροδέκτες διασύνδεσης εισόδων- εξόδων του μικροελεγκτή για την χρήση του σε συστήματα σταθερών πτερυγίων (αεροπλάνα), διαθέτει σύστημα ασφάλειας servo failsafe, σε περίπτωση δηλαδή προβλήματος επαναφέρει τους σερβοκινητήρες στην αρχική τους θέση μέσω ενός επεξεργαστή ARM Cortex-M3 με συχνότητα 24Mhz. Η πλακέτα αυτή δέχεται τροφοδοσία από 6 μέχρι 18 Volt DC και μπορεί με τη σειρά της να τροφοδοτήσει τις εξόδους με 2 Ampere στα 5 Volt, καθώς επίσης και την συνδεδεμένη σε αυτήν πλακέτα του PX4fmu. Ταυτόχρονα παρέχει προστασία πολικότητας σε όλες τις εισόδους ρεύματος. Υπάρχουν 8 έξοδοι για την σύνδεση σερβοκινητήρων (μέχρι 400Hz ταχύτητα), καθώς και συμβατότητα για σύνδεση Futaba S.BUS σερβοκινητήρων. Τέλος, υπάρχει είσοδος για δέκτη τηλεκατεύθυνσης σήματος PPM Sum. Η PX4IO διαθέτει, επίσης, χειροκίνητο διακόπτη εναλλαγής έλεγχου μεταξύ χρήστη-μικροελεγκτή.

Συγκεντρωτικά τα χαρακτηριστικά της PX4IO, όπως παρουσιάζονται στο επίσημο site είναι:

- 24 Mhz Cortex-M3 failsafe microcontroller
- 6-18V wide supply in, 5V / 2 A output
- Reverse polarity protection on all power inputs
- 8 high-speed servo outputs (up to 400 Hz)
- Futaba S.Bus compatible servo output
- PPM, Spektrum and Futaba S.Bus compatible receiver inputs (PPM in sum signal format, many compatible receivers, all channels on one connector)
- 2x 0-40 V, 1 A solid-state relays (MOSFET)
- 2x 5 V, 500mA current-limited, switched 5V power outputs
- Analog port with voltage divider (differential pressure sensors)
- PX4 Expansion bus (stacked on PX4FMU)



2-3. Οι δύο όψεις μιας πλακέτας μικροελεγκτή PX4IO.

Λειτουργικό σύστημα του PX4

Ο μικροελεγκτής PX4, όπως ήδη αναφέρθηκε, χρησιμοποιεί ένα λειτουργικό σύστημα πραγματικού χρόνου, μία συλλογή δηλαδή βασικών προγραμμάτων, η οποία ελέγχει τη λειτουργία του μικροελεγκτή συνολικά και χρησιμοποιείται ως υπόβαθρο για την εκτέλεση όλων των υπόλοιπων προγραμμάτων, για τη διαχείριση των περιφερειακών συσκευών και για την εξασφάλιση της επικοινωνίας μεταξύ χρήστη και μικροελεγκτή. Στην πράξη πρόκειται για ένα επίπεδο λογισμικού που μεσολαβεί μεταξύ του υλικού και των εκτελούμενων προγραμμάτων σε έναν μικροελεγκτή. Αποτελείται από ένα σύνολο μηχανισμών μέσω των οποίων επιτυγχάνεται η αυτόματη διαχείριση των πόρων ενός μικροελεγκτή και η ελεγχόμενη κατανομή τους στις εκτελούμενες εφαρμογές, ώστε οι τελευταίες να είναι σε θέση να προσπελάσουν εύκολα τους πόρους και τις συσκευές του συστήματος χωρίς να χρειάζεται να γνωρίζουν με ακρίβεια τη δομή του υλικού (hardware). Είναι το πρώτο πρόγραμμα που καλείται να εκτελεστεί την στιγμή εκκίνησης του μικροελεγκτή.

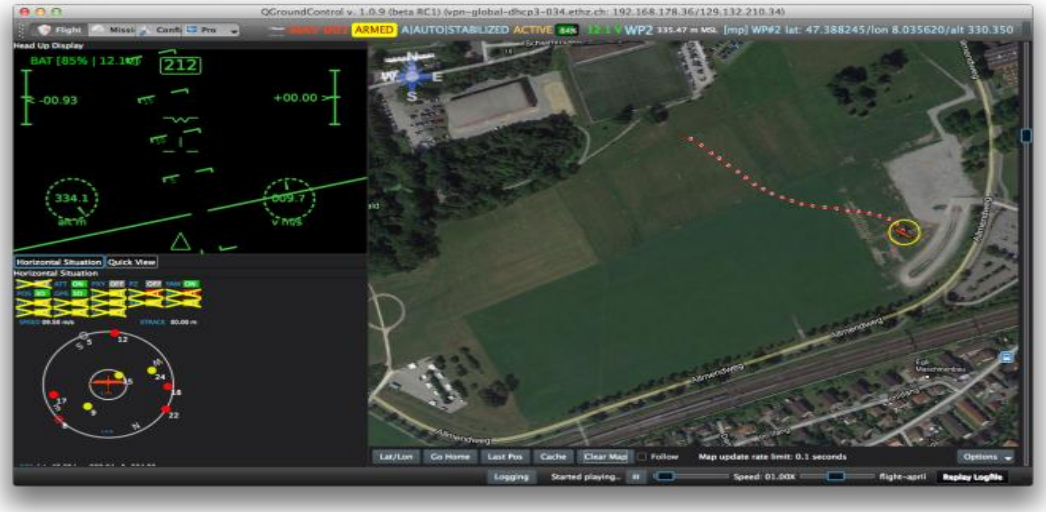
Συγκεκριμένα το λειτουργικό σύστημα πραγματικού χρόνου, το Nuttx, είναι ένα σύστημα ειδικού σκοπού και χρησιμοποιείται όταν υπάρχουν αυστηρές χρονικές απαιτήσεις για την ολοκλήρωση μιας λειτουργιάς. Ως τέτοιο εξυπηρετεί εφαρμογές πραγματικού χρόνου, όπως τους υπολογισμούς των δεδομένων περιβάλλοντος, θέσης κ.ά. από τα αισθητήρια για την διαμόρφωση επιθυμητής πορείας και συμπεριφοράς του αεροπλάνου, για την αλλαγή του σημείου διεύθυνσης πτήσης, αλλά και για την γενική λειτουργία του αυτόματου πιλότου, ενώ ήδη είναι στον αέρα. Ένα σύστημα πραγματικού χρόνου θεωρείται ότι λειτουργεί σωστά, μόνο όταν επιστρέφει ορθά αποτελέσματα μέσα στους αυστηρούς χρονικούς περιορισμούς που έχουν καθοριστεί.

Το λογισμικό του λειτουργικού συστήματος πραγματικού χρόνου σε έναν μικροελεγκτή PX4 παρέχει επιπλέον τις βιβλιοθήκες που επιτρέπουν να εκτελεστεί η διαδικασία του αυτόματου πιλότου. Το λογισμικό του είναι open source, δίνει δηλαδή την δυνατότητα επέμβασης σε αυτό και μεταβολής του κατά βούληση, κατά συνέπεια ο κώδικας μπορεί να διαφοροποιηθεί με σκοπό την δημιουργία λειτουργιών-διαδικασιών

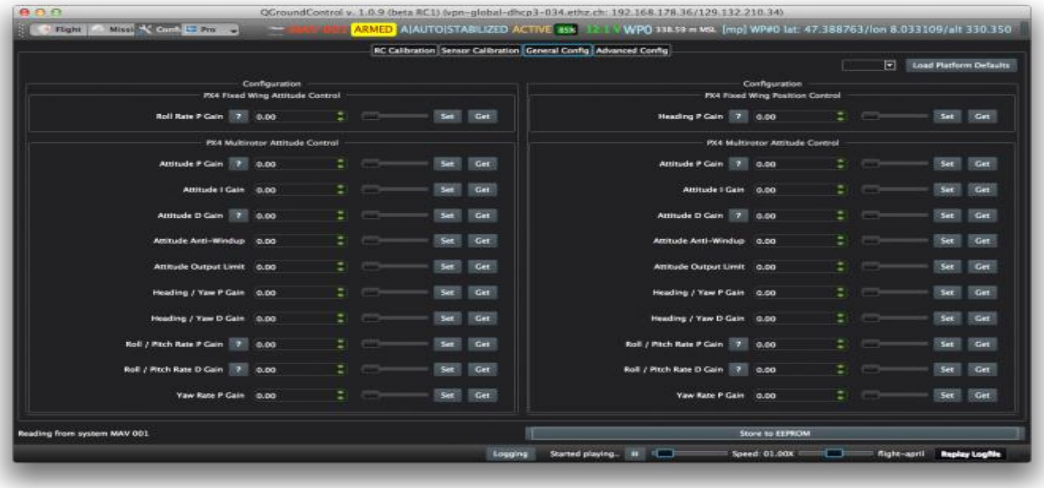
που δεν υπάρχουν στις υπάρχουσες βιβλιοθήκες, ενώ παράλληλα μπορούν και να βελτιωθούν. Χαρακτηριστικές είναι οι βιβλιοθήκες, οι επιφορτισμένες με την εκτίμηση της θέσης του σκάφους (position estimator), του ύψους του (fixedwing_att_control), καθώς και των παραμέτρων που προκύπτουν μετά από την επεξεργασία των δεδομένων των αισθητηρίων από το φίλτρο Kalman (att_pos_estimator_ekf, attitude_estimator_ekf). Οι βιβλιοθήκες, όπως και το σύνολο του κώδικα είναι γραμμένα στις γλώσσες προγραμματισμού C και C++. Ο κώδικας είναι αποθηκευμένος στο Github όπου γίνονται αλλαγές, διορθώσεις ή προσθήκες από τους προγραμματιστές του PX4, καθώς και από άλλους χρήστες της πλατφόρμας. Η επεξεργασία του κώδικα γίνεται με το πρόγραμμα Eclipse για Windows και η μεταφόρτωσή του στον PX4 γίνεται με την σύνδεση του micro USB καλωδίου.

Το πλαίσιο της επικοινωνίας του μικροελεγκτή PX4 με τον χειριστή αναλαμβάνει το πρόγραμμα Qground Control (version 1 & 2). Πρόκειται για ένα open source σταθμό ελέγχου MAV με γενική άδεια δημόσιας χρήσης ελεύθερου λογισμικού GPLv3, που δημιουργήθηκε από την ίδια ομάδα μεταπτυχιακών φοιτητών του τμήματος Computer Vision and Geometry group για την χρήση του με τις πλατφόρμες Autopilot (PX4, Ardupilot, κ.α) και πλέον αναπτύσσεται με μια κοινή προσπάθεια όλης της κοινότητας που το χρησιμοποιεί.

Το βασικό πρωτόκολλο επικοινωνίας του Qground Control είναι το MAVLINK, ένα δυαδικό σειριακό πρωτόκολλο, μέσω του οποίου το Qground Control μπορεί να λάβει δεδομένα για τον προσανατολισμό, την θέση του από το GPS, και την ταχύτητα του οχήματος, καθώς και να κάνει Κυκλικό Έλεγχο Πλεονασμού (Cyclic Redundancy Check, CRC), μια τεχνική ανίχνευσης σφαλμάτων κατά την διάρκεια μετάδοσης δεδομένων.



2-4. Το QGround Control σε άμεση επικοινωνία με τον μικροελεγκτή.



2-5. Ρύθμιση παραμέτρων ελεγκτών μέσω του QGround Control.

2-6. Καθορισμός checkpoint για την πτήση του MAV.

Βασικά χαρακτηριστικά του QGround Control είναι:

- Open source πρωτόκολλο επικοινωνίας MAV (Mavlink) με ελαφρές συναρτήσεις serialization για μικροελεγκτές.
- Υποστήριξη Windows / Linux / MacOS λειτουργικών συστημάτων.
- 2/3D εναέριοι χάρτες (υποστήριξη google earth) με drag-and-drop σημεία πορείας του σκάφους.
- Δυνατότητα ελέγχου της πορείας του σκάφους μέσω των σημείων πορείας και της εναλλαγής τους σε πραγματικό χρόνο.
- Γραφική αναπαράσταση των δεδομένων των αισθητηρίων και της τηλεμετρίας σε πραγματικό χρόνο.

- Υποστήριξη πολλαπλών autopilot πλατφορμών (PXIMU, ArduPilotMega, SLUGS, MatrixPilot / UAVDevBoard, κ.α.).
- Το πρωτόκολλο Mavlink υποστηρίζει μέχρι 255 οχήματα παράλληλα.
- Δυνατότητα μετάδοσης και λήψης σήματος video.

Εγκατάσταση PX4 Toolchain

Το PX4 Toolchain είναι το αρχείο που περιέχει το σύνολο των απαραίτητων εργαλείων για τον χειρισμό και την διαχείρηση του μικροελεγκτή PX4. Τα προγράμματα που το αποτελούν είναι οι drivers για την αναγνώριση του μικροελεγκτή από τον Η/Υ, το PX4 Software Download για την λήψη του firmware από το GitHub, το PX4 Eclipse για την τροποποίηση και εγκατάσταση του firmware στον μικροελεγκτή και το Teraterm για την σειριακή επικοινωνία μεταξύ υπολογιστή και λειτουργικού συστήματος του PX4.

Ύστερα από την εγκατάσταση του PX4 Toolchain από την σελίδα (http://www.inf.ethz.ch/personal/lomeier/downloads/px4_toolchain_installer_v06_win.exe) επιλέγουμε το πρόγραμμα PX4 Software Download ώστε να γίνει η λήψη του firmware στον φάκελο C:\px4. Στη συνέχεια εκκινούμε το PX4 Eclipse και επιλέγουμε File →New → Makefile Project with Existing Code. Στο νέο παράθυρο που εμφανίζεται επιλέγουμε Browse και τον φάκελο Firmware από την διεύθυνση C:\px4\Firmware και Ok. Στη συνέχεια επιλέγουμε το Cross GCC και Finish. Περιμένουμε μέχρι το Eclipse να αρχικοποιήσει τον φάκελο firmware. Κατόπιν μπορούμε να επεξεργαστούμε ή να τροποποιήσουμε τον κώδικα και να τον αποστείλουμε στον μικροελεγκτή. Για να γίνει δυνατή η μετατροπή του κώδικα σε εκτελέσιμο αρχείο θα πρέπει να γίνει δημιουργία των επόμενων targets. Στο δεξί παράθυρο του Eclipse επιλέγουμε Make Targets και κάνουμε δεξί κλίκ στον φάκελο Firmware επιλέγοντας add make target και δημιουργώντας το archives. Πρόκειται για την εντολή εκκίνησης της διαδικασίας μετατροπής των γραμμών σε εκτελέσιμο κώδικα ώστε να ενσωματωθεί στο NuttX. Με παρόμοια διαδικασία φτιάχνουμε τα targets: το all για την μετατροπή σε εκτελέσιμο κώδικα του Autopilot software, το distclean για την εκκαθάριση μη εκτελέσιμων γραμμών κώδικα από το NuttX, το clean για την εκκαθάριση μη εκτελέσιμων γραμμών κώδικα από το Autopilot

software και το upload px4fmu-v1_default για την μεταφορά του κώδικα στην πλακέτα. Η σειρά εκτέλεσης των ενεργειών, ύστερα από τροποποίηση του κώδικα είναι clean, distclean, archives, all και τέλος upload px4fmu-v1_default. Μετά την επιλογή της τελευταίας ενέργειας θα παρουσιαστεί μήνυμα στο κάτω παράθυρο Console για αναμονή σύνδεσης της πλακέτας του μικροελεγκτή σε μία θύρα COM. Πραγματοποιώντας την σύνδεση θα έχουμε την μεταφορά του προγράμματος και την επιβεβαίωση λήψης με τον χαρακτηριστικό ήχο εκκίνησης του μικροελεγκτή.

AUTOPILOT

Η τεχνολογική ανάπτυξη των UAV τα τελευταία χρόνια είναι ραγδαία και αφορά κυρίως στην κατασκευή μπαταριών υψηλής πυκνότητας, ασύρματων συσκευών μεγάλης εμβέλειας με μικρή κατανάλωση, οικονομικών ατράκτων, ισχυρότατων μικροελεγκτών και ηλεκτρικών κινητήρων, με αποτέλεσμα τα UAVs να γίνουν προσιτά και ευκολόχρηστα στο ευρύ κοινό. Τα μικρά αυτά αεροσκάφη σχεδιασμένα για χαμηλές πτήσεις με κύριο στόχο την παρατήρηση της επιφάνειας της Γής, ενέχουν τον κίνδυνο πρόσκρουσης και για την αποφυγή τέτοιων προβλημάτων απαιτούν από τον χειριστή ή τον μικροελεγκτή την δυνατότητα άμεσων αντιδράσεων και ακαριαίων χειρισμών.

Βασικές αρχές ελέγχου

Ένα αεροσκάφος έχει την δυνατότητα να περιστραφεί γύρω από τρείς άξονες (x,y,z), που έχουν ως σημείο αναφοράς το κέντρο βάρους του σκάφους. Αυτές οι τρείς περιστροφές (roll, pitch, yaw) καθορίζονται από την κίνηση των πτερυγίων του αεροσκάφους, ώστε να γέρνει δεξιά και αριστερά (στο εξής roll), να βυθίζεται ή να ανυψώνεται η μύτη του (στο εξής pitch) και να στρέφεται δεξιά και αριστερά (στο εξής yaw).

Οι δυνατές, εν πτήση, ενέργειες περιορίζονται στην κίνηση των οριζοντίων φτερών, στην αντίστοιχη του πηδαλίου και τέλος, στην αυξομείωση της ταχύτητας, ικανές παρόλα αυτά για τον πλήρη έλεγχο του αεροσκάφους με την χρήση μίας από αυτές ή σε συνδυασμό τους. Με αυτές άλλωστε τις ενέργειες μπορούμε να πραγματοποιήσουμε την περιστροφή του αεροσκάφους γύρω από τους τρείς προαναφερθέντες άξονες (x,y,z).

Αξονες αναφοράς

Η μελέτη και η κατανόηση της συμπεριφοράς των UAV θα πραγματοποιηθεί μέσω των εξής συστημάτων συντεταγμένων: 1. Inertial Frame, 2. Body Frame, 3. Wind Frame και 4. Stability Frame

Στο σύστημα συντεταγμένων, inertial frame, η ανάλυση των δυνάμεων θα γίνει με την εφαρμογή των εξισώσεων του Νεύτωνα. Με αυτό το σύστημα θα οριστεί ο σχεδιασμός της αποστολής του αεροσκάφους, καθώς και οι πληροφορίες των χαρτών για την παρακολούθηση της πορείας του UAV, δεδομένου ότι σε αυτό το σύστημα θα λαμβάνονται και οι μετρήσεις από το GPS και το μαγνητόμετρο. Ταυτόχρονα είναι δυνατός ο υπολογισμός της γωνίας με την οποία ταξιδεύει το UAV, αλλά και η ταχύτητα πτήσης του.

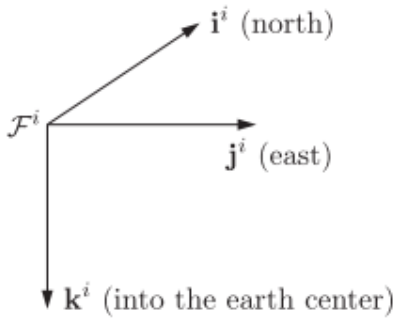
Το δεύτερο σύστημα, που ονομάζεται body frame, καθιστά εύκολη την μελέτη της κίνησης του αεροσκάφους, μέσα από την ανάλυση των δυνάμεων και των ροπών, που ασκούνται στο σώμα του. Από αυτό το σύστημα άλλωστε προκύπτουν οι μετρήσεις που δίνουν ορισμένα όργανα, όπως για παράδειγμα το γυροσκόπιο ή το επιταχυνσιόμετρο, με βάση το κέντρο βάρους του αεροσκάφους.

Το τρίτο κατά σειρά σύστημα, το wind frame, χρησιμοποιείται για τις μετρήσεις με βάση τον άνεμο. Οι δυνάμεις άντωσης, η αντίσταση του αέρα και οι δυνάμεις που περιστρέφουν το σώμα του αεροσκάφους, εκτιμώνται με βάση την ροή του αέρα γύρω από το σώμα του. Τέλος, το τέταρτο σύστημα, το stability frame είναι ο συνδυασμός των δύο προηγούμενων, του wind frame και του body frame.

Inertial frame

Το σύστημα συντεταγμένων inertial frame, γνωστό και με την ονομασία NED (North-East-Down system), έχει ως αναφορά ένα σταθερό σημείο στην επιφάνεια της Γής, όπως αυτό ορίζεται από τον παρατηρητή. Το σταθερό αυτό σημείο δηλώνει την αρχή των αξόνων και με βάση τον κανόνα του δεξιού χεριού, όπως φαίνεται και στο

σχέδιο (3-1), ορίζεται πως το διάνυσμα i^i δείχνει τον Βορρά, το αντίστοιχο j^i την Ανατολή, ενώ το διάνυσμα k^i δείχνει το κέντρο της Γής.

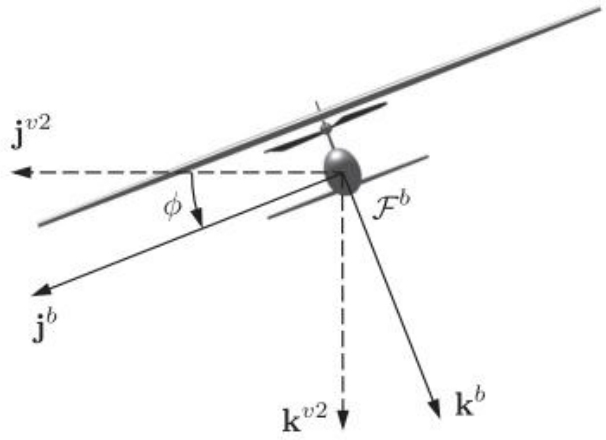


3-1. Inertial Frame [πηγή: Small Unmanned Aircraft Theory and Practice].

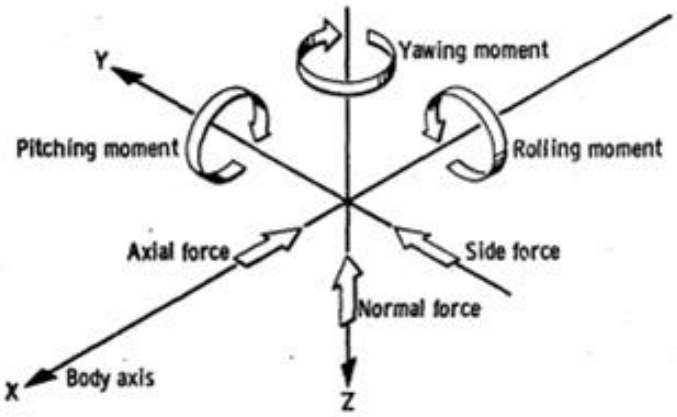
Body frame

Το δεύτερο σύστημα συντεταγμένων, body frame, έχει την δυνατότητα να κινείται μαζί με το σώμα του αεροσκάφους. Αρχή των συντεταγμένων είναι το κέντρο βάρους του UAV, ενώ οι άξονες είναι ορισμένοι με βάση το σώμα του, με αποτέλεσμα να μπορούμε να αναλύσουμε οποιαδήποτε δύναμη ασκείται σε αυτό και να εξάγουμε άμεσα συμπεράσματα.

Το σύστημα αυτό χρησιμοποιείται περισσότερο για τις μετρήσεις που αφορούν στο σύστημα πρόωσης του UAV, ώστε να αναλύονται οι δυνάμεις της αντίστασης του αέρα, καθώς και της ώθησης των κινητήρων. Στο σχέδιο (3-2) παρατηρούμε πως το διάνυσμα i^b είναι συνέχεια του άκρου της ατράκτου, το διάνυσμα j^b είναι προέκταση της δεξιάς πτέρυγας και το διάνυσμα k^b είναι κάθετο από την κοιλιά του αεροσκάφους. Παρατηρούμε ότι και αυτό, το διάνυσμα k^b με την σειρά του είναι το vehicle frame-2 μετατοπισμένο αριστερόστροφα κατά γωνία φ.



3-2.Body frame [πηγή: Small Unmanned Aircraft Theory and Practice].



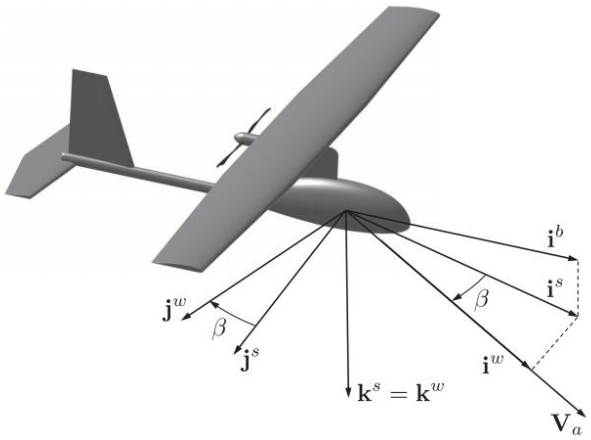
3-3. Body frame axis system [πηγή: Exploring in Aeronautics (NASA)].

Wind frame

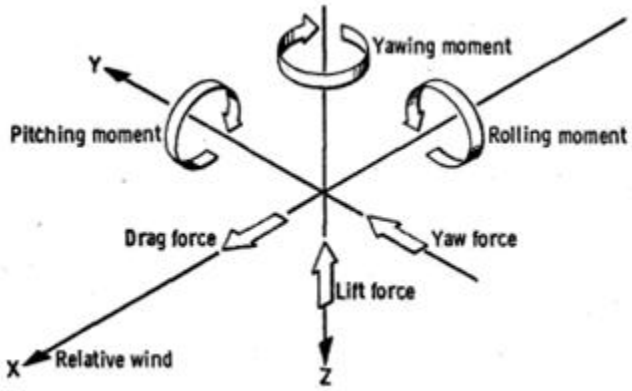
Το σύστημα συντεταγμένων, wind frame, βασίζεται στον ορισμό της γωνίας εκτροπής (sideslip angle). Παρατηρώντας ένα σκάφος κατακόρυφα από πάνω (3-4) διαπιστώνεται ότι το διάνυσμα της πορείας του μπορεί να είναι διαφορετικό από την νοητή προέκταση της γραμμής που ενώνει τα δύο άκρα της ατράκτου. Η γωνία, που

δημιουργείται, ονομάζεται sideslip angle, συμβολίζεται με το ελληνικό γράμμα β και είναι συνέπεια των δυνάμεων του αέρα που ενεργούν πάνω στο σώμα του αεροσκάφους, με αποτέλεσμα αυτό να πλαγιολισθαίνει για να έχει την επιθυμητή πορεία.

Σύμφωνα με τα παραπάνω το σύστημα wind frame ορίζεται σύμφωνα με το κέντρο βάρους του αεροσκάφους και το διάνυσμα i^w με την προβολή του διανύσματος Vα στο επίπεδο i, το διάνυσμα j^w να είναι ίδιο με το διάνυσμα j^s μετατοπισμένο κατά γωνία β, ενώ το διάνυσμα k^w να είναι ίδιο με το k^s.



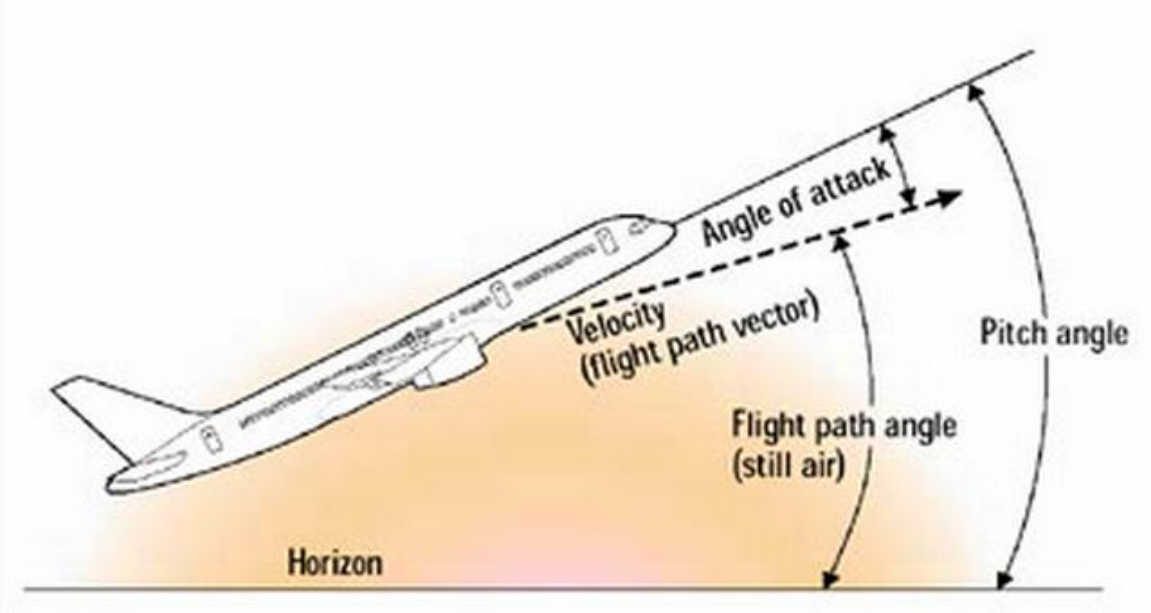
3-4. Wind frame [πηγή: Small Unmanned Aircraft Theory and Practice].



3-5. Wind frame axis system [πηγή:Exploring in Aeronautics (NASA)].

Stability frame

Το σύστημα συντεταγμένων, stability frame, βασίζεται στη γωνία επίθεσης (angle of attack) και για το λόγο αυτό κρίνεται απαραίτητη η ερμηνεία αυτής της έννοιας (angle of attack).

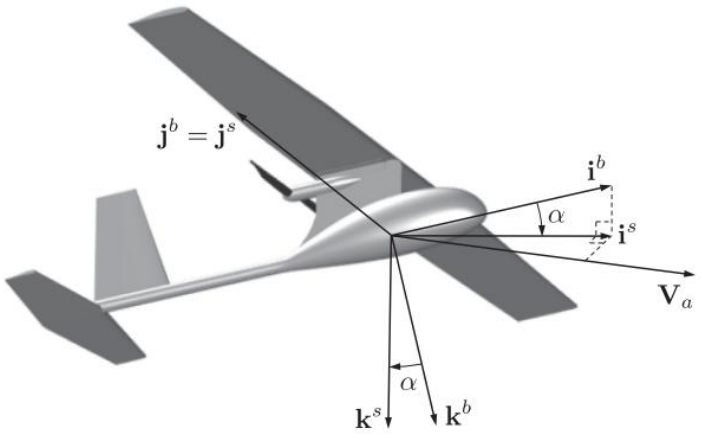


3-6.Angle of attack [πηγή:Boeing Aeromagazine].

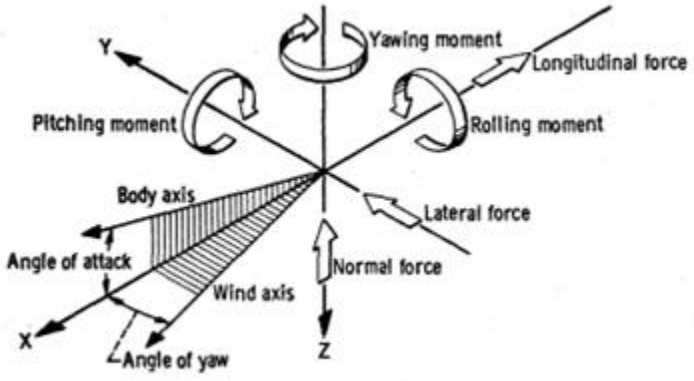
Το διάνυσμα της κίνησης ενός αεροσκάφους μέσα στον αέρα (Vα), μπορεί να διαφέρει από το διάνυσμα που προβάλει κάθετα από το άκρο της ατράκτου. Οι δυνάμεις του αέρα, που ασκούνται πάνω στο σώμα του αεροσκάφους, έχουν ως αποτέλεσμα το άκρο της ατράκτου να στοχεύει σε άλλη κατεύθυνση από αυτή στην οποία κινείται το αεροσκάφος. Στο σχέδιο (3-6) όπου απεικονίζεται το σκάφος κατά τομή, δηλώνεται σαφώς η γωνία που σχηματίζεται από το διάνυσμα της ταχύτητας (πορεία) και από την νοητή προέκταση της μύτης του αεροσκάφους. Η γωνία αυτή ορίζεται ως angle of attack (AOA) και συμβολίζεται με το ελληνικό γράμμα α. Ανάλογα με τις δυνάμεις του αέρα,

ένα αεροσκάφος μπορεί να βρίσκεται κανονικά σε πτήση, έχοντας θετική ή και αρνητική angle of attack.

Όταν θέλουμε να ορίσουμε ως θετική την angle of attack είμαστε υποχρεωμένοι να θεωρήσουμε δεξιόστροφη την περιστροφή του body frame. Προκύπτει ότι στο σύστημα stability frame το διάνυσμα i^s είναι ίδιο με την προβολή του διανύσματος (Vα) της πορείας του αεροσκάφους στο επίπεδο i, το j^s είναι ίδιο με το j^b ενώ το k^s είναι το k^b μετατοπιζόμενο κατά θετική γωνία α.



3-7. Stability frame [πηγή: Small Unmanned Aircraft Theory and Practice].



3-8. Stability frame axis system [πηγή:Exploring in Aeronautics (NASA)].

Η μαθηματική περιγραφή της περιστροφής ενός διανύσματος

Τα παραπάνω συστήματα αναφοράς αξόνων σχετίζονται μεταξύ τους για να γίνεται ο έλεγχος της μετατόπισης του σκάφους ανάμεσα σε δύο χρονικές στιγμές. Η μαθηματική αναγωγή, που χρειάζεται να γίνει από το ένα σύστημα στο άλλο, είναι δυνατή με μόνη προϋπόθεση την γνώση των παραμέτρων της μετατόπισης του ενός συστήματος αξόνων στο άλλο. Εν προκειμένω, την γωνία με την οποία θα περιστραφεί το αρχικό σύστημα, ώστε να καταλήξει στο τελικό σύστημα αντίστοιχα, καθώς και την φορά της περιστροφής.

Στην περίπτωση των UAV, το ενδιαφέρον επικεντρώνεται στη μετατροπή του συστήματος συντεταγμένων body frame, σύμφωνα με το οποίο είναι εκφρασμένα τα μεγέθη που μετράνε τα αισθητήρια, στο σύστημα συντεταγμένων inertial frame, το σύστημα δηλαδή συντεταγμένων ενός εξωτερικού παρατηρητή.

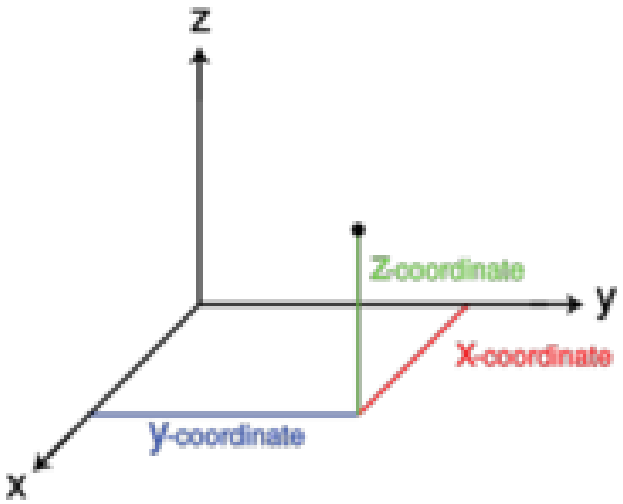
Η τελική μετατροπή θα προκύψει από ένα σύνολο διαδοχικών μεταβάσεων από τα συστήματα body frame, vehicle-2 frame, vehicle-1 frame, καταλήγοντας στο inertial frame. Τα συστήματα vehicle-1 frame και vehicle-2 frame είναι παράγωγα του body frame και προκύπτουν από αυτό με μια δεξιόστροφη περιστροφή κάθε φορά του προηγούμενου συστήματος. Σκοπός του κεφαλαίου αυτού είναι η επεξήγηση των βασικών αρχών που αφορούν την μετατροπή μεταξύ των συστημάτων αναφοράς αξόνων, δηλαδή της περιστροφής ενός διανύσματος και της μαθηματικής περιγραφής της νέας του θέσης.

Η μετάβαση αυτή μαθηματικά μπορεί να γίνει με τη χρήση των πινάκων περιστροφής, που όπως ήδη αναφέρθηκε, για να τους χρησιμοποιήσουμε, χρειαζόμαστε την γωνία και την φορά της περιστροφής.

Δεξιόστροφη και αριστερόστροφη περιστροφή

Όταν στον τρισδιάστατο χώρο ένα διάνυσμα στρέφεται, σημαίνει ότι κινείται πάνω σε ένα επίπεδο. Παραμένει σταθερό ως προς έναν από τους τρεις άξονες και κινείται πάνω στο επίπεδο που ορίζουν οι άλλοι δύο. Οι κανόνες που περιγράφουν την

δεξιόστροφη ή αριστερόστροφη στροφή ενός διανύσματος είναι πολλοί. Στην παρούσα μελέτη κρίθηκε καταλληλότερος και παρατίθεται ο κανόνας των δεικτών του ρολογιού. Έστω ένα τρισδιάστατο σύστημα συντεταγμένων x,y,z.



3-9. Σύστημα συντεταγμένων X,Y,Z.

Ό άξονας X ορίζει την ‘’μέσα-έξω’’ κατεύθυνση, ο άξονας Z την ‘’πάνω κάτω’’ και ο άξονας Y την ‘’αριστερά-δεξιά’’. Έστω τώρα ένα διάνυσμα το οποίο έχει αρχή την αρχή των αξόνων και εκτείνεται πάνω στον άξονα Z (προς τα πάνω). Έστω ότι αυτό το διάνυσμα στρέφεται ως προς (ή γύρω από) τον άξονα X (δηλαδή παραμένει σταθερό ως προς αυτό τον άξονα και μετατοπίζεται πάνω στο επίπεδο που ορίζουν οι άξονες Y και Z). Υποθέτουμε ότι ξεκινά από την αρχική του θέση (αρχή το [0,0,0] και ενώ το διάνυσμα βρίσκεται πάνω στον άξονα Z) και κινείται με την φορά των δεικτών του ρολογιού πάνω στο επίπεδο YZ, τείνει δηλαδή να βρεθεί πάνω στον θετικό άξονα Y. Αυτή η κίνηση, ανάλογη με την φορά των δεικτών του ρολογιού, ονομάζεται

αριστερόστροφη και η αντίθετή της (δηλαδή αν ξεκινούσε από το ίδιο σημείο και κατέληγε στον αρνητικό άξονα Υ) δεξιόστροφη.

Πίνακες περιστροφής

Οι πίνακες περιστροφής είναι (3x3) και συμπληρώνονται ως εξής:

Αρχικά πρέπει να ξέρουμε την γωνία με την οποία περιστράφηκε το διάνυσμα πάνω στο επίπεδο (αυτό το πληροφορούμαστε από την έξοδο του γυροσκοπίου). Υποθέτουμε ότι σε ένα σύστημα αξόνων X,Y,Z ένα διάνυσμα περιστράφηκε δεξιόστροφα κατά γωνία θ ως προς τον άξονα Z. Ορίζουμε το κάθε κελί του πίνακα του πίνακα (1) με τις εξής συντεταγμένες:

Πίνακας 1.

XX	XY	XZ
YX	YY	YZ
ZX	ZY	ZZ

Στο στοιχείο που περιγράφεται μόνο από τον άξονα γύρω από τον οποίο γίνεται η περιστροφή δίνουμε τιμή 1 (δηλαδή το ZZ) και στα υπόλοιπα στοιχεία στα οποία υπάρχει ο σταθερός άξονας δίνουμε τιμή 0 (δηλ. στα XZ,YZ,ZX,ZY). Προκύπτει δηλαδή ο ακόλουθος πίνακας (2):

Πίνακας 2.

XX	XY	0
YX	YY	0
0	0	1

Τα υπόλοιπα τέσσερα στοιχεία συμπληρώνονται βάζοντας στην πρώτη και στη δεύτερη διαθέσιμη γραμμή τα στοιχεία Cosθ και Sinθ (πίνακας 3), ως εξής:

Πίνακας 3.

Cosθ	Sinθ	0
Sinθ	Cosθ	0
0	0	1

Τέλος, αν η περιστροφή είναι δεξιόστροφη βάζουμε αρνητικό πρόσημο στο Sin που βρίσκεται στη γραμμή πάνω από τις τιμές 0 και 1 (αν οι τιμές 0 και 1 βρίσκονται στην πάνω ($1^{η}$) γραμμή γιατί η περιστροφή ήταν γύρω από τον X άξονα τότε το πρόσημο "–" πάει στο Sin της ($3^{ης}$ γραμμής) και προκύπτει ο τελικός πίνακας (4):

Πίνακας 4.

Cosθ	Sinθ	0
-Sinθ	Cosθ	0
0	0	1

Αν η περιστροφή ήταν αριστερόστροφη θα βάζαμε αρνητικό πρόσημο στο Sin της γραμμής κάτω από τις τιμές 0 και 1. Ο πίνακας (5) για αριστερή περιστροφή του ίδιου διανύσματος στο ίδιο επίπεδο και κατά ίδια γωνία θα ήταν:

Πίνακας 5.

Cosθ	-Sinθ	0
Sinθ	Cosθ	0
0	0	1

Η γραμμή κάτω από τις τιμές 0 και 1 αφού δεν υπάρχει θεωρείται ως η $1^{η}$ γραμμή.

Ο πίνακας περιστροφής συμβολίζεται με R_a^b και ορίζει την στροφή με αρχική θέση το a και τελική τη b. Ένα διάνυσμα λοιπόν, που περιστράφηκε, περιγράφεται τελικώς από την αρχική του θέση και τον πίνακα περιστροφής.

$$R_{τελ} = R_{αρχ} * R_{περιστροφής}$$

Ισχύουν οι ιδιότητες: i. $(R_a^b)^{-1} = (R_a^b)^T = R_b^a$

δηλαδή, αν πάρουμε τον πίνακα, που περιγράφει την μετάβαση από το a στο b, και τον αντιστρέψουμε, παίρνουμε τον πίνακα που περιγράφει την μετάβαση από το b στο a.

ii. $R_b^c R_a^b = R_a^c$

Αν πολλαπλασιάσουμε τον πίνακα που περιγράφει την μετάβαση από το a στο b με τον πίνακα που περιγράφει την μετάβαση από b στο c, παίρνουμε σαν αποτέλεσμα τον πίνακα με την μετάβαση από το a στο c.

iii. $det(R_a^b) = 1$

Η ορίζουσα είναι 1.

INERTIAL MEASUREMENT UNIT

Το Inertial Measurement Unit (IMU) είναι η ηλεκτρονική συσκευή μέσω της οποίας λαμβάνονται πληροφορίες σχετικά με την τοποθεσία του αεροσκάφους MAV, την τοποθέτησή του στον χώρο και τις δυνάμεις που ασκούνται σε αυτό συνεχώς. Πρόκειται δηλαδή για τον συνδυασμό οργάνων, όπως επιταχυνσιόμετρων, γυροσκόπιων, και ορισμένες φορές μαγνητόμετρων, τα οποία προσφέρουν τις απαραίτητες αυτές πληροφορίες για ένα αεροσκάφος. Τα τρία αυτά όργανα προτιμάται να χρησιμοποιούνται ταυτόχρονα, δεδομένου ότι το ένα συμπληρώνει και επαληθεύει τις μετρήσεις του άλλου, με αποτέλεσμα τη μικρότερη δυνατή απόκλιση στις μετρήσεις και στους υπολογισμούς του κάθε οργάνου, που είναι δυνατόν να προκύψουν λόγω σφαλμάτων από ηλεκτρικά πεδία, μηχανικούς θορύβους ή άλλες αιτίες. Πρόσφατα ξεκίνησε και βρίσκεται σε συνεχή εξέλιξη η κατασκευή συσκευών GPS με την προσθήκη του συστήματος IMU. Το αποτέλεσμα αυτού του συνδυασμού είναι η δυνατότητα συνεχούς υπολογισμού της θέσης του συστήματος, ακόμα και όταν το GPS χάνει το σήμα του.

Όργανα μετρήσεως

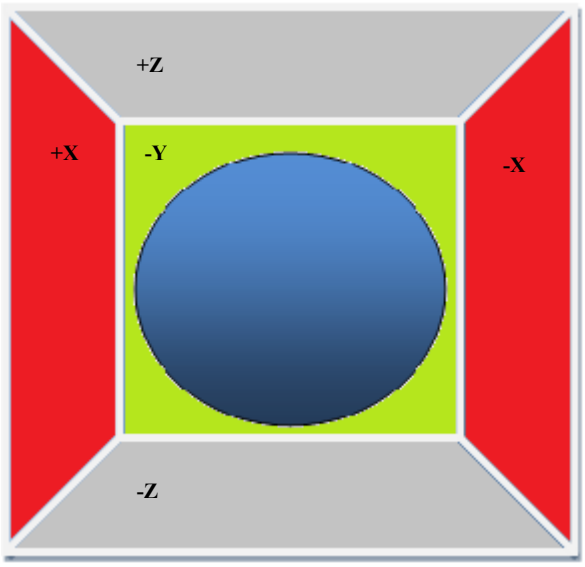
Επιταχυνσιόμετρο

Το επιταχυνσιόμετρο είναι μια ηλεκτρομηχανική συσκευή μέτρησης της επιτάχυνσης των δυνάμεων που ασκούνται πάνω σε ένα αεροσκάφος. Οι δυνάμεις αυτές μπορεί να είναι συνεχείς, όπως η βαρύτητα ή και δυναμικές, όπως η δόνηση της συσκευής.

Για να εξηγήσουμε τον τρόπο λειτουργίας αυτής της συσκευής θα παραθέσουμε ορισμένα συγκεκριμένα παραδείγματα.

