ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Πρόγραμμα Μεταπτυχιακών Σπουδών

Επιστήμη και Τεχνολογία της Πληροφορικής και των Υπολογιστών
Ειδίκευση Λογισμικού και Πληροφοριακών Συστημάτων

Διπλωματική Εργασία

Ανάπτυξη UAV με χρήση Ardupilot

Φοιτητής: Αντώνιος Σκάρος
ΑΜ: 18044

Εισηγητής: Ιωάννης Βογιατζής, Καθηγητής

Αθήνα - Αιγάλεω, Σεπτέμβριος 2022

University of West Attica

Faculty of Engineering

Department of Informatics and Computer Engineering

MSc Programme

Science and Technology of Informatics and Computers
Specialization Hardware and Informatics Systems

Diploma Thesis

Development of a UAV using Ardupilot

Student: Antonios Skaros
RN: 18044

Supervisor
Ioannis Voyiatzis
Professor

Athens - Egaleo, September 2022

Ανάπτυξη UAV με χρήση Ardupilot

Εξεταστική Επιτροπή

Ioannis Voyiatzis Digitally signed by Ioannis Voyiatzis Date: 2022.10.31 15:26:15 +02'00'	Panagiotis Giannakopoulos Digitally signed by Panagiotis Giannakopoulos Date: 2022.10.31 17:58:46 +02'00'	Panagiotis Karkazis Digitally signed by Panagiotis Karkazis Date: 2022.10.31 16:40:35 +02'00'

Ημερομηνία εξέτασης 30/09/2022

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

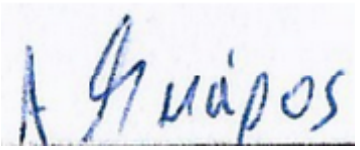
Ο κάτωθι υπογεγραμμένος Σκάρος Αντώνιος του Μιχαήλ, με αριθμό μητρώου 18044 φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής ΜΗΧΑΝΙΚΩΝ του Τμήματος ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ του Προγράμματος Μεταπτυχιακών Σπουδών ΕΠΙΣΤΗΜΗ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑ ΤΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του διπλώματός μου.»

Ο Δηλών



Αντώνιος Σκάρος

Ευχαριστίες

Θα ήθελα να εκφράσω τις ευχαριστίες μου στον επιβλέποντα καθηγητή μου κύριο Ιωάννη Βογιατζή και στον κύριο Δημήτριο Ψιλιά, υποψήφιο διδάκτορα για την βοήθειά τους.
Ακόμη θα ήθελα να ευχαριστήσω την Ευαγγελία Κάρρα, την Μαρία Τσαμπίκα Σκάρου και την Μαρία Θηρεσία Βάθη, συγγενείς και φίλους για την συνεχή στήριξη που μου παρέχουν.

“All we have to decide is what to do with the time that is given to us.”
— Gandalf the Gray

Περίληψη

Η παρούσα διπλωματική εργασία αφορά την ανάπτυξη ενός UAV με χρήση Ardupilot. Γίνεται μια θεωρητική αναφορά στην πλακέτα του Ardupilot, τα στοιχεία που συνθέτουν το UAV μας και την λειτουργία του Ardupilot. Στη συνέχεια γίνεται μια ανάλυση στο Matlab/Simulink και τις χρησιμότητές τους στην προσομοίωση πτήσεων και γενικότερα εφαρμογών της ρομποτικής. Τέλος γίνεται μια μετατροπή ενός παραδείγματος του Matlab του asbQuadcopter. Η μετατροπή αυτή ως στόχο έχει να μας δώσει πληροφορίες σχετικά με την επιλογή του τύπου μοντελοποίησης που θα ακολουθήσουμε. Μετατρέπουμε κάποια Μη Γραμμικά Μοντέλα (Non Linear Models) σε Γραμμικά(Linear) ή τα διαγράφουμε εντελώς και ελέγχουμε πως επηρεάζουν την ευστάθεια του συστήματος. Αυτή η μετατροπή μας δίνει πληροφορίες αν το σύστημα μπορεί να λειτουργήσει με “χειροκίνητες” αλλαγές ή είναι άρρηκτα συνδεδεμένο με αυτά τα Μη Γραμμικά Μοντέλα.

Λέξεις-Κλειδιά

Μη Επανδρωμένο Όχημα, Μη Επανδρωμένο Ιπτάμενο Όχημα, Τετρακόπτερο, Προσομοίωση πτήσης, Πλακέτα Ardupilot, Προσομοίωση Ρομποτικών συστημάτων,

Πίνακας με συντομογραφίες

MEA - Μη Επανδρωμένα Αεροσκάφη
UAV - Unmanned Aerial Vehicles (Μη Επανδρωμένα Ιπτάμενα Οχήματα)
SITL - Software In The Loop
GCS - Ground Control Station (Επίγειος Σταθμός Ελέγχου)
FPV - First Person View (Όψη Πρώτου Προσώπου)
RTL - Ready To Launch (Έτοιμο προς Απογείωση)
ASL - Above Sea Level (Πάνω από την επιφάνεια του νερού)
AGL - Above Ground Level (Πάνω από την επιφάνεια του εδάφους)
IMU - Inertial Measurement Unit (Αδρανειακή Μονάδα Μέτρησης)
EKF - Extended Kalman Filter
DCM - Direction Cosine Matrix
AHRS - Attitude Heading Reference System
UART - Universal Asynchronous Receiver Transmitter
OSD - On Screen Display
HIL - Hardware In the Loop
FFT - Fast Fourier Transmitter
DTED - Digital Transmitter Elevation Data
STL - Standard Triangle Language
FCS - Flight Control System
UDP - User Datagram Protocol
PID - Proportional Integral Derivative
FDM - Fused Deposition Modeling

Abstract

The following Master Thesis is a development of a UAV using Ardupilot. In the first part there is a theoretical reference to the Ardupilot Board, the components that UAV includes and the operation of the Ardupilot. In the second part there is an analysis in Matlab/Simulink and their usage in flight simulation and in robotics applications in general. In the final part a Matlab example of asbQuadcopter is converted. This conversion aims to give us information about which type of modeling to follow. We convert some Non Linear Models to Linear or delete them completely and check how they affect the stability of the system. This conversion helps us come to a conclusion if the system can work with "manual" changes or if it is inextricably linked to these Nonlinear Models.