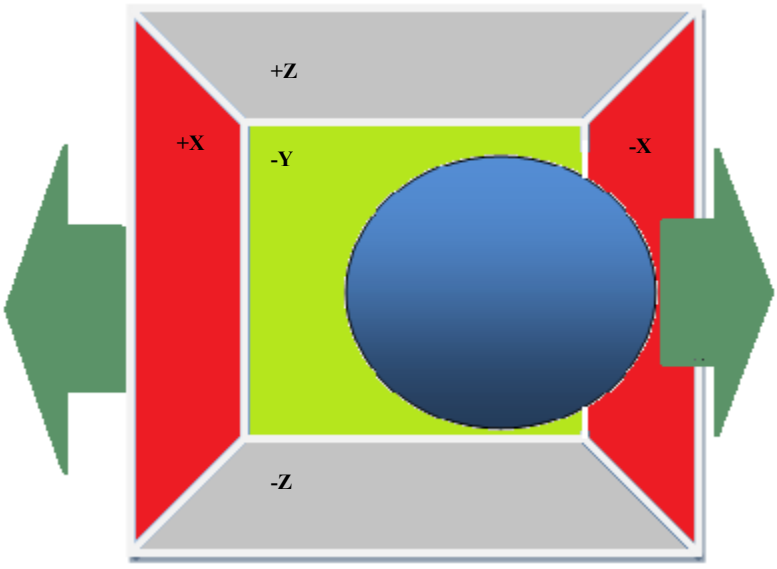
Στο πρώτο παράδειγμα φανταζόμαστε ένα κουτί σε σχήμα κύβου, με μία σφαίρα στο εσωτερικό του, στο οποίο δεν ασκείται καμία απολύτως δύναμη,

συμπεριλαμβανομένης και της δύναμης της βαρύτητας, σαν να το κουτί βρίσκεται στο διάστημα. Οι απέναντι πλευρές του κουτιού ορίζουν ένα ζευγάρι και το κάθε ένα από αυτά αφορά σε έναν άξονα (στην εικόνα σημειώνεται μόνο η πλευρά -Y, ενώ η αντίστοιχη +Y έχει αφαιρεθεί για να βλέπουμε το εσωτερικό του κουτιού). Θεωρούμε ως δεδομένο ότι όλα τα τοιχώματα είναι ευαίσθητα στη πίεση, οπότε η σφαίρα ισορροπεί στο κέντρο του κουτιού.



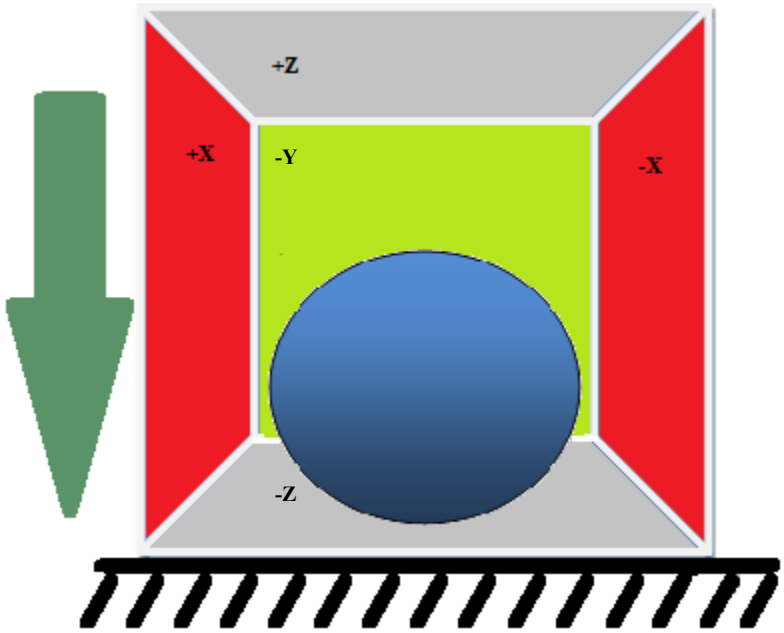
4-1.Κουτί με σφαίρα στο εσωτερικό του.

Αν κινήσουμε το κουτί προς τα αριστερά, χρησιμοποιώντας μια επιτάχυνση $1g=9,81m/s^2$, θα προκαλέσουμε πρόσκρουση της σφαίρας στο τοίχωμα (-X), με πίεση ίση με -1g, όπως φαίνεται στο σχήμα 2-2. Το επιταχυνσιόμετρο θα ανιχνεύσει μια δύναμη, στη πραγματικότητα αντίθετης κατεύθυνσης από το διάνυσμα της επιτάχυνσης.



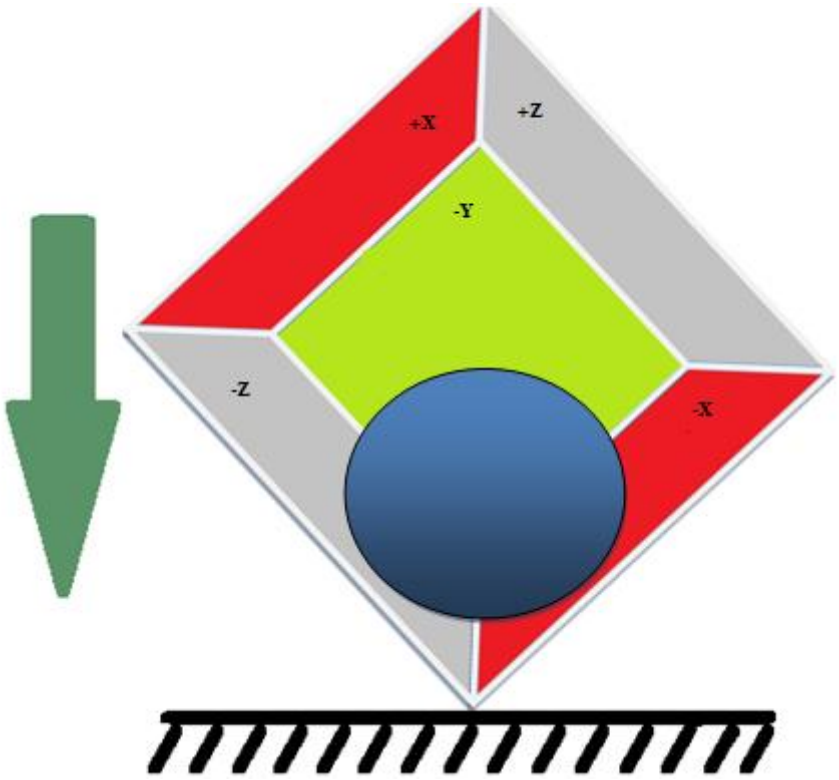
4-2. Μετακίνηση του κουτιού προς τα αριστερά.

Αν μεταφέρουμε το κουτί στην επιφάνεια της Γής, θα παρατηρήσουμε αντίστοιχα ότι η σφαίρα θα πιέζει, λόγω της βαρύτητας, το τοίχωμα (-Z), με πίεση ίση με 1g, όπως παρουσιάζεται στο σχήμα (3-3). Σε αυτή την περίπτωση το κουτί δεν μετακινείται προς μια κατεύθυνση, αλλά δέχεται την μόνιμη επίδραση της βαρύτητας πάνω στη σφαίρα. Υπάρχουν και άλλες περιπτώσεις όπου το κουτί δεν μετακινείται, αλλά έχουμε σταθερές δυνάμεις πίεσης στα τοιχώματά του, όπως για παράδειγμα αν η σφαίρα είναι μεταλλική και ασκείται η επίδραση ενός μαγνήτη πάνω της. Με το παραπάνω παράδειγμα εξηγείται, τι ακριβώς συμβαίνει αν η σφαίρα ασκήσει πίεση σε ένα από τα τοιχώματα του κουτιού.



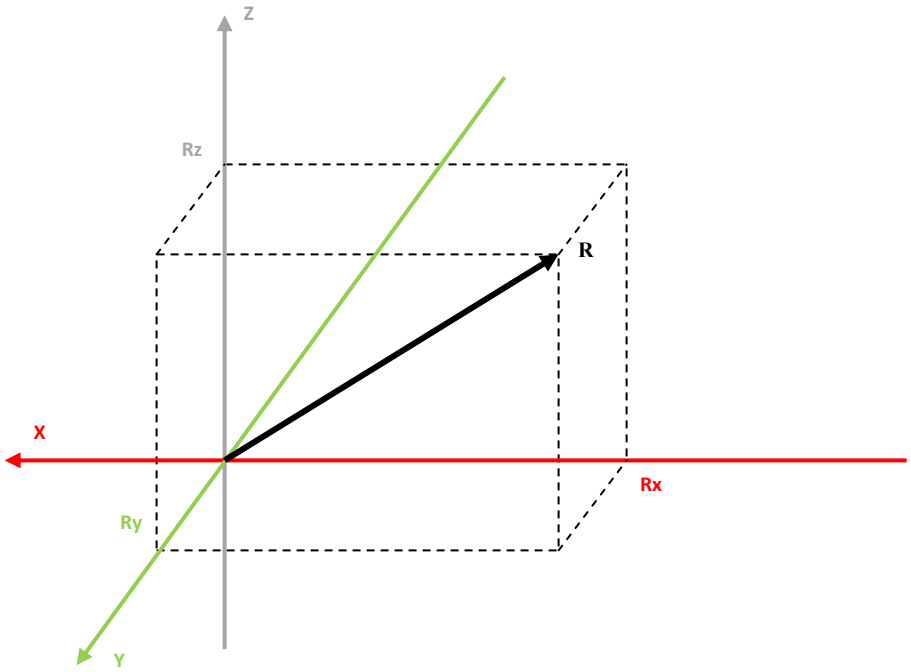
3-3. Το κουτί στην επιφάνεια της Γής με την επίδραση της δύναμης της βαρύτητας.

Με την εξέλιξη της χρήσης των επιταχυνσιόμετρων μας δίνεται πλέον η δυνατότητα μέτρησης περισσότερων κινήσεων, και στους τρείς δηλαδή άξονες ταυτόχρονα. Με βάση αυτή την δυνατότητα, ας δούμε τι θα συμβεί εάν περιστρέψουμε το κουτί κατά 45°. Όπως παρατηρούμε στο σχήμα (3-4), με την περιστροφή του κουτιού κατά 45°, η σφαίρα θα ασκήσει πίεση σε δύο από τα τοιχώματά του, το (-Z) και το (-X). Προκύπτει λοιπόν μία τιμή πίεσης ίση με 0,71 για κάθε ένα από τα τοιχώματα του κουτιού όπου ασκείται πίεση από την σφαίρα. Η τιμή αυτή είναι η προσέγγιση της Sqrt (1/2).



3-4. Απεικόνιση του κουτιού με περιστροφή κατά 45°.

Με το παράδειγμα του κουτιού, μπορέσαμε να εξηγήσουμε πως το επιταχυνσιόμετρο αντιδράει στις εξωτερικές δυνάμεις, είτε αυτές είναι σταθερές, είτε στιγμιαίες. Για να μπορέσουμε να κάνουμε όμως τους υπολογισμούς, που απαιτούνται, θα πρέπει να δημιουργήσουμε ένα σύστημα συντεταγμένων, σύμφωνα με τους άξονες του επιταχυνσιόμετρου.



3-5. Σύστημα συντεταγμένων επιταχυνσιόμετρου.

Διατηρούμε τα χρώματα των αξόνων ίδια με τα τοιχώματα του προηγούμενου παραδείγματος, για να είναι εύκολη η σύγκριση, ορίζουμε το διάνυσμα της δύναμης R και βλέπουμε τις προβολές του πάνω στους άξονες X, Y, Z ως Rx, Ry, και Rz αντίστοιχα. Με βάση το Πυθαγόρειο Θεώρημα μπορούμε να καταλήξουμε εύκολα στην εξίσωση

$$R^2 = Rx^2 + Ry^2 + Rz^2 \qquad \textbf{(Εξ.1)}$$

Εφαρμόζουμε την εξίσωση στο προηγούμενο παράδειγμα με το κουτί εικόνα (3-4) και αντικαθιστώντας τις τιμές, προκύπτει το εξής:

$$1^2 = (-\sqrt{1/2}^{\,2} + 0^{\,2} + (-\sqrt{1/2}^{\,2})$$

τέλος, με μια απλή σύγκριση των εξισώσεων έχουμε

$$R = 1,\ Rx = -\sqrt{1/2},\ Ry = 0,\ Rz = -\sqrt{1/2}$$

Στο σημείο αυτό κρίνεται σημαντικό να εξηγήσουμε τον τρόπο με τον οποίο τα επιταχυνσιόμετρα μάς μεταδίδουν την πληροφορία. Τα επιταχυνσιόμετρα χωρίζονται σε δύο κατηγορίες, τα αναλογικά και τα ψηφιακά. Στην περίπτωση των ψηφιακών χρησιμοποιείται ένα πρωτόκολλο όπως το I^2C, το SPI ή, όπως στην δική μας περίπτωση το USART, ενώ στη δεύτερη περίπτωση των αναλογικών επιταχυνσιόμετρων, δεδομένου ότι αυτά δίνουν στην έξοδο τους μια τάση εντός προκαθορισμένων ορίων, που θα πρέπει να μετατραπεί σε ψηφιακή τιμή, χρησιμοποιείται ένας μετατροπέας ADC (analog to digital converter) μέσω του οποίου προκύπτει η ζητούμενη ψηφιακή τιμή. Κρίνεται χρήσιμη μια σύντομη αναφορά δύο παραδειγμάτων μετατροπέα ADC, δεδομένου ότι δεν είναι δυνατό να αναλυθεί στην παρούσα μελέτη η λειτουργία των ADC. Ένας μετατροπέας ADC 10-bit θα έχει ως εύρος τιμών από 0 έως 1023 ($1023 = 2^{10} – 1$), ενώ ένας 12-bit ADC θα έχει εύρος τιμών από 0 έως 4095 ($4095 = 2^{12} - 1$).

Αναλυτικότερα αναφέρουμε το εξής παράδειγμα: έστω ότι έχουμε έναν 10-bit ADC, ο οποίος μας δίνει τις εξής τιμές για κάθε κανάλι του επιταχυνσιόμετρου:

AdcRx = 629

AdcRy = 597

AdcRz = 612

Κάθε μετατροπέας ADC έχει μία τιμή αναφοράς (Vreference). Ας υποθέσουμε πως στην περίπτωσή μας είναι 3.0 Volt. Για να μετατρέψουμε την έξοδο του ADC σε τάση θα χρησιμοποιήσουμε την εξής εξίσωση:

VoltsRx = AdcRx * Vref / 1023

οπότε έχουμε

VoltsRx = 629 * 3.0 / 1023 = 1.84 V (στρογγυλοποίηση των 2 δεκαδικών ψηφίων)

VoltsRy = 597 * 3.0 / 1023 = 1.75 V

VoltsRz = 612 * 3.0 / 1023 = 1.79 V

Επίσης κάθε μετατροπέας έχει μία κατώτερη τάση λειτουργίας που σχετίζεται με την τιμή 0g του επιταχυνσιόμετρου. Υποθέτουμε στην περίπτωση μας πως η τιμή αυτή είναι VzeroG=1,55V. Άρα μπορούμε να υπολογίσουμε την τιμή που δίνει το επιταχυνσιόμετρο σε Volt.

DeltaVoltsRx = VoltsRx – VzeroG

DeltaVoltsRx = 1.84 – 1.55 = 0.29V

DeltaVoltsRy = 1.75 – 1.55 = 0.20V

DeltaVoltsRz = 1.79 – 1.55 = 0.24V

Τέλος, για να μετατρέψουμε την τάση σε δύναμη g, θα πρέπει να ελέγξουμε την ευαισθησία του επιταχυνσιόμετρου. Υποθέτοντας πως η ευαισθησία είναι Sensitivity=376mV/g και διαιρώντας θα έχουμε:

Rx = DeltaVoltsRx / Sensitivity

Rx = 0.29V / 0.376V/g = 0.77g

Ry = 0.20V / 0.376V/g = 0.53g

Rz = 0.24V / 0.376V/g = 0.63g

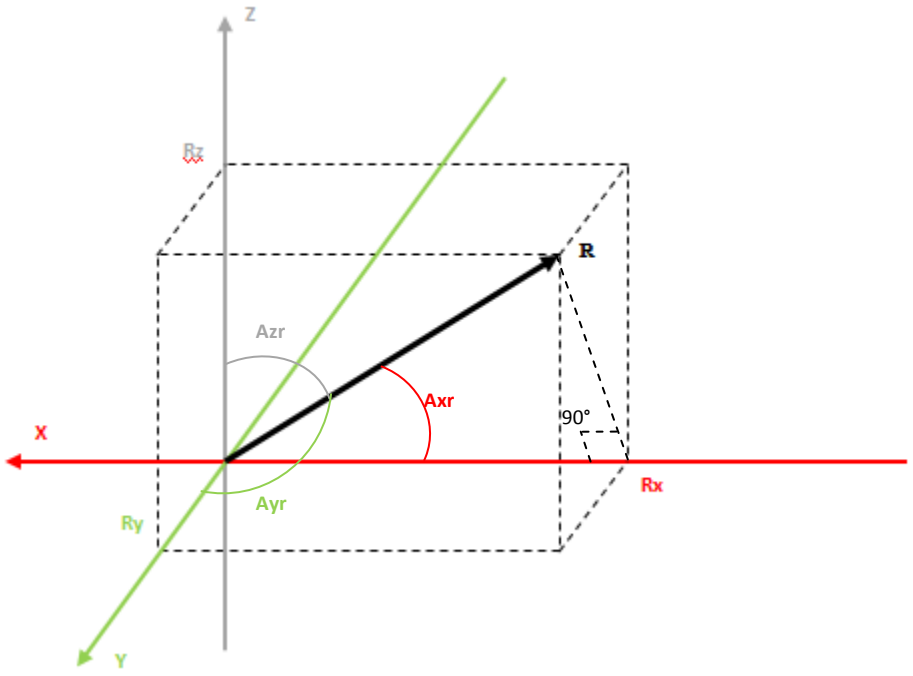
Μπορούμε τώρα να δώσουμε έναν γενικό τύπο, για να μετατρέπουμε τις τιμές εξόδου του ADC σε δυνάμεις g του κάθε άξονα:

$$Rx = (AdcRx * Vref / 1023 – VzeroG) / Sensitivity \quad \textbf{(Εξ.2)}$$

$$Ry = (Adcyx * Vref / 1023 – VzeroG) / Sensitivity$$

$$Rz = (AdcRz * Vref / 1023 – VzeroG) / Sensitivity$$

Σημειώνουμε ότι η τιμή αναφοράς (Vref), η τιμή τάσης που αντιστοιχεί σε μηδενική δύναμη (VzeroG), καθώς και η ευαισθησία (Sensitivity) του κάθε επιταχυνσιόμετρου αναφέρεται στο φυλλάδιο με τα τεχνικά χαρακτηριστικά του κάθε οργάνου. Για το συγκεκριμένο παράδειγμα, οι τιμές που χρησιμοποιήθηκαν ήταν τυχαίες.



3-6. Σύστημα συντεταγμένων επιταχυνσιόμετρου.

Επιστρέφοντας στο τελευταίο σύστημα συντεταγμένων που ορίσαμε, μας ενδιαφέρει να μάθουμε την γωνία μεταξύ του διανύσματος R σε σχέση με τον κάθε άξονα του συστήματος των συντεταγμένων μας. Για αυτό τον λόγο ονομάζουμε τις γωνίες με τους άξονες X, Y, Z ως Axr, Ayr και Azr αντίστοιχα. Παρατηρώντας τα τρίγωνα, που σχηματίζονται στο σχήμα 3-6, μπορούμε να καταλήξουμε στις εξής εξισώσεις:

Cos(Axr) = Rx / R

Cos(Ayr) = Ry / R

Cos(Azr) = Rz / R

Επομένως, λύνοντας τις παραπάνω εξισώσεις ως προς τις γωνίες και γνωρίζοντας τα διανύσματα θα έχουμε:

Axr = arcos(Rx/R)

Ayr = arcos(Ry/R)

Azr = arcos(Rz/R)

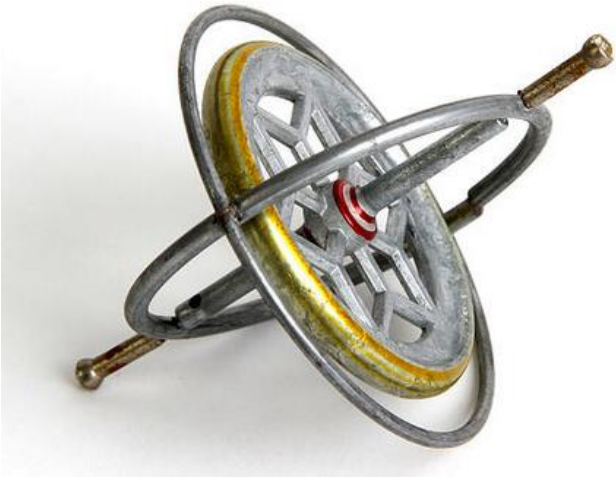
Αξίζει να σημειωθεί, πως από την Ευκλείδεια Γεωμετρία έχουμε την σχέση:

$$\sqrt{cosX^2 + cosY^2 + cosZ^2} = 1$$

όπου οι γωνίες X, Y, Z είναι οι γωνίες, που σχηματίζει ένα διάνυσμα, με τις τρείς προβολές του, στους άξονες του Καρτεσιανού συστήματος συντεταγμένων. Ο τύπος αυτός μπορεί να μας φανεί χρήσιμος, στις περιπτώσεις που δεν γνωρίζουμε το μήκος του διανύσματος R, γιατί μας ενδιαφέρει απλά η κατεύθυνση του.

Γυροσκόπιο

Το γυροσκόπιο είναι η συσκευή μέτρησης ή διατήρησης του προσανατολισμού. Πρόκειται για ένα σύνολο αποτελούμενο από μεταλλικούς δακτυλίους, ενωμένους κατακόρυφα μεταξύ τους και έναν άξονα που φέρει μεταλλικό δίσκο, ή σφαίρα στο μέσο του και τοποθετείται στο κέντρο των δακτυλίων. Προκαλώντας την περιστροφή του εσωτερικού δίσκου, με μεγάλη ταχύτητα, παρατηρούμε ότι το γυροσκόπιο αντιτίθεται στην οποιαδήποτε αλλαγή της θέσης του, αψηφώντας πολλές φορές και την βαρύτητα. Το φαινόμενο αυτό είναι το αποτέλεσμα της αρχής διατήρησης της στροφορμής. Η γυροσκοπική πυξίδα αποτελεί εφαρμογή των αρχών του γυροσκοπίου και για τον λόγο αυτό οι ενδείξεις της θεωρούνται αληθείς, σε αντίθεση με εκείνες της μαγνητικής πυξίδας που χρήζουν διορθώσεων. Τα γυροσκόπια σήμερα χρησιμοποιούνται ευρέως σε πολλές τεχνολογικές εφαρμογές, μεταξύ των οποίων και η ανίχνευση της τοποθέτησης ενός αεροσκάφους στον χώρο αλλά και η όποια μεταβολή της θέσης του σε αυτόν.



3-7. Γυροσκόπιο [πηγή: fineartamerica.com].

Το ηλεκτρονικό γυροσκόπιο μετράει τον ρυθμό μεταβολής των γωνιών σε σχέση με τους άξονες του συστήματος συντεταγμένων. Για να εξηγήσουμε τις μετρήσεις που λαμβάνουμε θα επιστρέψουμε στο τελευταίο σύστημα αξόνων που χρησιμοποιήσαμε και στο επιταχυνσιόμετρο, ορίζοντας ως Axz και Ayz τις γωνίες που σχηματίζουν οι προβολές του διανύσματος R, (Rxz, Ryz) στα επίπεδα XZ και YZ αντίστοιχα (σχ. 3-8).

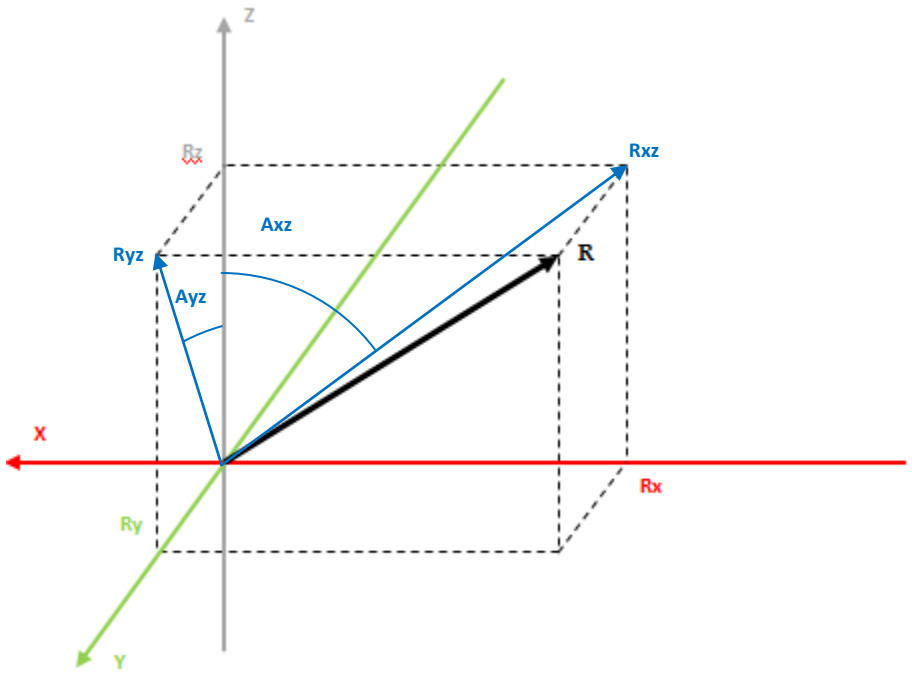
Με την εφαρμογή του Πυθαγόρειου Θεωρήματος καταλήγουμε στις εξισώσεις:

$$Rxz^2 = Rx^2 + Rz^2$$

και παρομοίως $$Ryz^2 = Ry^2 + Rz^2$$

Επίσης $$R^2 = Rxz^2 + Ry^2$$

$$R^2 = Ryz^2 + Rx^2$$



3-8. Γωνίες μέτρησης γυροσκόπιου.

Για να υπολογίσουμε τον ρυθμό μεταβολής των γωνιών υποθέτουμε πως μετράμε την γωνία Axz. Την χρονική στιγμή t_0 θα έχει τιμή Axz_0, ενώ την χρονική στιγμή t_1 θα έχει τιμή Axz_1. Έτσι ο ρυθμός μεταβολής υπολογίζεται με τον εξής τύπο:

$$RateAxz = (Axz_1 – Axz_0) / (t_1 – t_0)$$

Επειδή, το γυροσκόπιο (εκτός ορισμένων περιπτώσεων), έχει ως έξοδο τάση, θα πρέπει αυτή να μετατραπεί σε deg/s , όπως στο επιταχυνσιόμετρο. Ακολουθώντας τα βήματα όπως προηγουμένως θα καταλήξουμε στον τύπο:

$$RateAxz = (AdcGyroXZ * Vref / 1023 – VzeroRate) / Sensitivity \qquad \textit{(Εξ.3)}$$

$$RateAyz = (AdcGyroYZ * Vref / 1023 – VzeroRate) / Sensitivity$$

Πρέπει να σημειωθεί ότι οι τιμές AdcGyroXZ και AdcGyroYZ, αποτελούν τις εξόδους του μετατροπέα ADC και αντιπροσωπεύουν την περιστροφή της προβολής του διανύσματος R, στα επίπεδα XZ και YZ αντίστοιχα. Το Vref, το VzeroRate και την ευαισθησία (Sensitivity) μπορούμε να τα βρούμε στο φυλλάδιο με τα χαρακτηριστικά του κάθε οργάνου. Για το παράδειγμα που ακολουθεί οι τιμές που θα χρησιμοποιηθούν επιλέχθηκαν τυχαία.

Έστω ότι ο μετατροπέας μας δίνει στην έξοδο του τις τιμές:

AdcGyroXZ = 456

AdcGyroYZ = 780

Χρησιμοποιώντας την εξίσωση ***(Εξ.3)*** και θεωρώντας τα Vref = 3.0V,

VzeroRate = 2,15V, Sensitivity = 0,0025V/(deg/s)

προκύπτει:

RateAxz = (456 * 3.0V / 1023 – 2.15V) / (0.0025V/(deg/s)) = -325.10 deg/s

RateAyz = (780 * 3.0V/ 1023 – 2.15V) / (0.0025V/(deg/s)) = 54,95 deg/s

Σαν αποτέλεσμα έχουμε, πως η συσκευή μας περιστρέφεται γύρω από τον Y άξονα (ή αλλιώς στο επίπεδο XZ) με ταχύτητα -325.10 deg/s και ταυτόχρονα γύρω από τον X άξονα (ή αλλιώς στο επίπεδο YZ) με ταχύτητα 54,95 deg/s. Το αρνητικό πρόσημο σημαίνει πως η περιστροφή γίνεται με αντίθετη φορά σε σχέση με τη συμβατική θετική κίνηση του οργάνου.

Σε αυτό το σημείο θα πρέπει να σημειωθεί, πως το επιταχυνσιόμετρο και το γυροσκόπιο εμφανίζουν ορισμένα σφάλματα. Το γυροσκόπιο για παράδειγμα, μετά την ολοκλήρωση, εμφανίζει μια γωνιακή μετατόπιση “angular drift”, η οποία αυξάνεται γραμμικά με το πέρασμα του χρόνου. Στην περίπτωση του επιταχυνσιόμετρου, κάνοντας

διπλή ολοκλήρωση για να υπολογιστεί η θέση, το σφάλμα του αισθητηρίου προκαλεί σφάλμα θέσης "positional drift" υψωμένο στην δύναμη του δύο. Για αυτό το λόγο απαιτείται ο συνδυασμός περισσότερων οργάνων, που θα συμπληρώνουν και επαληθεύουν το ένα τις μετρήσεις του άλλου, για την ορθή πληροφόρησή μας.

Μαγνητόμετρο

Το μαγνητόμετρο είναι ένα όργανο μέτρησης και χρησιμοποιείται για τη μέτρηση της έντασης, την ανίχνευσή της κατεύθυνσης ή τέλος για την ανίχνευση μεταβολών σε ένα μαγνητικό πεδίο. Το μαγνητικό πεδίο της Γής έχει τιμές μεταξύ 0,5 και 0,6 Gauss και έχει συνιστώσες παράλληλες με την επιφάνειά της, «προσανατολισμένες» πάντα προς τον μαγνητικό Βορρά. Αυτή είναι η αρχή λειτουργίας για κάθε μαγνητική πυξίδα. Σημειώνεται ότι ο μαγνητικός Βορράς, είναι διαφορετικός από τον πραγματικό Βορρά που συναντάται στους γεωγραφικούς χάρτες. Η διαφορά τους ορίζεται στις 11,5°, ενώ σε διαφορετικά σημεία της Γής μπορεί να φτάσει και τις ±25°.

Σήμερα υπάρχουν αρκετοί τύποι ηλεκτρικών μαγνητικών αισθητήρων, όπως οι αισθητήρες Μαγνητοαντίστασης (MR), Σύνθετης Μαγνητοαντίστασης, Μαγνητοσυστολής (MI), Ηλεκτρομαγνητικής επαγωγής και φαινομένου Hall. Η επιλογή του κατάλληλου εξοπλισμού εξαρτάται από τα χαρακτηριστικά του κάθε μαγνητόμετρου. Από τα πιο σημαντικά χαρακτηριστικά είναι η ευαισθησία και η δυναμική του οργάνου, αν είναι κλιμακωτό ή βαθμιδωτό, ο χρόνος απόκρισής του στις μεταβολές του πεδίου, καθώς και η συχνότητα λειτουργίας του για την αποφυγή θορύβων που θα αλλοιώσουν τις μετρήσεις.

Οι ενδείξεις της πυξίδας είναι προκαθορισμένες από το μαγνητικό πεδίο της Γής, όταν αυτή είναι τοποθετημένη σε οριζόντιο επίπεδο. Για να χρησιμοποιήσουμε την μέτρηση αυτή θα πρέπει να διορθωθεί πρώτα η γωνία tilt και η επίδραση άλλων σωμάτων ή άλλων πεδίων στην παραμόρφωση του μαγνητικού πεδίου του οργάνου μας και τέλος να υπολογιστεί η διαφορά του πραγματικού Βορρά με τον μαγνητικό Βορρά.

Οι μέθοδοι διόρθωσης των σφαλμάτων αυτών ποικίλουν και εξαρτώνται από την επιθυμητή ακρίβεια, την ανάλυση, την ταχύτητα, το μέγεθος και την αξία του οργάνου.

Οι αισθητήρες Μαγνητοαντίστασης, αποτελούν ίσως την καλύτερη επιλογή για την κατασκευή συστημάτων IMU γιατί προσφέρουν μεγάλη ευαισθησία και καλή επαναληψιμότητα σε συνδυασμό με το μικρό μέγεθος και τη μεγάλη ακρίβεια.

Αισθητήρας Πίεσης

Οι αισθητήρες πίεσης χρησιμοποιούνται συνηθέστερα για την μέτρηση πίεσης σε υγρά και αέρια. Η πίεση είναι μια έκφραση της δύναμης, που απαιτείται, για να σταματήσει ένα ρευστό από την διαστολή του και μετράται σε μονάδες δύναμης προς εμβαδόν. Ανάλογα με την πίεση που δέχεται, ο αισθητήρας έχει σαν έξοδο ένα σήμα, που στις περισσότερες περιπτώσεις είναι μία τάση. Μπορούμε να συναντήσουμε αισθητήρες που είναι χωρητικοί, ηλεκτρομαγνητικοί, οπτικοί, που χρησιμοποιούν το πιεζοηλεκτρικό φαινόμενο κ.ά. Επίσης μπορούν να ποικίλουν ανάλογα με την ακρίβεια, την ευκρίνεια και το εύρος τους.

Ανάμεσα στις πολλές εφαρμογές που συναντώνται οι αισθητήρες πίεσης είναι και η μέτρηση του ύψους. Χρησιμοποιούνται για παράδειγμα σε πυραύλους και αεροσκάφη και υπολογίζουν το υψόμετρο, με βάση την διαφορά της ατμοσφαιρικής πίεσης από την επιφάνεια της θάλασσας στο σημείο όπου αυτά βρίσκονται. Ο τύπος που χρησιμοποιείται για τον υπολογισμό αυτό είναι:

$$h = (1 - (P / Pref)^{0.190284}) * 145366.45ft$$

Η εξίσωση αυτή μπορεί να χρησιμοποιηθεί μέχρι το όριο των 36,090 feet, (11.000m). Αν το όριο αυτό ξεπεραστεί, τότε εισάγεται ένα μόνιμο σφάλμα, που μπορεί να υπολογιστεί με διάφορους τρόπους ανάλογα με το κάθε αισθητήριο. Η ακρίβεια των

αισθητήρων πίεσης μπορεί να είναι μικρότερη του ενός μέτρου (ανάλογα το αισθητήριο), σαφώς καλύτερη από αυτήν που προσφέρουν τα συστήματα GPS (ακρίβεια περίπου 20 μέτρα) και για τον λόγο αυτό κατέχουν πια καθοριστική θέση στα συστήματα πλοήγησης.

ΕΛΕΓΚΤΗΣ PID

Ο ελεγκτής PID (proportional-integral-derivative) είναι ένας μηχανισμός ελέγχου ανάδρασης με ευρεία χρήση στα βιομηχανικά συστήματα ελέγχου. Ο ελεγκτής PID υπολογίζει μία τιμή σφάλματος, την διαφορά δηλαδή ανάμεσα σε μια πραγματική τιμή μιας μεταβλητής της διαδικασίας και στην επιθυμητή τιμή (setpoint) της μεταβλητής αυτής και στη συνέχεια επιχειρεί να ελαχιστοποιήσει αυτό το σφάλμα προσαρμόζοντας τις εισόδους ελέγχου της διαδικασίας.

Ο αλγόριθμος υπολογισμού των εισόδων ελέγχου του ελεγκτή περιλαμβάνει τρεις σταθερές παραμέτρους, από το στοιχείο αυτό και η ονομασία "ελεγκτής τριών όρων". Οι παράμετροι αυτές είναι: α. ο αναλογικός όρος (proportional), β. ο όρος ολοκλήρωσης (integral) και γ. ο όρος διαφόρισης (derivative) και συμβολίζονται ως P,I και D αντίστοιχα. Οι όροι αυτοί μπορούν να ερμηνευθούν με βάση τον χρόνο, στον οποίο αναφέρεται το σφάλμα, που υπολογίζουν. Ο P όρος εξαρτάται από το παρόν σφάλμα, ο I από την συσσώρευση προηγούμενων σφαλμάτων και ο D αποσκοπεί στην πρόβλεψη μελλοντικών σφαλμάτων με βάση τον στιγμιαίο ρυθμό μεταβολής του σφάλματος. Το άθροισμα των ενεργειών αυτών (άθροισμα με βάρη) χρησιμοποιείται για να προσαρμόσει την διαδικασία μέσω ενός στοιχείου ελέγχου, όπως η θέση μιας βαλβίδας ελέγχου, ενός αποσβεστήρα ή η ισχύς που παρέχεται σε θερμικό στοιχείο.

Με την παραμετροποίηση του ελεγκτή έχουμε την δημιουργία κατάλληλων σημάτων με σκοπό την επίτευξη των επιθυμητών στόχων. Συνηθέστερα, σε περιπτώσεις άγνωστης διεργασίας, ο ελεγκτής PID θεωρείται ως ο καταλληλότερος. Υπάρχουν όμως περιπτώσεις, που η χρήση του μπορεί να προκαλέσει αστάθεια στο σύστημα, ενώ μπορεί να προκύψει ικανοποιητικός έλεγχος με χρήση ενός ή και συνδυασμού των δύο εκ των τριών παραμέτρων του ελεγκτή, με αποτέλεσμα την χρήση P, PI ή PD ελέγχου.

Βαθμίδες ελεγκτή

Ορίζουμε την μεταβλητή e(t), ως το σφάλμα που προκύπτει από την μέτρηση της πραγματικής τιμής, της ελεγχόμενης μεταβλητής και την διαφορά της από την επιθυμητή τιμή (setpoint). Αυτό το σφάλμα προωθείται στις τρεις βαθμίδες του ελεγκτή (P, I και D), αποδίδοντας έναν όρο στην τελική εξίσωση του σήματος ελέγχου, που θα αποτελέσει την έξοδο του ελεγκτή u(t).

Η βαθμίδα P, που αποτελεί τον όρο αναλογίας πολλαπλασιάζει το σφάλμα με ένα κέρδος Kp, και αποδίδει στην τελική εξίσωση τον όρο K_p * e(t). Εάν το κέρδος Kp είναι μεγάλο, ως αποτέλεσμα του πολλαπλασιασμού, θα έχουμε μία μεγάλη αλλαγή στην έξοδο, για κάθε αλλαγή στην τιμή του σφάλματος, στοιχείο που μπορεί να οδηγήσει σε αστάθεια το σύστημα. Σε αντίθετη περίπτωση, ένα μικρό αναλογικό κέρδος έχει ως αποτέλεσμα μικρές μεταβολές στην έξοδο του ελεγκτή, με συνέπεια την μεγαλύτερη ευστάθεια του συστήματος, αν και επηρεάζει αρνητικά τον χρόνο αντίδρασης του ελεγκτή. Τέλος, όταν το κέρδος, που οριστεί, είναι πολύ μικρό, υπάρχει ο κίνδυνος οι διαταραχές του συστήματος να μην αντιμετωπιστούν εγκαίρως από τον ελεγκτή.

Τα βασικά χαρακτηριστικά του αναλογικού ελέγχου είναι η μείωση του μόνιμου σφάλματος, χωρίς όμως την εξάλειψή του, η μείωση του χρόνου ανύψωσης της καμπύλης απόκρισης του συστήματος, η αύξηση της υπερύψωσης (η μέγιστη απόκλιση από την επιθυμητή τιμή όταν η έξοδος υπερβεί το setpoint) και η μικρή αλλαγή στον χρόνο αποκατάστασης του συστήματος.

$$P_{out} = K_p * e(t).$$

Η βαθμίδα I, που αποτελεί τον όρο ολοκλήρωσης, επηρεάζεται από το μέγεθος του σφάλματος, καθώς και από τον χρόνο διάρκειάς του. Ο όρος της ολοκλήρωσης σε έναν ελεγκτή είναι το άθροισμα των στιγμιαίων τιμών του σφάλματος, σε ένα χρονικό διάστημα, πολλαπλασιασμένο με το κέρδος K_i.

$$I_{out} = K_i * \int_0^t e(t)dt$$

Βασικό χαρακτηριστικό της βαθμίδας ολοκλήρωσης είναι η εξάλειψη του μόνιμου σφάλματος αλλά και η μείωση του χρόνου ανύψωσης της καμπύλης εξόδου, η αύξηση της υπερύψωσης και η αύξηση του χρόνου αποκατάστασης του συστήματος.

Η βαθμίδα D, αποτελεί τον όρο διαφόρισης του ελεγκτή. Η παράγωγος του σφάλματος e(t) υπολογίζεται καθορίζοντας την κλίση της καμπύλης του σφάλματος στον χρόνο και πολλαπλασιάζοντας αυτόν τον ρυθμό μεταβολής με μία σταθερά κέρδους K_d, η οποία στην πραγματικότητα δείχνει το μέγεθος της συνεισφοράς του όρου διαφόρισης στην συνολική δράση ελέγχου.

Τα βασικά χαρακτηριστικά της βαθμίδας διαφόρισης είναι η μικρή αλλαγή στην τιμή του μόνιμου σφάλματος, η μικρή αλλαγή στον χρόνο ανύψωσης της καμπύλης εξόδου, η μείωση της υπερύψωσης, καθώς και η μείωση του χρόνου αποκατάστασης.

$$D_{out} = K_d * ((d / dt) * e(t))$$

Ως αποτέλεσμα, η συνολική εξίσωση του σήματος ελέγχου u(t), που θα αποτελεί την έξοδο του PID ελεγκτή, είναι το άθροισμα των παραπάνω βαθμίδων:

$$u(t) = K_p * e(t) + I_{out} + K_i * \int_0^t e(t)dt + K_d * ((d / dt) * e(t))$$

Παραμετροποίηση των K_p, K_i, K_d σταθερών του PID αγνώστου διεργασίας

Για τον έλεγχο διεργασιών, των οποίων δεν γνωρίζουμε την συνάρτηση μεταφοράς, χρησιμοποιούμε ορισμένες τεχνικές, ώστε να παραμετροποιήσουμε τις μεταβλητές K_p, K_i και K_d.

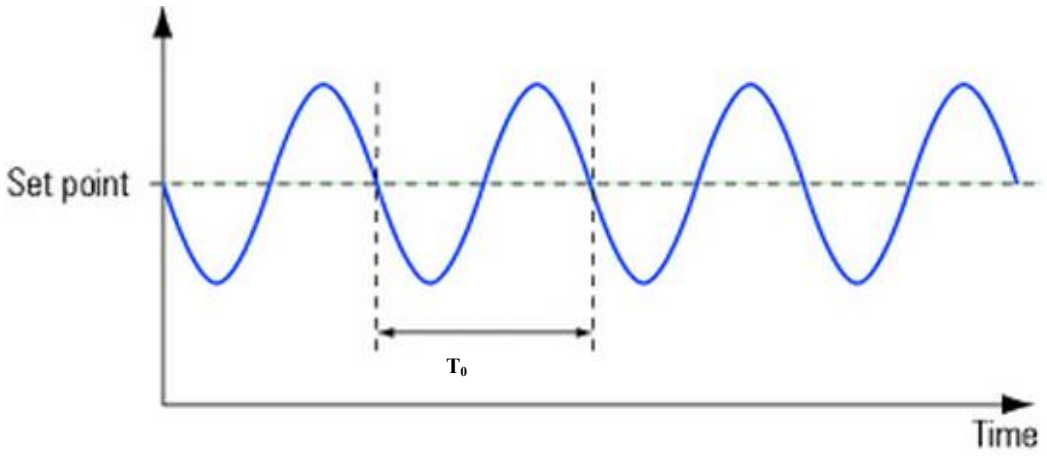
Ο απλούστερος τρόπος προκύπτει με την βοήθεια ενός παλμογράφου, καθώς παίρνουμε την γραφική παράσταση της εξόδου του συστήματος σε συνάρτηση με τον

χρόνο. Θέτουμε τις τρεις παραμέτρους K_p, K_i, K_d στην ελάχιστη δυνατή τιμή και στη συνέχεια αυξάνουμε σταδιακά την τιμή του κέρδους αναλογίας K_p, έως ότου η έξοδος του συστήματος γίνει μία ταλάντωση σταθερού πλάτους. Η τιμή του κέρδους, που προκαλεί την ταλάντωση, ορίζεται ως $K_{critical}$. Με αυτόν τον τρόπο μπορούμε να υπολογίσουμε τα K_p, K_i και K_d από τις σχέσεις:

$K_p = 0.6 * k_{critical}$

$K_i = 1 / \tau_i$, όπου $\tau_i = T_0 / 2$

$K_d = \tau_d$, όπου $\tau_d = T_0 / 8$



5-1.Ταλάντωση σταθερού πλάτους.

PID στον ψηφιακό χρόνο

Ο έλεγχος ενός συστήματος είναι δυνατόν να πραγματοποιηθεί σε δύο επίπεδα χρόνου, τον συνεχή και τον διακριτό. Ένα σύστημα του διακριτού χρόνου (ψηφιακό) προκύπτει ύστερα από δειγματοληψία από ένα σύστημα συνεχούς χρόνου, με σταθερή περίοδο δειγματοληψίας. Πρακτικά αυτό σημαίνει, ότι ενώ το αρχικό σύστημα έχει τιμές

σε όλο το εύρος του άξονα του χρόνου, εμείς προσπαθούμε να αποκτήσουμε την καλύτερη δυνατή απεικόνιση του συστήματος αυτού, παίρνοντας τιμές ανά σταθερές χρονικές στιγμές. Αυτή η σταθερή χρονική απόσταση μεταξύ των λαμβανόμενων δειγμάτων, καλείται περίοδος δειγματοληψίας T_s (Tsampling). Κατά συνέπεια σε ένα χρονικό διάστημα t sec, προκύπτει αριθμός δειγμάτων $n=t/T_s$. Το αρχικό αναλογικό σύστημα μπορεί να αναπαρασταθεί στον ψηφιακό χρόνο με το σύνολο των n δειγμάτων. Προκύπτει, ότι όσο μικρότερη είναι η περίοδος δειγματοληψίας, δηλαδή όσο μεγαλύτερη είναι η συχνότητα δειγματοληψίας fs, τόσο ακριβέστερη είναι η μεταφορά του συστήματος από το αναλογικό στο ψηφιακό επίπεδο.

Είδαμε, ότι στον συνεχή χρόνο η έξοδος του ελεγκτή είναι:

$$u(t)= Kp * e(t)+)+ I_{out} = K_i \int_0^t e(t)dt \quad + Kd *((d/dt) *e(t))$$

Για την εξίσωση του διακριτού χρόνου, θα πρέπει να εξετάσουμε τους συντελεστές μεμονωμένα.

Αρχικά το u(t) θα μετατραπεί σε u(n), αφού πλέον το σύστημα, άρα και η έξοδος του ελεγκτή που το ελέγχει, δεν υφίσταται κάθε χρονική στιγμή αλλά μόνο διακριτές στιγμές n (n+ Ts, n+2* Ts, n+3* Ts …). Η έξοδος του ελεγκτή θα προκύπτει κάθε n+κ*Ts, όπου κ ακέραιος. Το σφάλμα από e(t) θα εκφράζεται ως e(n).

Όσον αφορά στον όρο διαφόρισης για να μεταφερθούμε στον ψηφιακό χρόνο δεν έχουμε παρά να ακολουθήσουμε τον ορισμό της έννοιας της παραγώγου. Η παράγωγος του σφάλματος ελέγχου στον συνεχή χρόνο αντιπροσωπεύει τον ρυθμό μεταβολής της τιμής του σφάλματος στον ελάχιστο δυνατό χρόνο, επομένως το ίδιο θα συμβαίνει και στον διακριτό. Στον συνεχή χρόνο η ελάχιστη χρονική απόσταση, που συμβολίζεται με το dt, θα αντικατασταθεί με το T_s, που είναι η σταθερή χρονική απόσταση μεταξύ δύο μετρήσεων, ενώ η όποια μεταβολή στην τιμή του μετρούμενου μεγέθους, που

αναπαριστάται με το d*e(t), θα αντικατασταθεί με την διαφορά δύο διαδοχικών μετρήσεων, δηλαδή e(n)-e(n-1).

Τέλος, για να μεταφέρουμε τον όρο ολοκλήρωσης από τον συνεχή στον διακριτό χρόνο, θα πρέπει να κάνουμε την αναγωγή σε αυτόν της διαδικασίας του ολοκληρώματος σε ένα ορισμένο διάστημα. Αυτό μπορεί να γίνει απλά, αν σκεφτούμε την φυσική έννοια του ολοκληρώματος, που είναι το εμβαδόν ενός χωρίου, που ορίζουν δύο ορισμένα άκρα, δηλαδή η πρόσθεση όλων των μεταξύ τους ελάχιστων χωρίων και κατά αναλογία στον διακριτό προσθέτουμε όλες τις μετρήσεις του εξεταζόμενου μεγέθους εντός του «κβαντωμένου χρονικά» διαστήματος που μας ενδιαφέρει. Επομένως το $K_i \int_0^t e(t)dt$ θα μετατραπεί σε: $K_i * \sum_{k=0}^{n} e(k)$.

Τελικά, η αναλογική εξίσωση $u(t) = Kp * e(t) + K_i \int_0^t e(t)dt + Kd * ((d/dt) * e(t))$,

θα μετατραπεί σε $u(n) = K_p * e(n) + K_i * \sum_{k=0}^{n} e(k) + K_d * [(e(n) - e(n-1)) / Ts]$

NUTTX OS

Το λειτουργικό σύστημα πραγματικού χρόνου Nuttx, που είναι εγκατεστημένο στον PX4fmu, δίνει έμφαση στα πρότυπα συμμόρφωσης και συμβατότητας του κώδικα προγραμματισμού (standarts compliance), είναι προσαρμοζόμενο και επεκτάσιμο σε αρχιτεκτονική μικροελεγκτών απο 8 μέχρι 32bit και έχει ελάχιστες απαιτήσεις σε μνήμη RAM του συστήματος. Τα κύρια πρότυπα συμμόρφωσης, που ικανοποιεί, είναι το POSIX (Portable Operating System Interface for Unix) και το ANSI (American National Standards Institute).

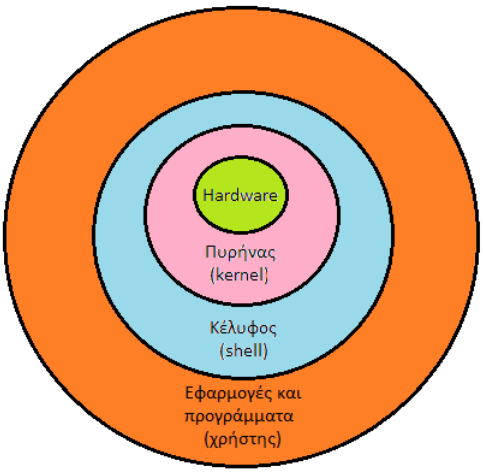
POSIX καλείται το όνομα μιας σειράς πρότυπων που καθορίζουν τις βασικές υπηρεσίες ενός λειτουργικού συστήματος, και κυρίως την διασύνδεση προγραμματισμού εφαρμογών-API των διαθέσιμων κλήσεων του συστήματος. Η διασύνδεση προγραμματισμού εφαρμογών-API αφορά στη διεπαφή των προγραμματιστικών διαδικασιών, στη δυνατότητα δηλαδή ενός λειτουργικού συστήματος, μιας βιβλιοθήκης ή μιας εφαρμογής να επιτρέπουν την λήψη αιτήσεων από άλλα προγράμματα ή/και την ανταλλαγή δεδομένων. Οι διαθέσιμες κλήσεις του συστήματος είναι ένα σύνολο υπηρεσιών που παρέχει ο πυρήνας του λειτουργικού συστήματος μέσω μίας προτυποποιημένης προγραμματιστικής διασύνδεσης.

Το πρότυπο συμμόρφωσης POSIX δημιουργήθηκε με στόχο την εξασφάλιση της συμβατότητας μεταξύ των ποικίλλων λειτουργικών συστημάτων UNIX. Το αποτέλεσμα είναι τα προγράμματα, που τρέχουν σε έναν υπολογιστή με αντίστοιχο λειτουργικό, να μπορούν να μεταγλωττιστούν σε άλλα συστήματα παρόμοιου λειτουργικού, χωρίς να χρειάζεται να τροποποιηθεί ο πηγαίος κώδικας ή μέρος αυτού. Η συμμόρφωση με το POSIX αποτελεί προϋπόθεση για να μπορεί ένα λειτουργικό σύστημα να ονομάζεται UNIX.

ANSI είναι η μέθοδος για τον έλεγχο της μορφοποίησης, του χρώματος, της κωδικοποίησης χαρακτήρων και άλλων επιλογών εξόδου στα τερματικά των

συστημάτων. Είναι συμβατή με αρκετές διεπαφές (API's) από το λειτουργικό σύστημα UNIX αλλά και από παρόμοια λειτουργικά συστήματα πραγματικού χρόνου. Το NuttX κυκλοφόρησε για πρώτη φορά το 2007 από τον Gregory Nutt, υπό την άδεια BSD.

NuttShell (Nsh) του NuttX OS



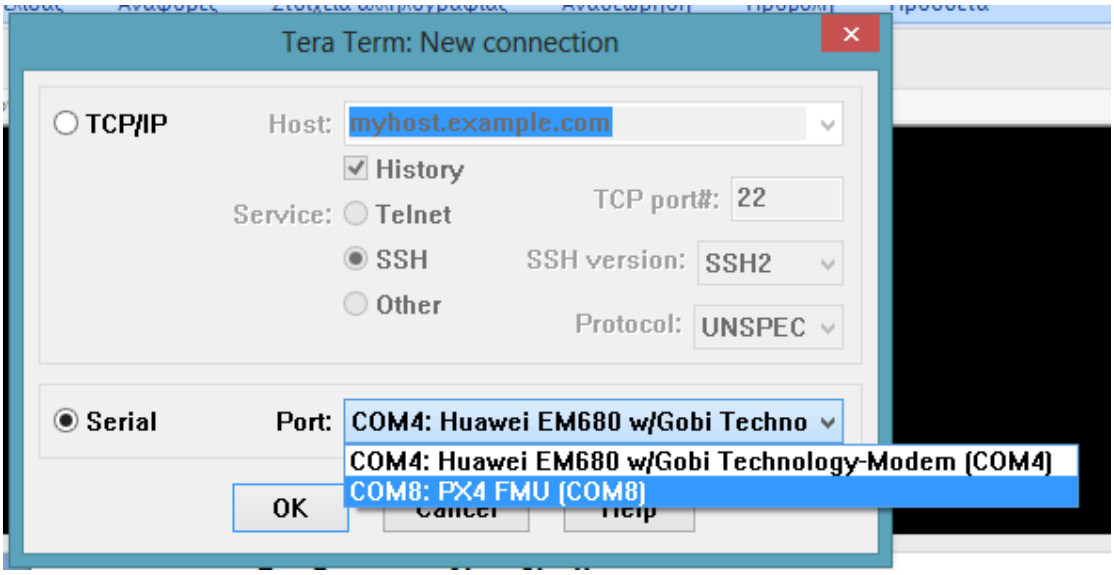
6-1. Διαστρωμάτωση λειτουργικού συστήματος.

Το Nuttshell είναι το ενσωματωμένο κέλυφος (αγγλικός όρος: shell) του λειτουργικού συστήματος NuttX. Το κέλυφος είναι το τμήμα του λογισμικού, που παρέχει σύνδεση στους χρήστες και έχει πρόσβαση στον πυρήνα του λειτουργικού συστήματος. Το συγκεκριμένο κέλυφος παρέχει διασύνδεση στον χρήστη γραμμής εντολών. Σε κάθε περίπτωση ο στόχος του Nuttshell είναι η κλήση και ενεργοποίηση άλλων προγραμμάτων, αν και συνήθως ένα κέλυφος έχει και άλλες ικανότητες όπως η προβολή των περιεχομένων των καταλόγων του συστήματος αρχείων. Το Nuttshell είναι αντίστοιχο του Bash (κέλυφος ελεύθερου λογισμικού του UNIX, προεπιλεγμένο στα περισσότερα συστήματα, που βασίζονται στον πυρήνα του Linux, καθώς και στο Mac

OS X), αναγνωρίζει ένα μεγάλο σύνολο εντολών και αλληλεπιδρά με τον χρήστη μέσω σειριακής επικοινωνίας με UART ή με USB.

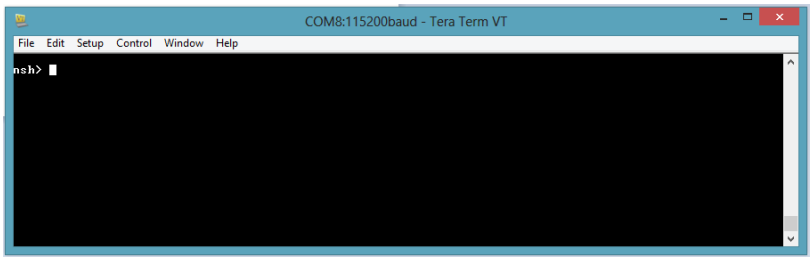
Σύνδεση στο NuttShell

Για να πραγματοποιήσουμε σύνδεση με το Nsh του PX4fmu, εγκαθιστούμε το PX4 Toolchain και συνδέουμε τον PX4fmu μέσω της θύρας USB στον υπολογιστή. Στη συνέχεια επιλέγουμε το πρόγραμμα τερματικού Teraterm από την λίστα προγραμμάτων του PX4 Toolchain που εγκαταστήσαμε. Μπορούμε να συνδεθούμε με οποιοδήποτε πρόγραμμα τερματικού π.χ. Windows Terminal emulator. Στο τερματικό επιλέγουμε νέα σειριακή σύνδεση με την θύρα που έχει ο PX4fmu και πατάμε OK.



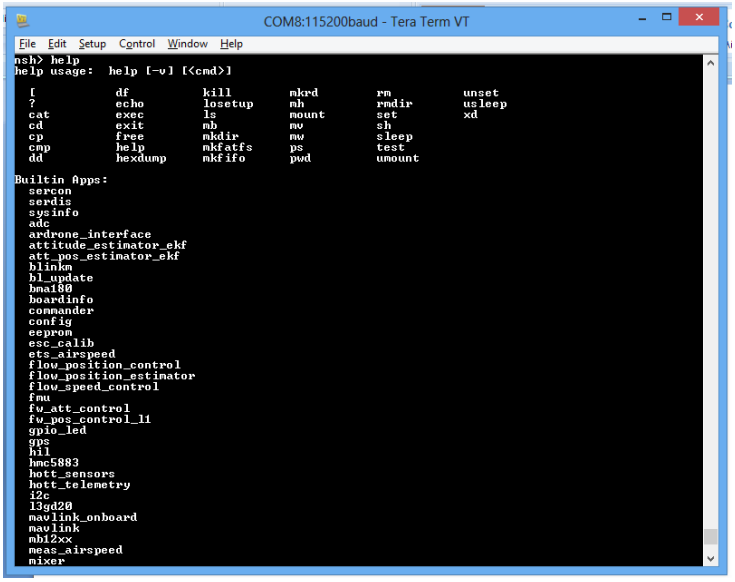
6-2. Επιλογή θύρας επικοινωνίας με τον PX4fmu.

Στο νέο παράθυρο του τερματικού πληκτρολογούμε τρεις (3) φορές enter και στην συνέχεια βλέπουμε την αναμονή εντολών του Nsh.



6-3. Περιβάλλον επικοινωνίας Teraterm.

Πληκτρολογούμε help και enter και βλέπουμε μια λίστα με εντολές κελύφους, που υποστηρίζει το Nsh και χαμηλότερα τις εφαρμογές, που μπορούμε να ξεκινήσουμε από το Nsh . Παρακάτω εμφανίζεται πίνακας με τις εντολές και την λειτουργία τους. Οι βασικές εφαρμογές αναλύονται σε άλλο κεφάλαιο.



6-4. Εντολές και εφαρμογές του λειτουργικού συστήματος.

6-1. Πίνακας εντολών Nsh

Εντολή	**Λειτουργία**
cat	Συνενώνει και εμφανίζει στην οθόνη αρχεία κειμένου (text files)
cd	Προκαλεί αλλαγή του τρέχοντος καταλόγου (directory)
cp	Αντιγράφει αρχεία κειμένου σε directory με το ίδιο ή διαφορετικό όνομα
cmp	Συγκρίνει δύο αρχεία οποιουδήποτε τύπου και τυπώνει το αποτέλεσμα στην έξοδο
dd	Αντιγραφή ενός αρχείου, μετατροπή και μορφοποίηση σύμφωνα με τις επιλογές
df	Εμφανίζει τον ελεύθερο χώρο που απομένει στο σύστημα αρχείων
echo	Προκαλεί εκτύπωση των παραμέτρων που ακολουθούν την echo
exec	Εκτελεί κώδικα του χρήστη
exit	Αποσύνδεση
free	Εμφανίζει την χρησιμοποιούμενη και την ελεύθερη μνήμη συστήματος
help	Οδηγός βοήθειας
hexdump	Φίλτρο που εμφανίζει συγκεκριμένα αρχεία, ή την είσοδο, εάν αυτά δεν έχουν καθοριστεί σε μορφή που έχει ορίσει ο χρήστης
kill	Τερματίζει διεργασίες

losetup	Χρησιμοποιείται για να συνδέσει συσκευές βρόχου με κανονικά αρχεία ή συσκευές μπλοκ
ls	Εμφανίζει λίστα με ταξινομημένα τα ονόματα αρχείων ενός καταλόγου
mv	Πρόσβαση στην μνήμη ram με byte size access
mkdir	Δημιουργεί έναν ή περισσότερους καταλόγους
mkfatfs	Χρησιμοποιείται για την κατασκευή ενός fat συστήματος αρχείων σε μια συσκευή, συνήθως ενός διαμερίσματος του σκληρού δίσκου
mkfifo	Δημιουργεί έναν χαρακτήρα FIFO (first in-first out) στο σύστημα
mkrd	Δημιουργεί Ramdisk
mh	Πρόσβαση στην μνήμη ram με 16-bit access
mount	Προσάρτηση σε κάποιον κατάλογο, ώστε να γίνουν ορατά τα περιεχόμενα
mv	Μετονομάζει αρχεία και καταλόγους
mw	Πρόσβαση στην μνήμη ram με 32-bit access
ps	Εμφανίζει πληροφορίες για τις διεργασίες που εκτελούνται στο σύστημα
pwd	Παρουσιάζει τον τρέχοντα κατάλογο (directory) που εργαζόμαστε (print working directory) μέσα από την συνολική διαδρομή (path)
rm	Διαγράφει ένα ή περισσότερα αρχεία

rmdir	Διαγράφει άδειους καταλόγους από το σύστημα αρχείων
set	Θέτει μια μεταβλητή περιβάλλοντος στην γραμμή εντολών
sh	Εκτελεί την ακολουθία των εντολών NSH στο αρχείο που αναφέρεται
sleep	Διακόπτει την εκτέλεση για μερικά seconds
test	Εκτελεί συγκεκριμένους ελέγχους
unmount	Αποπροσάρτηση κάποιου καταλόγου
unset	Διαγράφει μια μεταβλητή περιβάλλοντος από την γραμμή εντολών
usleep	Διακόπτει την εκτέλεση για μερικά micro seconds
xd	Δεκαεξαδική μορφή της μνήμης

Δουλεύοντας στο Nuttshell

Στη συνέχεια παρουσιάζουμε τον τρόπο εκτέλεσης ορισμένων εντολών και προγραμμάτων. Για να πραγματοποιήσουμε έναν έλεγχο των ενσωματωμένων αισθητηρίων στον μικροελεγκτή (επιταχυνσιόμετρο, γυροσκόπιο, μαγνητόμετρο, αισθητήρας πίεσης και θερμοκρασίας) πληκτρολογούμε την εντολή test sensors και enter. Στην οθόνη θα εμφανιστεί μια λίστα με τις ονομασίες των αισθητηρίων, οι τρέχουσες τιμές τους, καθώς και η ειδοποίηση σωστής λειτουργίας τους. Αν θέλουμε να ελέγξουμε το σύνολο του μικροελεγκτή (αισθητήρια, οπτική σήμανση, ηχείο, το ρολόι του επεξεργαστή, τους απαριθμητές κ.α.) τότε πληκτρολογούμε tests all και enter. Θα γίνει ένας πλήρης έλεγχος στις λειτουργίες της πλακέτας και στη συνέχεια θα εμφανιστούν τα αποτελέσματα. Για να πάρουμε δεδομένα θέσης από το GPS τότε πληκτρολογούμε gps status και enter, ώστε να εμφανιστεί η θύρα σύνδεσης του GPS, οι γεωγραφικές

συντεταγμένες και το υψόμετρο. Πληκτρολογώντας top και enter εμφανίζονται στατιστικά πραγματικού χρόνου για τις εφαρμογές που είναι ενεργοποιημένες και για την κατανάλωση σε πόρους του συστήματος, όπως μνήμη και επεξεργαστική ισχύς. Για την επιστροφή στην αναμονή του command line θα χρειαστεί να πατήσουμε το πλήκτρο Esc. Η εντολή reboot προκαλεί την επανεκκίνηση της πλακέτας PX4fmu. Μετά από την χρήση της θα χρειαστεί εκ νέου σύνδεση της πλακέτας με τον υπολογιστή. Τέλος, για να ξεκινήσουμε οποιαδήποτε εφαρμογή, όπως την attitude_estimator_ekf, τοτε πληκτρολογούμε την εντολή attitude_position_ekf start. Όταν οι εντολές έναρξης εφαρμογής δεν μας επιστρέφουν αποτέλεσμα, μπορούμε να πραγματοποιήσουμε έλεγχο της κατάστασης που βρίσκονται. Αυτό γίνεται είτε με την εντολή top, είτε με το όνομα της εφαρμογής συνοδευόμενο από την εντολή status, όπως για παράδειγμα attitude_estimator_ekf status. Για τον τερματισμό των εφαρμογών που έχουμε εκκινήσει μπορούμε να πληκτρολογήσουμε το όνομα της εφαρμογής μαζί με το stop π.χ. attitude_estimator_ekf stop ή την εντολή kill -9 και τον αριθμό διεργασίας PID που εμφανίζεται δίπλα στην εφαρμογή στα στατιστικά της top π.χ. kill -9 75.

Πρέπει να σημειωθεί πως πολλές εφαρμογές κατά την εκκίνηση τους στέλνουν συνεχώς δεδομένα στο παράθυρο τερματικού του υπολογιστή μας, χωρίς να υπάρχει η δυνατότητα να τις σταματήσουμε. Σε αυτήν την περίπτωση καταγράφουμε ό,τι δεδομένα χρειαζόμαστε και κάνουμε reset την πλακέτα του PX4fmu χειροκίνητα από τον διακόπτη reset δίπλα από την θύρα USB. Η συγκεκριμένη ενέργεια δεν αντιμετωπίζεται σαν πρόβλημα, διότι η εφαρμογή λειτουργεί με αυτόν τον τρόπο από κατασκευής της.

Αν θέλουμε να δούμε τα περιεχόμενα της κάρτας microSD που είναι τοποθετημένη στον PX4fmu θα πληκτρολογήσουμε την εντολή mount και enter. Τότε εμφανίζεται η ονομασία της κάρτας και το σύστημα αρχείων της π.χ. /fs/microsd type vfat. Για την πλοήγηση στους καταλόγους της κάρτας πληκτρολογούμε ls /fs/microsd και enter. Με την εντολή cd συνοδευόμενη από το path του κάθε φάκελου πετυχαίνουμε την είσοδο σε αυτούς, ενώ για να επιστρέψουμε στον κεντρικό φάκελο πληκτρολογούμε cd και enter.

Αναβάθμιση του λογισμικού του PX4IO μέσω του Nuttshell

Την πρώτη φορά που θα συνδέσουμε την πλακέτα του PX4IO με τη πλακέτα του PX4fmu θα πρέπει να εγκαταστήσουμε το firmware στην τελευταία. Για να το επιτύχουμε αυτό θα πρέπει να κατεβάσουμε το αρχείο px4io.bin από την ιστοσελίδα https://pixhawk.ethz.ch/px4/downloads και να το τοποθετήσουμε στην κάρτα microSD. Στη συνέχεια συνδέουμε τον μικροελεγκτή στην θύρα USB κρατώντας ταυτόχρονα πατημένο τον διακόπτη ασφαλείας του PX4IO. Συνδεόμαστε στο Nsh, ελέγχουμε και τερματίζουμε την εφαρμογή px4io, εάν έχει εκκινήσει, ώστε να γίνει η ενημέρωσή της κατά την επόμενη εκκίνηση με την εντολή px4io update. Περιμένουμε μέχρι να ολοκληρωθεί η διαδικασία και κάνουμε επανεκκίνηση του ελεγκτή.

Αυτόματη εκκίνηση εφαρμογών στο NuttX

Κατά την εκκίνησή του ο PX4 ελέγχει την πλακέτα, με την οποία είναι συνδεδεμένος (PX4IO ή PX4IOAR) και διαβάζει το αρχείο εκκίνησης rcS, που είναι αποθηκευμένο στο firmware (/etc/init.d/rcS). Στο αρχείο rcS καθορίζεται ο τύπος του αεροσκάφους και η λειτουργία της πτήσης π.χ. fw_attitude_control, η ενεργοποίηση των αισθητηρίων π.χ. sh /etc/init.d/rc.sensors, καθώς και η εκκίνηση των φίλτρων Kalman π.χ. attitude_estimator_ekf. Όταν θέλουμε να δοκιμάσουμε έναν νέο τύπο αεροσκάφους ή θέλουμε να εκκινήσουμε παράλληλα μία εφαρμογή, έχουμε την δυνατότητα να προσθέσουμε ένα αρχείο εκκίνησης rc.txt στην κάρτα microSD (/fs/microsd/etc/rc.txt). Σε επόμενη εκκίνηση του μικροελεγκτή θα γίνει έλεγχος για την ύπαρξη αρχείου, αν αυτό εντοπιστεί θα εκτελεστούν οι εντολές του, στην αντίθετη περίπτωση θα εκτελεστούν οι προκαθορισμένες εντολές του αρχείου rcS.

Δημιουργία αρχείου αυτόματης εκκίνησης rc.txt για σκάφος fixedwing

Για την δημιουργία ενός αρχείου rc.txt για την πτήση ενός αεροπλάνου fixedwing ανοίγουμε έναν απλό κειμενογράφο (όπως το σημειωματάριο των Windows) και γράφουμε τις εντολές που αναφέρονται παρακάτω, ενώ στη συνέχεια τοποθετούμε το αρχείο στην κάρτα microSD. Σε κάθε εκκίνηση θα εκτελούνται οι εντολές αυτές και θα

έχουμε έτοιμο τον μικροελεγκτή για πτήση, αφού θα έχουμε ορίσει όλες τις απαραίτητες παραμέτρους και εφαρμογές (τύπος αεροσκάφους, αισθητήρια, επικοινωνία, εφαρμογή υπολογισμού συμπεριφοράς σκάφους, καταγραφέα δεδομένων πτήσης, κ.ά.).

```
//!nsh//

//PX4FMU startup script//

//έναρξη σειριακής κονσόλας//

if sercon
then
        echo "USB connected"
fi

// έναρξη εφαρμογής uorb//

uorb start

//έλεγχος ύπαρξης αρχείου παραμέτρων params//


        param select /fs/microsd/params
        if [ -f /fs/microsd/params ]
        then
                if param load /fs/microsd/params
                then
                        echo "Parameters loaded"
                else
                        echo "Parameter file corrupt - ignoring"
                fi
        fi

//fi//
//επιλογή τύπου MAV //

 MAV_TYPE     1 = fixed wing, 2 = quadrotor, 13 = hexarotor
param set MAV_TYPE 1
//έναρξη των οπτικών ενδείξεων του μικροελεγκτή//
```

```
if rgbled start
then
	echo "Using external RGB Led"
else
	if blinkm start
	then
		blinkm systemstate
	fi
fi
```

// σύνδεση τερματικού υπολογιστή με το Nuttshell μέσω USB//

```
nshterm /dev/ttyACM0 &
```

//αναβάθμιση του PX4IO αν υπάρχει νεότερη έκδοση στην κάρτα microsd//

```
if px4io detect
then
	echo "PX4IO running, not upgrading"
else
	echo "Attempting to upgrade PX4IO"
	if px4io update
	then
		if [ -d /fs/microsd ]
		then
			echo "Flashed PX4IO Firmware OK" > /fs/microsd/px4io.log
		fi
```

//χρόνος για την επανασύνδεση του PX4IO αν γίνει αναβάθμιση//

```
		usleep 200000
	else
		echo "No PX4IO to upgrade here"
	fi
fi
```

//τερματισμός της κονσόλας του Nuttshell στο τέλος των εντολών//

```
set EXIT_ON_END no
```

//ρυθμίσεις PX4fmu και IO//

if px4io detect
then

//έναρξη mavlink για επικοινωνία Nuttshell με QGround //

mavlink start

//έναρξη commander//

commander start

//εκτέλεση εντολών αρχείου rc.io//

sh /etc/init.d/rc.io
else

//έναρξη mavlink για επικοινωνία Nuttshell με QGround control στην θύρα USB//

mavlink start -d /dev/ttyS0
commander start

// διαμόρφωση PWM για τα servo//

fmu mode_pwm
param set BAT_V_SCALING 0.004593
set EXIT_ON_END yes
fi

//έναρξη αισθητήρων//

sh /etc/init.d/rc.sensors

//έναρξη καταγραφής αισθητήρων //

sh /etc/init.d/rc.logging

//έναρξη GPS//

gps start

//έναρξη attitude και position estimator //

att_pos_estimator_ekf start

//έναρξη έλεγχου εξόδων για τα servo//

mixer load /dev/pwm_output /etc/mixers/FMU_AERT.mix

//έναρξη attitude και position controller//

```
  fw_att_control
  fw_pos_control_l1
```

//δημιουργία αρχείου καταγραφής δεδομένων πτήσης με συχνότητα 100Hz //

```
  sdlog2 start -r 100 -a
if [ $EXIT_ON_END == yes ]
then
        exit
fi
```

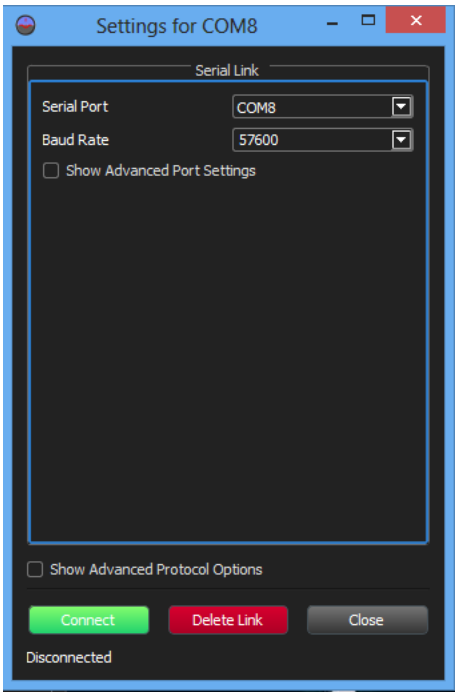
Σύνδεση στο γραφικό περιβάλλον QGround control

Η σύνδεση του γραφικού περιβάλλοντος QGround control με το Nuttshell γίνεται για να μας βοηθήσει να χειριστούμε τον μικροελεγκτή με ευκολία, χωρίς να πληκτρολογούμε εντολές στο τερματικό. Από το περιβάλλον μας παρέχεται η δυνατότητα χρήσης χάρτη για την επιλογή των σημείων πορείας, καθώς και γραφική αναπαράσταση όλων των παραμέτρων που αφορούν την πτήση του σκάφους. Για να συνδέσουμε τον PX4 στον υπολογιστή κάνουμε πρώτα εγκατάσταση το PX4 Toolchain για να γίνει η εγκατάσταση των απαραίτητων drivers. Εν συνεχεία εκκινούμε το interface

επικοινωνίας μεταξύ χρήστη και ελεγκτή και έπειτα επιλέγουμε το εικονίδιο PX4 autopilot.

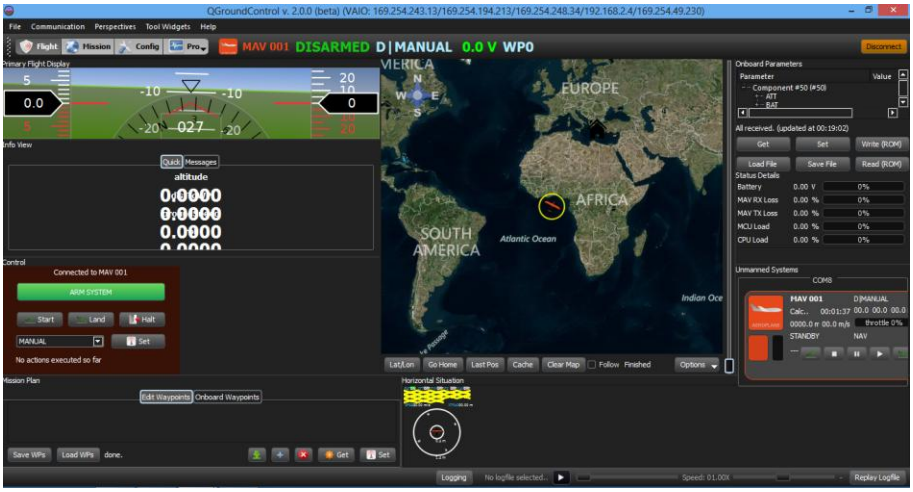
7-1. Επιλογή πλατφόρμας autopilot.

Για την ρύθμιση της θύρας επικοινωνίας στο QGround control επιλέγουμε από τις αναδυόμενες λίστες του μενού Communication, την επιλογή Add Link. Στο παράθυρο που εμφανίζεται (7-2), επιλέγουμε την σειριακή θύρα του PX4 και Baud Rate 57600 και επιλέγουμε Connect.



7-2. Επιλογή θύρας επικοινωνίας στο QGround control.

Σε περίπτωση επιτυχούς συνδέσεως του QGround control με τον PX4 εμφανίζεται η επόμενη οθόνη (7-3), που απεικονίζει την κατάσταση του ελεγκτή.



7-3. Περιβάλλον επικοινωνίας QGround control.

ΟΠΤΙΚΗ ΣΗΜΑΝΣΗ

Στην ενότητα που ακολουθεί, γίνεται αναφορά στην οπτική σήμανση, με χρήση φωτοδίοδων LED, των ενεργειών που λαμβάνουν μέρος στην πλακέτα του μικροελεγκτή PX4 FMU, καθώς και στην πλακέτα εισόδων και εξόδων του (I/O board). Οι λειτουργίες ξεχωρίζουν από τους συνδυασμούς των χρωμάτων, αλλά και από τον ρυθμό μεταβολής της κατάστασης των LED.

Στην πλακέτα PX4-FMU παρατηρούμε:

Πράσινο LED:

i. Η πράσινη φωτοδίοδος είναι σταθερά αναμμένη στην περίπτωση ενεργοποίησης της πλακέτας.

Μπλε LED:

i. Στην περίπτωση μη λειτουργίας της συγκεκριμένης φωτοδιόδου, έχουμε την απώλεια επικοινωνίας με τη συσκευή του GPS, ή άλλης συσκευής.
ii. Ο αργός ρυθμός μεταβολής της κατάστασης της φωτοδιόδου, σημαίνει πως επιτεύχθηκε η επικοινωνία με την συσκευή του GPS, ή άλλης συσκευής.
iii. Ο ταχύς ρυθμός μεταβολής της κατάστασης, σημαίνει πως υπάρχει δυσλειτουργία.
iv. Η σταθερή λειτουργία, σηματοδοτεί πως η συσκευή του GPS, έχει κλειδώσει την τοποθεσία που βρίσκεται.

Κόκκινο LED:

i. Ο αργός ρυθμός μεταβολής της κατάστασης του σηματοδοτεί την ετοιμότητα της συσκευής για να οπλιστεί.

ii. Ο ταχύς ρυθμός μεταβολής της κατάστασης, σηματοδοτεί δυσλειτουργία του συστήματος, στην διαδικασία αυτό-ελέγχου.
iii. Η σταθερή λειτουργία της φωτοδιόδου δηλώνει πως η συσκευή είναι οπλισμένη.

Ο συνδυασμός του κόκκινου-μπλε LED, με ηχητική ειδοποίηση, δηλώνει την ανάγκη για calibration των αισθητήρων.

Η ταχεία μεταβολή κατάστασης στο κόκκινο LED, αμέσως μετά την ενεργοποίηση της λειτουργίας του συστήματος, σηματοδοτεί την ετοιμότητα για εγκατάσταση νέου firmware, αν αυτή γίνεται μέσω USB.

Η σταθερή λειτουργία του μπλε LED, σε συνδυασμό με ηχητική ειδοποίηση (SOS, σε κώδικα Morse), δηλώνει δυσλειτουργία κατά την διαδικασία της ενεργοποίησης της πλακέτας. Πιθανών να οφείλεται στην απουσία της κάρτας microSD.

Στην πλακέτα I/O παρατηρούμε:

Πράσινο LED:

i. Σηματοδοτεί, με την αδιάκοπη λειτουργία του, την ενεργοποίηση της πλακέτας.

Μπλε LED:

i. Η αλλαγή κατάστασης με ρυθμό 2Hz, σηματοδοτεί την επικοινωνία της πλακέτας με το πρόγραμμα του ground control.
ii. Η κατάσταση μόνιμης λειτουργίας, σηματοδοτεί την εγκατάσταση νέου firmware, για την πλακέτα I/O.

Κόκκινο LED:

i. Η γρήγορη αλλαγή κατάστασης σηματοδοτεί την ενεργοποίηση του Bootloader (στην παρούσα περίπτωση, το μπλε LED θα πρέπει να είναι απενεργοποιημένο).
ii. Η αλλαγή κατάστασης με ρυθμό 4Hz, δηλώνει την απώλεια επικοινωνίας με την πλακέτα FMU.

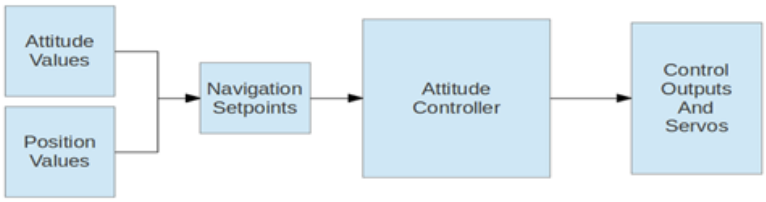
iii. Η μόνιμη κατάσταση λειτουργίας, δηλώνει δυσλειτουργία στην διαδικασία ενεργοποίησης. Θα πρέπει να γίνει έλεγχος των παραμέτρων του RC config.

Διακόπτης ασφάλειας:

i. Όταν βρίσκεται σε συνεχή λειτουργία δηλώνει την κατάσταση όπλισης των πλακετών FMU και I/O.
ii. Η αργή μεταβολή της κατάστασης του, σηματοδοτεί την ετοιμότητα του συστήματος για να οπλιστεί.
iii. Η γρήγορη μεταβολή της κατάστασης, σηματοδοτεί δυσλειτουργία και την άρνηση και όπλιση.
iv. Η κατάσταση μακράς απενεργοποίησης, που ακολουθείται από διπλή ενεργοποίηση, σηματοδοτεί τον οπλισμό της πλακέτας I/O, αλλά όχι της πλακέτας FMU.
v. Η κατάσταση μακράς απενεργοποίησης, που ακολουθείται από τετραπλή ενεργοποίηση, δηλώνει τον οπλισμό της πλακέτας FMU, αλλά όχι της πλακέτας I/O.

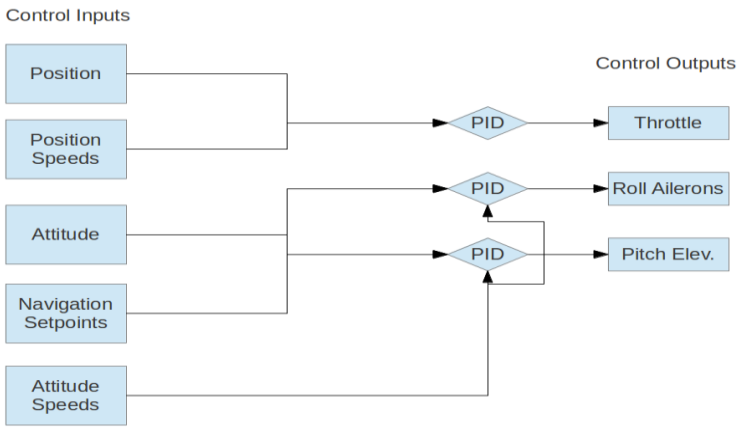
ΔΟΜΗ ΕΛΕΓΚΤΗ

Η κύρια δομή της εφαρμογής fixedwing control, αποτελείται από τρία παράλληλα τμήματα, όπως παρουσιάζονται και στο επόμενο σχήμα.



9-1. Δομή ελεγκτή fixedwing control [πηγή: www. pixhawk.ethz.ch.].

Χρησιμοποιώντας ένα σύνολο από τρείς ελεγκτές, γίνεται προσπάθεια για σταθεροποίηση και εξομάλυνση των κινήσεων του αεροσκάφους, σε σχέση με το roll, το pitch και την πρόωση. Λόγω του ότι τα αισθητήρια λαμβάνουν τις μετρήσεις τους, με βάση το earth-frame, θα πρέπει να υπάρξει μετατροπή των τιμών στο body frame για την χρήση τους από τον ελεγκτή. Τα αποτελέσματα των υπολογισμών θα αποθηκευτούν στις μεταβλητές rollb, pitchb και yawb αντίστοιχα.



9-2. Ελεγκτές κάθε μεταβλητής [πηγή: www. pixhawk.ethz.ch.].

Στον τομέα της πλοήγησης θα έχουμε έναν διαφορετικό βρόγχο. Σε αυτόν θα υπολογίζονται οι επιθυμητές τιμές του ύψους και της κατεύθυνσης του αεροσκάφους, καθώς και η σωστή πλοήγηση προς το επόμενο επιθυμητό σημείο (waypoint). Στον προγραμματισμό του μικροελεγκτή δεν έχει υπολογιστεί η περίπτωση του yaw rudder, επομένως η κίνηση του αεροσκάφους πραγματοποιείται μέσω των επιθυμητών τιμών του roll.

Το τελικό τμήμα των servos έχει ως σκοπό την σωστή τροφοδότηση των καναλιών των servo με τάση, ύστερα από την σωστή κλιμάκωση των εξόδων του ελεγκτή.

Η ροή δεδομένων γίνεται λαμβάνοντας από την εφαρμογή Orb τις παγκόσμιες συντεταγμένες, την θέση, τα setpoint, τις παραμέτρους και τις πληροφορίες προερχόμενες από το κανάλι της τηλεκατεύθυνσης και γίνονται αναγνώσιμες από όλες τις δομές. Οι δομές αυτές είναι:

- το Plane Data, που περιέχει τις μεταβλητές που περιγράφουν την κατάσταση του αεροσκάφους
- το Control Outputs, που περιέχει τις πληροφορίες για τις εξόδους του ελεγκτή (roll, pitch, yaw, πρόωση).
- Το PID Structure, που περιέχει τα κέρδη των παραμέτρων του ελεγκτή, τα όρια ολοκλήρωσης και την τελευταία κατάσταση του ελεγκτή.

Ο ελεγκτής τροφοδοτείται με αντίγραφα των δεδομένων και ύστερα από τον βρόγχο ελέγχου. Τα δεδομένα εξόδου δημοσιεύονται στην εφαρμογή Orb.

Βοηθητικές Συναρτήσεις

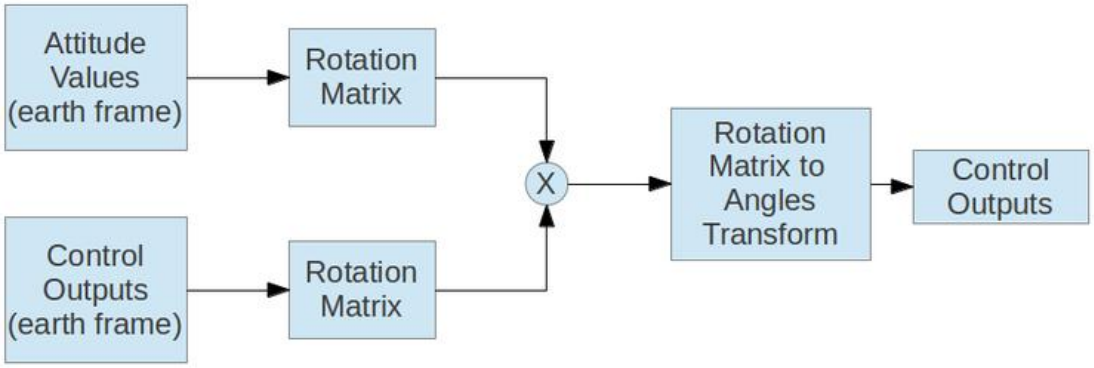
Στη συνέχεια παρουσιάζονται λειτουργίες βοηθητικών συναρτήσεων, που σκοπό έχουν τον υπολογισμό παραμέτρων και την ανάγνωση ή γραφή τιμών, για λογαριασμό άλλων συναρτήσεων.

Η Get Parameters πραγματοποιεί την λήψη των παραμέτρων από το Mission Planner (APM Planner, ή QGround Control).

Η Calculate PID Output αποτελείται από έναν αλγόριθμο PID με την προσθήκη ενός φίλτρου χαμηλών συχνοτήτων (Low Pass Filter) με σκοπό την κλιμάκωση των εξόδων του ελεγκτή.

Βοηθητικές Συναρτήσεις Πινάκων Περιστροφής

Ένα σύνολο τριών συναρτήσεων μας προσφέρει την μετατροπή των γωνιών της θέσης του αεροσκάφους σε πίνακες περιστροφής, την μετατροπή των πινάκων περιστροφής σε γωνίες θέσης του αεροσκάφους, με την εξαγωγή των γωνιών Euler και τον πολλαπλασιασμό των πινάκων.



9-3. Βοηθητικές Συναρτήσεις Πινάκων Περιστροφής [πηγή: www. pixhawk.ethz.ch.].

Βοηθητικές Συναρτήσεις Πλοήγησης

Ένα σύνολο από βοηθητικές συναρτήσεις συμβάλουν στην βέλτιστη πλοήγηση του αεροσκάφους. Προκύπτουν λοιπόν τα εξής: ο καθορισμός της κατεύθυνσης του

αεροσκάφους με βάση την πιο πρόσφατη θέση του, καθώς και την θέση του επόμενου waypoint, ο υπολογισμός της ταχύτητας σε σχέση με την επιφάνεια της Γής, ο υπολογισμός της απόστασης μέχρι το επόμενο waypoint και ο καθορισμός τον Plane Mode. Τα modes είναι: takeoff, cruise, loiter και land.