Keywords

UAV, Matlab, Simulink, Ardupilot, Simulation, Quadcopter, Drone,

Acronyms

UAV - Unmanned Aerial Vehicles
SITL - Software In The Loop
GCS - Ground Control Station
FPV - First Person View
RTL - Ready To Launch
ASL - Above Sea Level
AGL - Above Ground Level
IMU - Inertial Measurement Unit
EKF - Extended Kalman Filter
DCM - Direction Cosine Matrix
AHRS - Attitude Heading Reference System
UART - Universal Asynchronous Receiver Transmitter
OSD - On Screen Display
HIL - Hardware In the Loop
FFT - Fast Fourier Transmitter
DTED - Digital Transmitter Elevation Data
STL - Standard Triangle Language
FCS - Flight Control System

ΠΕΡΙΕΧΟΜΕΝΑ

Κεφάλαιο 1: Εισαγωγή

1.1 Αντικείμενο της διπλωματικής εργασίας

Στην παρούσα διπλωματική εργασία γίνεται θεωρητική ανάλυση της αυτόνομης πτήσης των μη επανδρωμένων ιπτάμενων οχημάτων. Αρχικά γίνεται μια σύντομη ιστορική αναδρομή στην πορεία τους. Αναφέρονται ενδεικτικά τα πρώιμα στάδια των UAVs και η πορεία τους μέχρι το σημερινό στάδιο.

Στην συνέχεια αναλύουμε το Matlab - Simulink και τον τρόπο που μπορούμε να το χρησιμοποιήσουμε σε προσομοιώσεις. Αναφέρονται τα πακέτα του Matlab που έχουν χρησιμοποιηθεί καθώς και μια αναφορά στις δυνατότητες που μας δίνει σαν πρόγραμμα σε σχέση με άλλα προγράμματα προσομοίωσης.

Τέλος δίνεται μια πρακτική εικόνα του asbQuadcopter του Matlab και πως μπορούμε να παρέμβουμε στον κώδικα του Simulink προκειμένου να δημιουργήσουμε τα δικά μας μοντέλα συμπεριφοράς του UAV.

1.2 Εισαγωγή - Ιστορικά

Ως μη επανδρωμένα αεροσκάφη ονομάζονται τα ιπτάμενα οχήματα τα οποία δεν περιέχουν κάποιον οδηγό ή χειριστή εντός της ατράκτου τους. Αντ'αυτού η κίνησή τους και η πορεία τους καθορίζεται από κάποιο τηλεχειριστήριο είτε αυτόματα. Άλλες συνήθεις ονομασίες με τις οποίες αναφερόμαστε σε αυτά είναι UAV από το ακρωνύμιο του Unmanned Aerial Vehicle, UAS από το Unmanned Aerial System και RPAS από το Remotely Piloted Aircraft Systems.

Στην καθημερινότητα όμως χρησιμοποιείται ο όρος drones. Ο όρος UAS αναφέρεται στο σύνολο των περιφερειακών συσκευών, των ατόμων που δουλεύουν πάνω σε αυτό καθώς και των διαδικασιών που συνδυάζονται ώστε να θεωρηθεί το UAV ένα ολοκληρωμένο σύστημα. Ο όρος RPAS καθιερώθηκε και νομοθετικά, ώστε για λόγους ασφαλείας να υπάρχει τουλάχιστον ένας "πιλότος" στο έδαφος προκειμένου να επιβλέπει την πτήση. Στο σύνολό τους τα μη επανδρωμένα ιπτάμενα οχήματα έχουν συνήθως μορφή παρόμοια με αυτή ενός μικρού αεροπλάνου, ελικοπτέρου ή ακόμη συνηθέστερα μια μορφή που ομοιάζει του ελικοπτέρου με ένα πλήθος ελίκων συνήθως 4,6 ή και 8.

1.2.1 Αναφορές και συντομεύσεις

Για την εύκολη ανάγνωση της διπλωματικής έχουν χρησιμοποιηθεί πολλές συντομεύσεις.

- Μη Επανδρωμένα Αεροσκάφη - MEA
- Unmanned Aerial Vehicle - UAV
- Ground Control Station - GCS
- Flight Control System - FCS

Εάν χρησιμοποιηθούν παρακάτω κάποιες συντομεύσεις ή και αγγλικοί όροι τότε έχουμε προηγουμένως ορίσει την σημασία τους.
Τα μη επανδρωμένα αεροσκάφη χωρίζονται σε 2 βασικές κατηγορίες:

1. Με σταθερές πτέρυγες, όπως για παράδειγμα αυτά που ομοιάζουν με τα αεροπλάνα
2. Με έλικες όπως τα ελικόπτερα

Επειδή αρκετές φορές τα drones έχουν περισσότερους έλικες χρησιμοποιείται ο όρος multicopters ή multirotors. Έτσι χρησιμοποιούνται όροι όπως quadcopter (4), hexacopter (6), octacopters (8) κ.ο.κ.

1.2.2 Ιστορία των Μη Επανδρωμένων Αεροσκαφών

Τα ΜΕΑ μετρούν πολλά χρόνια χρήσης τόσο σε στρατιωτικές επιχειρήσεις όσο και σε υπηρεσίες της καθημερινής μας ζωής. Στο σύνολο των ΜΕΑ περιλαμβάνονται εξ ολοκλήρου αυτόνομα οχήματα και οχήματα τηλεχειριζόμενα από το έδαφος. Έχουν την δυνατότητα να πραγματοποιήσουν σταθερή και ελεγχόμενη πτήση, έχουν ένα σαφές σχέδιο πτήσης στο οποίο ορίζεται το σημείο εκκίνησης η πορεία που θα ακολουθήσουν και τέλος το σημείο στο οποίο θα προσγειωθούν ολοκληρώνοντας την πτήση τους. Η κίνησή τους βασίζεται στους κινητήρες η επιλογή των οποίων γίνεται από μια ποικιλία κινητήρων με βάση τα συνολικά χαρακτηριστικά του ΜΕΑ, όπως για παράδειγμα το ύψος της πτήσης που θέλουμε να πετύχουμε, το μέγεθος του σκελετού και το συνολικό βάρος που θα έχει το ΜΕΑ μας. Τα ΜΕΑ λοιπόν διαφέρουν από κατευθυνόμενους πυραύλους. Παρότι και τα 2 είναι μη επανδρωμένα οχήματα κατασκευασμένα και προγραμματισμένα για να επιτύχουν έναν σκοπό τα UAVs έχουν την διαφορά ότι σαν βασικό στόχο έχουν την δυνατότητα χρήσης τους και σε άλλες αποστολές.
Ιστορικά η χρήση των ΜΕΑ μπορούμε να πούμε ότι χωρίζεται σε 2 περιόδους. Σε αυτήν κατά την οποία έχουν μια κίνηση που βασίζεται αποκλειστικά σε χρήση ως επαναχρησιμοποιούμενα εργαλεία και στην παλαιότερη που είχαν κατά βάση περιορισμένη διάρκεια ζωής.