Βοηθητικές Συναρτήσεις Υπολογισμού των Setpoint

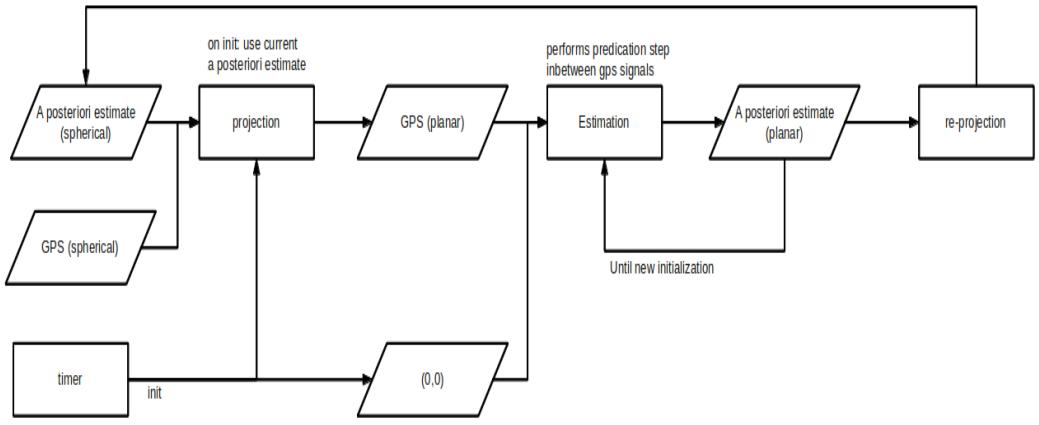
Το roll setpoint έχει ως σκοπό την διόρθωση της κατεύθυνσης του αεροσκάφους. Το setpoint προκύπτει από την διαφορά μεταξύ της γωνίας κατεύθυνσης και της γωνίας yaw του αεροσκάφους. Το pitch setpoint έχει ως σκοπό της διόρθωση του υψόμετρου. Το setpoint υπολογίζεται λαμβάνοντας υπόψη την διαφορά του ύψους του αεροσκάφους με το ύψος του επιθυμητού σημείου, καθώς και της απόστασης από το επιθυμητό σημείο. Το throttle setpoint, καθορίζει την πρόωση του αεροσκάφους. Αυτή εξαρτάται από το mode που βρίσκεται το αεροσκάφος, π.χ. το takeoff απαιτεί μέγιστη πρόωση, το land απαιτεί μηδενική πρόωση, ενώ το cruise χρειάζεται μία μέση τιμή.

Συναρτήσεις Εξόδων

Οι συναρτήσεις roll output, pitch output και throttle output παρέχουν τις τελικές τιμές εξόδων από τον ελεγκτή. Το σφάλμα, που προκύπτει από την αφαίρεση των πραγματικών μετρήσεων του αεροσκάφους με τις επιθυμητές τιμές, θα πολλαπλασιαστεί με το αντίστοιχο κέρδος και ως αποτέλεσμα θα έχουμε τις εξόδους του ελεγκτή.

Position Estimator

Ο εκτιμητής θέσης -position estimator- δέχεται σαν είσοδο τις τιμές του GPS, το πραγματικό αλλά και το επιθυμητό υψόμετρο με σκοπό να υπολογίσει την θέση του UAV στον χώρο. Το σήμα του GPS, καθώς και η έξοδος της εφαρμογής position estimator εκφράζονται σε σχέση με το Γεωγραφικό Σύστημα Συντεταγμένων (Γεωγραφικό Μήκος, Γεωγραφικό Πλάτος, Υψόμετρο). Δεδομένου ότι οι υπολογισμοί γίνονται σε σχέση με το σώμα του UAV, θα πρέπει να υπάρξει μεταφορά των τιμών σε άλλο σύστημα συντεταγμένων και επαναφορά αυτών στο αρχικό, ύστερα από το πέρας των υπολογισμών. Για την μεταφορά στα συστήματα συντεταγμένων χρησιμοποιείται η Αζιμουθιακή Προβολή. Στην διάρκεια εξέλιξης θα αποτελέσει μια σημαντική είσοδο για την PX4FLOW Smart Camera.



9-4. Block διάγραμμα του position estimator [πηγή: www.pixhawk.ethz.ch].

Οι εξισώσεις μετασχηματισμού που προκύπτουν είναι:

$$X = K' \, Cos(\varphi) \, Sin(\lambda - \lambda_0)$$

$$Y = K' \, [\, Cos(\varphi_1) \, Sin(\varphi) - Sin(\varphi_1) \, Cos(\varphi) \, Cos(\lambda - \lambda_0) \,]$$

Όπου:

φ_1 : το γεωγραφικό πλάτος της προβολής

λ_0 : το γεωγραφικό μήκος της προβολής

$K' = \frac{c}{Sin\,(c)}$

c: η γωνιακή απόσταση από το κέντρο

$Cos(c) = Sin(\varphi_1)\,Sin(\varphi) + Cos(\varphi_1)\,Cos(\varphi)\,Cos(\lambda - \lambda_0)$

Για τον ανάστροφο μετασχηματισμό έχουμε τις παρακάτω εξισώσεις:

$Y = \lambda_0 + Tan^{-1}\left(\frac{x\,Sin\,c}{c\,Cos(\varphi 1)Cos\,c - y\,Sin(\varphi 1)Sin\,c}\right)$ για $\varphi_1 \neq \pm 90°$

$Y = \lambda_0 + Tan^{-1}\left(-\frac{x}{y}\right)$ για $\varphi_1 \neq \pm 90°$

$Y = \lambda_0 + Tan^{-1}\left(\frac{x}{y}\right)$ για $\varphi_1 \neq \pm 90°$

Όπου $c = \sqrt{x^2 + y^2}$

Ο εκτιμητής κατασκευάστηκε αρχικά στο Matlab και στη συνέχεια μεταφέρθηκε στη γλώσσα προγραμματισμού C. Το αρχείο position_estimator_main.c περιέχει τον κώδικα επικοινωνίας μεταξύ του Matlab και του κώδικα C. Επίσης, είναι το βασικό εργαλείο που διαβάζει τις τιμές των οργάνων μέτρησης και πραγματοποιεί τους απαραίτητους μετασχηματισμούς. Η διαδικασία της εκτίμησης πραγματοποιείται με την

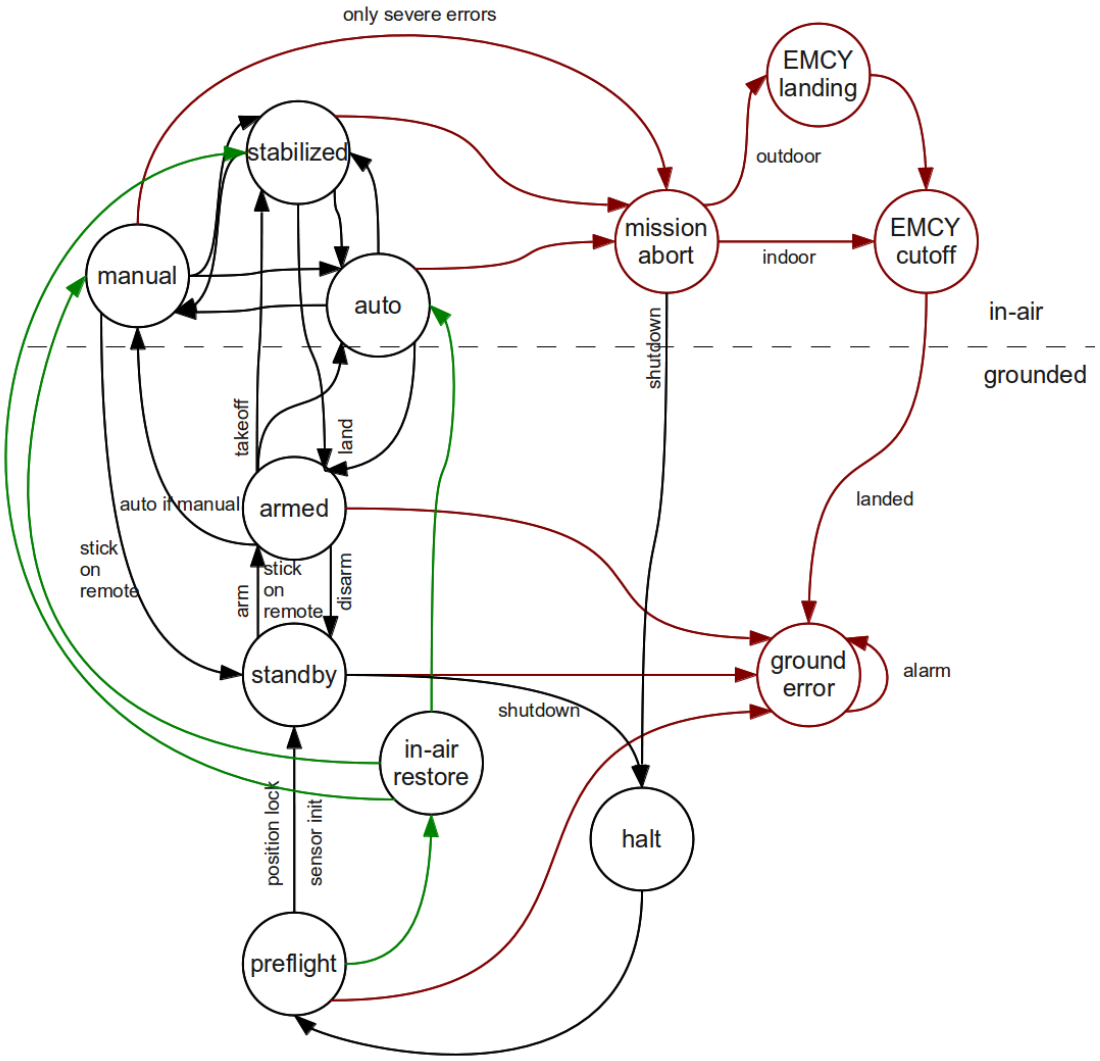
κλήση της συνάρτησης position_estimator. Το σύνολο του κώδικα C, μπορεί να βρεθεί στο position_estimator/codegen subdirectory.

COMMANDER

Πρόκειται για την εφαρμογή που μετατρέπει τα αποτελέσματα των ψηφιακών κυκλωμάτων σε εξόδους, με βάση την Λογική Πρώτου Βαθμού. Ο πίνακας που ακολουθεί αποτελεί το σύνολο των διακριτών καταστάσεων και των εξόδων της εφαρμογής.

10-1. Πίνακας διακριτών καταστάσεων και εξόδων [πηγή: www.pixhawk.ethz.ch].

State name	Description	armed	att mode	vel mode	pos mode	pos setpoint
PREFLIGHT	Uninitialized system	false	any	off	off	none
STANDBY	Ready to be armed	false	any	off	off	none
GROUND_READY	Armed, still on ground	true	any	off	off	none
MANUAL	Armed, > 30% thrust	true	rate or att	off	off	none
STABILIZED	Armed, takeoff cmd	true	att	on / off	on / off	stick/offboard
AUTO	Armed, takeoff cmd	true	att	on / off	on / off	waypoints
MISSION_ABORT		true	any	on / off	on / off	any
EMCY_LANDING		true	any	on / off	on / off	any
EMCY_CUTOFF		true	any	on / off	on / off	any
GROUND_ERROR		false	any	on / off	on / off	any



10-1. Διάγραμμα ροής μεταξύ των καταστάσεων [πηγή: www.pixhawk.ethz.ch].

Στην χειροκίνητη πτήση (με χρήση της τηλεκατεύθυνσης) η σειρά λογικής που ακολουθείται είναι:

1. Preflight, οπού αμέσως ακολουθείται από το
2. Standby
3. Ground_ready, οπλίζοντας (παραμένει στο έδαφος, με τους κινητήρες σε λειτουργία)
4. Manual, όπου δίνεται το 30% της πρόωσης για χρόνο 0,5s
5. Ground_ready, αφοπλίζοντας και ακολουθούμενο από το
6. Standby

Στην αυτόνομη πτήση η σειρά λογικής θα είναι παρόμοια με μόνη διαφορά το τέταρτο βήμα:

1. Preflight, οπού αμέσως ακολουθείται από το
2. Standby
3. Ground_ready, οπλίζοντας (παραμένει στο έδαφος, με τους κινητήρες να περιστρέφονται)
4. Auto, με την εντολή απογείωσης
5. Ground_ready, με την εντολή προσγείωσης (και παραμένοντας στο έδαφος με τους κινητήρες σε λειτουργία)
6. Standby, με την εντολή αφόπλισης

Στην συνέχεια θα αναλύσουμε χωριστά την λειτουργία της κάθε κατάστασης.

Preflight: Με την εκκίνηση του συστήματος έχουμε είσοδο στην κατάσταση Preflight. Την στιγμή αυτή, το σύστημα δεν μπορεί να οπλιστεί και όλα τα περιφερειακά συστήματα πρέπει να είναι απενεργοποιημένα, λόγω του ότι δεν είναι γνωστό αν το

σύστημα είναι σε κατάσταση ετοιμότητας. Σε αυτό το στάδιο πρέπει να πραγματοποιηθεί το calibration των αισθητήρων και οποιαδήποτε ενέργεια συντήρησης του συστήματος.

Standby: Στην κατάσταση standby, το σύστημα έχει κάνει έλεγχο των βασικών λειτουργιών και επαλήθευση της σωστής λειτουργίας των κύριων περιφερειακών συσκευών του.

Ground_ready: Το σύστημα είναι έτοιμο προς απογείωση. Σε περίπτωση δυσλειτουργίας (μηχανισμού ή αισθητήρα), το σύστημα ενημερώνει τον χειριστή με ανάλογο μήνυμα.

Manual: Στην κατάσταση manual, το σύστημα είναι σε πτήση με απομακρυσμένο έλεγχο. Στην περίπτωση βλάβης έχουμε τις εξής αντιδράσεις:

- Σε περιπτώσεις πολύ σοβαρών δυσλειτουργιών το σύστημα μεταβαίνει σε κατάσταση ματαίωσης της αποστολής. Δυσλειτουργίες μικρότερου βαθμού εμφανίζονται στον σταθμό του χειριστή, ώστε στην περίπτωση του ικανού χειριστή, να δίνεται η δυνατότητα για χειροκίνητη προσγείωση του συστήματος.
- Το κάθε όργανο έχει διαφορετική βαρύτητα για ξεχωριστά συστήματα. Στην περίπτωση δυσλειτουργίας του γυροσκόπιου, ένα σύστημα αεροσκάφους, μπορεί να συνεχίσει την αποστολή του, ενώ το σύστημα των multirotors, θα πρέπει αναγκαστικά να μεταβεί σε κατάσταση ματαίωσης της αποστολής.

Stabilized: Στην κατάσταση stabilized, το σύστημα σταθεροποιεί το ύψος πτήσης, ενώ ο έλεγχος πορείας του γίνεται από τον απομακρυσμένο έλεγχο.

Auto: Το σύστημα μεταβαίνει στα καθορισμένα σημεία, όπως αυτά έχουν οριστεί από τον χειριστή, καθώς ταυτόχρονα ελέγχει το ύψος και την θέση του.

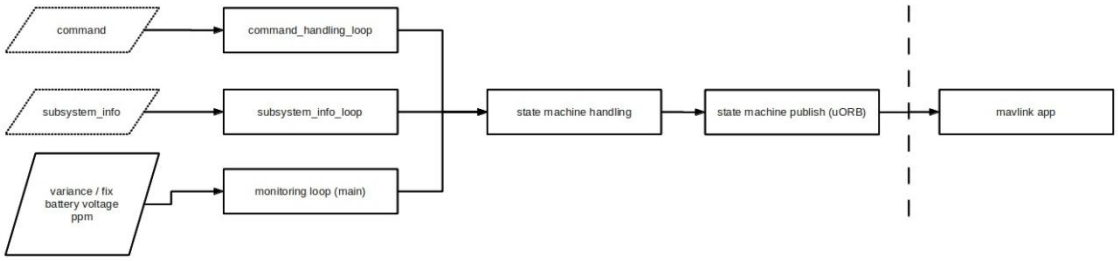
Mission_abort: Στην περίπτωση σοβαρής δυσλειτουργίας αυτόματα πραγματοποιείται επανεκκίνηση του συστήματος. Αν οι επανειλημμένες επανεκκινήσεις δεν έχουν διορθώσει το πρόβλημα, το σύστημα μεταβαίνει στην κατάσταση EMCY landing (έκτακτη προσγείωση), ή στην κατάσταση EMCY cutoff (έκτακτη απενεργοποίηση), ανάλογα με την κατάσταση της πτήσης (παράμετρος FLIGHT_ENV).

EMCY_landing: Ο ελεγκτής προσπαθεί να προσγειώσει το σύστημα και μεταβαίνει στη κατάσταση EMCY cutoff.

EMCY_cutoff: Το σύνολο των κινητήρων απενεργοποιούνται και το σύστημα μεταβαίνει στη κατάσταση ground error.

Ground_error: Το σύστημα είναι προσγειωμένο και με την χρήση χαρακτηριστικού ήχου ενημερώνει τον χειριστή για την κατάσταση δυσλειτουργίας.

Η δομή λειτουργίας της εφαρμογής commander δίνεται με βάση το επόμενο διάγραμμα.



10-2. Block διάγραμμα λειτουργίας commander [πηγή: www.pixhawk.ethz.ch].

Η εφαρμογή περιέχει ορισμένους βρόγχους που λειτουργούν αυτόνομα.

- Command_handling_loop: Η εφαρμογή Mavlink προωθεί τις εντολές του χειριστή στην εφαρμογή commander. Ανάλογα την εντολή, αλλάζει η κατάσταση του συστήματος.
- Subsystem info loop: Εδώ γίνεται η ενημέρωση για την λειτουργία, ή όχι, των υποσυστημάτων (π.χ. αισθητήρων). Πιθανές ενημερώσεις είναι present / not present, enabled / disabled, healthy / unhealthy.
- Monitoring loop: Πραγματοποιεί τους ακόλουθους ελέγχους και καθορίζει την μετάβαση του συστήματος σε κάποια κατάσταση,
    - Την επικοινωνία των αισθητήρων,
    - Την είσοδο ppm, από τον δέκτη της τηλεκατεύθυνσης, για την θέση του μοχλού σε οπλισμένο, ή αφοπλισμένο σύστημα,
    - Την τάση της μπαταρίας.

ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΚΩΔΙΚΑ ΕΛΕΓΧΟΥ

Οι βιβλιοθήκες του κώδικα αλλά και το σύνολό του γενικά, που συμπεριλαμβάνονται στην πλατφόρμα του PX4 και συνοδεύουν το hardware του μικροελεγκτή, υλοποιούν το project του αυτόματου πιλότου. Οι βιβλιοθήκες που υπάρχουν, υποστηρίζουν UAV πτήσεις τόσο της κατηγορίας fixed-wing (αντικείμενο της παρούσας εργασίας) όσο και της κατηγορίας multirotor (π.χ. quad-copters). Εκτός των αρχείων header που η C++ περιλαμβάνει στον μεταγλωττιστή της, οι προγραμματιστές του project έχουν ορίσει και δικά τους αντίστοιχα αρχεία, τα οποία μπορούν να συμπεριληφθούν στην βιβλιοθήκη χρόνου εκτέλεσης. Οι συναρτήσεις των αρχείων αυτών πραγματεύονται τον υπολογισμό των παραμέτρων ενός ελεγκτή τον έλεγχο της μεταβολής των γωνιών Euler με σκοπό την επιθυμητή πορεία του σκάφους κ.ά.

Ενδεικτικά παραθέτουμε ορισμένα αρχεία προγράμματος ελέγχου της συμπεριφοράς του σκάφους, τα οποία μπορούν να κατηγοριοποιηθούν ως εξής:

- ενεργοποίηση IMU και καταγραφή δεδομένων (sensors_params.c, sensors.cpp)

- υπολογισμός στιγμιαίας θέσης σκάφους (position_estimator_main.c, position_estimator_inav_main.c, position_estimator_inav_params.c)

- έλεγχος συμπεριφοράς σκάφους (fw_att_control_main.cpp, fw_att_control_params.c, fixedwing_pos_control_main.c, fixedwing_att_control_main.c, fixedwing_att_control_att.c, fixedwing_att_control_rate.c, attitude_estimator_ekf_main.cpp, kalman_main.cpp, navigator_main.cpp, attitude_estimator_so3_comp_main.cpp κ.α).

Επίσης, υπάρχουν αρχεία κώδικα που αφορούν στη βαθμονόμηση των οργάνων μέτρησης και των αισθητηρίων (accelerometer_calibration.cpp, gyro_calibration.cpp).

ΕΠΙΚΟΙΝΩΝΙΑ ΜΕΤΑΞΥ ΔΙΑΔΙΚΑΣΙΩΝ ΚΑΙ ΕΦΑΡΜΟΓΩΝ ΣΤΟΝ PX4

Για την λειτουργία του μικροελεγκτή PX4 είναι απαραίτητη η επικοινωνία των διαδικασιών και των εφαρμογών που τρέχουν σε αυτόν. Για παράδειγμα η εφαρμογή που φιλτράρει τα δεδομένα των αισθητηρίων μέσω Kalman έχει ανάγκη να επικοινωνεί με την εφαρμογή, όπου καταγράφονται οι μετρήσεις των αισθητηρίων, ώστε να επεξεργάζεται τις μετρήσεις αυτές. Η αρχιτεκτονική του λογισμικού στον PX4 για την επικοινωνία αυτή χρησιμοποιεί το πρότυπο publish-subscribe, ένα πρωτόκολλο ανταλλαγής μηνυμάτων, μεταξύ των διαδικασιών και των εφαρμογών που λειτουργούν στον μικροϋπολογιστή, μέσω διακριτών διαύλων (topics) με καθορισμένο όνομα. Κάθε δίαυλος στον PX4 περιέχει μόνο ένα είδος μηνύματος, π.χ. το topic vehicle_attitude μεταφέρει ένα μήνυμα που περιέχει τις χαρακτηριστικές τιμές του struct attitude.

Οι διαδικασίες/εφαρμογές, συχνά αναφέρονται και ως nodes, μπορούν να κάνουν publish ένα μήνυμα σε ένα bus/topic ώστε να στείλουν δεδομένα ή να κάνουν subscribe σε ένα topic και να λαμβάνουν τα δεδομένα που μία άλλη εφαρμογή κάνει publish. Στην πραγματικότητα, ένας node δεν γνωρίζει με ποιον επικοινωνεί κάθε φορά.

Η διαδικασία αυτή μπορεί να προσομοιαστεί με το κοινωνικό δίκτυο twitter. Ο χρήστης Α κάνει follow (subscribe) ένα λογαριασμό Β (topic) και όταν ο διαχειριστής του λογαριασμού Β κάνει μια δημοσίευση (publish) σε αυτό όσοι έχουν κάνει follow τον λογαριασμό Β ενημερώνονται για κάθε του ανανέωση.

Διαδικασίες Publishing- Subscribing

Publishing: Η διαδικασία του publishing περιλαμβάνει τρεις δραστηριότητες, αυτήν του καθορισμού του topic, της δημοσίευσής του, αλλά και της δημοσίευσης ανανεώσεών. Για να καθοριστεί ένα topic ο publisher δημιουργεί ένα header αρχείο διαθέσιμο στους πιθανούς subscribers χρησιμοποιώντας την εντολή orb_declare. Η

δημοσίευση γίνεται μετά την δημιουργία του topic κάνοντας χρήση των εντολών orb_define και orb_advertise και η ενημέρωση του topic γίνεται με την χρήση του orb_publish. Με την χρήση της orb_declare και με παράμετρο σε αυτήν το όνομα του topic γίνεται δημιουργία του header file. Ακολούθως της orb_declare δίνεται η μορφή της δημοσίευσης που θα γίνεται.

Π.χ.
ORB_DECLARE(random_integer);
struct random_integer_data {
int r;
};

Εδώ δηλώνεται ένα topic με τίτλο random_integer μέσω του οποίου θα γίνεται publish μία δομή (*random_integer_data).* Μετά την δημιουργία του topic, η συνάρτηση orb_define με παράμετρο το όνομα του, δηλώνει ότι το ακολουθεί. Έπειτα δηλώνεται η μεταβλητή στην οποία θα αποθηκεύονται οι ενημερώσεις του topic οι οποίες θα γίνονται publish μέσω της orb_publish.

Subscribing: Για να ενημερωθεί μια εφαρμογή από ένα topic στο οποίο έχει κάνει subscribe, χρησιμοποιεί τη συνάρτηση orb_subscribe και δίνει σε μια μεταβλητή την τιμή του topic από το οποίο ενημερώνεται. Στη συνέχεια, αφού ελέγξει αν το topic ανανεώθηκε, με την συνάρτηση orb_check αντιγράφει με το orb_copy τα νέα δεδομένα του διαύλου.

Παράδειγμα διαδικασίας publishing-subscribing:

topic.h
ORB_DECLARE(random_integer);
struct random_integer_data {int r;
};
publisher.c
#include <topic.h>

```
ORB_DEFINE(random_integer);

static int topic_handle;
int
init()
{
struct random_integer_data rd = { .r = random(), };
topic_handle = orb_advertise(ORB_ID(random_integer), &rd);
}
int
update_topic()
{
struct random_integer_data rd = { .r = random(), };
orb_publish(ORB_ID(random_integer), topic_handle, &rd);
}
```

subscriber.c

```
#include <topic.h>
static int topic_handle;
int
init()
{
topic_handle = orb_subscribe(ORB_ID(random_integer));
}
void
check_topic()
{
bool updated;
struct random_integer_data rd;
orb_check(topic_handle, &updated);
if (updated) {
orb_copy(ORB_ID(random_integer), topic_handle, &rd);
printf("Random integer is now %d\n", rd.r);
}
}
```

ATTITUDE CONTROLLER

```
#include <nuttx/config.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <math.h>
#include <poll.h>
#include <time.h>
#include <drivers/drv_hrt.h>
#include <uORB/uORB.h>
#include <uORB/topics/vehicle_global_position.h>
#include <uORB/topics/vehicle_global_position_setpoint.h>
#include <uORB/topics/vehicle_attitude.h>
#include <uORB/topics/vehicle_status.h>
#include <uORB/topics/vehicle_attitude_setpoint.h>
#include <uORB/topics/manual_control_setpoint.h>
#include <uORB/topics/actuator_controls.h>
#include <uORB/topics/vehicle_rates_setpoint.h>
#include <uORB/topics/vehicle_global_position.h>
#include <uORB/topics/parameter_update.h>
#include <systemlib/param/param.h>
#include <systemlib/pid/pid.h>
#include <geo/geo.h>
#include <systemlib/perf_counter.h>
#include <systemlib/systemlib.h>
#include <systemlib/err.h>
#include "params.h"
```

//Στο κομμάτι του κώδικα που προηγήθηκε, παρατηρούμε την χρήση της δεσμευμένης λέξης include, που χρησιμοποιείται για να συμπεριλάβει μία σειρά από header files, τα οποία περιέχουν έτοιμες προς χρήση συναρτήσεις, που ο χρήστης μπορεί να χρησιμοποιήσει χωρίς να τις ορίσει προηγουμένως.//

```
__EXPORT int ex_fixedwing_control_main(int argc, char *argv[]);
int fixedwing_control_thread_main(int argc, char *argv[]);
static void usage(const char *reason);
void control_attitude(const struct vehicle_attitude_setpoint_s *att_sp, const struct
vehicle_attitude_s *att,float speed_body[], struct vehicle_rates_setpoint_s *rates_sp,
struct actuator_controls_s *actuators);
```

//Στο δεύτερο τμήμα κώδικα, γίνονται δηλώσεις κάποιων πρωτοτύπων συναρτήσεων, ώστε να μπορεί να τις καλέσει αργότερα ο χρήστης. Σε επόμενο βήμα γίνεται μετά τη δήλωση των πρωτοτύπων και ο πλήρης ορισμός της κάθε ρουτίνας, δηλαδή ο επιστρεφόμενος τύπος, οι παράμετροι και ο κώδικας.//

```
static bool thread_should_exit = false;
static bool thread_running = false;
static int deamon_task;
static struct params p;
static struct param_handles ph;
```

//Στο σημείο αυτό, δηλώνονται κάποιες μεταβλητές και μέλη δεδομένων, όπου με την χρήση του static αποκτούν internal linkage (δηλαδή δεν είναι το όνομα τους ορατό έξω από το αρχείο στο οποίο ορίζονται) και επιπλέον αποκτούν static duration (δηλαδή όταν το πρόγραμμα που τις δημιούργησε τελειώσει οι μεταβλητές αυτές ‘‘αποσυντίθενται’’ ελευθερώνοντας χώρο στη μνήμη).//

//Εδώ ορίζεται η συνάρτηση control_attitude της οποίας το πρωτότυπο έχει δηλωθεί προηγουμένως.//

```
void control_attitude(const struct vehicle_attitude_setpoint_s *att_sp, const struct
vehicle_attitude_s *att,float speed_body[], struct vehicle_rates_setpoint_s *rates_sp,
struct actuator_controls_s *actuators)
{
        float roll_err = att->roll - att_sp->roll_body;
```

```
        actuators->control[0] = roll_err * p.roll_p;
        float pitch_err = att->pitch - att_sp->pitch_body;
        actuators->control[1] = pitch_err * p.pitch_p;
}
```

//Η συνάρτηση control_attitude πραγματοποιεί έλεγχο αναλογίας στις δύο από τις τρεις γωνίες Euler, τη γωνία roll και τη γωνία pitch. Δέχεται σαν παραμέτρους τέσσερις μεταβλητές τύπου struct (δηλαδή γενικές μεταβλητές που αποτελούνται από άλλες πολλές μεταβλητές, συναφείς μεταξύ τους ενώ μπορεί να είναι διαφορετικού τύπου και οι οποίες υπομεταβλητές καλούνται μέλη των struct) και μία float η οποία εδώ δεν χρησιμοποιείται, υπολογίζει τις δύο μεταβλητές roll_err και pitch_err κάνοντας χρήση των μελών roll και pitch της μεταβλητής att. Αυτές, είναι οι πραγματικές τιμές των γωνιών τη στιγμή της μέτρησης και των μελών roll_body και pitch_body της μεταβλητής att_sp που είναι οι επιθυμητές τιμές για τις γωνίες. Από τη διαφορά πραγματικού-επιθυμητού προκύπτει το σφάλμα. Στη συνέχεια έχουμε τον πολλαπλασιασμό του σφάλματος με ένα κέρδος p (τα μέλη roll_p και pitch_p της μεταβλητής p αντίστοιχα) και το αποτέλεσμα αποθηκεύεται στον πίνακα control μέλος της struct actuators. Στον πίνακα προβλέπονται οι παρακάτω θέσεις για τα πεδία ελέγχου με το εύρος να κυμαίνεται μεταξύ των εξής τιμών:

Control Group 0 (attitude):

0 - roll (-1..+1)

1 - pitch (-1..+1)

2 - yaw (-1..+1)

3 - thrust (0..+1)

4 - flaps (-1..+1).//

//Η συνάρτηση control_heading ελέγχει την κατεύθυνση του σκάφους μέσω των γωνιών yaw και roll.//

```
void control_heading(const struct vehicle_global_position_s *pos, const struct
vehicle_global_position_setpoint_s *sp,const struct vehicle_attitude_s *att, struct
vehicle_attitude_setpoint_s *att_sp)
{
        float bearing = get_bearing_to_next_waypoint(pos->lat/1e7d, pos->lon/1e7d,
sp->lat/1e7d, sp->lon/1e7d);
        float yaw_err = att->yaw - bearing;
        float roll_command = yaw_err * p.hdng_p;
        if (att_sp->roll_body < -0.6f) {
                att_sp->roll_body = -0.6f;
        } else if (att_sp->roll_body > 0.6f) {
                att_sp->roll_body = 0.6f;
        }
}
```

//Η συνάρτηση control_heading δέχεται σαν παραμέτρους, τέσσερις μεταβλητές struct. Η pos που αντιπροσωπεύει την στιγμιαία θέση του σκάφους σε longitude και latitude συντεταγμένες, η sp που είναι η επιθυμητή θέση και εκφράζεται επίσης σε παγκόσμιες συντεταγμένες, η att που περιέχει όλες τις τιμές των ελεγχόμενων παραμέτρων (roll,pitch,yaw….) και τέλος η att_sp που είναι η επιθυμητή att.

Η μεταβλητή bearing περιέχει τη γωνία yaw που πρέπει να φτάσει το σκάφος, ώστε η κατεύθυνση του να το οδηγήσει στο επόμενο waypoint. Η γωνία αυτή υπολογίζεται μέσα στην get_bearing_to_next_waypoint, η οποία δέχεται σαν παραμέτρους τις στιγμιαίες παγκόσμιες συντεταγμένες και αυτές του επόμενου waypoint. Αφού τις μετατρέψει σε γωνίες κατεύθυνσης εκφρασμένες σε rad, υπολογίζει και επιστρέφει την μεταξύ τους απόκλιση. Στη συνέχεια υπολογίζεται το yaw_err όπως προηγουμένως, δηλαδή η διαφορά μεταξύ του στιγμιαίου yaw και του επιθυμητού, που σημαίνει η διαφορά μεταξύ του μέλους yaw της struct μεταβλητής att και του baering. Επειτα υπολογίζεται η μεταβλητή που θα μεταβιβάζεται σαν εντολή ελέγχου του roll, yaw_err πολλαπλασιαζόμενο με το κέρδος p που προβλέπεται για τη συγκεκριμένη περίσταση, δηλαδή το μέλος hdng_p της μεταβλητής p. Τέλος γίνεται ένας έλεγχος της τιμής της μεταβλητής που ελέγχει την έξοδο του ελεγκτή, όσον αφορά την γωνία roll. Είδαμε ότι

το εύρος των τιμών ελέγχου είναι το (-1,1). Επειδή οι μεταβολές θέλουμε να είναι ελεγχόμενες και η κίνηση του σκάφους όσο το δυνατόν ομαλότερη, με τον έλεγχο που γίνεται, μειώνεται το εύρος της μεταβλητής από (-1, 1) στο (-0.6, 0.6) το οποίο έχει επίπτωση και στην μέγιστη τιμή της γωνίας roll που επιτρέπει ο ελεγκτής να πάρει το σκάφος.//

//Το main thread του προγράμματος.//

```
int  fixedwing_control_thread_main(int argc, char *argv[])
{
bool verbose = false;
for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-v") == 0 || strcmp(argv[i], "--verbose") == 0) {
        verbose = true;
                }
}
warnx("[example fixedwing control] started");
parameters_init(&ph);
parameters_update(&ph, &p);
```

//Το πρόγραμμα εμφανίζει ένα μήνυμα έναρξης της διαδικασίας στην οθόνη και στη συνέχεια αρχικοποιεί τις βοηθητικές παραμέτρους των οργάνων-αισθητηρίων.//

//Στη συνέχεια δηλώνει και με τη βοήθεια της συνάρτησης memset αρχικοποιεί με την τιμή μηδέν όλες τις δομές (structs) που στη συνέχεια θα χρησιμοποιήσει. Οι δομές αυτές είναι οι μεταβλητές που περιέχουν παραμέτρους όπως οι γωνίες Euler, οι γωνιακές ταχύτητες του σκάφους, οι παγκόσμιες συντεταγμένες της στιγμιαίας του θέσης και του waypoint, οι έξοδοι των ελεγκτών των γωνιών κλπ.//

```
struct vehicle_attitude_s att;
memset(&att, 0, sizeof(att));
struct vehicle_attitude_setpoint_s att_sp;
memset(&att_sp, 0, sizeof(att_sp));
struct vehicle_rates_setpoint_s rates_sp;
memset(&rates_sp, 0, sizeof(rates_sp));
struct vehicle_global_position_s global_pos;
memset(&global_pos, 0, sizeof(global_pos));
struct manual_control_setpoint_s manual_sp;
memset(&manual_sp, 0, sizeof(manual_sp));
struct vehicle_status_s vstatus;
memset(&vstatus, 0, sizeof(vstatus));
struct vehicle_global_position_setpoint_s global_sp;
memset(&global_sp, 0, sizeof(global_sp));
struct actuator_controls_s actuators;
memset(&actuators, 0, sizeof(actuators));
```

//Εδώ, μηδενίζονται όλες τις τιμές του πίνακα control, μέλους της actuators, ο οποίος περιέχει τα σήματα ελέγχου που θα στείλει ο ελεγκτής στους κινητήρες.//

```
for (unsigned i = 0; i < NUM_ACTUATOR_CONTROLS; i++) {
                actuators.control[i] = 0.0f;
        }
```

//Στο τμήμα που ακολουθεί, ανακοινώνει ότι θα κάνει publish και σε εφαρμογές εκτός του παρόντος προγράμματος τις μεταβλητές actuator_pub και rates_pub.//

```
orb_advert_t actuator_pub =
orb_advertise(ORB_ID_VEHICLE_ATTITUDE_CONTROLS, &actuators);
orb_advert_t rates_pub = orb_advertise(ORB_ID(vehicle_rates_setpoint), &rates_sp);
```

//Εδώ, πραγματοποιείται το subscribe, σε topics άλλων εφαρμογών, για να ενημερώνεται για τις μεταβολές μεταβλητών όπως η θέση του σκάφους, οι γωνίες Euler, τα setpoint κ.α.//

```
int att_sub = orb_subscribe(ORB_ID(vehicle_attitude));
int att_sp_sub = orb_subscribe(ORB_ID(vehicle_attitude_setpoint));
int global_pos_sub = orb_subscribe(ORB_ID(vehicle_global_position));
int manual_sp_sub = orb_subscribe(ORB_ID(manual_control_setpoint));
int vstatus_sub = orb_subscribe(ORB_ID(vehicle_status));
int global_sp_sub = orb_subscribe(ORB_ID(vehicle_global_position_setpoint));
int param_sub = orb_subscribe(ORB_ID(parameter_update));
float speed_body[3] = {0.0f, 0.0f, 0.0f};
bool throttle_half_once = false;
struct pollfd fds[2] = {{ .fd = param_sub, .events = POLLIN },
{ .fd = att_sub, .events = POLLIN }};
```

//Μέσα στο while γίνονται οι εξής έλεγχοι: Καταρχάς μέσω του poll (fds, 2, 500), το πρόγραμμα ψάχνει για αιτία εξόδου από το βρόγχο εκτέλεσης, κάθε 500ms. Αν η συνάρτηση poll επιστρέψει αρνητική τιμή στο ret, τότε εμφανίζεται μήνυμα λάθους, αν η τιμή είναι μηδέν αυτό σημαίνει ότι τίποτα δεν άλλαξε τα τελευταία 500ms, ενώ αν η τιμή είναι θετική μπαίνει σε ένα βρόγχο επιλογών else. Με το **if** (fds[0].revents & POLLIN), ελέγχει για τυχόν αλλαγές στο topic parameter_update και εφόσον υπάρχουν, ενημερώνεται για αυτές. Στη συνέχεια με χρήση της orb_check ελέγχει αν υπάρχουν νέες μετρήσεις, όσον αφορά τη θέση του σκάφους ή νέα επιθυμητά setpoint. Αν υπάρχει κάποιο update ενημερώνει τις μεταβλητές θέσης και setpoint με τις νέες τιμές (orb_copy(ORB_ID(vehicle_global_position_setpoint), global_sp_sub, &global_sp);) και επιπλέον ενημερώνει και τον πίνακα speed_body με τις τιμές της ταχύτητας του κάθε άξονα του σκάφους.

```
while (!thread_should_exit) {
        int ret = poll(fds, 2, 500);
        if (ret < 0) {
                warnx("poll error");
        } else if (ret == 0) {
        } else {
                if (fds[0].revents & POLLIN) {
                struct parameter_update_s update;
                orb_copy(ORB_ID(parameter_update), param_sub, &update);
```

```
                parameters_update(&ph, &p);
                }
                if (fds[1].revents & POLLIN) {
                bool pos_updated;
                 orb_check(global_pos_sub, &pos_updated);
                bool global_sp_updated;
                 orb_check(global_sp_sub, &global_sp_updated);
                bool manual_sp_updated;
                orb_check(manual_sp_sub, &manual_sp_updated);
                orb_copy(ORB_ID(vehicle_attitude), att_sub, &att);
                if (global_sp_updated)
                orb_copy(ORB_ID(vehicle_global_position_setpoint),global_sp_sub,
&global_sp);
                if (pos_updated) {
                orb_copy(ORB_ID(vehicle_global_position), global_pos_sub,
&global_pos);
                 if (att.R_valid) {
speed_body[0] = att.R[0][0] * global_pos.vx + att.R[0][1] * global_pos.vy +
att.R[0][2] * global_pos.vz;
speed_body[1] = att.R[1][0] * global_pos.vx + att.R[1][1] * global_pos.vy +
att.R[1][2] * global_pos.vz;

speed_body[2] = att.R[2][0] * global_pos.vx + att.R[2][1] * global_pos.vy +
att.R[2][2] * global_pos.vz;


} else {
        speed_body[0] = 0;
        speed_body[1] = 0;
        speed_body[2] = 0;


        warnx("Did not get a valid R\n");
        }
}
```

//Στο τμήμα αυτό, ελέγχει αν έχει δώσει ο χρήστης νέο setpoint χειροκίνητα.//

```
if (manual_sp_updated)
orb_copy(ORB_ID(manual_control_setpoint), manual_sp_sub, &manual_sp);
```

//Ελέγχει αν το throttle ξεπέρασε το 50% κάποια στιγμή.Αν ναι κάνει true την throttle_half_once και μπαίνει αργότερα σε failsafe mode//

```
if (isfinite(manual_sp.throttle) &&  (manual_sp.throttle >= 0.6f) &&
(manual_sp.throttle <= 1.0f)) {
throttle_half_once = true;
}
```

//Ελέγχει την κατάσταση του σκάφους και το mode πτήσης.//

```
orb_copy(ORB_ID(vehicle_status), vstatus_sub, &vstatus);

#if 0
```

//Εδώ γίνεται έλεγχος, για το αν το σκάφος βρίσκεται σε mode πτήσης αυτόματου πιλότου ή αν η κατάσταση είναι τέτοια, που δεν απαιτείται διόρθωση και σαν συνέπεια μηδενίζει όλες τις εξόδους του ελεγκτή.//

```
if (vstatus.navigation_state == NAVIGATION_STATE_AUTO_ ||
 vstatus.navigation_state == NAVIGATION_STATE_STABILIZED) {
                        control_heading(&global_pos, &global_sp, &att, &att_sp);
                  actuators.control[1] = 0.0f;
                 actuators.control[2] = 0.0f;
                control_attitude(&att_sp, &att, speed_body, &rates_sp, &actuators);
                actuators.control[3] = att_sp.thrust;
                actuators.control[4] = 0.0f;
}
```

//Στη συνέχεια, ελέγχεται εάν το mode της πτήσης είναι χειροκίνητο. Αρχικά γίνεται ένας έλεγχος έκτακτων συνθηκών, όπως η διακοπή επικοινωνίας με τον σταθμό ελέγχου. Αν το RC σήμα χαθεί, το throttle μειώνεται στο 60% και το σκάφος μπαίνει σε διαδικασία loiter, δηλαδή πραγματοποιεί κυκλική κίνηση (att_sp.roll_body = 0.3f;) για ένα διάστημα μέχρι να αποκατασταθεί η επικοινωνία, διατηρώντας το ύψος του (att_sp.pitch_body = 0.0f;). Αν η επικοινωνία δεν αποκατασταθεί θα μπει σε διαδικασία ομαλής αυτόματης

προσγείωσης. Σε διαφορετική περίπτωση (όπου η επικοινωνία με τον σταθμό εδάφους δεν αντιμετωπίζει πρόβλημα) ο έλεγχος του σκάφους περνά στον χειριστή του σταθμού ελέγχου: att_sp.roll_body = manual_sp.roll;att_sp.pitch_body = manual_sp.pitch;att_sp.yaw_body = 0;
att_sp.thrust = manual_sp.throttle;).//

```
else if (vstatus.navigation_state == NAVIGATION_STATE_MANUAL) {
                                    }
else if (vstatus.state_machine == SYSTEM_STATE_MANUAL) {
        if (vstatus.manual_control_mode ==
VEHICLE_MANUAL_CONTROL_MODE_SAS) {
        if (vstatus.rc_signal_lost && throttle_half_once) {
        att_sp.roll_body = 0.3f;
        att_sp.pitch_body = 0.0f;
        if (isfinite(manual_sp.throttle) && (manual_sp.throttle >= 0.0f) &&
         (manual_sp.throttle <= 1.0f)) {
        att_sp.thrust = 0.6f * manual_sp.throttle;
        }
        else {att_sp.thrust = 0.0f;
        }
        att_sp.yaw_body = 0;
        }
        else {
        att_sp.roll_body = manual_sp.roll;
        att_sp.pitch_body = manual_sp.pitch;
        att_sp.yaw_body = 0;
        att_sp.thrust = manual_sp.throttle;
        }
         att_sp.timestamp = hrt_absolute_time();
```

//Εδώ, καλείται η συνάρτηση control_attitude, που είδαμε προηγουμένως και αποθηκεύεται η τιμή του throttle στην μεταβλητή ελέγχου.//

```
         control_attitude(&att_sp, &att, speed_body, &rates_sp, &actuators);
        actuators.control[3] = att_sp.thrust;
```

//Ελέγχει αν τα flaps ελέγχονται χειροκίνητα και σε διαφορετική περίπτωση τα θέτει σε μηδενική τιμή.//

```
***if*** *(isfinite(manual_sp.flaps)) {*
*actuators.control[4] = manual_sp.flaps;*
*}*
 ***else*** *{*
*actuators.control[4] = 0.0f;*
*}*
*}*
```

//Στο σημείο αυτό, ελέγχεται η πιθανότητα ο χειριστής να καθορίσει συγκεκριμένες τιμές roll, pitch, yaw, throttle και flaps.//

```
        else if (vstatus.manual_control_mode ==
VEHICLE_MANUAL_CONTROL_MODE_DIRECT) {
                actuators.control[0] = manual_sp.roll;
                actuators.control[1] = manual_sp.pitch;
        actuators.control[2] = manual_sp.yaw;
        actuators.control[3] = manual_sp.throttle;
        if (isfinite(manual_sp.flaps)) {
        actuators.control[4] = manual_sp.flaps;
        }
        else {
        actuators.control[4] = 0.0f;
        }
        }
}
#endif

orb_publish(ORB_ID(vehicle_rates_setpoint), rates_pub, &rates_sp);
```

//Εδώ κάνει publish τις εξόδους του επενεργητή.//

```
if (isfinite(actuators.control[0]) && isfinite(actuators.control[1]) &&
  isfinite(actuators.control[2]) &&   isfinite(actuators.control[3])) {
orb_publish(ORB_ID_VEHICLE_ATTITUDE_CONTROLS, actuator_pub, &actuators);
}
}
}
}

printf("[ex_fixedwing_control] exiting, stopping all motors.\n");
thread_running = false;
```

//Εξοδος από το βρογχο while.//

//Μηδενισμός όλων των εξόδων του επενεργητή και νέο publish. Τέλος main_thread.//

```
for (unsigned i = 0; i < NUM_ACTUATOR_CONTROLS; i++)
actuators.control[i] = 0.0f;
orb_publish(ORB_ID_VEHICLE_ATTITUDE_CONTROLS, actuator_pub, &actuators);
fflush(stdout);
return 0;
}
```

//Ορισμός της συνάρτησης usage, που δηλώθηκε στην αρχή. Η usage εκτελεί ένα print screen.//

```
static void
usage(const char *reason)
{
        if (reason)
                fprintf(stderr, "%s\n", reason);

        fprintf(stderr, "usage: ex_fixedwing_control {start|stop|status}\n\n");
        exit(1);
}
```

//Ορισμός της ex_fixedwing_control_main που δηλώνεται στην αρχή και που λειτουργεί σαν daemon. Δηλαδή εκκινεί και σταματά το πρόγραμμα, ελέγχει την κατάστασή του κλπ.//

```
int ex_fixedwing_control_main(int argc, char *argv[])
{
if (argc < 1)
usage("missing command");
if (!strcmp(argv[1], "start")) {
if (thread_running) {
printf("ex_fixedwing_control already running\n")

exit(0);

}

thread_should_exit = false;

deamon_task = task_spawn_cmd("ex_fixedwing_control",
SCHED_DEFAULT,
 SCHED_PRIORITY_MAX - 20,2048,fixedwing_control_thread_main,(argv) ? (const
char **)&argv[2] : (const char **)NULL);
thread_running = true;
exit(0);
}

if (!strcmp(argv[1], "stop")) {
thread_should_exit = true;
exit(0);
}

if (!strcmp(argv[1], "status")) {
if (thread_running) {
printf("\tex_fixedwing_control is running\n");
}
else {
printf("\tex_fixedwing_control not started\n");
}
exit(0);
```

```
}
usage("unrecognized command");

exit(1);

}
```

NAVIGATOR

// Ο κώδικας του αρχείου navigator_main.cpp ελέγχει τη συμπεριφορά του σκάφους κατά τη διάρκεια μιας αποστολής, όπως και τη συμπεριφορά του όταν εισέλθει σε failsafe mode.//

```
#include <nuttx/config.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <math.h>
#include <poll.h>
#include <time.h>
#include <drivers/drv_hrt.h>
#include <arch/board/board.h>
#include <uORB/uORB.h>
#include <uORB/topics/airspeed.h>
#include <uORB/topics/vehicle_global_position.h>
#include <uORB/topics/vehicle_global_position_set_triplet.h>
#include <uORB/topics/vehicle_attitude_setpoint.h>
#include <uORB/topics/manual_control_setpoint.h>
#include <uORB/topics/actuator_controls.h>
#include <uORB/topics/vehicle_rates_setpoint.h>
#include <uORB/topics/vehicle_attitude.h>
#include <uORB/topics/vehicle_status.h>
#include <uORB/topics/parameter_update.h>
#include <uORB/topics/mission.h>
#include <systemlib/param/param.h>
#include <systemlib/err.h>
#include <geo/geo.h>
#include <systemlib/perf_counter.h>
#include <systemlib/systemlib.h>
#include <mathlib/mathlib.h>
```

//Η χρήση της εντολής include γίνεται με σκοπό να συμπεριλάβει το αρχείο, στις βιβλιοθήκες χρόνου εκτέλεσης, μια σειρά από header files, που περιέχουν έτοιμες συναρτήσεις προς χρήση και οι οποίες ορίζονται εντός των header files αυτών.//

//Με την εντολή extern "C" ο μεταγλωττιστής του προγράμματος ειδοποιείται, ότι η εν λόγω συνάρτηση είναι ορισμένη σε κάποια βιβλιοθήκη της C, αλλά θα μεταγλωττιστεί σε ένα module τύπου C++ μαζί με τις υπόλοιπες συναρτήσεις.//

//Δήλωση προτύπου της συνάρτησης navigator_main. Σε άλλο σημείο του προγράμματος θα πρέπει να γίνει ο πλήρης ορισμός της.//

```
extern "C" __EXPORT int navigator_main(int argc, char *argv[]);
```

//Δήλωση της κλάσης navigator. Αρχικά αναφέρονται τα μέλη της, ιδιωτικά και δημόσια και στη συνέχεια γίνεται πλήρης ορισμός των μεθόδων της.//

```
class Navigator
{
public:
```

//Οι **Navigator**() και **~Navigator**() συναρτήσεις, είναι οι constructor και destructor συναρτήσεις της κλάσης, οι οποίες θα ορισθούν πλήρως μετά το τέλος της δήλωσης της κλάσης. Η πρώτη έχει σαν σκοπό να δώσει αρχικές τιμές στα private μέλη της κλάσης, ενώ η άλλη να αποδεσμεύσει την μνήμη που απασχολούν τα εν λόγω μέλη μετά την ολοκλήρωση της εκτέλεσης του προγράμματος.//

```
Navigator();
~Navigator();
```

//Η start() επίσης θα ορισθεί πλήρως στη συνέχεια. Σκοπός της συνάρτησης είναι να εκκινήσει το sensor task, δηλαδή να θέσει τους αισθητήρες σε διαδικασία λήψης μετρήσεων.//

int ***start();***

//Ορισμός των ιδιωτικών μελών της κλάσης.//

private*:*

bool *_task_should_exit;* *//Αν γίνει true τερματίζεται το πρόγραμμα.//*
int *_navigator_task;*

//Οι παρακάτω μεταβλητές θα χρησιμοποιηθούν για να κάνει subscribe το πρόγραμμα σε διάφορα topics, τα οποία αφορούν καίριες παραμέτρους της διαδικασίας πλοήγησης, όπως τα κανάλια της ασύρματης επικοινωνίας σκάφους και ground control station, η ταχύτητα του, ειδοποιήσεις αλλαγών σε setpoints κλπ.//

int *_global_pos_sub;*
int *_att_sub;*
int *_attitude_sub;*
int *_airspeed_sub;*
int *_vstatus_sub;*
int *_params_sub;*
int *_manual_control_sub;*
int *_mission_sub;*

//Μέσω της _triplet_pub θα γίνεται το publishing που θα αφορά στο setpoint θέσης.//

orb_advert_t *_triplet_pub;*

//Οι παρακάτω δομές θα χρησιμοποιηθούν από το πρόγραμμα σαν πολυμεταβλητές όπου θα αποθηκεύονται όλα τα δεδομένα και οι παράμετροι από τα όργανα μέτρησης, όλα τα setpoints που αφορούν την πτήση, τα mode πτήσης κλπ.//

```
struct vehicle_attitude_s                    _att;
struct vehicle_attitude_setpoint_s           _att_sp;
struct manual_control_setpoint_s             _manual;
struct airspeed_s                            _airspeed;
struct vehicle_status_s                              _vstatus;
struct vehicle_global_position_s             _global_pos;
struct vehicle_global_position_set_triplet_s _global_triplet;

perf_counter_t_loop_perf;
```

//Μεταβλητές που θα χρησιμεύσουν στον έλεγχο κάθε αποστολής ξεχωριστά. Αριθμός αντικειμενικών στόχων αποστολής, εγκυρότητα αυτής, χώρος στη μνήμη για την αποθήκευση των μεταβλητών ελέγχου.//

```
unsigned    _mission_items_maxcount;
struct      mission_item_s                   * _mission_items;
bool        _mission_valid;
```

//Μεταβλητές καταστάσεων manual ελέγχου.//

```
float       _seatbelt_hold_heading;
float       _loiter_hold_lat;
float       _loiter_hold_lon;
float       _loiter_hold_alt;
bool        _loiter_hold;
```

//Δημιουργία της struct _parameters.//

```
struct {
float throttle_cruise;
}_parameters;
```

//Δημιουργία της struct _parameter_handles.//

struct {
param_t throttle_cruise;
}_parameter_handles;

//Η **parameters_update**() ενημερώνει την μνήμη για τις μεταβολές των τιμών των μεταβλητών της struct _parameters.//

int parameters_update();

//Η **control_update**() ενημερώνει τις μεταβλητές ελέγχου. Γωνίες Euler, στροφές κινητήρα κλπ.//

void control_update();

//Η **vehicle_status_poll**() πραγματοποιεί έλεγχο για τυχόν αλλαγή στο mode ελέγχου του σκάφους.//

void vehicle_status_poll();

//Η **vehicle_attitude_poll**() ελέγχει τις αλλαγές στη θέση του σκάφους.//

void vehicle_attitude_poll();

//Η **mission_poll**() ελέγχει για τυχόν νέο setpoint.//

void ***mission_poll****();*

//Δήλωση πρωτοτύπων συναρτήσεων ελέγχου του throttle και του pitch.//

float ***control_throttle(float*** *energy_error);*
float ***control_pitch(float*** *altitude_error);*

//Δήλωση πρωτοτύπων συναρτήσεων υπολογισμού σφάλματος ταχύτητας και ύψους.//

void calculate_airspeed_errors*();*
void calculate_gndspeed_undershoot*();*
void calculate_altitude_error*();*

static void ***task_main_trampoline(int*** *argc,* ***char*** **argv[]);*

//Η **task_main**() **__attribute__**((noreturn)) είναι η συνάρτηση που πραγματοποιεί τη συλλογή δεδομένων από τα αισθητήρια.//

void ***task_main() __attribute__****((noreturn));*
};

//Εδώ δημιουργείται ένα namespace με το όνομα navigator, το οποίο ελέγχει σε επίπεδο προεπεξεργαστή, δηλαδή πριν τη μεταγλώττιση του κώδικα, αν έχει καθοριστεί μεταβλητή με το όνομα error. Στη περίπτωση που έχει οριστεί την καταργεί και συνέχεια ορίζει τη μεταβλητή error με τιμή -1.Έτσι εξασφαλίζεται ότι το πρόγραμμα δεν χρησιμοποιεί αλλού το error και αποφεύγονται πιθανές συγκρούσεις και λάθη κατά τη μεταγλώττιση. Ορίζεται επίσης μία μεταβλητή τύπου navigator.//

namespace *navigator*
{
#ifdef *ERROR*
 # undef *ERROR*
#endif
static const int *ERROR = -1;*
*Navigator *g_navigator;*

}

//Στη συνέχεια, γίνεται ο πλήρης ορισμός της constructor συνάρτησης της κλάσης navigator η οποία σαν σκοπό έχει να αρχικοποιηθούν οι τιμές των private μεταβλητών της.//

Navigator::Navigator *() :*

*_task_should_exit(**false**),*
_navigator_task(-1),

//Μεταβλητές που θα χρησιμοποιηθούν για subscriptions.//

_global_pos_sub(-1),
_att_sub(-1),
_airspeed_sub(-1),
_vstatus_sub(-1),
_params_sub(-1),
_manual_control_sub(-1),
//Μεταβλητές για publications.//
_triplet_pub(-1),

_loop_perf(perf_alloc(PC_ELAPSED, "navigator")),

//Μεταβλητές ελέγχου κατάστασης του mode πτήσης.//

_mission_items_maxcount(20),
*_mission_valid(**false**),*
*_loiter_hold(**false**)*

//Βλέπουμε το κύριο σώμα της constructor, όπου ελέγχεται το μέγεθος της μνήμης που χρειάζεται για να αποθηκευτούν όλα τα δεδομένα που αφορούν στους αντικειμενικούς σκοπούς της αποστολής (setpoints και τρέχουσες τιμές ταχυτήτων, γωνιών, ύψους κλπ).

Έπειτα δεσμεύεται αντίστοιχος χώρος μνήμης. Σε περίπτωση που δεν υπάρχει αρκετή διαθέσιμη μνήμη για τις ανάγκες της αποστολής, απορρίπτονται τα ζητούμενα waypoints και εμφανίζεται αντίστοιχο μήνυμα στην οθόνη. Επιπλέον ενεργοποιείται η συνάρτηση parameters_update() που μεταβιβάζει τιμές στις μεταβλητές που αποθηκεύουν τις τρέχουσες τιμές γωνιών, ταχυτήτων, ύψους και όποιας άλλης αφορά στην στιγμιαία κατάσταση του σκάφους. //

```
{
_mission_items = (mission_item_s*)malloc(sizeof(mission_item_s) * _mission_items_maxcount);
if (!_mission_items) {
_mission_items_maxcount = 0;
warnx("no free RAM to allocate mission, rejecting any waypoints");
}

_parameter_handles.throttle_cruise = param_find("NAV_DUMMY");

parameters_update();
}
```

//Εδώ ορίζεται η destructor συνάρτηση της κλάσης navigator. Γίνεται έλεγχος για το αν έχει αλλάξει η τιμή της _navigator_task από -1. Με την επιβεβαίωση της αλλαγής και την αναμονή 1 sec το πρόγραμμα σταματά και παράλληλα ελευθερώνεται η μνήμη που είχε δεσμευτεί από την κλάση.//

```
Navigator::~Navigator()
{
if (_navigator_task != -1) {

_task_should_exit = true;

unsigned i = 0;
do {

usleep(20000);
```

```
if (++i > 50) {

task_delete(_navigator_task);
break;
}
} while (_navigator_task != -1);
}

navigator::g_navigator = nullptr;
}
```

//Ενημέρωση παραμέτρων κατάστασης σκάφους.//

```
int
Navigator::parameters_update()
{
return OK;
}
```

//Η **vehicle_status_poll**() ελέγχει για ενδεχόμενη αλλαγή στο mode της πτήσης του σκάφους.//

```
void
Navigator::vehicle_status_poll()
{
bool vstatus_updated;

orb_check(_vstatus_sub, &vstatus_updated);

if (vstatus_updated) {

orb_copy(ORB_ID(vehicle_status), _vstatus_sub, &_vstatus);
}
}
```

//Ο ορισμός της **vehicle_attitude_poll**(), η οποία ελέγχει για αλλαγές στην κατάσταση του σκάφους (όλες τις μεταβλητές που αφορούν τον όρο "κατάσταση") μέσω της δομής _att και ενημερώνει την vehicle_attitude.//

```
void
Navigator::vehicle_attitude_poll()
{
bool att_updated;
orb_check(_att_sub, &att_updated);

if (att_updated) {
orb_copy(ORB_ID(vehicle_attitude), _att_sub, &_att);
}
}
```

//Η **mission_poll**() ελέγχει αν υπάρχει νέο setpoint κατεύθυνσης (νέο waypoint). Εφόσον υπάρχει ορίζει μια δομή mission τύπου mission_s, την οποία και ενημερώνει με το νέο setpoint και στη συνέχεια ελέγχει μέσω του mission.count, αν η απαίτηση σε μνήμη της αποστολής είναι εντός των ορίων που μπορούν να εξυπηρετηθούν. Αν είναι, τότε δίνει στην Boolean μεταβλητή _mission_valid την τιμή true, το οποίο συνεπάγεται την έγκριση έναρξης εκτέλεσης της αποστολής, ενώ σε αντίθετη περίπτωση εμφανίζει μήνυμα σφάλματος, που έχει να κάνει με την διαθέσιμη και την απαιτούμενη για την εκτέλεση μνήμη//

```
void
Navigator::mission_poll()
{
bool mission_updated;
orb_check(_mission_sub, &mission_updated);

if (mission_updated) {

struct mission_s mission;
```

```
orb_copy(ORB_ID(mission), _mission_sub, &mission);


if (mission.count <= _mission_items_maxcount) {
irqstate_t flags = irqsave();



memcpy(_mission_items, mission.items, mission.count * sizeof(struct mission_item_s));
_mission_valid = true;

irqrestore(flags);
}
 else {
warnx("mission larger than storage space");
        }
}
}
```

//Ενεργοποίηση του main task της navigator.//

```
void
Navigator::task_main_trampoline(int argc, char *argv[])
{
navigator::g_navigator->task_main();
}
```

//H main task.//

```
void
Navigator::task_main()
{
```

//Ενημέρωση για την έναρξη της διαδικασίας ελέγχου της πλοήγησης.//

```
warnx("Initializing..");
fflush(stdout);
```

//Εδώ γίνονται subscriptions σε topics που αφορούν στην θέση του σκάφους εκφρασμένη σε παγκόσμιες συντεταγμένες, στις παραμέτρους της εν ισχύ αποστολής, σε παραμέτρους που αφορούν την κατάσταση του σκάφους ως προς τις στιγμιαίες γωνίες, ταχύτητες, ύψος, καθώς και στο mode ελέγχου της πτήσης.//

_global_pos_sub = orb_subscribe(ORB_ID(vehicle_global_position));
_mission_sub = orb_subscribe(ORB_ID(mission));
_att_sub = orb_subscribe(ORB_ID(vehicle_attitude));
_airspeed_sub = orb_subscribe(ORB_ID(airspeed));
_vstatus_sub = orb_subscribe(ORB_ID(vehicle_status));
_params_sub = orb_subscribe(ORB_ID(parameter_update));
_manual_control_sub = orb_subscribe(ORB_ID(manual_control_setpoint));

//Με την χρήση της συνάρτησης orb_set_interval το πρόγραμμα καθορίζει το ρυθμό με τον οποίο θα ενημερώνονται οι μεταβλητές global_pos_sub, vstatus_sub, δηλαδή ο ρυθμός με τον οποίο θα ανανεώνεται η τιμή της θέσης του σκάφους σε παγκόσμιες συντεταγμένες και το mode ελέγχου της πτήσης. Ο ρυθμός καθορίζεται στα 50 Hz και 5 Hz αντίστοιχα.//

orb_set_interval(_vstatus_sub, 200);
orb_set_interval(_global_pos_sub, 20);

//Η **parameters_update**() ενημερώνει την μνήμη για τις μεταβολές των τιμών των μεταβλητών της struct _parameters.//

parameters_update();

//Δημιουργία ενός πίνακα fds 2 θέσεων τύπου pollfd.//

***struct** pollfd fds[2];*

//Στο fds[0] το πρόγραμμα κρατάει τις παραμέτρους κατάστασης του σκάφους, ενώ στο fds[1] τη στιγμιαία θέση του σκάφους.//

```
fds[0].fd = _params_sub;
fds[0].events = POLLIN;
fds[1].fd = _global_pos_sub;
fds[1].events = POLLIN;
```

//Εδώ γίνονται όλοι οι έλεγχοι για τις πιθανές καταστάσεις του σκάφους και τα πιθανά mode ελέγχου όσο βρίσκεται το πρόγραμμα σε εκτέλεση. Ο έλεγχος για την ισχύ της συνθήκης λειτουργίας γίνεται με την τιμή της μεταβλητής _task_should_exit.

Όσο αυτή είναι false το πρόγραμμα εκτελείται.//

```
while (!_task_should_exit) {

int pret = poll(&fds[0], (sizeof(fds) / sizeof(fds[0])), 100);
if (pret == 0)
continue;
if (pret < 0) {
warn("poll error %d, %d", pret, errno);
continue;
}
```

//Ελεγχος κατάστασης του σκάφους.//

```
perf_begin(_loop_perf);
vehicle_status_poll();
```

//Εδώ το πρόγραμμα ελέγχει αν έχουν αλλάξει/ενημερωθεί οι παράμετροι της κατάστασης του σκάφους και αν ναι τις αντιγράφει στην parameter_update.//

```
if (fds[0].revents & POLLIN)
{
struct parameter_update_s update;
orb_copy(ORB_ID(parameter_update), _params_sub, &update);

parameters_update();
 }
```

//Εδώ το πρόγραμμα πραγματοποιεί έλεγχο για τυχόν αλλαγή στη θέση του σκάφους και αν ναι τότε ενεργοποιεί τον ελεγκτή. Παράλληλα μέσω της vehicle_attitude_poll() και της mission_poll() ελέγχει αν υπάρχει νέα αποστολή για το σκάφος (νέο waypoint).//

```
if (fds[1].revents & POLLIN) {

static uint64_t last_run = 0;
float deltaT = (hrt_absolute_time() - last_run) / 1000000.0f;
last_run = hrt_absolute_time();

if (deltaT > 1.0f)
deltaT = 0.01f;

orb_copy(ORB_ID(vehicle_global_position), _global_pos_sub, &_global_pos);
vehicle_attitude_poll();
mission_poll();
math::Vector2f ground_speed(_global_pos.vx, _global_pos.vy);
```

//Τρέχον waypoint.//

```
math::Vector2f  next_wp(_global_triplet.current.lat / 1e7f, _global_triplet.current.lon / 1e7f);
```

// Θέση σκάφους σε παγκόσμιες συντεταγμένες.//

math::Vector2f current_position(_global_pos.lat / 1e7f, _global_pos.lon / 1e7f);

//Εδώ γίνεται ο έλεγχος για το mode της πτήσης στο οποίο βρίσκεται το σκάφος αφού προηγουμένως έχει διερευνηθεί αυτό με την mission_poll. Η τιμή 1 παραπέμπει σε πτήση με mode αυτόματου πιλότου. Αν η αποστολή είναι έγκυρη (mission valid) αποθηκεύεται στην μεταβήτή prev_wp το επόμενο waypoint (στο μέλος setx το γεωγραφικό πλάτος και στο sety το γεωγραφικό μήκος) αλλιώς το σκάφος μπαίνει σε διαδικασία διατήρησης της τρέχουσας κατεύθυνσης και στην μεταβλητή prev_wp διατηρούνται οι συντεταγμένες του τελευταίου έγκυρου waypoint.//

```
if (1 ) {

if (_mission_valid) {

math::Vector2f prev_wp;

if (_global_triplet.previous_valid) {
prev_wp.setX(_global_triplet.previous.lat / 1e7f);
prev_wp.setY(_global_triplet.previous.lon / 1e7f);
}
else {
prev_wp.setX(_global_triplet.current.lat / 1e7f);
prev_wp.setY(_global_triplet.current.lon / 1e7f);

}
```

//Ο επόμενος έλεγχος, με δεδομένο ότι βρισκόμαστε σε mode αυτόματου πιλότου και με έγκυρο επόμενο waypoint, είναι αν αυτό απαιτεί για τη μετάβαση του σκάφους εκεί μία 'κανονική' απλή πλοήγηση ή αν μιλάμε για πτήση τύπου loiter (δηλαδή αν το νέο waypoint απαιτεί μία κυκλική σταθερή τροχιά γύρω από ένα σταθερό σημείο).//

```
if (_global_triplet.current.nav_cmd == NAV_CMD_WAYPOINT) {
}
```

```
else if (_global_triplet.current.nav_cmd == NAV_CMD_LOITER_TURN_COUNT ||
_global_triplet.current.nav_cmd == NAV_CMD_LOITER_TIME_LIMIT ||
_global_triplet.current.nav_cmd == NAV_CMD_LOITER_UNLIMITED) {
}
_loiter_hold = false;
}
```

//Εδώ ορίζονται οι παράμετροι της πτήσης τύπου loiter.//

```
 else {
if (!_loiter_hold) {
_loiter_hold_lat = _global_pos.lat / 1e7f;
_loiter_hold_lon = _global_pos.lon / 1e7f;
_loiter_hold_alt = _global_pos.alt;
_loiter_hold = true;
}
}
```

//Η τιμή ένα, αντιπροσωπεύει ένα mode αυτόματου πιλότου. Το μηδέν, το οποίο ελέγχεται εδώ σαν ενδεχόμενο, αντιπροσωπεύει το mode seatbelt και το όποιο άλλο ενδεχόμενο, αντιπροσωπεύει manual mode πτήσης.//

```
else if (0) {
continue;
}
else {
continue;
}
if (_global_triplet.current.nav_cmd == NAV_CMD_RETURN_TO_LAUNCH) {
}
 else if (_global_triplet.current.nav_cmd == NAV_CMD_LAND) {
}
else if (_global_triplet.current.nav_cmd == NAV_CMD_TAKEOFF) {

if (_global_pos.alt < _global_triplet.current.altitude) {
_att_sp.pitch_body = math::max(_att_sp.pitch_body, _global_triplet.current.param1);
}
```

```
}
```

//Εδώ γίνεται publish το attitude setpoint.//

```
if (_triplet_pub > 0) {
orb_publish(ORB_ID(vehicle_global_position_set_triplet), _triplet_pub,
&_global_triplet);}
```

//Advertise και Publish της _triplet_pub.//

```
 else {
_triplet_pub = orb_advertise(ORB_ID(vehicle_global_position_set_triplet),
&_global_triplet);
}
}
perf_end(_loop_perf);
}
warnx("exiting.\n");
_navigator_task = -1;
_exit(0);
}
```

```
int
Navigator::start()
{
ASSERT(_navigator_task == -1);

_navigator_task = task_spawn_cmd("navigator",  SCHED_DEFAULT,
 SCHED_PRIORITY_MAX - 5,2048,(main_t)&Navigator::task_main_trampoline,
 nullptr);

if (_navigator_task < 0) {
warn("task start failed");
return -errno;
}

return OK;
```

```
}
```

//Το πρόγραμμα πραγματοποιεί μία σειρά ελέγχων στο daemon function κομμάτι του προγράμματος. Ανάλογα με τις εντολές που δίνει ο χρήστης και την κατάσταση του προγράμματος εμφανίζει αντίστοιχα μηνύματα όπως "start", "already running", "start failed" κλπ.//

```
int navigator_main(int argc, char *argv[])
{
if (argc < 1)
errx(1, "usage: navigator {start|stop|status}");

if (!strcmp(argv[1], "start")) {

if (navigator::g_navigator != nullptr)
errx(1, "already running");

navigator::g_navigator = new Navigator;

if (navigator::g_navigator == nullptr)
errx(1, "alloc failed");

if (OK != navigator::g_navigator->start()) {
delete navigator::g_navigator;
navigator::g_navigator = nullptr;
err(1, "start failed");
}


exit(0);
}

if (!strcmp(argv[1], "stop")) {
if (navigator::g_navigator == nullptr)
errx(1, "not running");

delete navigator::g_navigator;
navigator::g_navigator = nullptr;
exit(0);
}
```

```
if (!strcmp(argv[1], "status")) {
if (navigator::g_navigator) {
errx(0, "running");

}
 else {
errx(1, "not running");
}
}
warnx("unrecognized command");
return 1;
}
```

ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗ PID

```
#include "pid.h"
#include <math.h>
```

//Εδώ περιλαμβάνονται στις βιβλιοθήκες χρόνου εκτέλεσης του προγράμματος τα αρχεία pid.h (το οποίο ορίστηκε από τον χρήστη και υπάρχει στις βιβλιοθήκες της C++) και math.h (βιβλιοθήκη της C++ όπου ορίζονται συναρτήσεις μαθηματικής φύσεως) και ορίζεται η μεταβλητή SIGMA με τιμή 0.000001//

```
#define SIGMA 0.000001f
```

//Εδώ το πρόγραμμα μέσω της pid_init αρχικοποιεί τις τιμές των μελών της πολυμεταβλητής pid.//

```
__EXPORT void pid_init(PID_t *pid, float kp, float ki, float kd, float intmax,float limit,
uint8_t mode, float dt_min)
{
pid->kp = kp;
pid->ki = ki;
pid->kd = kd;
pid->intmax = intmax;
pid->limit = limit;
pid->mode = mode;
pid->dt_min = dt_min;
pid->count = 0.0f;
pid->saturated = 0.0f;
pid->last_output = 0.0f;
pid->sp = 0.0f;
pid->error_previous = 0.0f;
pid->integral = 0.0f;
}
```

```
__EXPORT int pid_set_parameters(PID_t *pid, float kp, float ki, float kd, float intmax,
float limit)
{

int ret = 0;
```

//Εδώ ελέγχεται αν το Kp είναι αριθμός συγκεκριμένος (όχι άπειρο, όχι κάτι διαφορετικό από αριθμό) και αν είναι καταχωρείται στο μέλος Kp της πολυμεταβλητής pid (τύπου pid_t). Σε αντίθετη περίπτωση το ret γίνεται 1. Το ίδιο γίνεται και με τα Ki, Kd, intmax και limit. Έτσι εξασφαλίζεται ότι οι τιμές που δόθηκαν στις μεταβλητές αυτές είναι αποδεκτές.//

```
if (isfinite(kp)) {
pid->kp = kp;

}
 else {
ret = 1;
        }

if (isfinite(ki)) {
pid->ki = ki;
}
 else {
ret = 1;
        }

if (isfinite(kd)) {
pid->kd = kd;
}
 else {
ret = 1;
        }
if (isfinite(intmax)) {
pid->intmax = intmax;
}
```

```
  else {

ret = 1;
}

if (isfinite(limit)) {
pid->limit = limit;
}
else {
ret = 1;
          }
return ret;
}
```

//Εδώ το πρόγραμμα, με την συνάρτηση pid_calculate, υπολογίζει τις παραμέτρους του ελεγκτή PID. Δέχεται σαν παραμέτρους την μεταβλητή τύπου PID_T και τέσσερις ακόμα που αντιπροσωπεύουν ένα setpoint, μία πραγματική μέτρηση, μία μονάδα χρόνου και μία παράγωγο μιας πραγματικής μέτρησης μέτρησης.//

```
__EXPORT float pid_calculate(PID_t *pid, float sp, float val, float val_dot, float dt)
{
```

//Αρχικά ελέγχεται αν όλες οι παράμετροι που δόθηκαν στην συνάρτηση είναι έγκυροι. Αν ναι, η συνάρτηση προχωρά στον υπολογισμό των νέων παραμέτρων του PID ενώ σε διαφορετική περίπτωση η έξοδος του ελεγκτή παραμένει ίδια με την τελευταία έγκυρη.//

```
if (!isfinite(sp) || !isfinite(val) || !isfinite(val_dot) || !isfinite(dt)) {
return pid->last_output;
}

float i, d;
pid->sp = sp;
```

// Εδώ υπολογίζεται το στιγμιαίο σφάλμα ως διαφορά μεταξύ setpoint και πραγματικής μέτρησης.//

float *error = pid->sp - val;*

// Εδώ ελέγχεται το mode του PID ελεγκτή που χρησιμοποιούμε και αντίστοιχα γίνονται οι απαραίτητοι υπολογισμοί.//

//Το PID_MODE_DERIVATIV_CALC αφορά PID ελεγκτή και υπολογίζει το Kd ως την παράγωγο του σφάλματος την οποία και αποθηκεύει στην μεταβλητή d. Η παράγωγος υπολογίζεται ως η διαφορά μεταξύ δύο διαδοχικών μετρήσεων του σφάλματος προς το χρόνο μεταξύ των δύο αυτών μετρήσεων.//

```
if (pid->mode == PID_MODE_DERIVATIV_CALC) {
d = (error - pid->error_previous) / fmaxf(dt, pid->dt_min);
pid->error_previous = error;
}
```

//Το PID_MODE_DERIVATIV_CALC_NO_SP αφορά PID ελεγκτή και υπολογίζει το Kd ως την παράγωγο της μετρούμενης και υπό εξέταση μεταβλητής (π.χ. κάποια γωνία Euler).//

```
 else if (pid->mode == PID_MODE_DERIVATIV_CALC_NO_SP) {
d = (-val - pid->error_previous) / fmaxf(dt, pid->dt_min);
pid->error_previous = -val;
}
```

//Το PID_MODE_DERIVATIV_SET αφορά PID ελεγκτή και χρησιμοποιείται αν το Kd είναι ήδη γνωστό (π.χ. μέσω Kalman).//

```
 else if (pid->mode == PID_MODE_DERIVATIV_SET) {
d = -val_dot;
}
```

//Το else αναφέρεται στο PID_MODE_DERIVATIV_NONE και αφορά ελεγκτή PI. Το Kd τίθεται μηδέν.//

```
 else {
d = 0.0f;
}
```

//Αν το τελικό Kd που υπολογίζεται δεν είναι έγκυρο τότε τίθεται μηδέν.//

```
if (!isfinite(d)) {
d = 0.0f;
}
```

//Αν το Ki (δηλαδή ο ολοκληρωτικός όρος) είναι μεγαλύτερος του μηδενός τότε υπολογίζεται το ολοκλήρωμα του σφάλματος και αποθηκεύεται στο i. Στη συνέχεια ελέγχει αν υπάρχει κορεσμός του ελεγκτή (δηλαδή αν η έξοδος του ελεγκτή είναι μέσα στα όρια τιμών εξόδου του ελεγκτή) και ενημερώνει την μεταβλητή saturated.//

```
if (pid->ki > 0.0f) {
i = pid->integral + (error * dt);

if ((pid->limit > SIGMA && (fabsf((error * pid->kp) + (i * pid->ki) + (d * pid->kd)) >
pid->limit)) ||fabsf(i) > pid->intmax) {
i = pid->integral;                  // If saturated then do not update integral value
pid->saturated = 1;

}
```

//Σε περίπτωση μη κορεσμού της εξόδου του PID τότε ελέγχεται αν η τιμή του i είναι έγκυρη ή όχι και σε περίπτωση μη εγκυρότητας μηδενίζεται. Σε περίπτωση έγκυρου i η τιμή του μεταβιβάζεται στο pid.integral όπου αποθηκεύεται το ολοκλήρωμα του σφάλματος.//

```
else {
if (!isfinite(i)) {
i = 0.0f;
}
pid->integral = i;
pid->saturated = 0;
}
}
```

//Σε περίτωση που το Ki είναι μηδέν, μηδενίζεται και ο όρος i.//

```
 else {
i = 0.0f;
pid->saturated = 0;
}
```

//Εδώ τελικώς υπολογίζεται η έξοδος του ελεγκτή η οποία μεταβιβάζεται στην μεταβλητή output. Η έξοδος όπως ακριβώς προβλέπει η θεωρία αποτελείται από τρεις όρους. Ο αναλογικός όρος πολλαπλασιάζει το σφάλμα με ένα κέρδος αναλογίας Kp, ο ολοκληρωτικός όρος πολλαπλασιάζει το ολοκλήρωμα του σφάλματος με ένα κέρδος ολοκλήρωσης Ki και ο όρος διαφόρισης πολλαπλασιάζει την παράγωγο του σφάλματος με ένα κέρδος διαφόρισης Kd. Στη συνέχεια ελέγχεται αν η έξοδος είναι έγκυρη και εντός ορίων των τιμών εξόδου του PID. Τέλος με την reset_integral μηδενίζεται το ολοκλήρωμα του σφάλματος για να υπολογιστεί από την αρχή και πάλι.//

```
float output = (error * pid->kp) + (i * pid->ki) + (d * pid->kd);
```

```
if (isfinite(output)) {
if (pid->limit > SIGMA) {
if (output > pid->limit) {
output = pid->limit;
}
 else if (output < -pid->limit) {
output = -pid->limit;
}
}
pid->last_output = output;
}
return pid->last_output;
}

__EXPORT void pid_reset_integral(PID_t *pid)
{
pid->integral = 0;
}
```

ΠΑΡΑΔΕΙΓΜΑ ΔΗΜΙΟΥΡΓΙΑΣ ΕΦΑΡΜΟΓΗΣ ΣΤΟ ECLIPSE

Στο παρόν κεφάλαιο παρατίθεται ένα παράδειγμα δημιουργίας εφαρμογής μέσω του προγράμματος Eclipse. Στην αρχή δίνεται ο πηγαίος κώδικας, ενώ στη συνέχεια παρουσιάζεται η διαδικασία για την μετατροπή του σε εκτελέσιμο αρχείο.

Το παράδειγμα είναι γραμμένο σε γλώσσα C++ και παρουσιάζει την μετατροπή της μέτρησης του επιταχυνσιόμετρου από μια τιμή τάσης σε μονάδες βαρύτητας g (ενδεικτικά αναφέρουμε ότι οι τιμές που μετρά το επιταχυνσιόμετρο είναι καθαροί αριθμοί τριών ή τεσσάρων δεκαδικών ψηφίων). Επιπλέον, οι αριθμοί πριν μετατραπούν σε g, φιλτράρονται μέσω φίλτρου Kalman. Εδώ πρέπει να σημειώσουμε, ότι για την υλοποίηση της διαδικασίας φιλτραρίσματος απαιτούνται ορισμένες παραδοχές, που βοηθούν στην απλοποίηση των εξισώσεων του φίλτρου, χωρίς ωστόσο να μειωθεί σημαντικά η αξιοπιστία του. Οι παραδοχές αυτές είναι οι εξής: οι A, B και H αντί για πίνακες συμμεταβλητότητας αντικαθιστώνται σε αριθμητικές σταθερές και ίσες με 1 για την απλοποίηση των πράξεων, το R (θόρυβος μέτρησης) βρίσκεται σε μία λογική μη μηδενική τιμή (10% του P) και τέλος απαλείφεται ο όρος Q.

```
#include <iostream.h>
#include <cstdlib.h>
using namespace std;

//Εδώ ορίζεται πλήρως η συνάρτηση conversion, η οποία δέχεται σαν παραμέτρους την μέτρηση του οργάνου και ορισμένες σταθερές του, όπως την τάση αναφοράς Vref, την τάση που αντιστοιχεί σε 0g, την ευαισθησία του επιταχυνσιόμετρου, καθώς και τον αριθμό των bit του ADC και πραγματοποιεί την μετατροπή.//

float conversion(int metrisi,int bits,float voltref,float voltzero,float sensitivity){

float a,b,g;
```

```
a=metrisi*voltref/(2^bits);
b=a-voltzero;
g=b*1000/sensitivity;
return(g);
}
```

//Η main συνάρτηση του κώδικα.//

```
int main(void) {
int bit,ran,A[100],i,j,k,l,c,P,m,h;
float vref,vzero,sens,B[100],C[101],R,Kg,D[100];
```

//Εδώ ζητείται από τον χρήστη να δώσει από το πληκτρολόγιο τις σταθερές του οργάνου μέτρησης και τον αριθμό των bits του μετατροπέα τα οποία στη συνέχεια θα χρησιμοποιηθούν στην κλήση της συνάρτησης conversion. Για το επιταχυνσιόμετρο της εργασίας οι ενδεικτικές τιμές είναι Vref=3, Vzero=1.5, Sensitivity=300 και ο ADC είναι της τάξεως των 10 bit.//

```
cout<<"ADC metatropeas,arithmos bit:"<<endl;
cin>>bit;
cout<<"V reference accelarometer:"<<endl;
cin>>vref;
cout<<"Vzerog:"<<endl;
cin>>vzero;
cout<<"accelerometer sensitivity se mV/g:"<<endl;
cin>>sens;
for (i=0;i<=99;i++)
{A[i]=1;
}
j=0;
```

//Εδώ παίρνουμε 100 τυχαίες μετρήσεις με την συνάρτηση rand() σε ελεγχόμενο εύρος.//

```
while (A[99]==1){
ran=rand();
if ((ran>=512) && (ran<=5120)){
A[j]=ran;
j=j+1;
```

```
}
}
C[0]=0;
P=2;
R=0.2;
```

//Φιλτράρισμα των μετρήσεων.//

```
for (m=1;m<=100;m++){
Kg=(P/(P+R));
C[m]=C[m-1]+Kg*(A[m-1]-C[m-1]);
P=(1-Kg)*P;
}
for (h=1;h<=100;h++)
{D[h-1]=C[h];
for (k=0;k<=99;k++)
{
l=D[k];
B[k]=conversion(l,bit,vref,vzero,sens);
if (B[k]<0)
{B[k]=0;
}
}
for(c=0;c<=99;c++)
{
```

//Απεικόνιση των μετρήσεων, αρχικών και φιλτραρισμένων, καθώς και η αντιστοιχία τους σε g.//

```
cout<<"H metrisi "<<D[c]<<"tou accelerometer antistoixei se"<<B[c]<<"Gs"<<endl;
}
}
return 0;
}
```

Στο σημείο αυτό ολοκληρώνεται ο πηγαίος κώδικας και θα παρουσιαστεί η διαδικασία μετατροπής του σε εκτελέσιμο αρχείο.

Αρχικά εκκινούμε το Eclipse και στον φάκελο Firmware/srs/modules δημιουργούμε έναν νέο φάκελο με την ονομασία paradeigma_app. Στο εσωτερικό του δημιουργούμε ένα αρχείο module.mk και γράφουμε τις εξής εντολές:
MODULE_COMMAND = paradeigma_app
SRCS = paradeigma_app.c

Στον φάκελο paradeigma_app δημιουργούμε ένα καινούριο αρχείο με το όνομα paradeigma_app.c που στο εσωτερικό του αντιγράφουμε το παραπάνω παράδειγμα. Στη συνέχεια ανοίγουμε το αρχείο Firmware/makefiles/config_px4fmu_default.mk στο οποίο προσθέτουμε την γραμμή Modules += modules/paradeigma_app και κλείνουμε το αρχείο κάνοντας save. Ακολούθως στο παράθυρο Make target του Eclipse ανοίγουμε τον φάκελο Firmware και επιλέγουμε (διπλό κλικ) το εικονίδιο archives (βλ. σελ. 30). Στο κάτω παράθυρο μπορούμε να επιλέξουμε την καρτέλα console για να βλέπουμε την πορεία της διαδικασίας. Μετά το πέρας της διαδικασίας επιλέγουμε με την ακόλουθη σειρά τα εικονίδια clean, upload px4fmu-v1_default και περιμένουμε να εμφανιστεί στο console μήνυμα για την σύνδεση του PX4fmu στη USB. Ύστερα από την λήξη της διαδικασίας θα ηχήσει ο χαρακτηριστικός ήχος εκκίνησης του PX4fmu.

Για να δοκιμάσουμε την εφαρμογή μας ενεργοποιούμε το Teraterm (βλ. σελ. 69) και στην αναμονή του Nsh πληκτρολογούμε help και enter. Στην λίστα με τις εφαρμογές που μας εμφανίζει θα δούμε διαθέσιμη και την paradeigma_app. Τώρα μπορούμε να την εκκινήσουμε πληκτρολογώντας paradeigma_app και enter.

ΦΙΛΤΡΟ KALMAN

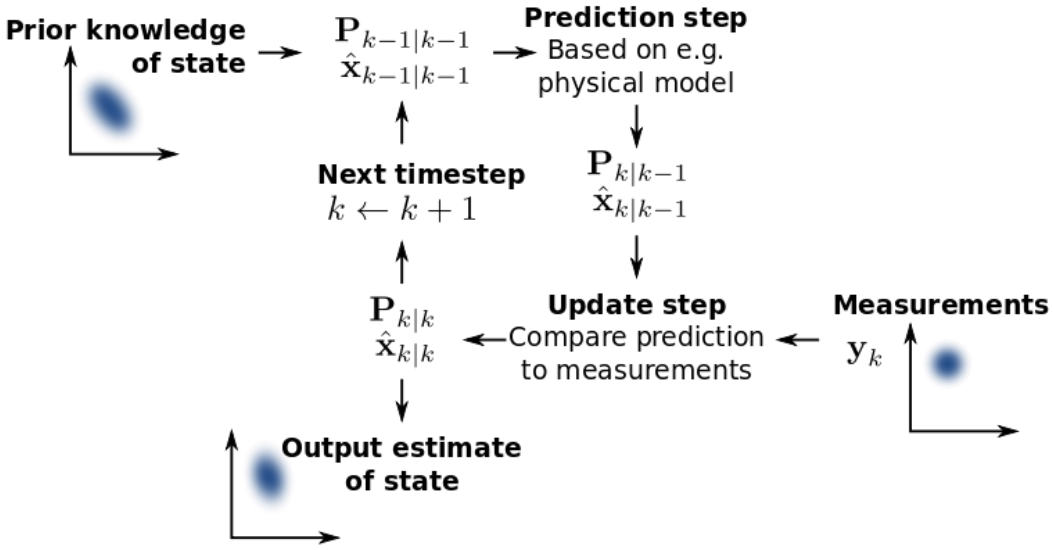
Στο παρόν κεφάλαιο παρουσιάζονται οι γενικές αρχές του φίλτρου Kalman, καθώς και του εκτεταμένου φίλτρου Kalman. Παρουσιάζεται, επίσης, ο κώδικας πραγματοποίησης και υπολογισμού του φίλτρου, όπως αυτός χρησιμοποιείται από τον μικροελεγκτή PX4.

Ο όρος φιλτράρισμα υποδηλώνει την λειτουργία απομάκρυνσης ανεπιθύμητων στοιχείων-παραγόντων από μια διαδικασία. Η εφαρμογή του φιλτραρίσματος κατά τη διάρκεια μιας μέτρησης της τιμής ενός μεγέθους μέσω ενός οργάνου έχει σαν σκοπό την όσο το δυνατόν μικρότερη απόκλιση της μετρούμενης τιμής από την πραγματική. Το φίλτρο Kalman, είναι ένα μαθηματικό στατιστικό εργαλείο, το οποίο χρησιμοποιεί μετρήσεις-παρατηρήσεις ενός μεγέθους σε διάρκεια χρόνου, που περιέχουν θόρυβο και άλλες ανακρίβειες, με σκοπό να παράγει εκτιμήσεις για την πραγματική τιμή του υπό εξέταση μεγέθους κάθε χρονική στιγμή. Συνοπτικά δηλαδή, το φίλτρο Kalman είναι ένας βέλτιστος εκτιμητής τιμής μιας μετρούμενης -με θόρυβο- μεταβλητής, και είναι απαραίτητο επειδή τα όργανα μέτρησης όλων των μεγεθών, υπό κάθε συνθήκη, μεταφέρουν στην μέτρησή τους παράλληλα με την πραγματική τιμή του μεγέθους και ένα σφάλμα ή αλλιώς θόρυβο. Με έναν "μαθηματικό ορισμό", το φίλτρο Kalman θα οριζόταν ως ένας εκτιμητής του γραμμικού προβλήματος ελαχίστων τετραγώνων, που καλείται να υπολογίσει την στιγμιαία κατάσταση ενός γραμμικού δυναμικού συστήματος, διαταρασσόμενο από λευκό θόρυβο. Ο εκτιμητής αυτός είναι στατιστικά βέλτιστος ως προς οποιαδήποτε συνάρτηση του τετραγώνου σφάλματος εκτίμησης.

$$F(e^2)=min$$

Το φίλτρο Kalman βρίσκει εφαρμογή ευρέως και σε διαφορετικά επιστημονικά πεδία και αποδεικνύεται σε κάθε περίπτωση εξαιρετικά χρήσιμο. Συχνή είναι η εφαρμογή του φίλτρου στην πλοήγηση και στον έλεγχο μη επανδρωμένων οχημάτων και ειδικά

UAV, καθώς και στη λειτουργία του αυτόματου πιλότου αεροσκαφών. Εκτεταμένη είναι η χρήση του φίλτρου και σε άλλους τομείς λόγω της ουσιαστικής συμβολής του, όπως για παράδειγμα στην οικονομετρία και στην ψηφιακή επεξεργασία σήματος. Ενεργό πεδίο δράσης βρίσκει και στους βιομηχανικούς χώρους, όπου μπορεί να χρησιμοποιηθεί στον έλεγχο χημικών διεργασιών ή στις εφαρμογές υπολογιστικής όρασης (3D-animation).