Ως μια πρώτη εμφάνιση των ΜΕΑ μπορούμε να θεωρήσουμε τα μη επανδρωμένα αερόστατα με τα οποία επιτέθηκαν οι Αυστριακοί στην Βενετία στις 22 Αυγούστου του 1849. Τα αερόστατα αυτά μετέφεραν εκρηκτικά και εκτοξεύτηκαν από το πλοίο Volcano. Ως στόχο είχαν ασφαλώς τον βομβαρδισμό της πόλης, με την ρίψη των εκρηκτικών από τα αερόστατα μέσω της εκπυρσοκρότησής τους. Είχαν ένα σύστημα με βάση τον ηλεκτρομαγνητισμό, ενός απομονωμένου σύρματος χαλκού και μιας γαλβανικής μπαταρίας τοποθετημένης στην κορυφή ενός κτιρίου. Παρ'όλα αυτά το σχέδιό τους δεν στέφθηκε με απόλυτη επιτυχία, καθώς μερικά από αυτά επέστρεψαν στις γραμμές των Αυστριακών εξαιτίας του ισχυρού ανέμου. Αυτά τα συστήματα δεν εμπίπτουν πλέον στην κατηγορία των ΜΕΑ.

Α' Παγκόσμιος Πόλεμος

Η ανάπτυξη των ΜΕΑ ξεκίνησε προς την λήξη του Α' Παγκοσμίου Πολέμου με πολυ πρωταρχικό παράδειγμα την κατασκευή του *Ruston Proctor Aerial Target* με τεχνικές ραδιοελέγχου. Λίγο καιρό μετά ακολούθησε η κατασκευή του "Hewitt-Sperry Αυτόματου Αεροπλάνου" όπως ονομαζόταν. Είχε το προσωνύμιο ιπτάμενη βόμβα και κατάφερε να ολοκληρώσει μια αυτόνομη πτήση. Εν γένει η ανάπτυξη ΜΕΑ της εποχής μοιάζει περισσότερο με τους σημερινούς τηλεκατευθυνόμενους πυραύλους παρά με τα σημερινά ΜΕΑ.

Β' Παγκόσμιος Πόλεμος

Το 1940 ο Ντένι ξεκίνησε την εταιρεία Radioplane και περισσότερα μοντέλα εμφανίστηκαν κατά την διάρκεια του Β 'Παγκοσμίου Πολέμου - που χρησιμοποιήθηκαν τόσο για την εκπαίδευση αντιαεροπορικών πυροβολητών όσο και για πτήσεις επιθετικών αποστολών. Η ναζιστική Γερμανία παρήγαγε και χρησιμοποίησε διάφορα αεροσκάφη UAV κατά την διάρκεια του πολέμου, όπως το Argus As 292 και την ιπτάμενη βόμβα V-1 με κινητήρα τζετ.

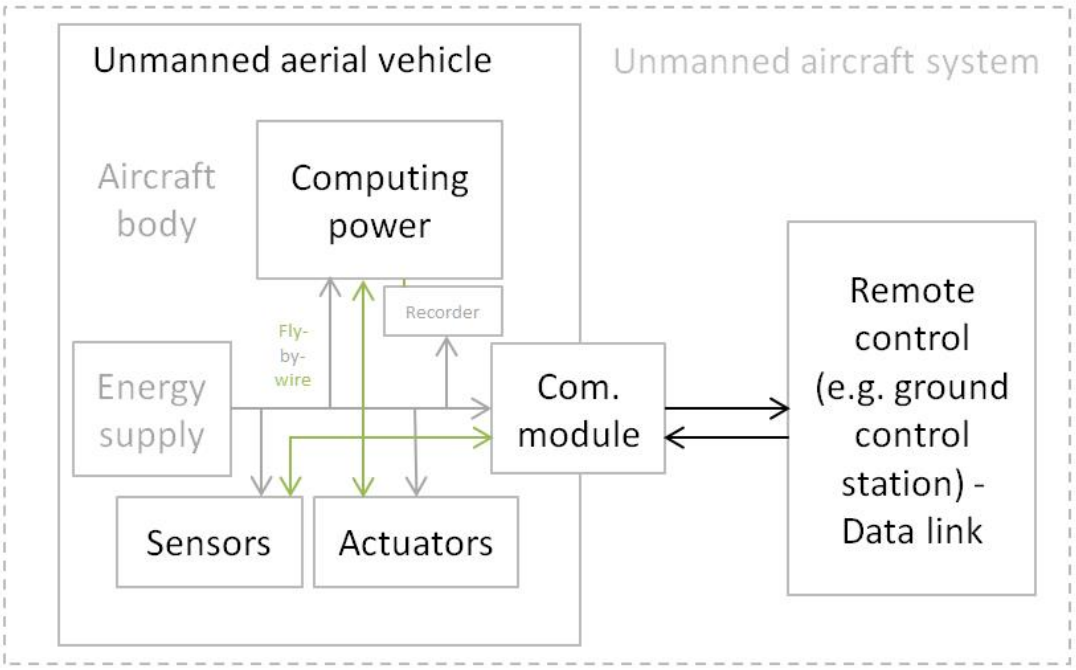
Εικόνα 1. Ο Τσώρτσιλ με ένα πρώιμο μη επανδρωμένο όχημα (6 Ιουνίου 1941)

Εικόνα 2. Μη επανδρωμένο όχημα το οποίο προοριζόταν ως target missile

1.2.3 Εξαρτήματα του UAV

Το UAV μας αποτελείται από ένα σύνολο εξαρτημάτων με σκοπό την κίνησή του. Κάποια από αυτά τα εξαρτήματα μας βοηθούν έμμεσα να επιτύχουμε μια ομαλή πτήση. Τα εξαρτήματα αυτά είναι:

- Μοτέρ DC
- Έλικες
- Σκελετός-Σασί
- Αισθητήρες
- Πλακέτα
- Bluetooth
- GPS
- Camera



Εικόνα 3. Γενική λογική αρχιτεκτονικής σχεδίασης του UAV

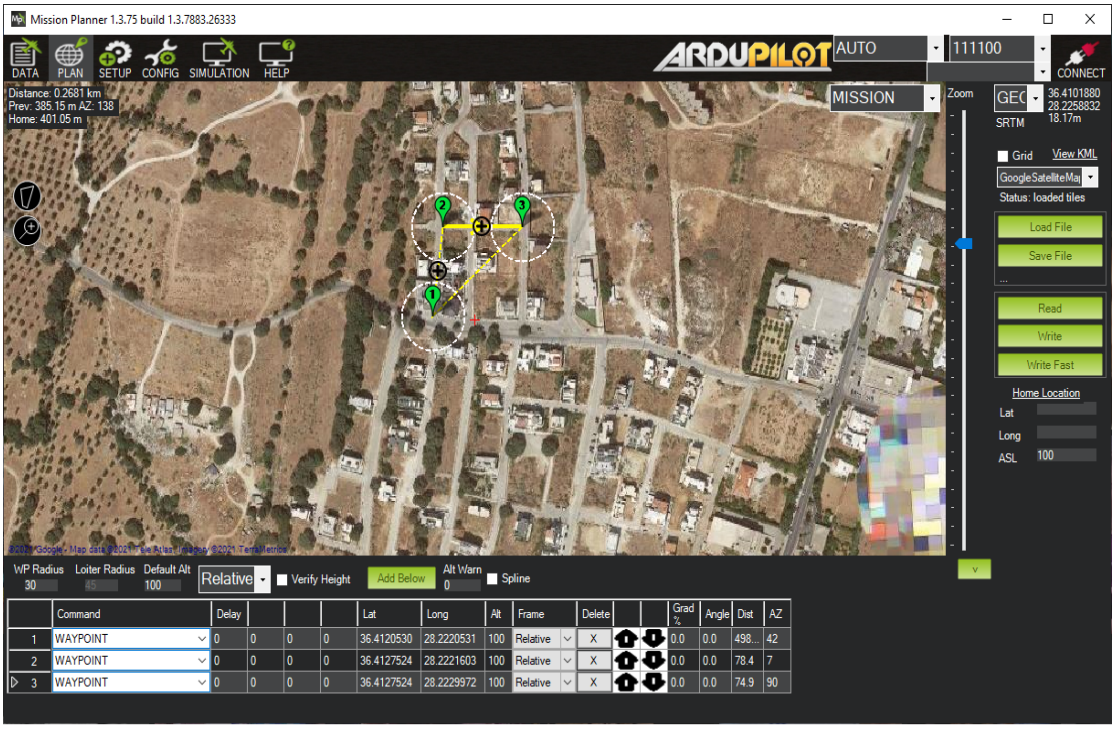
1.3 Εγκαταστάσεις και Ενέργειες πριν την πρώτη πτήση

Πριν την πρώτη μας πτήση με το Ardupilot θα πρέπει να βεβαιωθούμε ότι έχουμε εγκαταστήσει το Ground Control Station (GCS). Κατά την διάρκεια αυτής της διαδικασίας θα πρέπει να εγκαταστήσουμε την **πλακέτα** στον **σκελετό** συνδέοντάς την με τον δέκτη, την τροφοδοσία και τα μοτέρ. Στην συνέχεια θα εκτελέσουμε τις ρυθμίσεις και την **βαθμονόμηση.**

Οι προγραμματιστές τείνουν να χρησιμοποιούν GCS που συνδέονται με κάποιον ηλεκτρονικό υπολογιστή καθώς αυτό δίνει την δυνατότητα για παρέμβαση εις βάθος τόσο στις παραμέτρους του οχήματος αλλά και στον έλεγχο σφαλμάτων. Παρ'όλα αυτά εάν κάποιος χομπίστας επιθυμεί να πετάξει απλώς ένα UAV έχει την δυνατότητα να το κάνει ακόμη και από το κινητό του, επιλέγοντας ένα GCS που απευθύνεται σε χρήστες κάποιου λειτουργικού όπως iOS, Android κ.α.

Μερικά από τα πιο δημοφιλή συστήματα GCS είναι τα παρακάτω:
Mission Planner: Αυτό το GCS είναι συμβατό με λειτουργικό Windows, Linux, Android και είναι σαφώς το πιο συμβατό, παρότι την βέλτιστη απόδοση την έχει στα Windows. Εντάσσει συνεχώς καινούργιες δυνατότητες και ενημερώσεις στο Ardupilot. Συνίσταται κατά βάση σε πιο προχωρημένους χρήστες, παρ'όλα αυτά μπορεί και ένας ερασιτέχνης να το χρησιμοποιήσει και να εκμεταλλευτεί τις δυνατότητές του.

QGroundControl: Είναι συμβατό με πλατφόρμες Windows, MacOS X, Linux, Android και iOS. Είναι το πιο δημοφιλές GCS για συστήματα που βασίζονται σε Android όπως για παράδειγμα smartphones.



Εικόνα 4. Το περιβάλλον του Mission Planner

1.3.1 Επισκόπηση του Mission Planner

Το Mission Planner είναι μια πλήρως εξοπλισμένη εφαρμογή GCS, ανοιχτού κώδικα. Μπορεί να χρησιμοποιηθεί ως βοηθητικό πρόγραμμα διαμόρφωσης ή ως συμπληρωματικό τμήμα του δυναμικού ελέγχου του αυτόνομου οχήματός μας. Παρακάτω ακολουθούν κάποιες από τις ιδιότητές του:

- Δυνατότητα ορισμού σημείων της διαδρομής (Way Points) με χρήση Google Maps και άλλων διαθέσιμων χαρτών.
- Δυνατότητα επιλογής εντολών πτήσης μέσα από το μενού του.
- Δυνατότητα κατεβάσματος του σχεδίου πτήσης για την ανάλυσή του και πιθανή διάθεσή του σε υπηρεσίες και αρχές σε περίπτωση που αυτό χρειαστεί.
- Δυνατότητα λειτουργίας αυτόματου πιλότου για το όχημά μας.
- Διεπαφή με προσομοιωτή πτήσης ώστε να δημιουργήσουμε ένα ολοκληρωμένο πρόγραμμα (Software In The Loop - SITL) UAV προσομοίωσης.
- Εκτελεί την δική του προσομοίωση SITL πολλών τύπων frames για όλα τα οχήματα ArduPilot.
- Καταγραφή σε πραγματικό χρόνο των logs της πτήσης που μας προσφέρουν περισσότερες πληροφορίες από αυτά που είναι σε θέση να καταγράψει το UAV μας και μας τα προσφέρει μετά την ολοκλήρωση της πτήσης, όταν επιστρέψει στην βάση.
- Επαφή με τα στοιχεία πτήσης και ανάλυση αυτών.
- Λειτουργία του UAV με κάμερα πρώτου προσώπου (FPV-First Person View).

Ιστορία του Mission Planner.

Το Mission Planner αποτελεί ένα πρόγραμμα ανοικτού κώδικα. Ως εκ τούτου υποστηρίζεται από την κοινότητα. Αρχικά δημιουργήθηκε από τον Michael Oborne για το APM AutoPilot Project.

1.3.2 Σχεδίαση Αποστολής.