16-1. Ακολουθία βημάτων από την μέτρηση μέχρι την εκτίμηση.

Βασικες εξισώσεις φίλτρου Kalman

Η τιμή, το αποτέλεσμα δηλαδή, από την μέτρηση ενός αισθητήριου-οργάνου μέτρησης εμπεριέχει και μία αλλοίωση της πραγματικής τιμής του μεγέθους, δηλαδή ένα θόρυβο. Έστω Z_κ η τιμή της μέτρησης την χρονική στιγμή κ και V_k ο θόρυβος αυτής. Προκύπτει ότι Z_κ=H*X_κ+V_k, δηλαδή, ότι η τιμή της μέτρησης που επιστρέφει ένα όργανο, είναι ένας γραμμικός συνδυασμός της πραγματικής τιμής του μεγέθους (X_κ) και

του θορύβου. Η μορφή του συντελεστή Η μπορεί να είναι είτε ένας αριθμός είτε ένας πίνακας. Η τιμή της, υπό εξέταση, μεταβλητής Χ την χρονική στιγμή κ (ή αλλιώς η κατάσταση του, υπό εξέταση, συστήματος την χρονική στιγμή κ), μπορεί να αποδοθεί ως ένας γραμμικός συνδυασμός τριών όρων: της προηγούμενης κατάστασης του συστήματος, ενός σήματος ελέγχου του συστήματος, αν αυτό υπάρχει (μία είσοδο της διεργασίας η οποία οδηγεί το σύστημα) και του θορύβου που υπεισέρχεται στην διεργασία. Επομένως προκύπτουν οι εξισώσεις:

$$X_\kappa = A * X_{\kappa-1} + B * U_k + W_{k-1} \qquad \textit{(Εξ.1)}$$

$$Z_\kappa = H * X_\kappa + V_k \qquad \textit{(Εξ.2)}$$

Στις παραπάνω συναρτήσεις, οι συντελεστές Α, Β και Η, αποτελούν πίνακες, χωρίς ωστόσο αυτό να αποκλείει σε ορισμένες περιπτώσεις, να είναι απλώς αριθμοί. Αυτό που μένει να προσδιοριστεί είναι οι συναρτήσεις W_{k-1} και V_k που αντιπροσωπεύουν τον θόρυβο στην διεργασία και τον θόρυβο στην μέτρηση αντίστοιχα, καθώς και την μέση τιμή του θορύβου και την αντίστοιχη μέση απόκλιση από αυτήν. Αν και είναι εξαιρετικά σπάνιο να συναντήσουμε ένα σήμα, με απόλυτα Κανονική κατανομή (Gauss), παρόλα αυτά μπορούμε σε πολλές περιπτώσεις να το υποθέσουμε ως τέτοιο, χωρίς να κινδυνεύουμε να υποπέσουμε σε εσφαλμένους μαθηματικούς υπολογισμούς, καθώς πολλά σήματα πλησιάζουν αρκετά αυτή την συμπεριφορά. Φυσικά ισχύει ότι όσο ακριβέστερα υπολογίσουμε τις παραμέτρους που αφορούν στον θόρυβο, τόσο ορθότερες θα είναι οι εκτιμήσεις των μεταβλητών κατάστασης του συστήματος. Στην προκειμένη περίπτωση τα W_{k-1}, V_k θεωρούνται σήματα με Κανονική κατανομή.

Η **Κανονική κατανομή**, γνωστή και ως Γκαουσιανή κατανομή, αναφέρεται σε συνεχείς μεταβλητές, αποτελώντας συνεχή συνάρτηση πυκνότητας-πιθανότητας. Χρησιμοποιείται ως μία πρώτη προσέγγιση για να περιγραφούν τυχαίες μεταβλητές

πραγματικών τιμών, οι οποίες τείνουν να συγκεντρώνονται γύρω από μια μέση τιμή. Η Κανονική κατανομή αποτελεί την πιο σημαντική κατανομή της στατιστικής μεθοδολογίας για τους εξής βασικούς λόγους:

- Την Κανονική κατανομή ακολουθούν είτε με ακρίβεια, είτε με μεγάλη προσέγγιση τα περισσότερα συνεχή φαινόμενα.

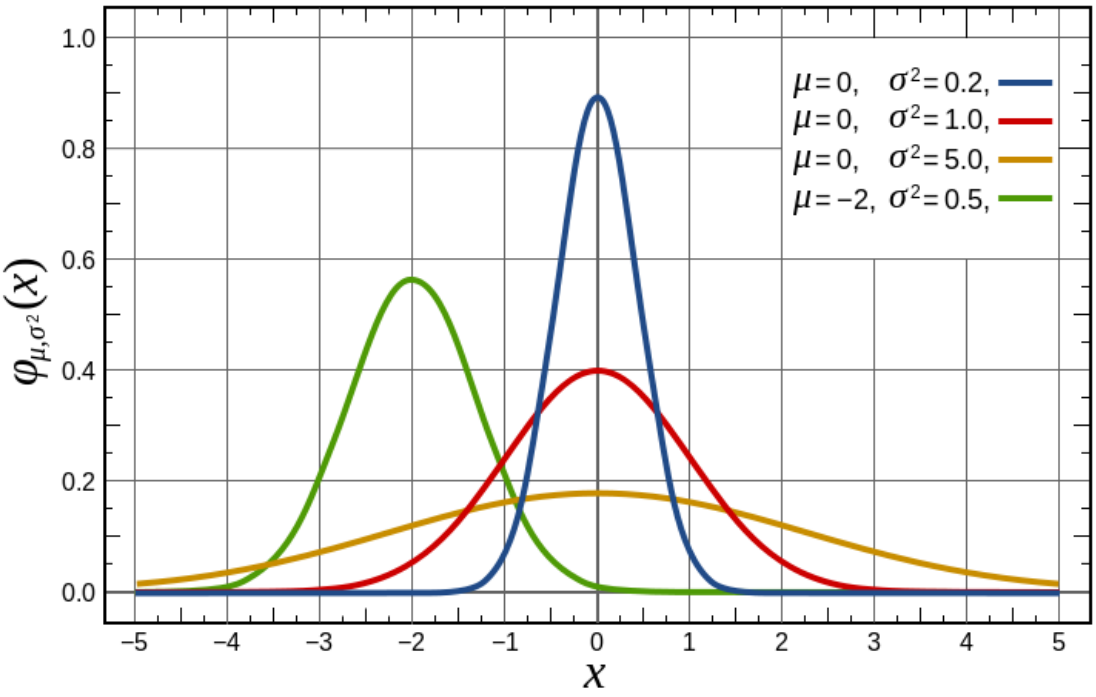
- Πολλές ασυνεχείς κατανομές πιθανοτήτων μπορούν να προσεγγιστούν μέσω της Κανονικής κατανομής (π.χ. πληθυσμιακά χαρακτηριστικά, όπως το ύψος, το βάρος, η βαθμολογία σε διαγώνισμα, κλπ.).

- Η Κανονική κατανομή αποτελεί σύμφωνα με το κεντρικό οριακό θεώρημα τη βάση της στατιστικής συμπερασματολογίας ή επαγωγικής στατιστικής (το άθροισμα ενός ικανοποιητικά μεγάλου αριθμού ανεξάρτητων και ισόνομων τυχαίων μεταβλητών, προσεγγίζεται από την κανονική κατανομή).

- Η Κανονική κατανομή αναφέρεται πολλές φορές και ως κατανομή σφαλμάτων γιατί τυχαία σφάλματα που εμφανίζονται σε διάφορες μετρήσεις έχουν Κανονική κατανομή.

Η γραφική παράσταση της σχετιζόμενης συνάρτησης πυκνότητας - πιθανότητας έχει σχήμα “καμπάνας”, και είναι γνωστή ως Γκαουσιανή συνάρτηση ή κωδωνοειδής καμπύλη:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



16-2. Κωδωνοειδής καμπύλη συνάρτησης Gauss.

Τυποποιημένη Κανονική κατανομή

Η Κανονική κατανομή, που έχει μέση τιμή 0 (μ=0) και τυπική απόκλιση 1 (σ=1, άρα και διασπορά 1), συμβολίζεται με N(0,1) και ονομάζεται **τυποποιημένη Κανονική κατανομή**. Μια τυχαία μεταβλητή που ακολουθεί την τυποποιημένη Κανονική κατανομή, έχει επικρατήσει να συμβολίζεται με Z και η συνάρτηση πυκνότητάς της με φ(z).

$$\phi(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$, όπου $-\infty < z < +\infty$

Η τυποποίηση των δεδομένων βασίζεται στην απόκλισή τους από το μέσο όρο σε όρους της σ, σύμφωνα με τον τύπο:

$$Z = \frac{X - \mu}{\sigma}$$

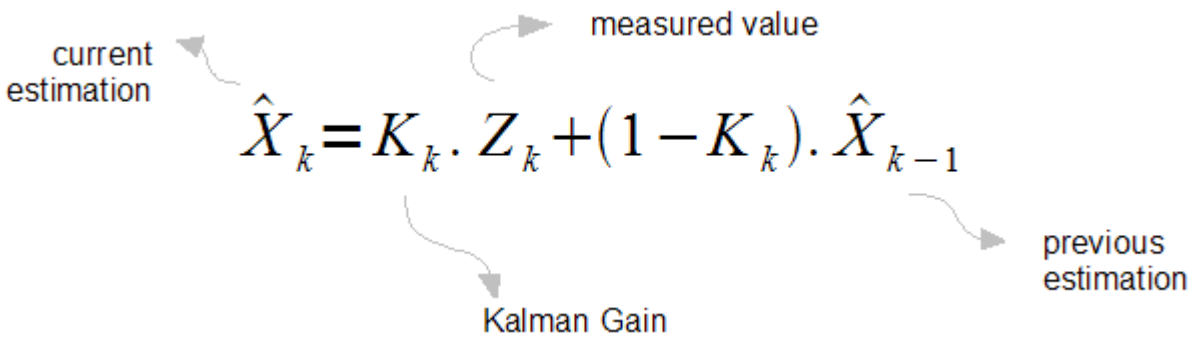
Εξισώσεις πρόβλεψης και διόρθωσης

Υπάρχουν δύο διακριτές ομάδες εξισώσεων που αφορούν στο φίλτρο, δηλαδή οι εξισώσεις πρόβλεψης (time update) και οι εξισώσεις διόρθωσης (measurement update).

16-1. Πίνακας εξισώσεων πρόβλεψης και διόρθωσης.

Time Update *(prediction)*	**Measurement Update** *(correction)*
$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$	$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
$P_k^- = AP_{k-1}A^T + Q$	$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$
	$P_k = (I - K_k H)P_k^-$

Στις εξισώσεις πρόβλεψης ο όρος P_k αντιπροσωπεύει τη διασπορά του σφάλματος , ενώ το Q είναι η διασπορά του θορύβου της διεργασίας. Στις εξισώσεις διόρθωσης ο όρος R αντιπροσωπεύει τη διασπορά του θορύβου μέτρησης, ενώ ο όρος K_κ αντιπροσωπεύει το κέρδος Kalman που υπολογίζεται για κάθε -διακριτή- χρονική στιγμή και αποτελεί την βάση του συνόλου της διαδικασίας. Σε μία απλουστευμένη αλλά συνήθη κατάληξη της εξίσωσης του φίλτρου έχουμε:



Αναλύοντας την παραπάνω εξίσωση, μπορούμε να καταλήξουμε στα παρακάτω:

Η εκτιμώμενη τιμή για την μεταβλητή $X_κ$ εξαρτάται από το άθροισμα της μετρηθείσας τιμής $Z_κ$ και της προηγούμενης εκτιμηθείσας $X_{κ-1}$, με συντελεστές στους όρους, το κέρδος Kalman (K) και το ποσοστιαίο του υπολοίπου (1-K) αντίστοιχα. Επομένως γίνεται σαφές, πόσο σημαντικός είναι ο βέλτιστος υπολογισμός του κέρδους K. Ουσιαστικά, πρόκειται για το ποσοστό αναλογίας μεταξύ μέτρησης και προηγούμενης εκτίμησης, το οποίο θα οδηγήσει την εξίσωση για να πάρουμε την νέα πρόβλεψη του X.

Ο κύκλος του Φίλτρου

Το φίλτρο στη λειτουργία του θυμίζει ένα κλειστό σύστημα ελέγχου, δηλαδή ένα σύστημα με βρόγχο ανατροφοδότησης. Αρχικά το φίλτρο πραγματοποιεί μία πρόβλεψη για μία δεδομένη χρονική στιγμή. Πρόκειται για την εκτίμηση $X_κ$ της μεταβλητής X τη χρονική στιγμή κ, ενώ στη συνέχεια έχουμε τη διόρθωση μέσω μιας ενθόρυβης μέτρησης $Z_κ$, την οποία λαμβάνει μέσω ανάδρασης για την βέλτιστη εκτίμηση της $X_κ$. Αναλύοντας την κυκλική λειτουργία του φίλτρου Kalman, προκύπτουν τα βήματα:
(Υποθέτουμε ότι βρισκόμαστε στην χρονική στιγμή κ-1, όπου έχει ήδη πραγματοποιηθεί η τελευταία μέτρηση.)

i. Πρόβλεψη της επόμενης κατάστασης του συστήματος για την στιγμή κ.
ii. Υπολογισμός του βέλτιστου κέρδους Kalman.
iii. Μέτρηση την χρονική στιγμή κ και ενημέρωση της εκτίμησης με βάση την νέα μέτρηση, προς διόρθωση της πρόβλεψης.
iv. Ενημέρωση συμμεταβλητότητας σφάλματος τη στιγμή κ (πίνακα διασποράς).
v. Εκτίμηση νέας διασποράς σφάλματος.
vi. Πρόβλεψη για τη στιγμή κ+1.

Ο κύκλος λειτουργίας του φίλτρου εκτελείται επαναληπτικά.

Εκτεταμένο φίλτρο Kalman

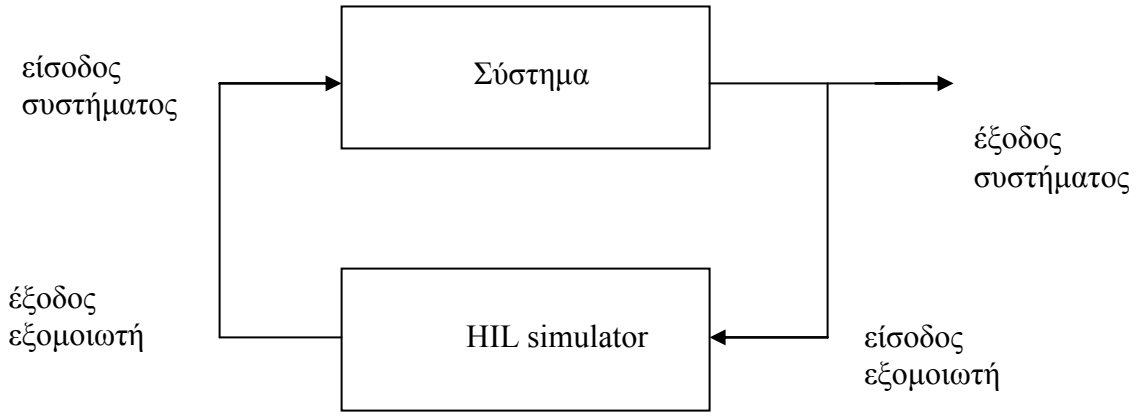
Κρίνεται απαραίτητο να αναφερθεί πως το φίλτρο στην αρχική του μορφή, σχεδιάστηκε για την αντιμετώπιση προβλημάτων εκτίμησης της κατάστασης γραμμικών συστημάτων. Η ανάγκη να χρησιμοποιηθεί σε μεγαλύτερο όγκο προβλημάτων οδήγησε στην επέκταση της χρήσης του σε μη γραμμικά συστήματα, με την μορφή του εκτεταμένου φίλτρου Kalman (Extended Kalman Filter, η αλλιώς EKF), για το οποίο στην παρούσα εργασία δεν θα γίνει εκτεταμένη αναφορά στις μεθόδους του. Η γενική τακτική του φίλτρου είναι να διαμορφώνει όσο το δυνατόν πιο γραμμικά την τελευταία τιμή του μέσου (mean) και της συμμεταβλητότητας (covariance).

ΠΡΟΣΟΜΟΙΩΣΗ ΜΕ ΤΟ ΥΛΙΚΟ ΕΝΤΟΣ ΤΟΥ ΒΡΟΓΧΟΥ ΕΛΕΓΧΟΥ

(HARDWARE IN THE LOOP SIMULATION)

Η προσομοίωση με το υλικό εντός βρόγχου ελέγχου (HIL simulation), είναι η τεχνική, που χρησιμοποιείται ευρέως σήμερα και προσφέρει μια αποτελεσματική πλατφόρμα για την ανάπτυξη πολύπλοκων ενσωματωμένων συστημάτων πραγματικού χρόνου (real-time embedded systems), καθώς και την δοκιμή τους. Το πολύπλοκο σύστημα, που τίθεται υπό έλεγχο, αναπαριστάται με μαθηματικές σχέσεις όλων των σχετικών δυναμικών συστημάτων, δημιουργώντας ένα «μοντέλο προσομοίωσης». Στη συνέχεια υπάρχει η δυνατότητα αλλαγής των δεδομένων που λαμβάνει το σύστημα με τα επιθυμητά , δηλαδή αυτά που θα δεχόταν αν βρισκόταν σε πραγματικές συνθήκες. Το μοντέλο προσομοίωσης και οι μαθηματικές σχέσεις που το αποτελούν επεξεργάζονται τα δεδομένα και κατά συνέπεια γίνεται έλεγχος των αντιδράσεων και των ενεργειών του συστήματος. Σκοπός του HIL simulation, λοιπόν, είναι να τροφοδοτεί συνέχεια το ελεγχόμενο σύστημα με ηλεκτρικά σήματα – δεδομένα, τέτοια ώστε να το εξαπατούν, αναγκάζοντάς το να αντιδρά σαν να βρίσκεται σε μια πραγματική κατάσταση.

Η χρησιμότητα του HIL simulation είναι πολύ σημαντική, καθώς εξασφαλίζει μικρότερο κόστος ανάπτυξης και τήρηση του χρονοδιαγράμματός της, μεγαλύτερη ασφάλεια, αλλά και περισσότερες και πολύωρες δοκιμές του συστήματος.



ΕΞΟΜΟΙΩΤΗΣ X-PLANE 10

Το X-plane είναι ο πιο περιεκτικός και ισχυρός εξομοιωτής πτήσης για PC σε παγκόσμιο επίπεδο, προσφέροντας τα πιο ρεαλιστικά μοντέλα πτήσης. Είναι ένα εργαλείο που μπορεί να χρησιμοποιηθεί για να προβλέψει την πορεία της πτήσης ενός fixed ή rotary wing σκάφους με εξαιρετική ακρίβεια και σε καμία περίπτωση ένα παιχνίδι εξομοίωσης.

Δεδομένου ότι το X-plane εξομοιώνει τον χειρισμό και την συμπεριφορά σχεδόν κάθε αεροσκάφους κατά την διάρκεια της πτήσης, για αυτή ακριβώς την δυνατότητά του χρησιμοποιείται από πιλότους, που θέλουν να βελτιώσουν τις δεξιότητες τους μέσω ενός εξομοιωτή, σε ρεαλιστικές συνθήκες. Οι μηχανικοί αεροσκαφών μπορούν με αυτόν τον εξομοιωτή να προβλέψουν την συμπεριφορά ενός νέου σκάφους και οι ερασιτέχνες πιλότοι έχουν την ευκαιρία να εξερευνήσουν τον κόσμο της πτητικής και δυναμικής συμπεριφοράς ενός αεροσκάφους σε ένα ασφαλές περιβάλλον.

18-1. Ρεαλιστική απεικόνιση αεροσκάφους , εν ώρα πτήσης.

Ο εξομοιωτής X-plane σήμερα

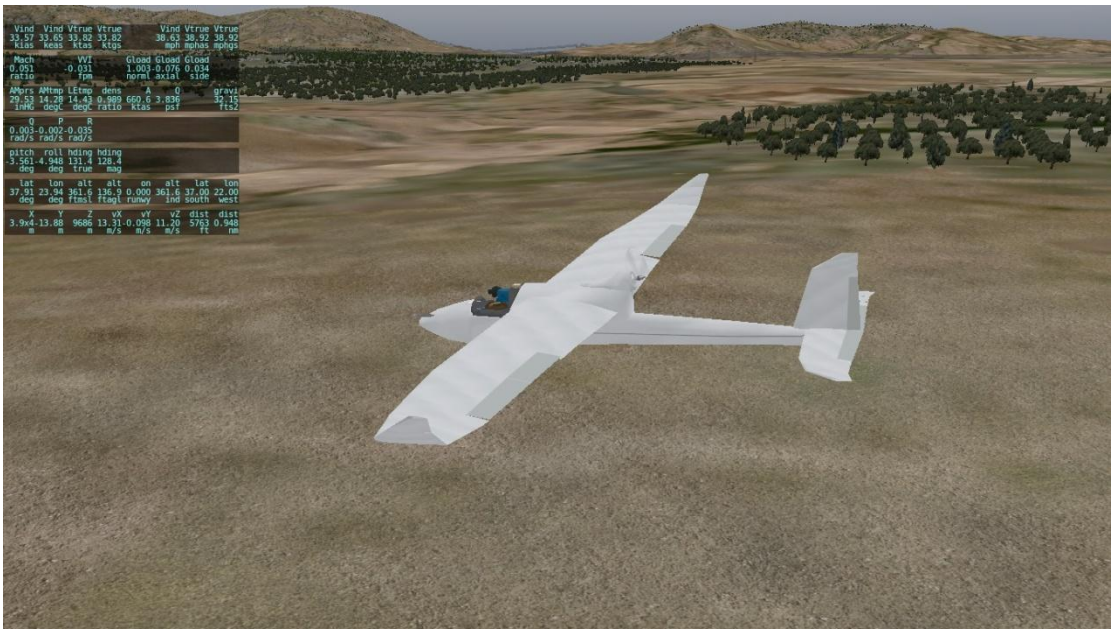
Η ποιότητα των χαρακτηριστικών του εξομοιωτή επιβεβαιώνεται, καθώς μηχανικοί από το Velocity, την Nasa, την Scaled Composites και την Carter Aviation έχουν χρησιμοποιήσει το X-Plane για να σχεδιάσουν, να αξιολογήσουν και να εξομοιώσουν τη διαδικασία πτήσης κάποιου/κάποιων αεροσκαφών. Το αμερικανικό εθνικό Test pilot school, επίσης, χρησιμοποιεί το X-Plane για να εκπαιδεύσει τους πιλότους σε μη συμβατικά συστήματα ελέγχου πτήσης και αεροσκαφών.

Οι περισσότεροι χρήστες, είτε για επαγγελματικούς, είτε για ψυχαγωγικούς σκοπούς, προτιμούν τον συγκεκριμένο εξομοιωτή για το εξαιρετικά υψηλό επίπεδο ρεαλισμού.

18-2. Εξομοίωση του πιλοτήριου.

Τα δεδομένα εξόδου του εξομοιωτή X-plane

Η επικοινωνία του X-Plane με τον χρήστη επιτυγχάνεται μέσω μιας σειράς δεδομένων εξόδου, προκειμένου να αναπαρασταθεί με την μεγαλύτερη δυνατή ακρίβεια η δυναμική συμπεριφορά τόσο του ελεγχόμενου σκάφους όσο και των εξωτερικών συνθηκών του περιβάλλοντος.



18-3. Πλάγια όψη εν ώρα πτήσης.

Για την κατανόηση των τιμών, που προκύπτουν, ως έξοδοι των οργάνων μέτρησης και αντιπροσωπεύουν όλες τις ενδείξεις του εξομοιωτή, είναι απαραίτητη η επεξήγηση ορισμένων όρων.

Οροι:

Groundspeed

Η ταχύτητα του σκάφους ως προς ένα σταθερό σημείο/παρατηρητή στη Γη.

Airspeed

Η ταχύτητα του σκάφους σε σχέση με την ταχύτητα του αέρα. Αν π.χ. ένας παρατηρητής πάνω στη Γη μετρά την ταχύτητα του σκάφους ίση με 50 μίλια/ώρα (V ground) και το σκάφος κινείται με ευνοϊκό άνεμο 20 μιλίων/ώρα, τότε το airspeed θα είναι 30 μίλια/ώρα. Αν, ως προς τον παρατηρητή στη Γη, η ταχύτητα του σκάφους είναι ίση με 50 μίλια/ώρα και ο άνεμος κινείται αντίθετα με ταχύτητα 20 μιλίων/ώρα τότε το airspeed θα ισούται με 70 μίλια/ώρα. Τέλος, αν υπήρχε περιβάλλον στο οποίο η ταχύτητα του ανέμου μπορούσε να είναι ίση με το μηδέν τότε το airspeed θα ταυτιζόταν με την ταχύτητα που θα μετρούσε ο παρατηρητής στη Γη (V ground).

Indicated Airspeed (IAS)

Στο εσωτερικό του αεροσκάφους βρίσκεται το όργανο με το οποίο καταμετράται η airspeed του σκάφους και ονομάζεται Airspeed Indicator. Η ένδειξη του οργάνου αυτού ορίζεται ως Indicated Airspeed (IAS).

True Airspeed (TAS)

Η πραγματική airspeed τιμή του σκάφους ονομάζεται True Airspeed (TAS). Η διαφορά μεταξύ της ένδειξης του οργάνου (Airspeed Indicator) και της πραγματικής τιμής (True airspeed) προκύπτει επειδή η μέτρηση επηρεάζεται από δύο στοιχεία του αέρα, την πίεση και την πυκνότητα. Εκ των δύο, το Airspeed Indicator αντιλαμβάνεται, μόνο τις μεταβολές που αφορούν στην πίεση του αέρα ενώ όσον αφορά την πυκνότητα, παρά το γεγονός ότι κινούμενο το σκάφος συναντά μόρια αέρα με διαφορετική πυκνότητα, δεν είναι σε θέση να αναγνωρίσει την μεταβολή αυτή. Έτσι η ένδειξη IAS είναι πρακτικά

ίδια με την TAS στο ύψος της θάλασσας κατά ISA (International Standard Atmosphere) και σε χαμηλές ταχύτητες.

Πρέπει να σημειωθεί πως το IAS έχει μια απόκλιση της τάξεως του 2%/1000 ft πάνω από το επίπεδο της θάλασσας σε σχέση με το TAS, κατά ISA.

Equivalent airspeed (EAS)

EAS= TAS * SQRT (p/p_0) όπου p η πραγματική πυκνότητα του αέρα και p_0 η πυκνότητα στο επίπεδο της θάλασσας κατά ISA.

Κόμβος

Μονάδα μέτρησης ταχύτητας. (1 κόμβος = 1,852 Km/h ή 1,151 miles/h)

Mach

Μονάδα μέτρησης ταχύτητας (1 Mach = 340 m/s περίπου). Η ταχύτητα του ήχου διαμέσου του αέρα.

Gload

Η αριθμητική αναλογία μιας ασκούμενης δύναμης σε σχέση με την δύναμη της βαρύτητας στο επίπεδο της επιφάνειας της Γης.

18-4. Μεγέθυνση ενδείξεων των οργάνων κατά την πτήση.

Στην φωτογραφία διακρίνονται οι εξής τιμές δεδομένων:

- Vind, kias

Η Indicated Airspeed σε κόμβους.

- Vind, keas

Η Equivalent airspeed σε κόμβους.

- Vtrue, ktas

Η True airspeed σε κόμβους.

- Vtrue, ktgs

Η True groundspeed σε κόμβους.

- Vind, mph

Η Indicated Airspeed σε μίλια ανά ώρα.

- Vtrue, mphas

Η True airspeed σε μίλια ανά ώρα (airspeed).

- Vtrue, mphgs

Η True airspeed σε μίλια ανά ώρα (groundspeed).

- Mach, ratio

Η ταχύτητα του σκάφους εκφρασμένη σε αναλογία με το 1 Mach.

- VVI, fpm

Η κατακόρυφη ταχύτητα του σκάφους σε feet/min.

- Gload, norml

Το g-load σε όλο το σκάφος ως προς το body axis system.

- Gload, axial

Το αξονικό gload του σκάφους.

- Gload, side

Το πλαϊνό gload του σκάφους.

- AMprs, inHG

Η βαρομετρική πίεση σε Inch of Mercury (1 HG= 3386.389 Pascal at 0 °C).

- AMtmp, degC

Η θερμοκρασία της ατμόσφαιρας σε °C.

- LEtmp, degC

Η θερμοκρασία της ατμόσφαιρας στο άκρο του φτερού, σε °C.

- dens, ratio

Η αναλογία πυκνότητας του αέρα μεταξύ δύο σημείων.

- A, ktas

Η airspeed μετρημένη σε knots.

- Q, psf

Το μέτρο της δυναμικής ενέργειας λόγω ύψους, ή λόγω πεδίου βαρύτητας σε Pounds per square foot.

- gravi, fts2

Η βαρυτική δύναμη σε feet/s^2.

- Q, rad/s

Ο ρυθμός μεταβολής του pitch ως προς το body axis system.

- P, rad/s

Ο ρυθμός μεταβολής του roll ως προς το body axis system.

- R, rad/s

Ο ρυθμός μεταβολής του yaw ως προς το body axis system.

- pitch, deg

Η γωνία pitch μετρημένη σε body axis.

- roll, deg

Η γωνία roll μετρημένη σε body axis.

- hding, true

Η πραγματική κατεύθυνση του σκάφους σε body-axis γωνίες Euler.

- hding, mag

Η μαγνητική κατεύθυνση του σκάφους σε μοίρες.

- lat, deg

Το γεωγραφικό πλάτος του σκάφους σε μοίρες.

- lon, deg

Το γεωγραφικό μήκος του σκάφους σε μοίρες.

- alt, ftmsl

Το ύψος του σκάφους σε πόδια πάνω από τη θάλασσα.

- alt, ftagl

Το ύψος του σκάφους σε πόδια πάνω από τη Γη.

- alt, ind

Το υψόμετρο που δείχνει το όργανο μέτρησης (σε πόδια).

- lat, south & lon, west

Η νοτιο-δυτική γωνία, ενός σημείου που έχει 3° longitude και 2° latitude.

- X, m

Το X σε μέτρα σε σχέση με το inertial frame system.

- Y, m

Το Y σε μέτρα σε σχέση με το inertial frame system.

- Z, m

Το Z σε μέτρα σε σχέση με το inertial frame system.

- vX, m/s

Η ταχύτητα στον άξονα X σε σχέση με το inertial frame system.

- vY, m/s

Η ταχύτητα στον άξονα Y σε σχέση με το inertial frame system.

- vZ, m/s

Η ταχύτητα στον άξονα Z σε σχέση με το inertial frame system.

- dist, ft

Η απόσταση που έχει διανυθεί σε πόδια.

- dist, nm

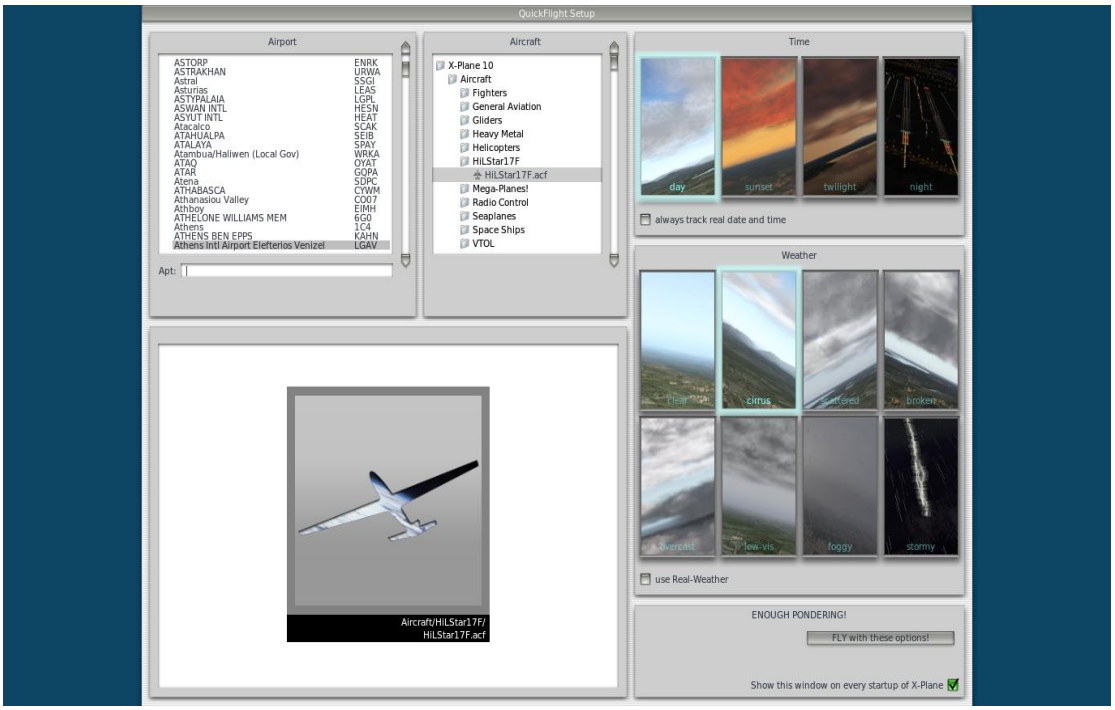
Η απόσταση που έχει διανυθεί σε ναυτικά μίλια.

ΕΓΚΑΤΑΣΤΑΣΗ HARDWARE IN THE LOOP SIMULATION ΓΙΑ ΤΟΝ PX4

Η αναγκαιότητα χρήσης της προσομοίωσης HIL σε μελέτες όπως η παρούσα έγινε απόλυτα σαφής, όπως άλλωστε και η δυνατότητα υλοποίησής της μέσω του μικροελεγκτή PX4. Η χρησιμότητα της προσομοίωσης έγκειται στο γεγονός ότι μπορούμε να εξετάσουμε τον κώδικα, που εκτελεί ο μικροελεγκτής, χωρίς να είναι απαραίτητη η απογείωση ενός μοντέλου τηλεκατευθυνόμενου αεροπλάνου. Με τον τρόπο αυτό τα δεδομένα των αισθητήρων κατευθύνονται από την προσομοίωση στον αυτόματο πιλότο και αυτός με την σειρά του δίνει τα αποτελέσματα του ελέγχου που κάνει στην προσομοίωση. Το ενδεχόμενο ενός προβλήματος είτε στον κώδικα, είτε σε μια άλλη παράμετρο, που δεν είχε εξαρχής υπολογιστεί μπορεί να λυθεί με ασφάλεια χωρίς τις καταστροφικές συνέπειες στο πραγματικό μοντέλο του τηλεκατευθυνόμενου αεροπλάνου. Το ίδιο ισχύει και στην περίπτωση αλλαγής ορισμένων παραμέτρων του συστήματος κατά τη βούληση του χρήστη.

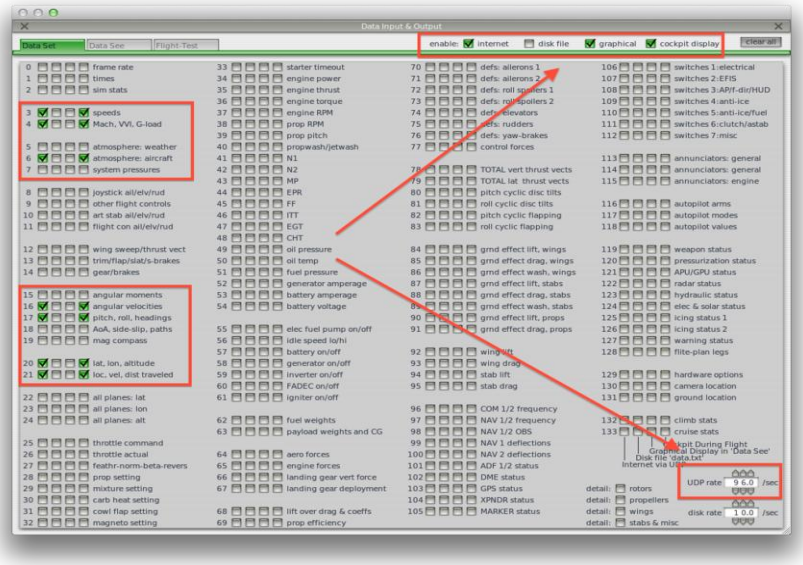
Για την υλοποίηση της προσομοίωσης θα πρέπει να υπάρχει εγκατεστημένο σε υπολογιστή το Qground control για επικοινωνία με τον μικροελεγκτή PX4, καθώς και το πρόγραμμα προσομοίωσης πτήσης X-Plane 10. Η σύνδεση του μικροελεγκτή με τον υπολογιστή πραγματοποιείται με καλώδιο USB και όχι ασύρματα μέσω της τηλεμετρίας.

Η διαδικασία έναρξης των εργασιών έχει ως εξής: Ανοίγουμε το πρόγραμμα Qground control, συνδέσουμε τον μικροελεγκτή PX4 με το καλώδιο USB, επιλέγουμε communication και connect και κατεβάζουμε το μοντέλο του τηλεκατευθυνόμενου αεροπλάνου από την διεύθυνση https://pixhawk.ethz.ch/px4/_media/users/diydroneshilstar17f0.3.2.zip. Στη συνέχεια πηγαίνουμε στον κατάλογο εγκατάστασης του X-Plane στον σκληρό δίσκο στον φάκελο Xpane\Aircraft και αντιγράφουμε το αρχείο HILStar17f που λάβαμε. Ξεκινάμε το X-Plane και επιλέγουμε το αεροσκάφος μας HILStar17f από την επιλογή select Aircraft.



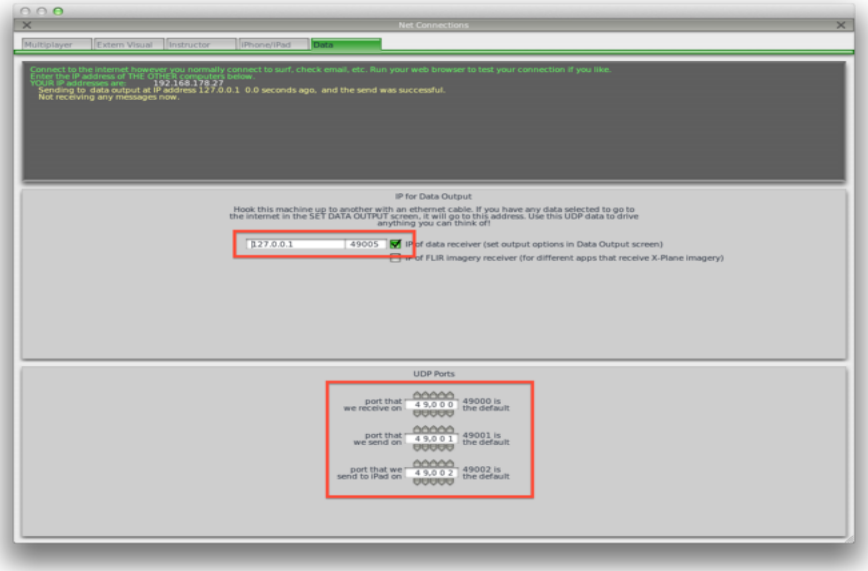
19-1. Επιλογή τύπου αεροσκάφους και αεροδρομίου.

Όταν το αεροσκάφος εμφανιστεί στην οθόνη, επιλέγουμε με το ποντίκι από το κύριο μενού στο πάνω μέρος settings και data input & output για να ρυθμίσουμε τους τρόπους επικοινωνίας του X-Plane. Στη νέα καρτέλα του μενού επιλέγουμε από τους αριθμούς 3,4,6,16,17,20,21 το πρώτο και το τελευταίο τετραγωνάκι και στο UDP rate πληκτρολογούμε 96.



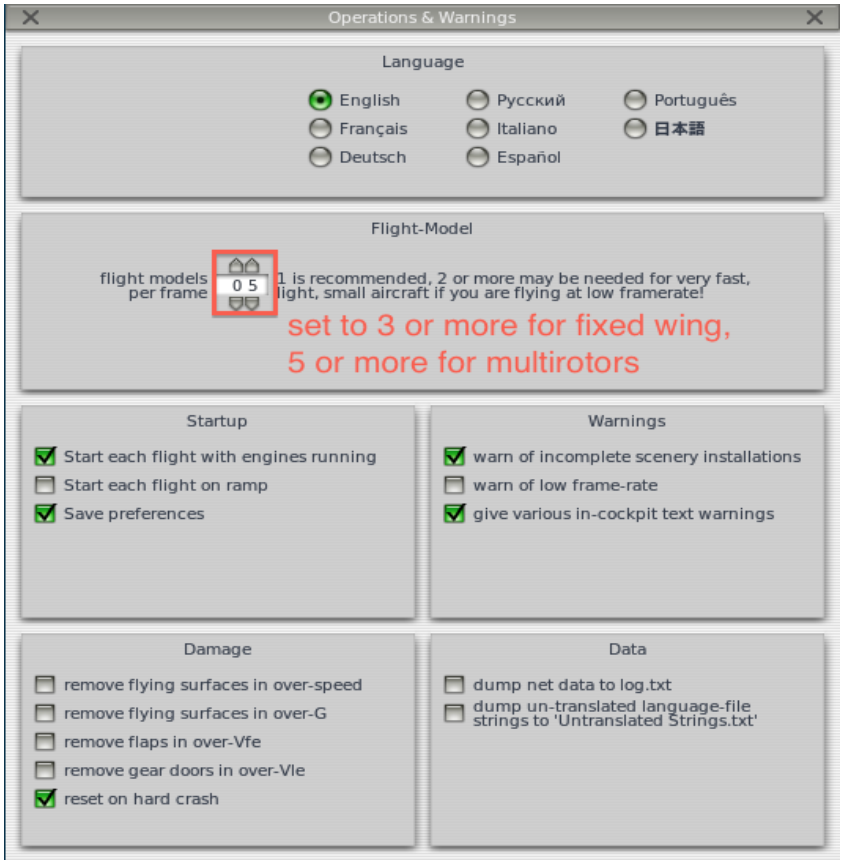
19-2. Επιλογή ρυθμίσεων επικοινωνίας στο XPlane 10.

Επιλέγουμε, εκ νέου από το μενού settings και net connections, την καρτέλα data. Στην IP βάζουμε 127.0.0.1 και στην UDP data port 49005, Port that we receive on 49000 και port that we send on 49001.



19-3. Επιλογή πυλών επικοινωνίας.

Από το μενού settings και operations & warnings επιλέγουμε και αυξάνουμε το flight models per frame σε 5.



19-4. Επιλογή flight models per frame.

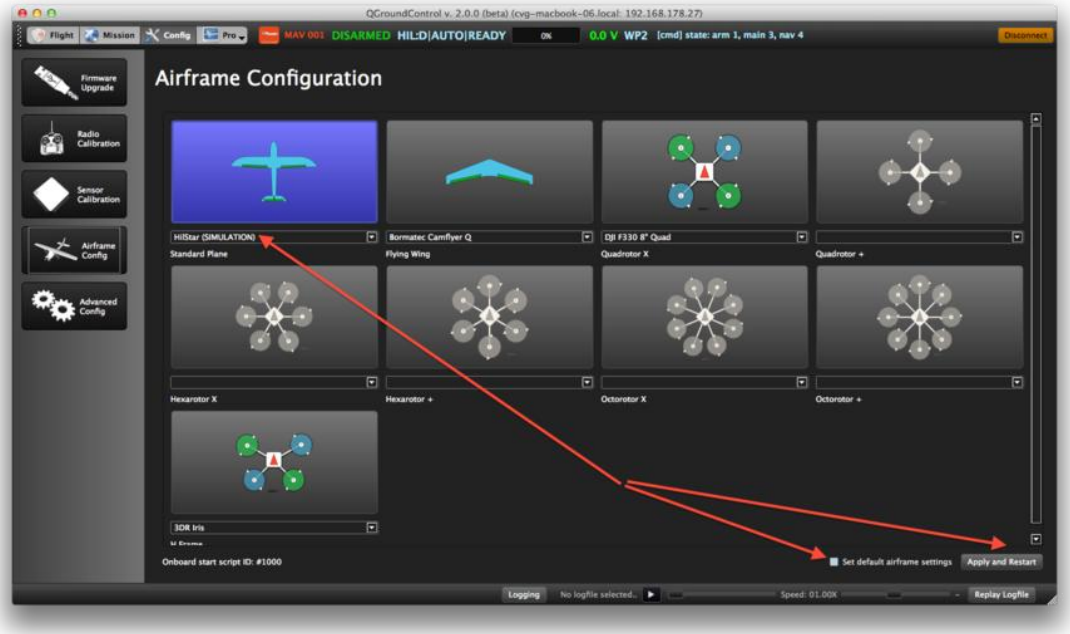
Αφήνουμε ανοιχτό το X-Plane και συνεχίζουμε στο Qground control.

Απαραίτητη προϋπόθεση για να είναι δυνατή η σύνδεση με τον PX4 είναι η αφαίρεση οποιουδήποτε αρχείου autostart (rc.txt) που υπάρχει στην κάρτα microSD. Επιπρόσθετα σημειώνεται, κατόπιν παρατήρησης για την συγκεκριμένη έκδοση του λογισμικού του PX4, ότι είναι καλύτερο να μην υπάρχουν αρχεία παραμέτρων (parameters) στον κυρίως φάκελο της κάρτας, καθώς και να μην έχει γίνει calibration αισθητήρων.

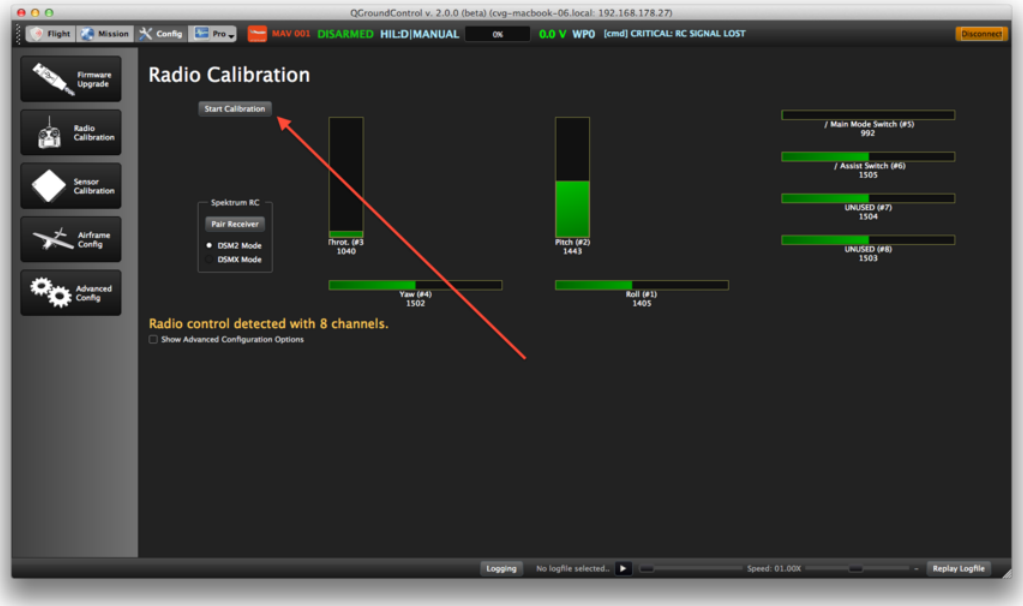
Επιλέγουμε από την επάνω μπάρα config, στα αριστερά airframe config και στο πάνω αριστερά εικονίδιο του αεροπλάνου το HILstar simulation. Κατόπιν πατάμε apply and restart και αφού ακούσουμε τον ήχο αποσυνδέουμε το καλώδιο USB και το ξανασυνδέουμε για να κάνουμε connect. Επιλέγουμε radio calibration από το μενού αριστερά και ακολουθούμε τα βήματα που θα ζητηθούν.

Σημείωση: Για να μπορούμε να ενεργοποιήσουμε όλα τα modes πτήσης από την τηλεκατεύθυνση, θα πρέπει να ρυθμίσουμε κάποιες παραμέτρους, ώστε να έχουμε στο χειριστήριο παλμούς διαφορετικούς από τους προεγκατεστημένους. Η διαδικασία που ακολουθείται (για το χειριστήριο Turnigy TGY 9X) είναι η εξής:

Επιλέγουμε menu → settings → AUX-CH και στο Channel 5 επιλέγουμε THRO HOLD. Ύστερα τοποθετούμε τον διακόπτη F.MODE στη θέση 2 και τον διακόπτη THRO HOLD στην θέση κάτω. Στο MENU πηγαίνουμε στο PROG.MIX και επιλέγουμε MIX1. Στο DNRATE που εμφανίζεται επιλέγουμε τιμή ώστε η τιμή του παλμού που εμφανίζεται στο menu Radio Calibration (εικ.19-6) να είναι 1555ms. Στη συνέχεια επαναφέρουμε τον διακόπτη THRO HOLD στην προηγούμενη θέση και μέσω του menu MIX1 (του PROG.MIX) ρυθμίζουμε την τιμή UPRATE έτσι ώστε στο Radio Calibration να έχουμε παλμό 1425ms. Επιστρέφουμε τον διακόπτη F.MODE στην θέση N και μέσα από το menu PROG.MIX επιλέγουμε το MIX2. Στο DNRATE που εμφανίζεται ρυθμίζουμε την τιμή ώστε να εμφανίζεται παλμός 1685ms στο Radio Calibration. Μετά γυρνάμε τον διακόπτη THRO HOLD προς τα πάνω και ρυθμίζουμε την τιμή UPRATE ώστε να παίρνουμε ένα παλμό σε τιμή 1295ms. Συνεχίζοντας τοποθετούμε τον διακόπτη F.MODE στην θέση 1 και τον THRO HOLD προς τα κάτω. Στο MIX3 του menu ρυθμίζουμε την τιμή DNRATE ώστε να δίνει παλμό 1815ms. Τέλος γυρίζουμε τον διακόπτη THRO HOLD στην πάνω θέση και ρυθμίζουμε την τιμή UPRATE του MIX3 για παλμό 1165ms.

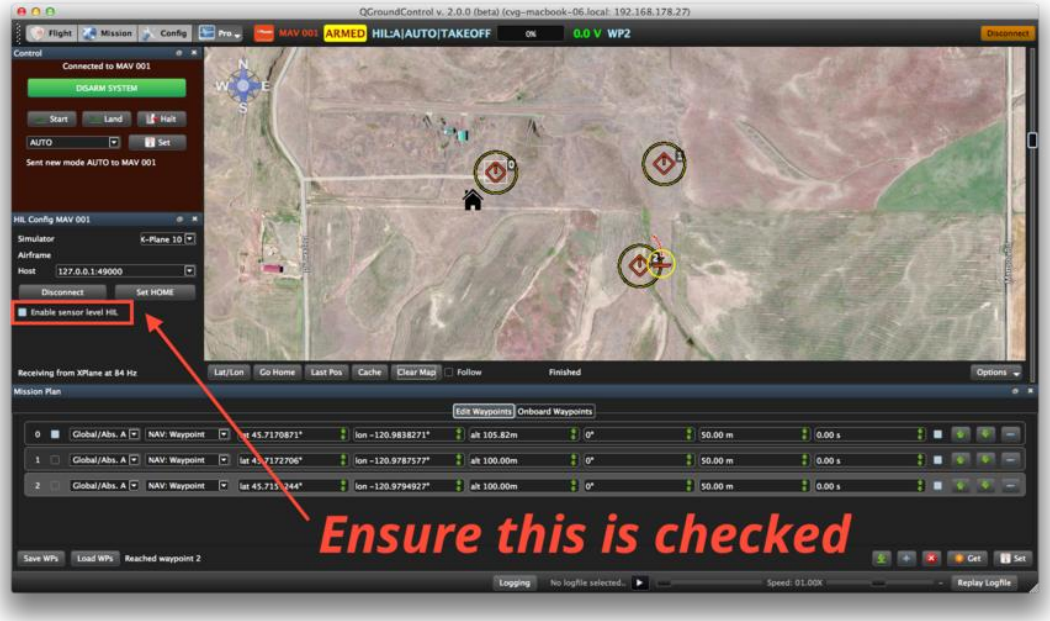


19-5. Επιλογή τύπου σκάφους στο (HIL simulation) QGround Control.



19-6. Calibration τηλεκατεύθυνσης στο QGround Control.

Στην συνέχεια επιλέγουμε pro και simulation από το μενού πάνω αριστερά. Στο πάνελ ρυθμίσεων Hil config MAV001 επιλέγουμε X-Plane 10, Host 127.0.0.1:4900, ενεργοποιούμε το enable sensor level HIL και πατάμε connect. Αμέσως μετά τον ήχο θα δούμε στο χάρτη του Qground control ότι το αεροπλάνο έχει πάρει την θέση που έχει και στο X-Plane (ίδιο αεροδρόμιο και αεροδιάδρομος).



19-7. Απεικόνιση σύνδεσης με XPlane, τοποθεσίας και σημείων διαδρομής στο QGround Control.

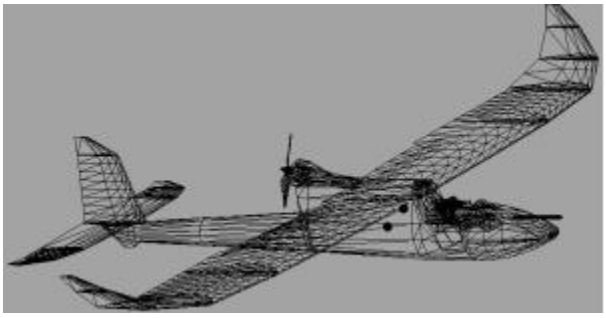
Μετά από όλες τις παραπάνω ενέργειες μπορούμε να επιλέξουμε σημεία διαδρομής στον χάρτη στο Qground control. Με διπλό κλικ στο σημείο, που επιθυμούμε, το επιλέγουμε και αυτό προστίθεται στο κάτω μέρος της οθόνης μαζί με άλλα σημεία διαδρομής σε λίστα, με γεωγραφικό μήκος και πλάτος, ύψος, loiter και άλλες λεπτομέρειες που μπορούμε να επεξεργαστούμε. Επιλέγοντας set όλα τα σημεία μεταφέρονται στον PX4. Προσέχουμε πάντα να υπάρχει σύνδεση μεταξύ Qground control και X-Plane παρατηρώντας το «Receiving from X-Plane at xx Hz». Πατάμε το διακόπτη ασφαλείας του PX4 μέχρι να αναβοσβήνει διπλά και συνεχόμενα, μόνο τότε μπορούμε να επιλέξουμε manual και set από το πάνελ ρυθμίσεων connected to MAV001

. Προσέχουμε να μην είναι συνδεδεμένοι οι κινητήρες στην πλακέτα του PX4. Επιλέγουμε ARM system και μπορούμε πλέον με την τηλεκατεύθυνση να απογειώσουμε το αεροσκάφος.

Στο X-Plane βλέπουμε σε πραγματικό χρόνο μια ρεαλιστική εικόνα του αεροσκάφους όπως στην πραγματικότητα και ταυτόχρονα στο Qground control το βλέπουμε να κινείται στον χάρτη. Στην συνέχεια αν επιλέξουμε auto και set ενεργοποιείται ο αυτόματος πιλότος και το αεροσκάφος κινείται προς το πρώτο σημείο που έχουμε ορίσει στον χάρτη. Μπορούμε να αλλάζουμε τα σημεία στο χάρτη ή να τα αφαιρούμε κατά την διάρκεια της πτήσης .

ΑΛΛΗΛΕΠΙΔΡΑΣΗ ΠΑΡΑΜΕΤΡΩΝ PID ΚΑΙ ΣΚΑΦΟΥΣ ΣΤΗΝ ΠΡΑΞΗ

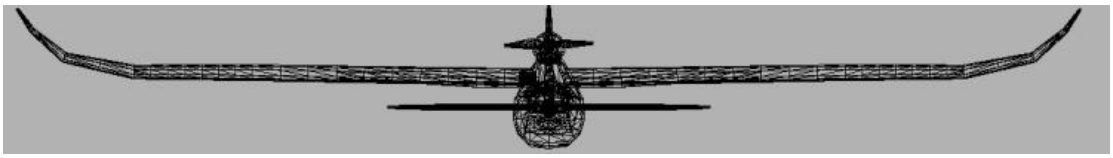
Στο παρόν κεφάλαιο παρουσιάζεται η επίδραση των μεταβολών των παραμέτρων του ελεγκτή στις γωνίες Euler, στην ευστάθεια της πτήσης και στην πορεία του σκάφους. Λαμβάνοντας τις ιδανικές παραμέτρους του ελεγκτή PID, αναλόγως το επιλεγμένο τύπο αεροσκάφους, από τις βιβλιοθήκες του PX4, πραγματοποιούμε δοκιμαστική πτήση στο πρόγραμμα εξομοίωσης. Στη συνέχεια, μεταβάλλουμε τις τιμές των παραμέτρων του ελεγκτή και παρατηρούμε αν το αποτέλεσμα της εξομοίωσης της πτήσης συμπίπτει της θεωρίας. Το μοντέλο της έρευνας (Hilstar 17), για την κίνηση του επενεργεί στις δύο από τις τρείς γωνίες Euler (pitch και roll).



20-1. HiLStar 17 [πηγή: diydrones.com].

Το μοντέλο που χρησιμοποιήθηκε στην εξομοίωση με το XPlane, είναι το HiLStar 17. Πρόκειται για ένα μοντέλο σχεδιασμένο με βάση το πραγματικό τηλεκατευθυνόμενο αεροσκάφος μοντελισμού Bixler της HobbyKing. Είναι κατασκευασμένο από αφρώδες υλικό (EPO foam) και μπορεί να ελεγχθεί πλήρως μέσω

μίας τετρακάναλης τηλεκατεύθυνσης, κάθε κανάλι της οποίας χειρίζεται τις μεταβολές των rudder, ailerons, elevator και throttle.



20-2. Οπίσθια όψη του HiLStar 17 [πηγή: diydrones.com].

Το βάρος είναι κατανεμημένο όπως στο πραγματικό μοντέλο, έχει άνοιγμα φτερών 2380mm και μήκος 1572mm. Ο κινητήρας του είναι brushless outrunner, το βάρος του είναι 2,26kg (με το σύνολο του εξοπλισμού πτήσης και τον ελεγκτή) και ο τύπος της προπέλας είναι APC 6x4E.

Έλεγχος του σκάφους

Ο έλεγχος της πορείας του σκάφους γίνεται μέσω των γωνιών pitch και roll. Η στροφή του σκάφους (δεξιά-αριστερά) δεν γίνεται μέσω του rudder και της γωνίας yaw αλλά μέσω της roll. Σε κάθε μία από τις δύο γωνίες αυτές Euler εφαρμόζεται έλεγχος. Για την ακρίβεια στην γωνία roll εφαρμόζεται PI έλεγχος και στην γωνία pitch απλός αναλογικός μηδενίζοντας τις παραμέτρους D και I,D του PID αντίστοιχα. Οι ιδανικές τιμές των κερδών αναλογίας, ολοκλήρωσης και διαφόρισης υπολογίζονται από τον PX4 μέσω ειδικών αλγορίθμων. Μέσω του εξομοιωτή και του Qground Control μεταβάλλουμε τις παραμέτρους του ελεγκτή και διαπιστώνουμε την επαλήθευση της θεωρίας ελέγχου σε κάθε περίπτωση και ανάλογα με το τι περιμέναμε να συμβεί πριν κάθε αλλαγή παραμέτρου.

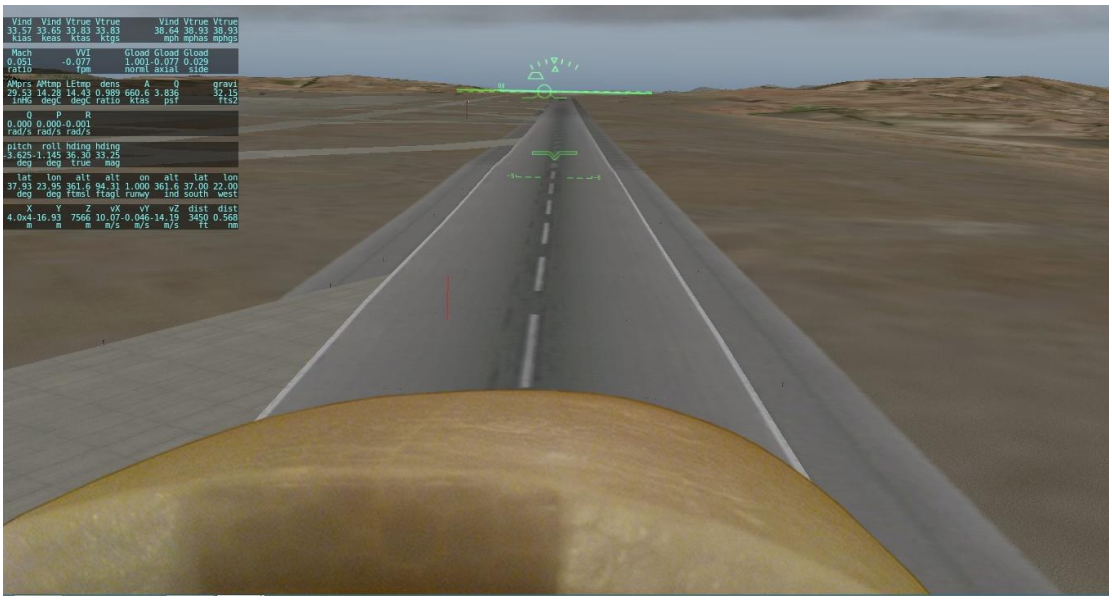
Ελεγκτής PID για την γωνία roll

Οι ιδανικές τιμές των παραμέτρων του ελεγκτή PID, όπως υπολογίζονται από τον αλγόριθμο του κώδικα των βιβλιοθηκών του μικροελεγκτή PX4 είναι οι εξής:

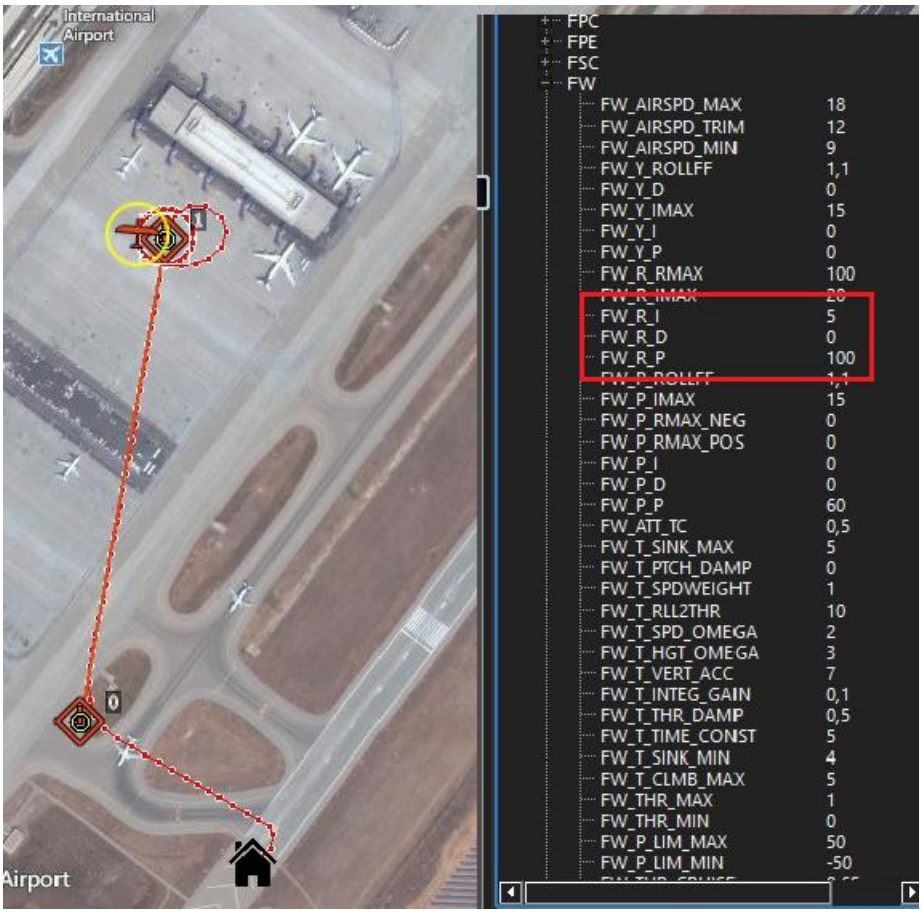
FW_R_P = 100 (fixedwing_roll_P)

FW_R_I = 5 (fixedwing_roll_I)

FW_R_D = 0 (fixedwing_roll_D)

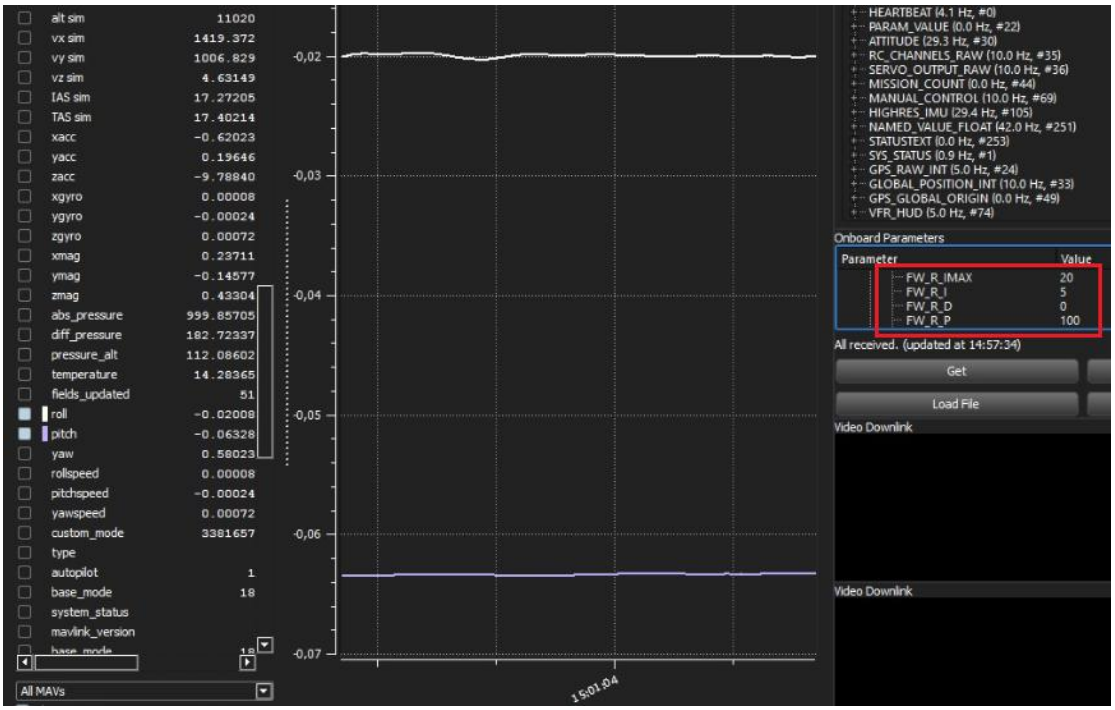


20-3. Στιγμιότυπο πτήσης του HiLStar17 στο XPlane.



20-4. Πορεία αεροσκάφους με ιδανικές παραμέτρους PID.

Παρατηρούμε ότι το σκάφος ακολουθεί ακριβώς την βέλτιστη διαδρομή μεταξύ των waypoints, όταν ο ελεγκτής είναι ιδανικά ρυθμισμένος ($k_p = 100$, $k_i = 5$, $k_d = 0$), ενώ παράλληλα διατηρεί και την μέγιστη ευστάθεια. Αυτό διακρίνεται από τη σταθερότητα των γωνιών roll και pitch στον χρόνο (19-5).

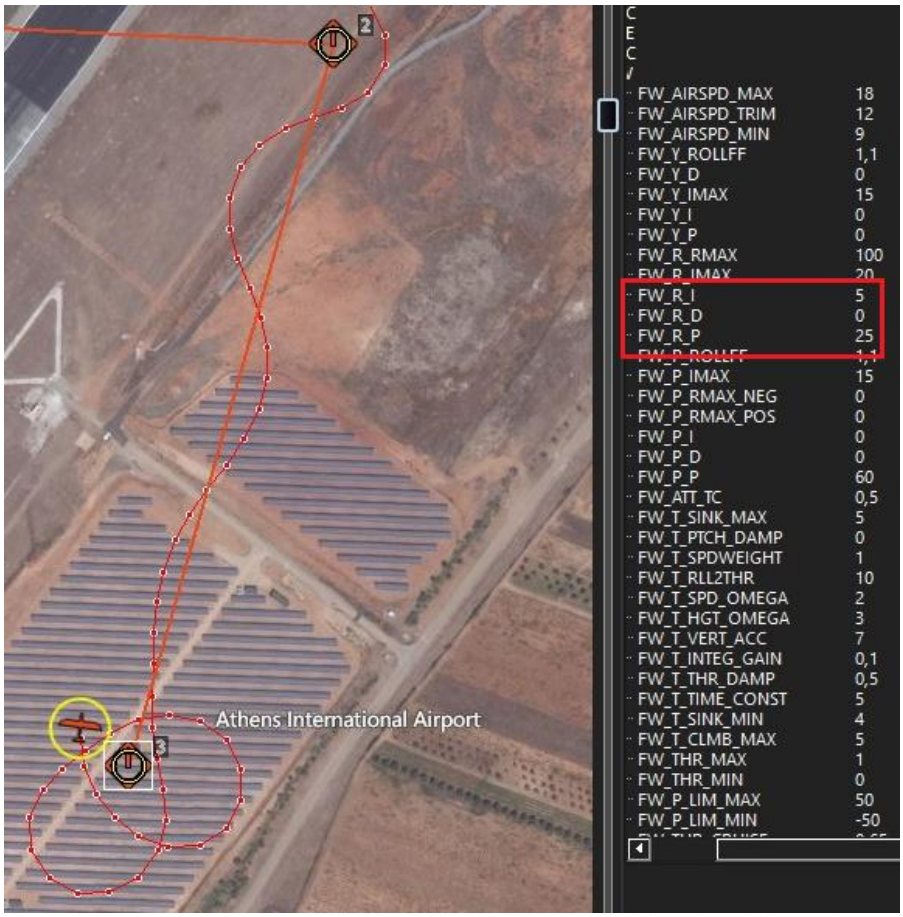


20-5. Γωνίες roll και pitch με ιδανικό PID.

Στη συνέχεια, διαφοροποιούμε τις τιμές της παραμέτρου P και καταγράφουμε την απόκλιση από την βέλτιστη διαδρομή, μεταξύ των δύο waypoints.

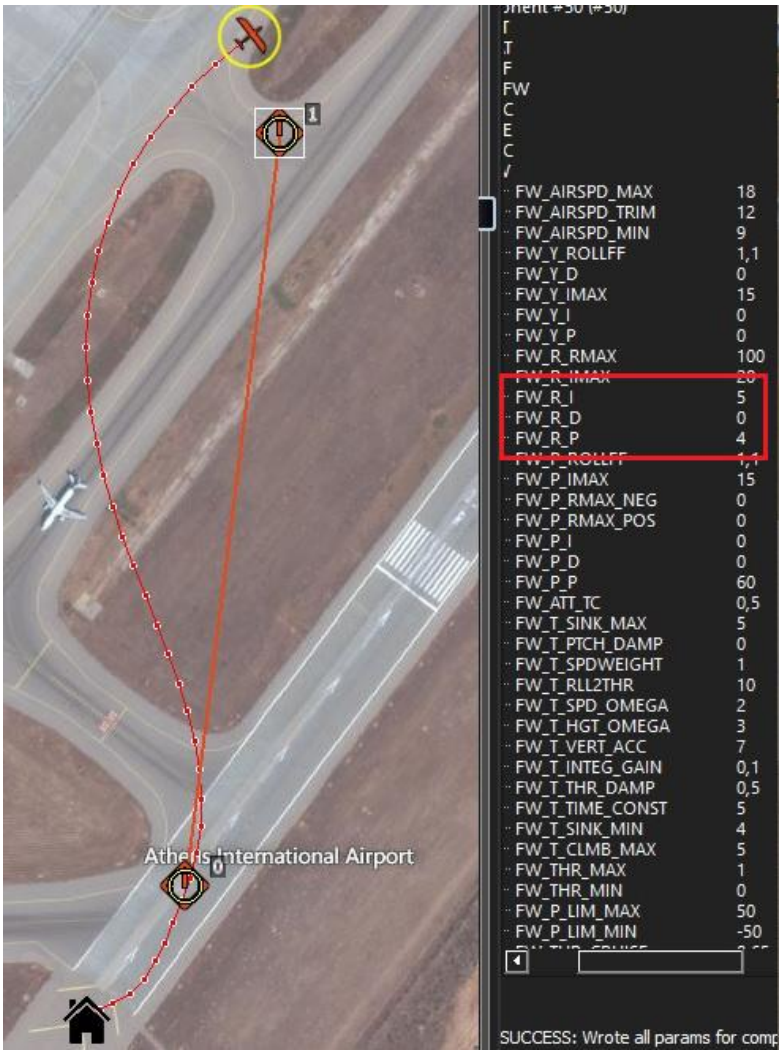
Αρχικά, μειώνουμε σταδιακά τις τιμές της παραμέτρου.

20-6. Πορεία με τιμές παραμέτρων $k_p = 40$, $k_i = 5$, $k_d = 0$.



20-7. Πορεία με τιμές παραμέτρων $k_p = 25$, $k_i = 5$, $k_d = 0$.

20-8. Πορεία με τιμές παραμέτρων $k_p = 12$, $k_i = 5$, $k_d = 0$.

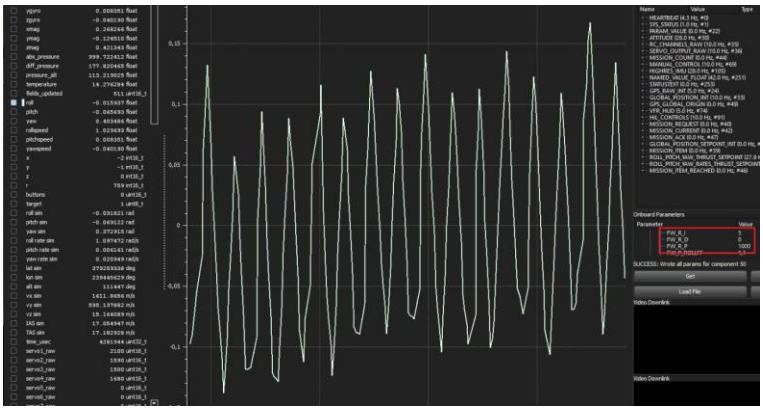


20-9. Πορεία με τιμές παραμέτρων $k_p = 4$, $k_i = 5$, $k_d = 0$.

Μειώνοντας την τιμή της παραμέτρου P του ελεγκτή, παρατηρούμε την σταδιακά αυξανόμενη απόκλιση του σκάφους από την βέλτιστη προδιαγεγραμμένη διαδρομή μεταξύ των δύο waypoints. Η απόκλιση αυτή οφείλεται στο γεγονός ότι καθώς η τιμή της παραμέτρου του ελεγκτή P απομακρύνεται από την ιδανική μειώνεται η δυνατότητα του ελεγκτή να ελαχιστοποιεί το σφάλμα μεταξύ πραγματικής τιμής και του setpoint. Στην

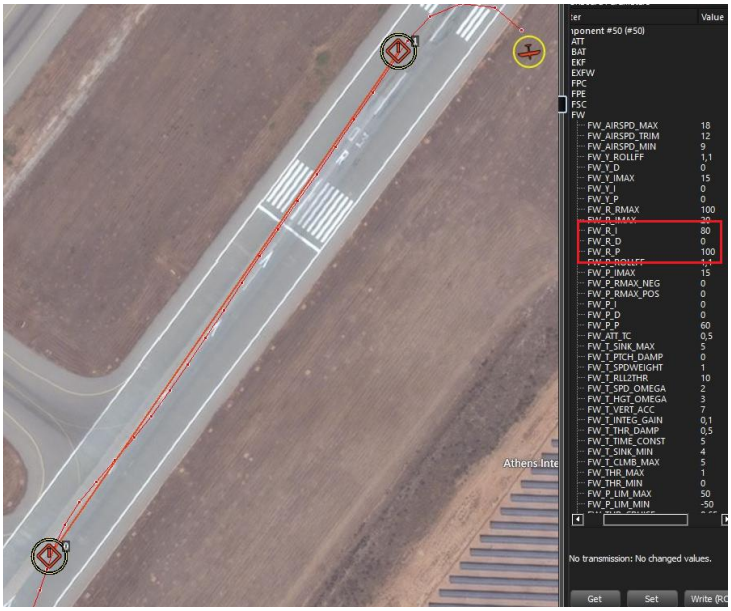
οριακή τιμή του k_p=4 παρατηρούμε την αδυναμία του αυτόματου πιλότου να οδηγήσει το σκάφος στο επιθυμητό waypoint.

Στη συνέχεια αυξάνουμε την τιμή του k_p =1000. Παρατηρούμε ότι το σκάφος ακολουθεί τη βέλτιστη δυνατή διαδρομή μεταξύ των waypoints καθώς η τιμή k_p του ελεγκτή του επιτρέπει να ελαχιστοποιήσει το σφάλμα, παράλληλα όμως έχουμε απώλεια στην ευστάθεια του σκάφους, το οποίο διακρίνεται από τις μεταβολές τις γωνίας roll στον χρόνο, όπως παρουσιάζεται στην εικόνα 19-10. Κάθε παρατήρηση μας πάνω στις μεταβολές, στην πορεία και την ευστάθεια του σκάφους συμπίπτει απόλυτα με την θεωρία ελέγχου. Με την σταδιακή μείωση του κέρδους αναλογίας περιμέναμε την όλο και αυξανόμενη απόκλιση του σκάφους από την επιθυμητή πορεία λόγω του ότι μειώνεται η ικανότητα του ελεγκτή να ελαχιστοποιεί το μόνιμο σφάλμα. Αυτό συμβαίνει γιατί όσο μειώνεται ο συντελεστής του όρου αναλογίας, τόσο μειώνεται η ισχύς του όρου στην εξίσωση και όπως είναι λογικό, σε τιμές κοντά στο μηδέν ο όρος σχεδόν εξαλείφεται. Η δουλειά του όρου P στον ελεγκτή είναι να μειώσει την απόσταση μεταξύ πραγματικής και επιθυμητής τιμής όσο περισσότερο γίνεται έτσι ώστε να ακολουθεί το σύστημα την επιθυμητή συμπεριφορά έστω με κάποια απόκλιση. Η εξάλειψη του μόνιμου σφάλματος αφορά στον όρο I. Στην περίπτωση της πολύ μεγάλης τιμής που δώσαμε στον ελεγκτή αναμέναμε ότι θα συνεχίσει να ακολουθεί την επιθυμητή πορεία ακριβώς διότι το κέρδος ήταν τέτοιο που επέτρεπε την μείωση του σφάλματος σε χαμηλά επίπεδα αλλά επίσης ήταν αναμενόμενο να έχουμε απώλεια στην ευστάθεια του σκάφους λόγω του ότι η απόκριση του ελεγκτή σε κάθε μεταβολή του σφάλματος γίνεται αρκετά "απότομη".

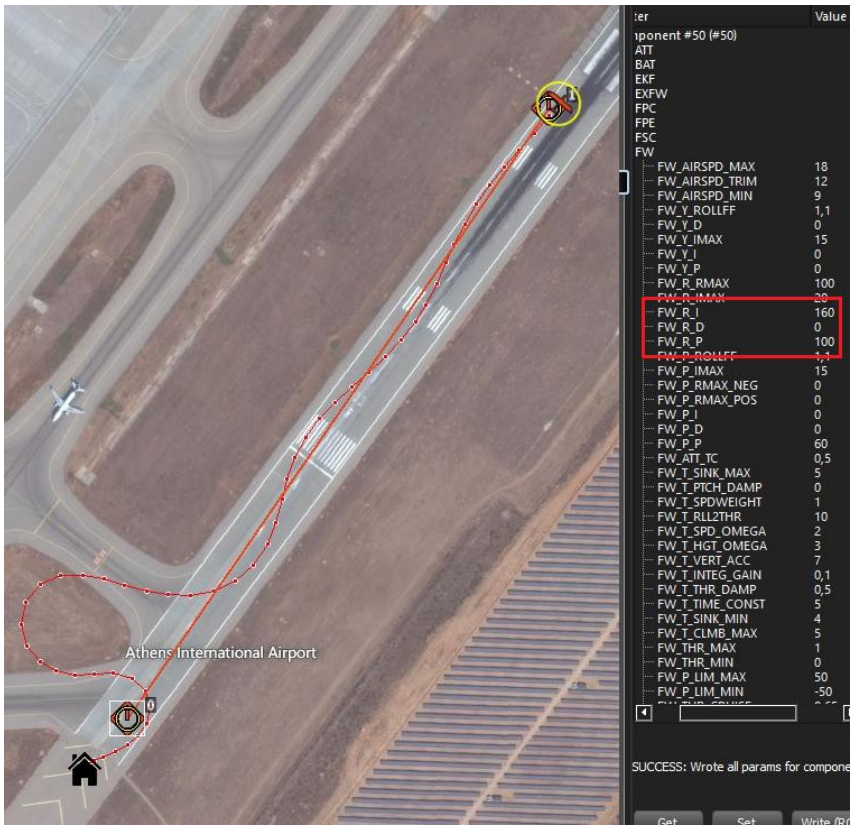


20-10. Γωνία roll με $k_p = 1000$, $k_i = 5$, $k_d = 0$.

Στη συνέχεια επαναφέρουμε την βέλτιστη τιμή για την παράμετρο $k_p = 100$ και μεταβάλλουμε την τιμή της παραμέτρου k_i. Στην πρώτη περίπτωση θέτουμε $k_i = 80$, στη δεύτερη θέτουμε $k_i = 160$, ενώ στη τρίτη περίπτωση $k_i = 200$.



20-11. Πορεία με τιμές παραμέτρων $k_p = 100$, $k_i = 80$, $k_d = 0$.

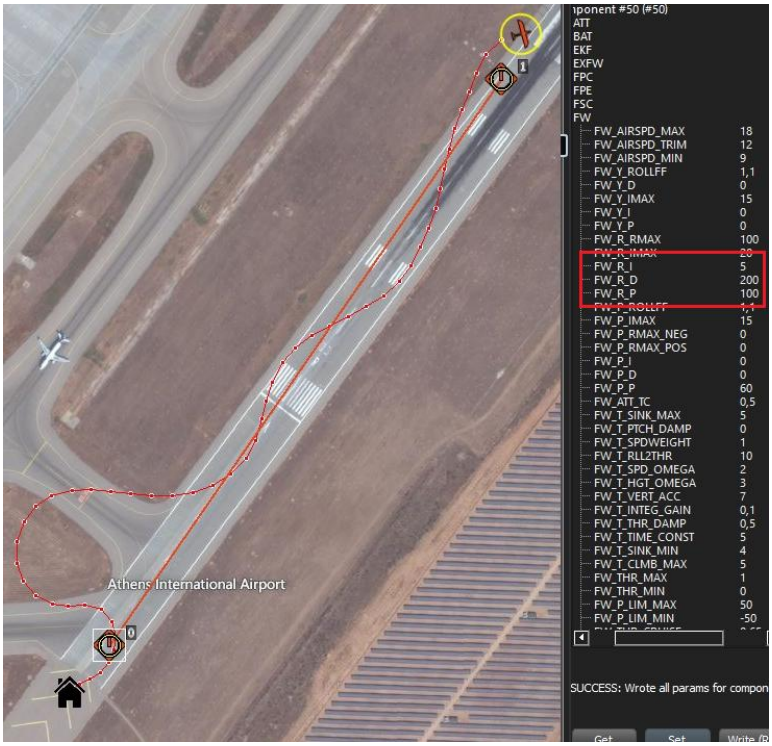


20-12. Πορεία με τιμές παραμέτρων $k_p = 100$, $k_i = 160$, $k_d = 0$.

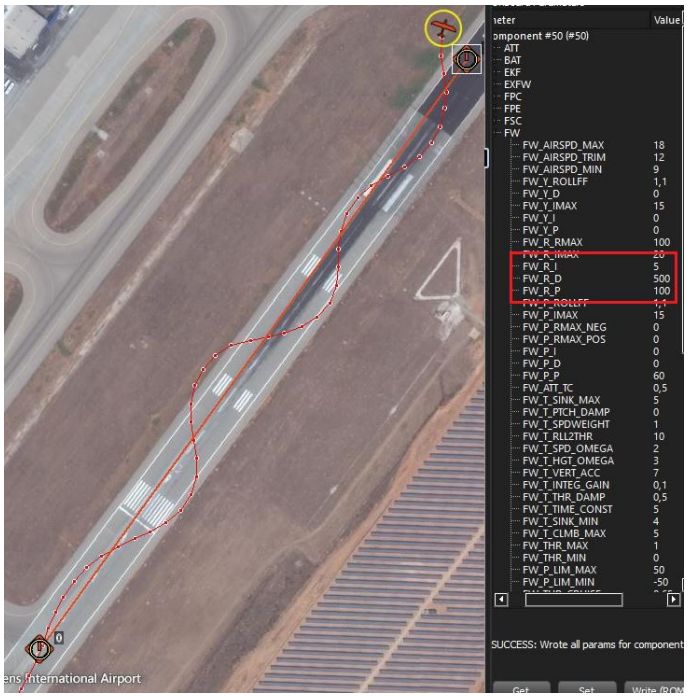
20-13. Πορεία με τιμές παραμέτρων $k_p = 100$, $k_i = 200$, $k_d = 0$.

Στην πρώτη περίπτωση παρατηρούμε ότι υπάρχει απόκλιση της πραγματικής από την επιθυμητή πορεία, το οποίο επιβεβαιώνει την θεωρία, διότι ο όρος I μηδενίζει το μόνιμο σφάλμα. Άρα με εσφαλμένη τιμή του όρου σωστά παρατηρείται μία απόκλιση. Στη δεύτερη περίπτωση παρατηρείται μία μεγαλύτερη απόκλιση, λόγω της μεγαλύτερης διαφοράς του όρου k_i από την βέλτιστη τιμή του. Τέλος με μία εντελώς εσφαλμένη τιμή του όρου k_i παρατηρούμε ότι το σκάφος δεν μπορεί να προσεγγίσει την επιθυμητή πορεία και χάνει τον προσανατολισμό του.

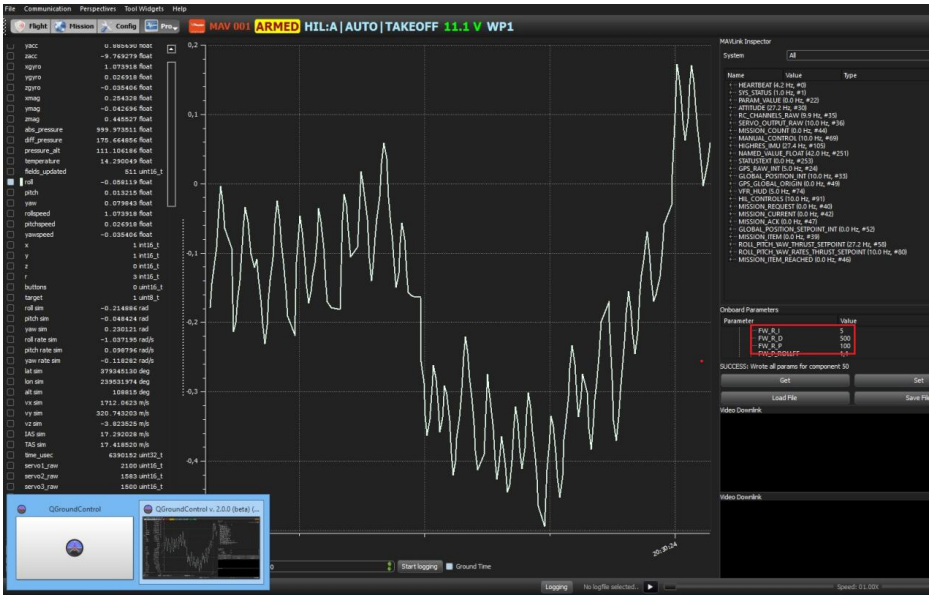
Τελικώς διαφοροποιούμε την τιμή της παραμέτρου k_d. Οι τιμές που χρησιμοποιούμε είναι $k_d = 200$ και $k_d = 500$.



20-14. Πορεία με τιμές παραμέτρων $k_p = 100$, $k_i = 5$, $k_d = 200$.



20-15. Πορεία με τιμές παραμέτρων $k_p = 100$, $k_i = 5$, $k_d = 500$.



20-16. Γωνία roll με $k_p = 100$, $k_i = 5$, $k_d = 500$.

Στην πρώτη περίπτωση παρατηρούμε μία απόκλιση από την επιθυμητή πορεία, λόγω του ότι η μεγάλη τιμή της παραμέτρου k_d εξαναγκάζει το σκάφος σε πολύ γρήγορες αντιδράσεις, με σκοπό την διόρθωση της πορείας του που όμως έχει ως αποτέλεσμα την μεγάλη υπερύψωση (μέγιστη απόκλιση από το setpoint) της ελεγχόμενης γωνίας. Στη δεύτερη περίπτωση με ακόμα μεγαλύτερο k_d παρατηρούμε μεγαλύτερη απόκλιση. Στην εικόνα 19-16, δίνεται η γραφική παράσταση της μεταβολής της γωνίας roll ως συνάρτηση του χρόνου, όπου παρατηρούμε τις βίαιες μεταβολές της γωνίας, αποτέλεσμα της επίδρασης του όρου k_d στον έλεγχο της.