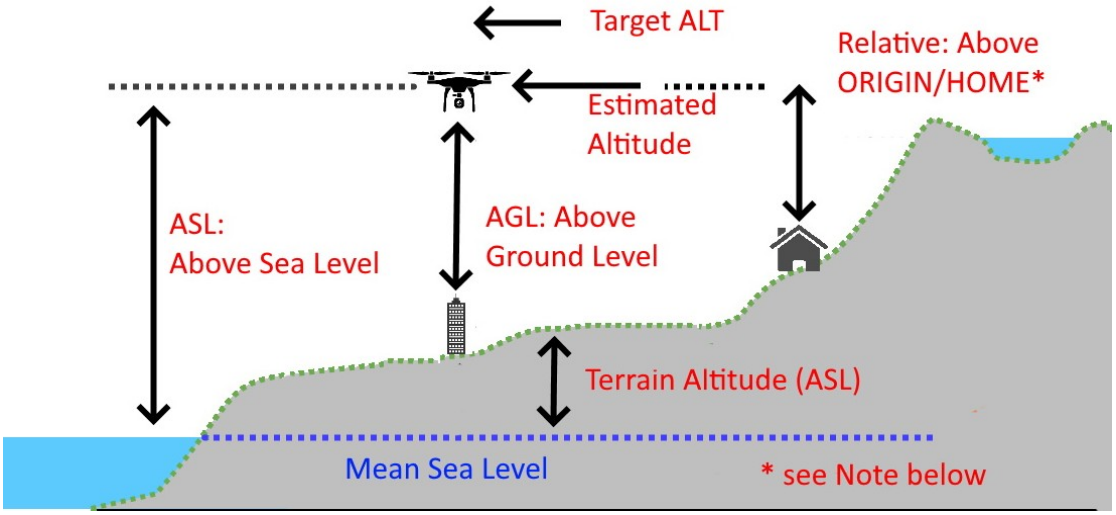
Σε αυτήν την ενότητα θα ασχοληθούμε με ένα σύνολο οδηγιών για τον σχεδιασμό των αποστολών του οχήματός μας στο Mission Planner, οι οποίες θα εκτελούνται όταν μπαίνει στην λειτουργία AUTO.

Σχεδίαση Αποστολής με Waypoints και Events

Στο σχέδιο πτήσης το πρώτο σημείο το οποίο ορίζουμε είναι το Home. Είναι το σημείο από το οποίο ξεκινάει το όχημά μας και αυτό στο οποίο καταλήγει συνήθως. Ο ορισμός του γίνεται αυτόματα ως το πρώτο σημείο το οποίο ορίζεται. Όπως αναφέραμε και νωρίτερα το πρόγραμμα μας δίνει την δυνατότητα να ορίσουμε Waypoints αυτόματα απλά με κλικ πάνω στον χάρτη.

Προεπιλεγμένο υψόμετρο: Σε αυτήν την παράμετρο επιλέγουμε το επιθυμητό ύψος που θα έχει το όχημά μας όταν εισέρχεται σε ένα νέο waypoint.
Το ORIGIN και το HOME ρυθμίζονται κατά την προετοιμασία εδάφους από το GPS και είναι συνήθως η ίδια τοποθεσία όπως προαναφέραμε. Ο χρήστης έχει την δυνατότητα να μετακινήσει το σημείο HOME κατά την διάρκεια της πτήσης (με την χρήση του Mission Planner) ώστε να ορίσει ένα νέο **RTL Point (Ready to Launch).**

Ορισμός Υψόμετρου (Altitude).



Εικόνα 5. Στο παραπάνω διάγραμμα εμφανίζονται διαφορετικοί τύποι υψόμετρου

Παρακάτω θα εξηγήσουμε την σημασία του κάθε ακρωνύμιου:

ASL (Altitude above Sea Level) Το υψόμετρο ορίζεται με βάση την απόσταση του οχήματος από την επιφάνεια της θάλασσας.

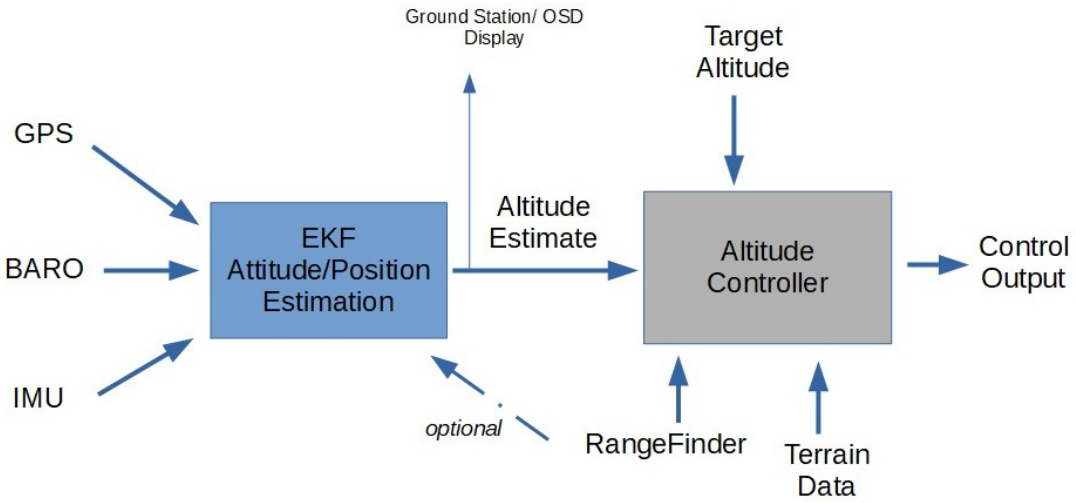
AGL (Altitude above Ground Level) Το υψόμετρο ορίζεται με βάση την απόσταση του οχήματος από την επιφάνεια του εδάφους. Επειδή προφανώς το έδαφος παρουσιάζει διαφορές στο υψόμετρό του η απόσταση αυτή υπολογίζεται σε σχέση με την υψομετρική απόσταση από το σημείο HOME.

Relative (Σχετική) Το υψόμετρο υπολογίζεται με βάση την απόσταση από το έδαφος την δεδομένη στιγμή.

Terrain Alt Το υψόμετρο υπολογίζεται με βάση την υψομετρική απόσταση εδάφους από την επιφάνεια της θάλασσας (κατά βάση αναφέρεται σε επίγεια μη επανδρωμένα οχήματα).

Προσεγγιστική Alt Το υψόμετρο υπολογίζεται εσωτερικά (από το altitude controller του UAV μας) σε σχέση με την ORIGIN θέση του. Χρησιμοποιείται σε περιπτώσεις τις οποίες έχουμε θέσει ως στόχο ένα συγκεκριμένο υψόμετρο το οποίο πρέπει να φτάσει το όχημά μας.