Ελεγκτής PID για την γωνία pitch

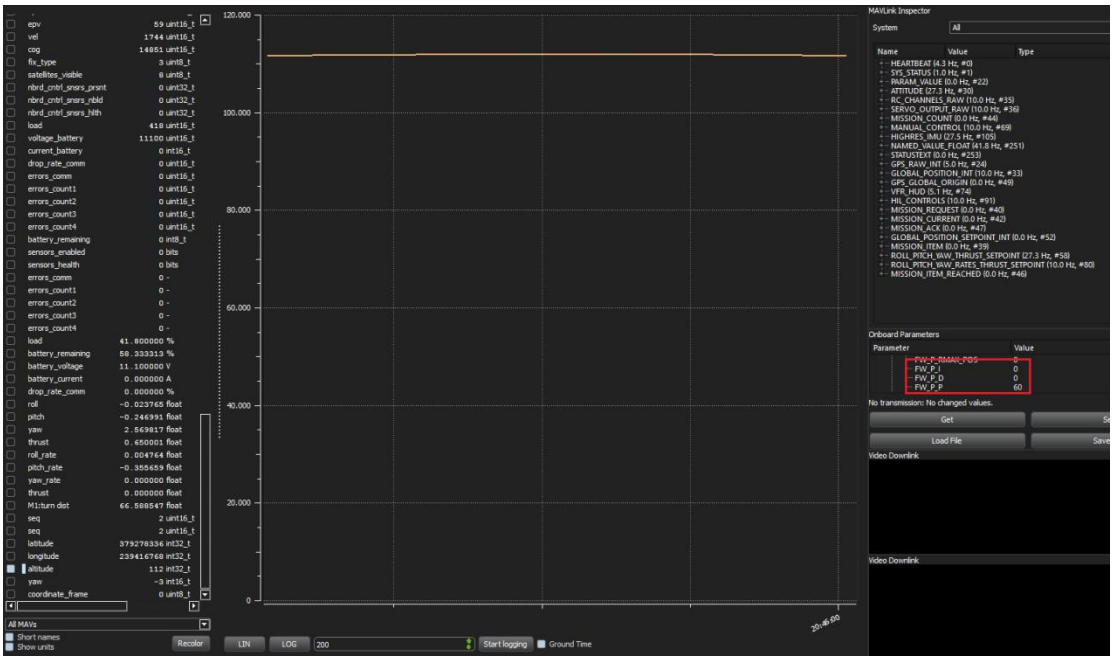
Οι ιδανικές τιμές των παραμέτρων του ελεγκτή PID, όπως υπολογίζονται από τον αλγόριθμο του κώδικα των βιβλιοθηκών του μικροελεγκτή PX4 είναι οι εξής:

FW_P_P = 60 (fixedwing_pitch_P)

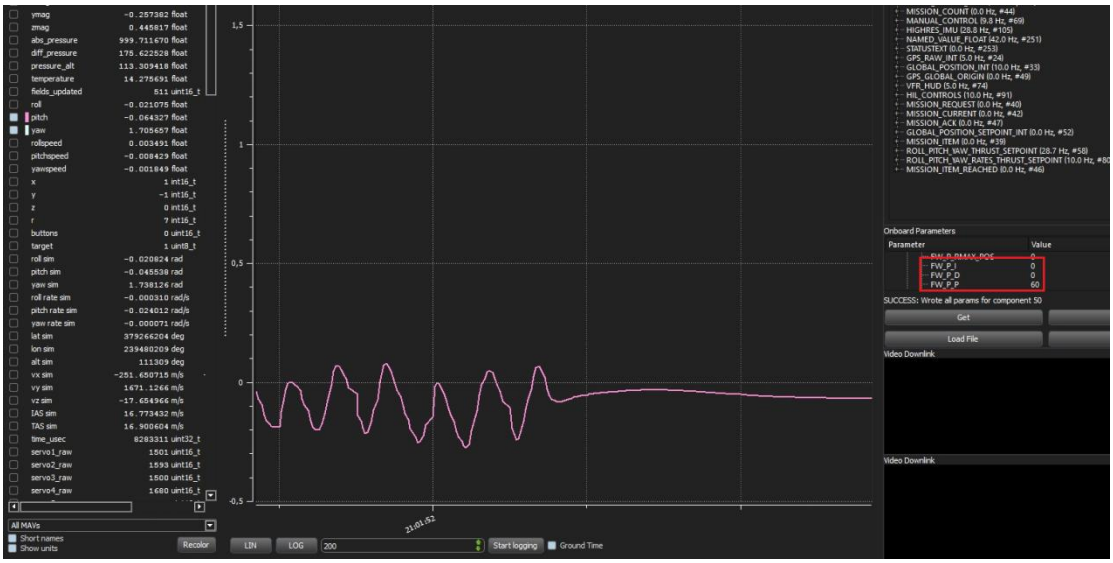
FW_P_I = 0 (fixedwing_ pitch _I)

FW_P_D = 0 (fixedwing_ pitch _D)

Το setpoint του ύψους που έχει επιλεγεί είναι 112m.

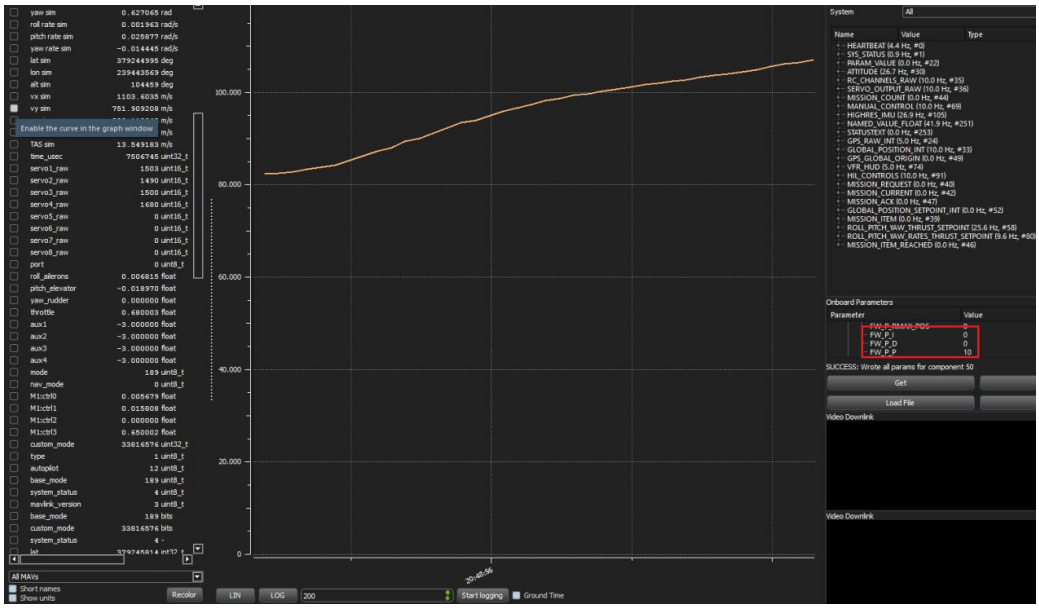


20-17. Γραφική παράσταση ύψους με $k_p = 60$, $k_i = 0$, $k_d = 0$.



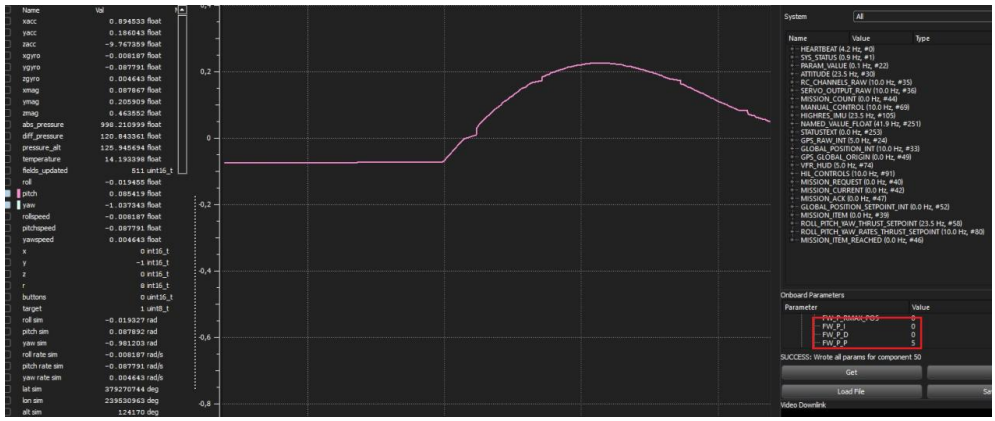
20-18. Γωνία pitch με $k_p = 60$, $k_i = 0$, $k_d = 0$.

Παρατηρούμε ότι με τις βέλτιστες συνθήκες ρύθμισης του ελεγκτή το επιθυμητό ύψος διατηρείται σταθερό και η τιμή της γωνίας pitch είναι στατιστικά σταθμισμένη γύρω από το μηδέν με πολύ μικρή διασπορά.



20-19. Γραφική παράσταση ύψους με $k_p = 10$, $k_i = 0$, $k_d = 0$.

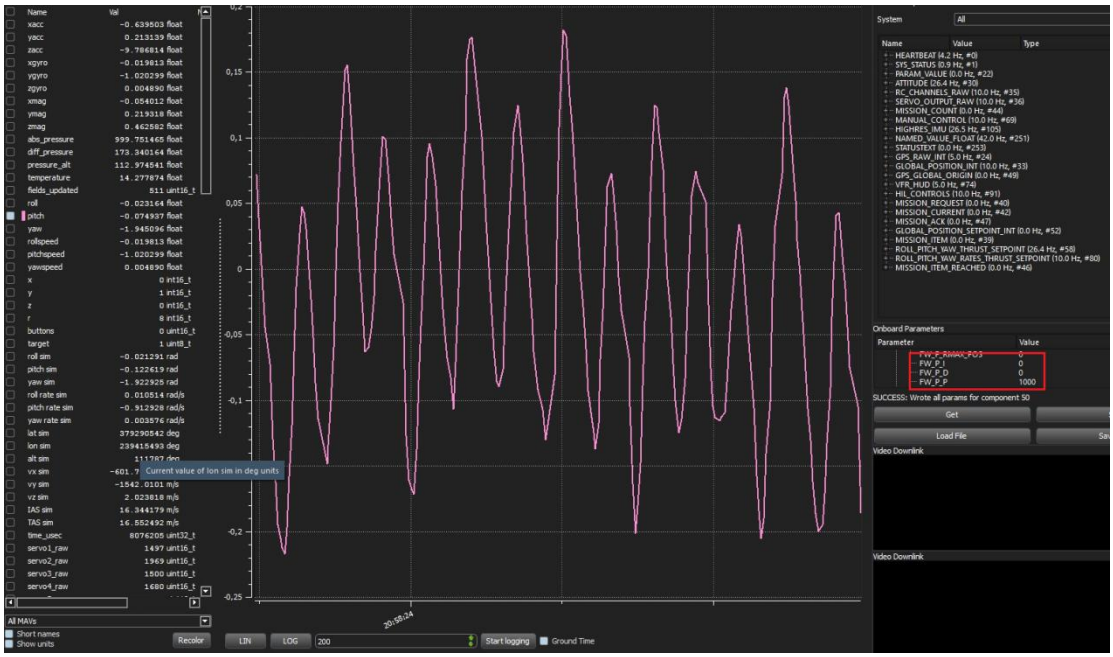
Όπως φαίνεται στην εικόνα 19-19 η μεταβολή του k_p από 60 σε 10, προκαλεί υπερύψωση του σκάφους πάνω από το επιθυμητό ύψος.



20-20. Γωνία pitch με $k_p = 5$, $k_i = 0$, $k_d = 0$.

Μεταβάλλοντας το k_p από 60 σε 5, παρατηρούμε τις αλλαγές στην τιμή της γωνίας pitch.

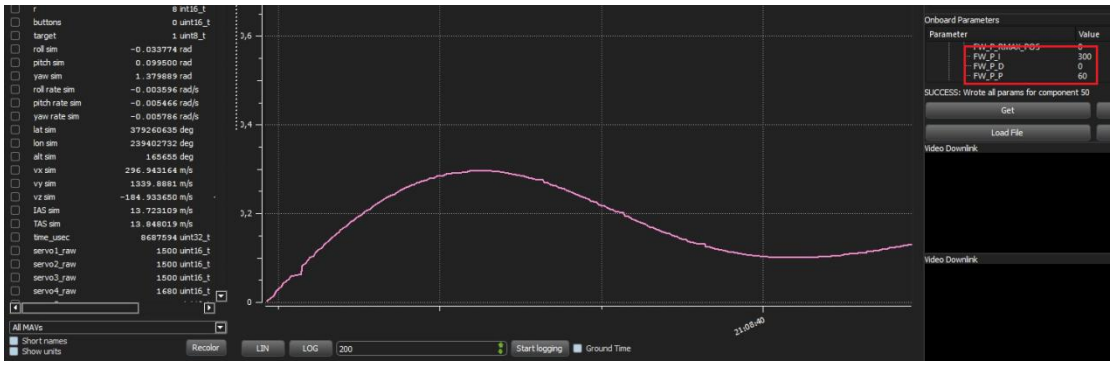
Στη συνέχεια δοκιμάζουμε τιμές για το k_p μεγαλύτερες από το 60.



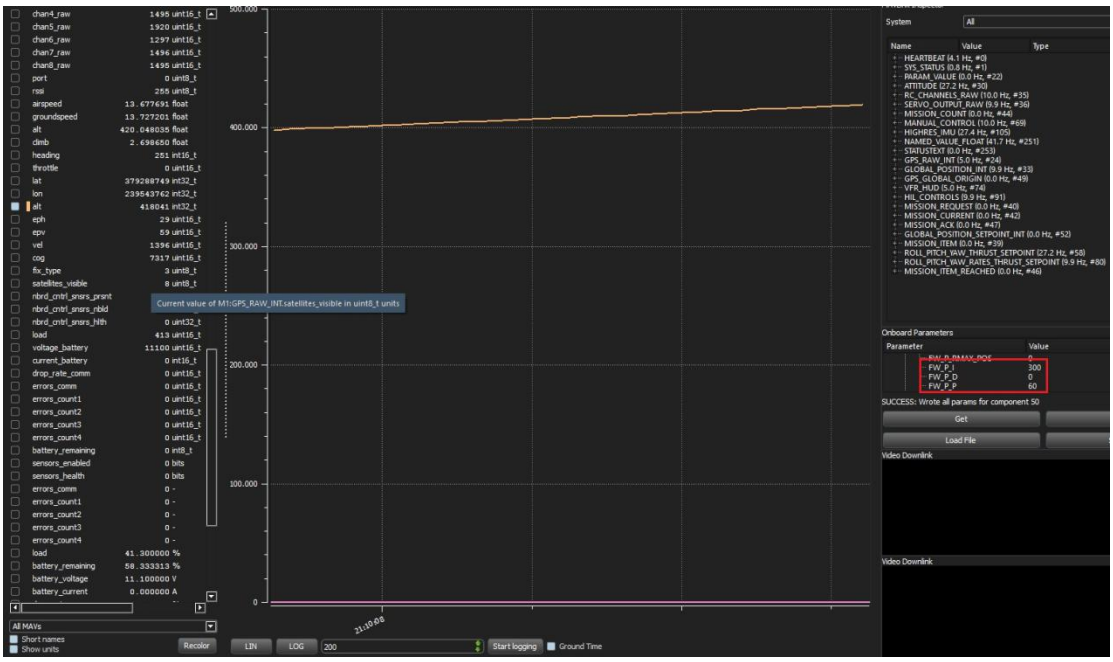
20-21. Γωνία pitch με $k_p = 1000$, $k_i = 0$, $k_d = 0$.

Παρατηρούμε την αστάθεια του συστήματος μέσω της γραφικής παράστασης της γωνίας pitch όπου φαίνονται οι συνεχείς μεταβολές της.

Τέλος δίνουμε μια πολύ μεγάλη τιμή στο μέλος I, το οποίο έχει ως αποτέλεσμα τον αποπροσανατολισμό του αεροσκάφους.

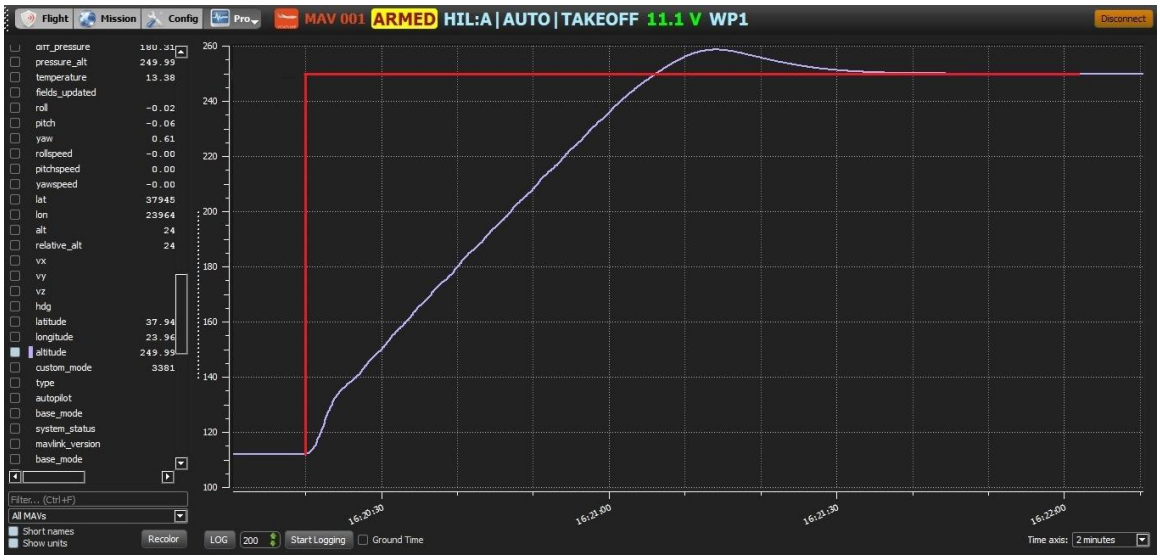


20-22. Γωνία pitch με $k_p = 60$, $k_i = 300$, $k_d = 0$.



20-23. Γραφική παράσταση ύψους με $k_p = 60$, $k_i = 300$, $k_d = 0$.

Μεταβάλλοντας την τιμή της παραμέτρου k_i σε 300 παρατηρούμε τη γωνία pitch η οποία αποκλίνει από την επιθυμητή τιμή. Το ύψος , ενώ έχουμε θέσει το setpoint στα 112m, βλέπουμε ότι έχει ξεπεράσει τα 400m λόγω της εσφαλμένης παραμετροποίησης του ελεγκτή.



20-24. Αλλαγή ύψους από 112m στα 250m.

Επιπλέον παρατίθεται η γραφική παράσταση, που αναπαριστά την μεταβολή του ύψους του σκάφους στον χρόνο (το νέο setpoint στα 250m) με σκοπό την αξιολόγηση του ελεγκτή. Παρατηρούμε ότι ο ελεγκτής είναι υψηλής ποιότητας, δεδομένου ότι παρουσιάζει μικρή υπερύψωση - μικρότερη του 4% -, έχει μέγιστη ακρίβεια, στην αποκατάσταση το μόνιμο σφάλμα είναι μηδέν, και τέλος έχει μεγάλη ταχύτητα, δεν υπάρχει δηλαδή ουσιαστική καθυστέρηση από την στιγμή ορισμού του setpoint μεχρι και την ενεργοποίηση του ελεγκτή.

ΣΥΜΠΕΡΑΣΜΑΤΑ – ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Η ολοκλήρωση της εργασίας μάς οδήγησε σε μια σειρά από συμπεράσματα σχετικά με τον έλεγχο Μη Επανδρωμένων Αεροσκαφών σε επίπεδο MAV. Κρίνεται σκόπιμο να τονιστεί και στο σημείο αυτό ότι η πλατφόρμα PX4fmu του Πανεπιστήμιου ΕΤΗ, που επιλέχθηκε από την ομάδα μελέτης έχει αποδειχθεί η καταλληλότερη για τον έλεγχο UAV. Η άρτια δομή της εν λόγω πλατφόρμας έχει καταρχήν έναν επεξεργαστή σύγχρονης αρχιτεκτονικής μεγάλης ταχύτητας υπολογισμών, με πλήρως εμπλουτισμένο παρεχόμενο software σε επίπεδο βιβλιοθηκών και επιπρόσθετα παρέχει την δυνατότητα σε έναν χρήστη με γνώσεις προγραμματισμού να επέμβει στο λογισμικό, ώστε να το μορφοποιήσει και να το επεκτείνει.

Επιπλέον, δεδομένου ότι ένα κλειστό σύστημα ελέγχου απαιτεί μεγάλη ακρίβεια στις μετρήσεις-εκτιμήσεις των μεγεθών του ολοκληρώνεται μόνο με την χρήση φίλτρων Kalman, επιβεβαιώνεται η πληρότητα της εν λόγω πλατφόρμας, μιας και παρέχεται από αυτήν η συγκεκριμένη δυνατότητα. Παρότι ο έλεγχος μέσω ενός PID ελεγκτή είναι επαρκής για την ολοκλήρωση αποστολών σε περιβάλλον με ήπιες εξωγενείς συνθήκες πτήσης, πρέπει να αναφερθεί πώς για τις αντίστοιχες συνθήκες με υψηλότερο βαθμό δυσκολίας πτήσης και πολυπλοκότερα σκάφη, είναι καταλληλότερες μορφές ελέγχου οι αυτοπροσαρμοζόμενοι και οι ασαφείς ελεγκτές, καθώς και τα νευρωνικά δίκτυα.

Στο πλαίσιο της μελέτης αυτής μας δόθηκε η δυνατότητα να δοκιμάσουμε και τελικά να επιβεβαιώσουμε, μέσω πειραματικών πτήσεων στον εξομοιωτή (X-Plane), τις γνώσεις μας πάνω στη θεωρία ελέγχου και επιπλέον να διαπιστώσουμε τις επιπτώσεις, που μπορεί να έχει σε μια πτήση ενός αεροσκάφους, η λανθασμένη παραμετροποίηση του ελεγκτή. Επιβεβαιώνεται ότι η απόκλιση του όρου P από την ιδανική τιμή προκαλεί την αδυναμία του σκάφους να ακολουθήσει την βέλτιστη διαδρομή πτήσης. Όσο η απόκλιση αυτή αυξάνει, τόσο λιγότερο προσεγγίζει η πορεία του σκάφους, την επιθυμητή. Για τον όρο I διαπιστώθηκε, πώς η μη ιδανική βαθμονόμηση του, δεν

επιτρέπει τον μηδενισμό του μόνιμου σφάλματος. Τέλος, λόγω των τεχνικών χαρακτηριστικών του αεροσκάφους πειραματισμού και της αργής απόκρισης του σε μεταβολές κίνησης, δεν είναι απαραίτητος ο έλεγχος διαφόρισης.

Το υλικό που προκύπτει με αφορμή την παρούσα μελέτη για μελλοντικές έρευνες είναι ποικίλο και με δυνατότητες ανάπτυξης σε πολλαπλά επίπεδα. Σε ένα πρώτο στάδιο και άμεσα εξαρτώμενο από τα έως τώρα αποτελέσματα είναι η έρευνα σχετικά με την αντικατάσταση του PID ελεγκτή με διαφορετικές μεθόδους ελέγχου, όπως με έναν αυτοπροσαρμοζόμενο ή ασαφή ελεγκτή, καθώς και νευρωνικά δίκτυα.

Ένα διαφορετικό πεδίο διερεύνησης θα μπορούσε να αποτελέσει η δοκιμή εγκατάστασης κάμερας σε επικοινωνία με τον μικροελεγκτή, ώστε να πραγματοποιείται μέσω αυτής αποστολές όπως η αναγνώριση προσώπων, η δυνατότητα εναέριας φωτογράφησης, ο έλεγχος κυκλοφορίας οχημάτων, η χαρτογράφηση του εδάφους κ.α. Επιπλέον ένα Μη Επανδρωμένο Αεροσκάφος θα μπορούσε να χρησιμοποιηθεί για τον εντοπισμό πυρκαγιών, για την συλλογή δεδομένων από δυσπρόσιτες για γεωλογικούς λόγους περιοχές, όπως για παράδειγμα ανενεργά ή κυρίως ενεργά ηφαίστεια.

Μεγαλεπήβολο αλλά όχι ανέφικτο είναι το πλάνο μελέτης και δημιουργίας ενός σμήνους UAV με βάση την μεταξύ τους επικοινωνία και την πλοήγηση τους στον χώρο. Η επίτευξη της εργασίας αυτής θα μπορεί να αποδώσει μεγαλύτερη ακρίβεια και ταχύτερη επεξεργασία των πληροφοριών, καθώς και ταχύτερη κάλυψη μεγαλύτερων επιφανειών, σε αντίστοιχες με τις προηγούμενες αποστολές.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ξένη Βιβλιογραφία

1. Randal W. Beard, Timothy W. McLain, *Small Unmanned Aircraft Theory and Practice* Princeton University Press, 2012.
2. Robert C. Nelson, *Flight Stability and Automatic Control*, WCB/McGraw-Hill, 1998 2nd Edition.
3. Michael J. Caruso, *Applications of Magnetoresistive Sensors in Navigation Systems*, Honeywell Inc., 1997.
4. Dmitry Budker, Michael Romalis, *Optical Magnetometry*, Vol. 3 (Department of Physics, University of California, Nuclear Science Division, Lawrence Berkley National Laboratory, Department of Physics, Princeton University), 2007.
5. Pierre-Jean Bristeau, Francois Callou, David Vissiere, Nicolas Petit, "*The Naavigation and Control technology iside the AR.Drone micro UAV*", in Proc. 18th IFAC World Congress Milano, 2011.
6. HaiYang Chao, YongCan Cao, YangQuan Chen, "*Autopilots for Small Unmanned Vehicles: A Survey*", International Journal of Control Automation and Systems, 2010.
7. Randal Beard, Derek Kingston, Morgan Quigley, Deryl Snyder, Reed Cristiansen, Walt Johnson, Timothy McLain, Michael A. Goodrich, *"Autonomous Vehicle Technologies for Small Fixed-Wing UAVs"*, Journal of Aerospace Computing, Information and Communication, Vol.2, 2005.
8. Christopher Paulson, Andras Sobester, *Bespoke Balloon Launched Sensorcraft for Atmospheric Research Missions*, American Institute of Aeronautics and Astronautics.

9. Brett B., Frank A., Dale D., Vian J. *"Real-Time Indoor Autonomous Vehicle Test Enviroment"* Control Systems IEEE, Vol. 28, Issue. 2, April 2008.
10. Brett Bethke, *Persistent Vision-Based Search and Track Using Multiple UAVs*, Massachusetts Institute of Technology, 2007.
11. David R. Gordon, *A Small Unmanned Aerial Vehicle for Military and Civilian Applications*, David R. Gordon, 1997.
12. Jung Soon Jang, Darren Liccardo, *Automation of Small UAVs Using A Low Cost Mems Sensor and Embedded Computing Platform*, Crossbow Technology Inc.
13. Dominik Honegger, Lorenz Meier, Petri Tanskanen, Mark Pollefeys, *An Open Source and Open Hardware Embedded Metric Optical Flow CMOS Camera for Indoor and Outdoor Applications*, ETH Zurich.
14. Greg Welch, Gary Bishop, *An Introduction to the Kalman Filter*, University of North Carolina at Chapel Hill, 2001.
15. Anderson J. David, *Introduction to Flight*, McGraw-Hill, 1989.
16. Blakelock H. John, *Automatic Control of Aircraft and Missiles*, John Wiley & Sons, 1991.
17. Michael V. Cook, *Flight Dynamics Principles*, John Wiley & Sons, 1997.
18. Pedro Castillo, Rogelio Lozano, Alejandro E. Dzul, *Modeling and Control of Mini-Flying Machines*, Springer, 2005.
19. Kimon P. Valavanis, *Advances in Unmanned Aerial Vehicles,* Springer, 2007.
20. N. Knoebel, S. Osborne, D. Snyder, T. McLain, R. Beard, A Eldredge, *"Preliminary modeling, control and trajectory desing for miniature autonomous tailsitters"* in Prac. AIAA Guidance, Navigation, Control Conf. Exhibit, 2006.

21. Air Force Study Board, *Automation in Combat Aircraft*, National Academy Press, Washington DC, 1982.
22. National Aeronautics and Space Administration, *"Exploring in Aeronautics"*, Washington DC, 1971.

Ελληνική Βιβλιογραφία

23. Κωστής Καραγεώργης, Κωνσταντίνος Πατούρας, *Θεωρία Πτήσης, Επιδόσεις, Τύποι, Όργανα και Συστήματα Αεροσκαφών*, Mark Garfinkel, 2001.
24. Χρήστος Κατσίκας, *Αισθητήρας Μαγνητικού Πεδίου Βασισμένος στο Φαινόμενο Μαγνητοσυστολής*, ΕΜΠ, 2010.
25. Αθανάσιος Γκέγκας, *Δομές Ολοκληρωμένου Δορυφορικού και Αδρανειακού Συστήματος Πλοήγησης*, ΕΜΠ, 2008.
26. Λιακόπουλος Δημοσθένης, *Η Τεχνολογία του Κακού*, Λιακόπουλος ΕΛ, 2007.
27. Χρήστος Δρόσος, *Αποτίμηση Τεχνολογικών Μεθόδων και Αλγορίθμων Ελέγχου Μη Επανδρωμένων Αεροσκαφών*, Πανεπιστήμιο Πειραιώς, 2011.

Ιστότοποι

28. https://pixhawk.thz.ch/px4/en/start
29. http://ctms.engin.umich.edu/CTMS/index.php?aux=Home
30. http://www.princeton.edu/~stengel/MAE331.html
31. www.diydrones.com
32. www.starlino.com
33. www.x-plane.com

34. www.wikipedia.org
35. www.boeing.com/commercial/aeromagazine
36. http://forums.x-plane.org
37. http://linux.about.com
38. http://plane.ardupilot.com/wiki/common-load-firmware-px4
39. http://groups.google.com/forum/#forum/px4users
40. http://nuttx.org/doku.php?id=documentation:nuttshell
41. http://qgroundcontrol.org
42. http://paparazzi.enac.fr/wiki/Main Page
43. http://mathworld.wolfram.com/EulerAngles.html
44. http://bilgin.emse.org/BotsBytes/KalmanFilterforDummies.aspx
45. http://www.fastgraph/makegames/3drotation
46. http://www.cplusplus.com

ΠΑΡΑΡΤΗΜΑ

PX4FMU – Flight Management Unit

QUICK START – HARDWARE VERSION 1.7

Description

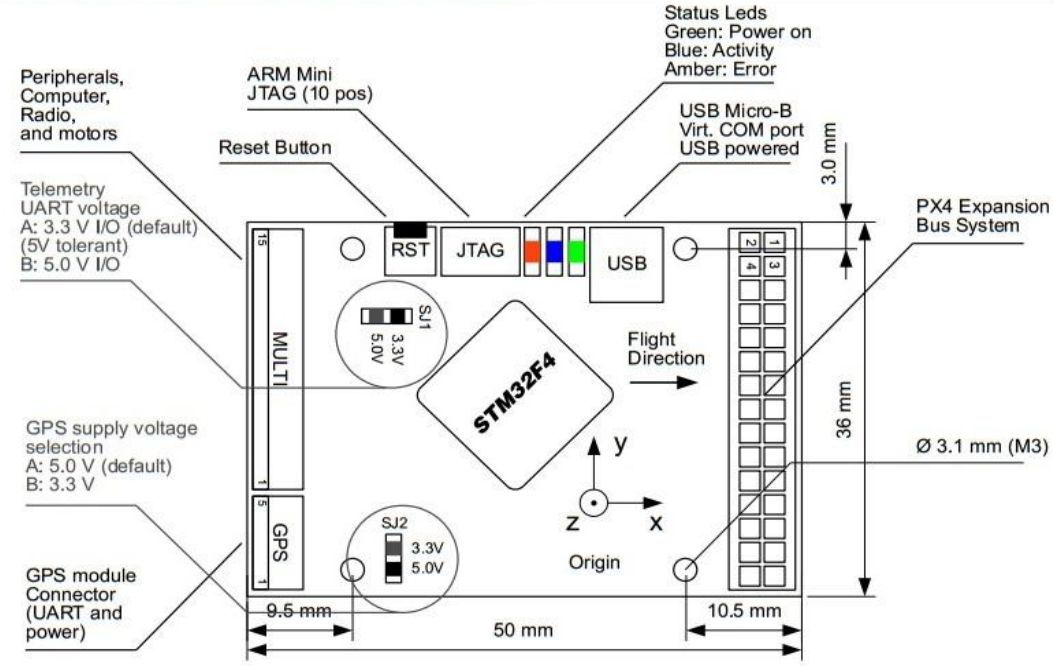
PX4FMU is an onboard management unit for micro air vehicles. It combines an autopilot and inertial measurement unit and enables the control of an aircraft using a single-board solution. Additional I/O can be easily connected via the 30-pin expansion bus.

http://pixhawk.ethz.ch/px4/

Features

- 168 Mhz Cortex-M4 CPU (196 KB RAM, 1 MB Flash)
- 250 mW typical power consumption
- Reverse polarity protection on all power inputs
- 3D gyro, accelerometer and magnetometer, pressure sensors
- I2C, 3x UART, PPM, analog, GPS, 2x 5V GPIO, 4x PWM / Servo
- MicroSD card slot
- Expansion bus: CAN, 2x I2C, SPI, 4x analog, 2x UART, GPIOs
- USB Serial Port (Virtual COM Port / VCP) and bootloader
- 50 x 36 x 6 mm (1.38x1.97x0.24"), 8g, 30x30 mm mounting holes
- 4.5-6 V wide supply input range (incl. USB power)
- Selectable 3.3 V or 5 V IO for UART2 and GPS ports

Connectors, Jumpers and Dimensions



Pinout and absolute maximum Ratings

- Input: 4.3-6 V (VDD_5V), 20 mA onboard use, max. 800 mA for max peripheral load. Reverse-polarity protected.
- Output: 3.3 V (VDD_3V3), fuse-limited 500 mA EXT, 3.3 V, fuse-limited 200 mA GPS

GPS

5	GND
4	NOT CONNECTED
3	USART6_RX
2	USART6_TX
1	VDD_GPS (5V default)

Males housing: Hirose DF13 "DF13-5S-1.25C", contacts: "DF13-2630SCF"; AWG 26-30

VDD_5V	1 2	VDD_5V	
GND	3 4	GND	
CAN2_RX	5 6	CAN2_TX	
USART1_RX	7 8	USART1_TX	
I2C3_SDA	9 10	I2C3_SCL	
-	11 12	ADC123_IN10	
UART6_RX	13 14	UART6_TX	
UART5_RX	15 16	UART5_TX	
I2C2_SDA	17 18	I2C2_SCL	
USART2_RTS	19 20	USART2_CTS	
USART2_RX	21 22	USART2_TX	
GPIO_EXT1	23 24	PPM_INPUT	
BUZZER	25 26	GPIO_EXT2	
ADC123_IN11	27 28	GND	
ADC123_IN13	29 30	ADC123_IN12	

Males 2 mm header: 3M "951230-2520-AR-PR"

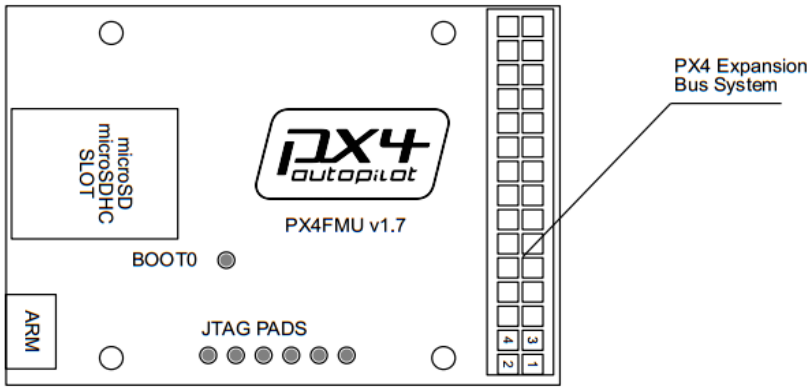
MULTI

15	GND
14	I2C1_SCL
13	I2C1_SDA
12	USART2_TX / SRV3 / AR.TX
11	USART2_CTS / SRV1 / AR.S4
10	USART2_RTS / SRV2 / AR.S3
9	UART2_RX / SRV4 / AR.RX
8	USART1_TX
7	USART1_RX
6	PPM_INPUT (3-5V)
5	BATTERY_MONITOR (3-18V)
4	GPIO_EXT2 / AR.S2
3	GPIO_EXT1 / AR.S1
2	VDD_3V3
1	VDD_5V

Males housing: Hirose DF13 "DF13-15S-1.25C", contacts: "DF13-2630SCF"; AWG 26-30

Additional connectors (bottom side)

The footprints on the bottom side of the connector can be used by advanced users to interface additional boards or sensors.



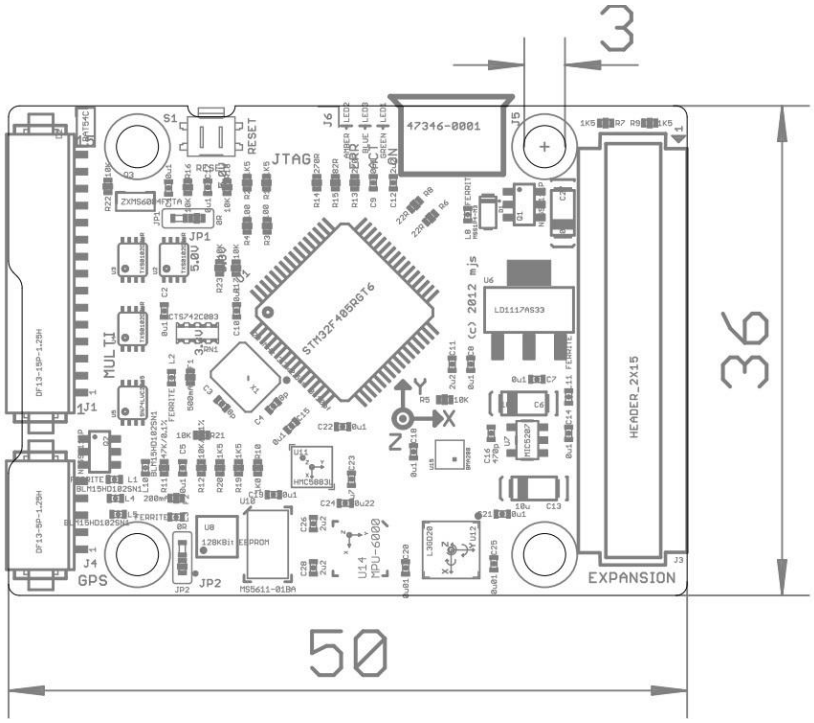
Software Tools / Getting Started

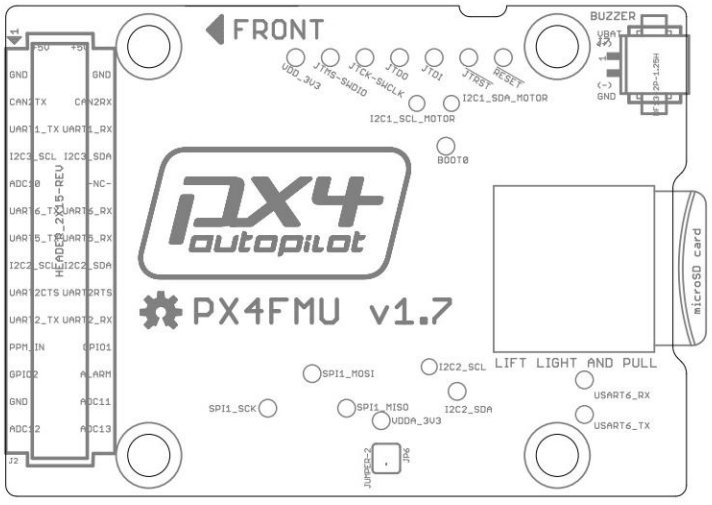
Please check the most recent user manual at https://pixhawk.ethz.ch/px4/users/

Upgrading Firmware / Developing Custom Code

Please check the most recent developer instructions at https://pixhawk.ethz.ch/px4/dev/

Open Hardware License


3
36
50
47346-0001
STM32F405RGT6
LD1117AS33
HEADER_2X15
EXPANSION
GPS
JTAG
RESET
MULTI
MS5611-01BA
MPU-6000


FRONT
BUZZER
VBAT
GND
VDD_3V3
JTMS-SWDIO
JTCK-SWCLK
JTDO
JTDI
JTRST
RESET
I2C1_SDA_MOTOR
I2C1_SCL_MOTOR
BOOT0
PX4
autopilot
PX4FMU v1.7
microSD card
LIFT LIGHT AND PULL
SPI1_MOSI
I2C2_SCL
USART6_RX
SPI1_SCK
SPI1_MISO
I2C2_SDA
VDDA_3V3
USART6_TX
HEADER 2X15-REV
+5V
GND
CAN2TX
CAN2RX
UART4_TX
UART4_RX
I2C3_SCL
I2C3_SDA
ADC10
-NC-
UART6_TX
UART6_RX
UART5_TX
UART5_RX
I2C2_SCL
I2C2_SDA
UART2CTS
UART2RTS
UART2_TX
UART2_RX
PPM_IN
GPIO1
GPIO2
A_ARM
GND
ADC11
ADC12
ADC13

PX4IO – Input / Output and Servo Module

QUICK START – HARDWARE VERSION 1.3

Description

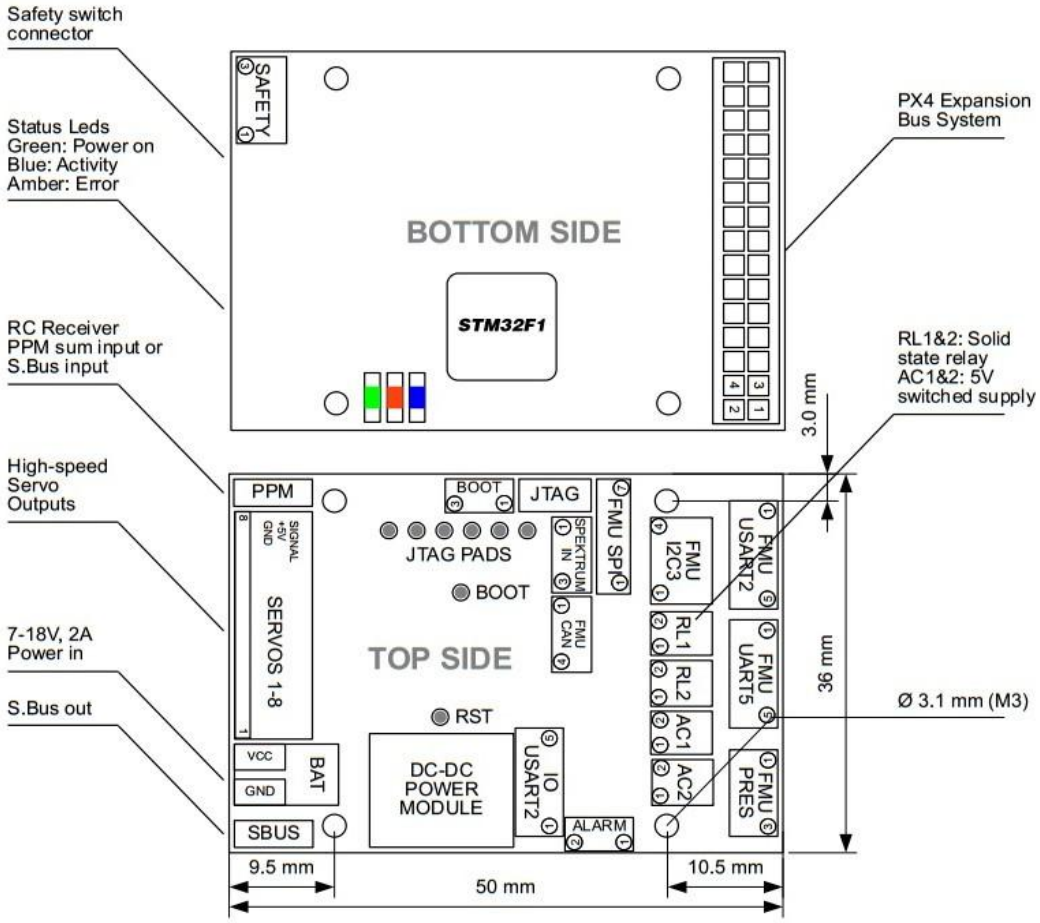
PX4IO is a power supply and expansion module for the PX4FMU f ight management unit. It provides servo outputs, receiver inputs, two solid state relays, two switched and current-limited 5V power outputs and a wide range of additional I/O connectors.

http://github.com/PX4/Hardware

Features

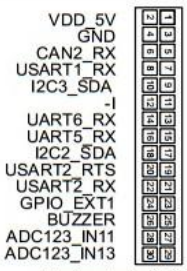
- 24 Mhz Cortex-M3 I/O multiplexer
- 6.3-18V wide supply in, 5V / 2 A output
- Reverse polarity protection on all power inputs
- 8 high-speed servo outputs (up to 400 Hz)
- PPM, Spektrum and Futaba S.Bus compatible receiver inputs
- 2x 0-40 V, 1 A solid-state relays (MOSFET)
- 2x 5 V, 500mA current-limited, switched 5V power outputs
- Analog port with dividers (differential pressure sensors)
- PX4 Expansion bus
- 50x36x14 mm (1.38x1.97x0.55"), 20g, 30x30 mm M3 mounting

Connectors, Jumpers and Dimensions



Pinout and Absolute Maximum Ratings

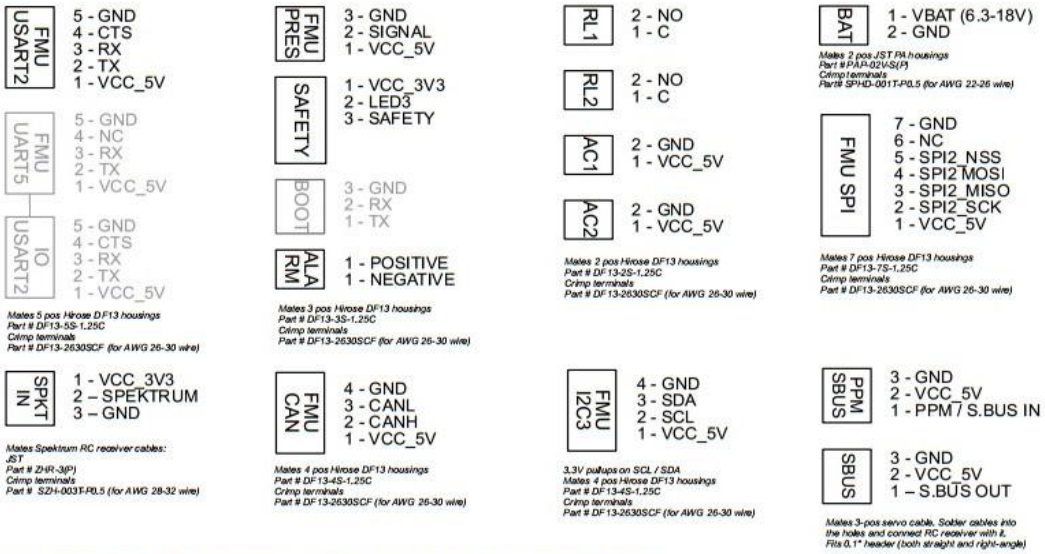
- Input: 6.3-18 V (BAT connector), max current: 2.5 A
- Accessory outputs: 5 V, each 500mA current limited
- Peripherals output: 5 V, 0.5 A current limited
- Servo Output: 5V, 1.0 A current limited
- NOT compatible per default with high-voltage servo systems providing more than 5V on the servo rail (remove L1 for high voltage systems and supply 6.5V on J16)



Mates 2 mm header: 3M "9532230-2000-AR-PR"

Back side of PCB
Mates standard servo plugs

All connectors below are oriented the same way as shown in the overview picture. Check the pin 1 markings in the overview if unsure.
Note: Gray printed connectors are internally connected to other peripherals and can only be used in certain configurations



Upgrading Firmware / Developing Custom Code

PX4IO is designed as a failsafe board with a stable codebase. Building custom firmware is only recommended for very advanced users.
To develop custom code, follow the PX4FMU/PX4IO toolchain guide at: https://pixhawk.ethz.ch/px4/dev/start

Open Hardware License