Στόχος Alt Μας ορίζει την απόσταση από την επιθυμητή τιμή υψόμετρου.



Εικόνα 6. Το παραπάνω διάγραμμα μας δείχνει πως εφαρμόζεται ο έλεγχος στο υψόμετρο στο Ardupilot

1.4 Αλγόριθμος EKF

Το **EKF** είναι ένας αλγόριθμος που χρησιμοποιείται στα Copter και Plane. Χρησιμοποιείται για να εκτιμήσει την θέση του οχήματος χρησιμοποιώντας την ταχύτητα και τον γωνιακό προσανατολισμό, με βάση το γυροσκόπιο, το επιταχυνσιόμετρο, την πυξίδα, το GPS, την ταχύτητα του αέρα και την βαρομετρική πίεση.

Ο EKF σε σχέση με άλλους αντίστοιχους αλγόριθμους έχει την δυνατότητα (με την χρήση όλων των στοιχείων που είναι διαθέσιμα σε αυτόν) να απορρίψει μετρήσεις με πολύ ακραία σφάλματα. Αυτό καθιστά το όχημα λιγότερο ευάλωτο σε τυχόν βλάβες που μπορεί να επηρεάσουν την ορθή λειτουργία κάποιου αισθητήρα. Στην διάθεση του αλγορίθμου είναι ακόμη οι μετρήσεις προαιρετικών αισθητήρων όπως ανιχνευτές οπτικής ροής και εύρους λέιζερ που βοηθούν στην πλοήγηση.

Η τρέχουσα έκδοση που χρησιμοποιεί το Ardupilot είναι η EKF2 κυρίως, με σκοπό την εκτίμηση στάσης και θέσης του οχήματος, με το DCM (έναν λιγότερο αποτελεσματικό αλγόριθμο ίδιου τύπου) να λειτουργεί στο παρασκήνιο. Σε περίπτωση που το όχημά μας διαθέτει 2 IMU αισθητήρες υπάρχει η δυνατότητα να λειτουργούν παράλληλα 2 ξεχωριστά EKF χρησιμοποιώντας ο καθένας από ένα ξεχωριστό. Ο IMU είναι ένας συγκεκριμένος τύπος αισθητήρα που μετρά τον γωνιακό ρυθμό, τη δύναμη και μερικές φορές το μαγνητικό πεδίο.

Οι IMU αποτελούνται από επιταχυνσιόμετρο 3 αξόνων και γυροσκόπιο 3 αξόνων, τα οποία θα θεωρούνταν IMU 6 αξόνων. Μπορούν επίσης να περιλαμβάνουν ένα πρόσθετο μαγνητόμετρο 3 αξόνων, το οποίο θα θεωρείται IMU 9 αξόνων.

Μόλις παράξουν έξοδο και οι 2 EKF “πυρήνες” επιλέγεται η έξοδος η οποία ορίζει την καλύτερη εικόνα του οχήματος από τις 2, σύμφωνα με τα δεδομένα των αισθητήρων.
Οι περισσότεροι χρήστες δεν χρειάζεται να τροποποιήσουν οποιεσδήποτε παραμέτρους EKF.
Ακόμη υπάρχει διαθέσιμος ο EKF3 αλλά σε γενικές γραμμές συνίσταται η χρήση του EKF2. Παρακάτω συγκρίνουμε τα δυνατά σημεία του ενός και του άλλου:

- Ο EKF2 θεωρείται ως μια προεπιλογή από τους περισσότερους χρήστες. Ως εκ τούτου έχει περισσότερες δοκιμές και έχει αποδειχθεί πιο σταθερός.
- Ο EKF2 έχει την δυνατότητα να δεχθεί εξωτερικές εκτιμήσεις από άλλα συστήματα.
- Δεδομένου ότι ο EKF3 υπολογίζει και τους 3 άξονες προκειμένου να πάρει απόφαση για την κατεύθυνση του οχήματος, ξοδεύει πολύ μεγαλύτερο χρονικό διάστημα σε αυτήν την διαδικασία.
- Ο EKF3 χρησιμοποιεί νεότερες πηγές αισθητήρων όπως: Beacons, Wheel Encoders και Visual Odometry.
- Ο EKF2 εκτιμά τις κλίμακες του γυροσκόπιου σε αντίθεση με το EKF3. Δεν είναι σημαντικός παράγοντας δεδομένου ότι οι τιμές της γυροσκοπικής κλίμακας είναι σχεδόν πάντα πολύ κοντά στο 1.0 αλλά μπορεί να είναι σημαντικό για οχήματα που περιστρέφονται πολύ γρήγορα.

Επιλέγοντας έκδοση EKF και τον αριθμό των πυρήνων.

Για την επιλογή έκδοσης EKF καθώς και για το πότε λειτουργεί και πότε όχι υπάρχουν οι κατάλληλες εντολές που θα ορίσουμε παρακάτω. Με την χρήση αυτών προσπαθούμε να βελτιστοποιήσουμε την λειτουργία του οχήματός μας.

AHRS_EKF_USE

Ορίζοντας αυτήν την παράμετρο σε “1” ενεργοποιούμε την χρήση του EKF. Όταν το ορίζουμε σε “0” τότε χρησιμοποιούμε άλλους παλαιότερους αλγόριθμους. Δεδομένου ότι έχουμε πλήρως ενεργοποιημένη την καταγραφή δεδομένων AHRS τότε οι αλγόριθμοι αυτοί θα λειτουργούν παράλληλα σε κάθε περίπτωση και θα υπάρξει καταγραφή δεδομένων και από τους 2.
Από την έκδοση 3.3 του Copter και μετά ο EKF είναι ενεργοποιημένος από προεπιλογή και αυτή η παράμετρος δεν είναι διαθέσιμη. Οι χρήστες Planes και Rover μπορούν να επιλέξουν την χρήση παλαιότερων αλγορίθμων.

AHRS_EKF_TYPE

Σε αυτήν την εντολή θέτουμε παράμετρο “2” ή “3” για την χρήση EKF2 ή EKF3 αντιστοίχως.

EK2_ENABLE_,EK3_ENABLE

Μια “μασκα” bit η οποία καθορίζει ποια IMUs χρησιμοποιούνται. Μια ξεχωριστή δραστηριότητα του EKF θα ξεκινήσει για κάθε ένα από αυτά όπως έχουμε προαναφέρει.

1: Ξεκινάει ένα EKF που χρησιμοποιεί το πρώτο IMU
2: Ξεκινάει ένα 2ο EKF που χρησιμοποιεί μόνο το δεύτερο IMU
3: Ξεκινάει 2 EKFs τα οποία χρησιμοποιούν αντίστοιχα το πρώτο και το 2ο IMU

EKF3_PRIMARY

Επιλέγουμε έναν “πυρήνα” ως κύριο. Η αρίθμηση ξεκινάει από το 0. Επομένως η εντολή EK3_IMU_MASK,0 ορίζεται στην πρώτη γραμμή.
Σημείωση: τα Plane και Rover θα επιστρέψουν αυτόματα από το EKF2 και το EKF3 στο DCM σε περίπτωση που το EKF κινδυνέψει να βλάψει το όχημά μας.

1.4.1 Εναλλαγή χρήσης αισθητήρων

Το EKF3 παρέχει την δυνατότητα να συνδυάσει τους αισθητήρες που έχει διαθέσιμους το όχημα ώστε να πάρει τις κατάλληλες αποφάσεις ώστε να κινηθεί. Χρησιμοποιεί και δευτερεύουσες πληροφορίες από τους αισθητήρες όπως πχ ταχύτητα αέρα, βαρόμετρο, πυξίδα και GPS. Αυτό επιτρέπει στο όχημα να συνδυάζει τις πληροφορίες προκειμένου να πάρει μια απόφαση ανάλογα με τις διαφορετικές συνθήκες. Επίσης το EKF3 είναι σε θέση να “κλείσει” τους αισθητήρες του GPS όταν εισέρχεται σε περιοχή που δεν μπορεί να χρησιμοποιηθεί.

1.4.2 Παράμετροι με δυνατότητα ρύθμισης

EK2_ALT_SOURCE

Μας ορίζει από ποιόν αισθητήρα θα πάρει την κύρια τιμή για το υψόμετρο.
Παράμετρος 0: χρησιμοποιείται αυτόματα το βαρόμετρο

Παράμετρος 1: χρησιμοποιείται ο ανιχνευτής απόστασης. Αυτή η παράμετρος χρησιμοποιείται σε εσωτερικούς χώρους όπου και κατά βάση το “έδαφος” παραμένει εντελώς επίπεδο. Σε περίπτωση που θέλουμε να πετάξουμε σε εξωτερικό χώρο μπορούμε να απενεργοποιήσουμε εντελώς αυτήν την λειτουργία.
Παράμετρος 2: χρησιμοποιείται το GPS. Προτιμώμενη επιλογή σε περίπτωση που το GPS είναι πολύ καλής ποιότητας και η μετατόπιση του βαρόμετρου μπορεί να δημιουργήσει πρόβλημα.

EK2_ALT_M_NSE

Η προεπιλογή του είναι το “1.0”. Όσο μειώνεται η τιμή τόσο μειώνεται και η εξάρτηση από τα επιταχυνσιόμετρα και αυξάνεται η εξάρτηση από τα βαρόμετρα.

EK2_GPS_TYPE

Ελέγχει τον τρόπο χρήσης του GPS.

0: Χρησιμοποιεί την 3D ταχύτητα και την 2D θέση από το GPS
1: Χρησιμοποιεί την 2D ταχύτητα και την 2D θέση από το GPS
2: Χρησιμοποιεί την 2D θέση από το GPS
3: Δεν χρησιμοποιεί το GPS

EK2_YAW_M_NSE

Ελέγχει το “κενό” ανάμεσα στο GPS και την πυξίδα κατά την διάρκεια υπολογισμού της κατεύθυνσης. Η προεπιλογή της τιμής είναι το “0,5”. Όσο μειώνεται η τιμή τόσο αυξάνεται η αξιοπιστία της πυξίδας.

Συνάφεια και αλλαγή λωρίδας
Ο αλγόριθμος EKF δημιουργεί πολλαπλές “παρουσίες” του φίλτρου που ονομάζονται λωρίδες (lanes). Η κύρια λωρίδα δίνει τις πληροφορίες για την εκτίμηση της κατάστασης και στο παρασκήνιο λειτουργούν οι υπόλοιπες λωρίδες, διαθέσιμες για αλλαγή σε περίπτωση που χρειαστεί. Οι λωρίδες που δημιουργούνται αντιστοιχούν στα IMU που έχουμε. Η κάθε λωρίδα μπορεί να χρησιμοποιήσει τους αισθητήρες της κύριας. Η κύρια λωρίδα μπορεί να οριστεί από τον χρήστη αλλά είναι σε θέση να αλλάξει ακόμα και εν ώρα πτήσης σε περίπτωση σφάλματος. Τα σύγχρονα οχήματα έχουν στην διάθεσή τους πολλούς αισθητήρες. Το Affinity δίνει την δυνατότητα να χρησιμοποιούνται οι κύριοι αισθητήρες από τις άλλες παρουσίες του EKF. Αυτό μας παρέχει καλύτερο στατιστικό δείγμα πάνω στις πληροφορίες που έχουμε διαθέσιμες και ως εκ τούτου καλύτερη επιλογή λωρίδας. Έτσι είναι σε θέση να απομονώσει αισθητήρες με υψηλό “θόρυβο” στις μετρήσεις μας.
Παρακάτω βλέπουμε έναν πίνακα ως παράδειγμα. Στο όχημά μας διαθέτουμε 1 βαρόμετρο, 2 GPS, 2 αισθητήρες ταχύτητα αέρας, 3 μαγνητόμετρα και 3 IMU.

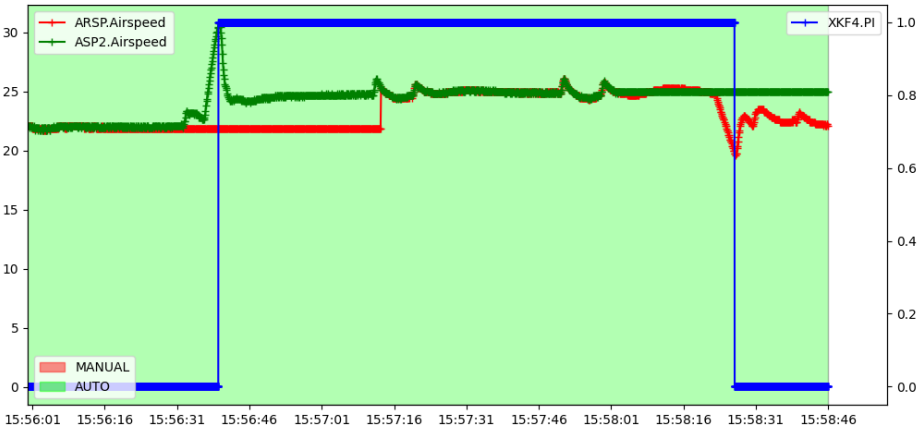
LANE	1	2	3
AIRSPEED	1	2	1
BAROMETER	1	1	1
GPS	1	2	1
MAGNETOMETER	1	2	3

Παράμετροι Διαμόρφωσης
Το Affinity είναι διαθέσιμο μόνο με το EKF3, επομένως πρέπει να βεβαιωθούμε ότι το χρησιμοποιούμε εφόσον είμαστε βέβαιοι ότι το EK3_ENABLE έχει οριστεί σε "1" και το AHRS_EKF_TYPE έχει οριστεί σε "3"

Η εντολή EK3_AFFINITY είναι μια παράμετρος bitmask που μας δίνει την δυνατότητα να επιλέξουμε ανάμεσα στους αισθητήρες που είναι συναφείς.
Η παράμετρος EK3_ERR_THRESH ορίζει την ευαισθησία αλλαγής λωρίδας. Τα σφάλματα των λωρίδων αθροίζονται σε σχέση με την κύρια λωρίδα. Ουσιαστικά αυτή η ευαισθησία μας δίνει την δυνατότητα να ορίζουμε την διαφορά σφάλματος ανάμεσα σε μια **μη κύρια λωρίδα** και την **κύρια** ώστε να μπορούμε να ξέρουμε πότε η πρώτη έχει καλύτερη απόδοση. Η μείωσή της αντίστοιχα θα φέρει πολύ “εύκολη” αλλαγή λωρίδων. Ως εκ τούτου καταλαβαίνουμε ότι η λάθος ρύθμιση αυτής της παραμέτρου μπορεί να προκαλέσει σοβαρά προβλήματα ή/και απώλεια του οχήματός μας.

Αποτελέσματα των Τεστ
Παρακάτω βλέπουμε κάποια γραφήματα δοκιμής με SITL που δείχνουν αλλαγή λωρίδας με δυνατότητα αυτόματης αλλαγής λωρίδας. Στα παραδείγματα που βλέπουμε οι αισθητήρες υποβάλλονται σε “θόρυβο”.

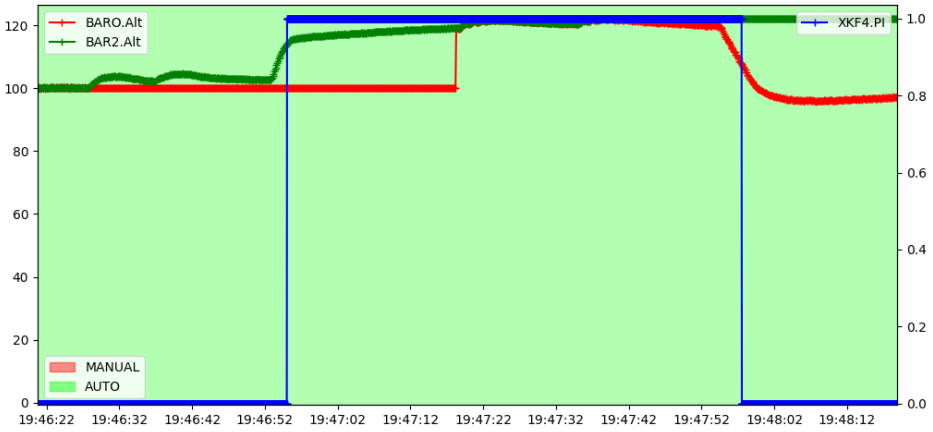
Ταχύτητα Αέρα
Στο 1ο παράδειγμα έχουμε ένα αεροπλάνο με 2 αισθητήρες ταχύτητας αέρα. Έχουμε λοιπόν 2 IMU άρα και 2 λωρίδες. Ο πρώτος αισθητήρας απέτυχε γι’αυτό και δείχνει σταθερή τιμή. Η ταχύτητα του οχήματος αυξάνεται και ενεργοποιείται η 2η λωρίδα (γίνεται κύρια). Ομοίως ο 2ος αισθητήρας αποτυγχάνει και γίνεται ξανά αλλαγή στις λωρίδες.



Εικόνα 7. Έξοδοι αισθητήρων ταχύτητας αέρα

Βαρόμετρο

Παρακάτω ακολουθεί ένα παράδειγμα εναλλαγής λωρίδας με την χρήση 2 βαρόμετρων. Υπάρχουν 2 βαρόμετρα επομένως 2 λωρίδες. Το πρώτο βαρόμετρο που χρησιμοποιείται στην κύρια λωρίδα αποτυγχάνει (γι'αυτό και είναι σταθερή η τιμή πίεσης που δείχνει). Αυτό αυξάνει το υψόμετρο του οχήματος και ενεργοποιείται η 2η λωρίδα ως κύρια. Το 2ο βαρόμετρο αποτυγχάνει, το υψόμετρο αυξάνεται επομένως ενεργοποιείται ξανά η προηγούμενη κύρια λωρίδα (τωρινή δεύτερη).

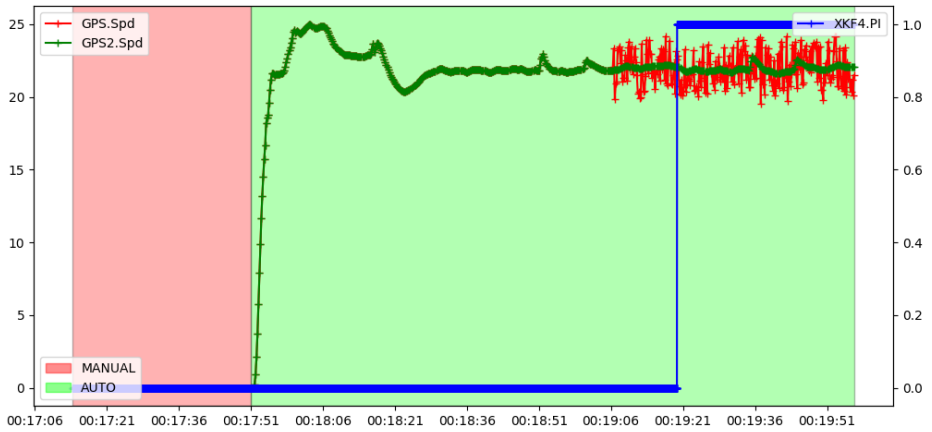


Εικόνα 8. Έξοδοι των βαρόμετρων

GPS

Στο παρακάτω παράδειγμα έχουμε 2 GPS επομένως έχουμε 2 λωρίδες. Στο GPS της κύριας λωρίδας δημιουργούμε (στην προσομοίωση) έναν θόρυβο και στους 3 άξονες. Την πραγματική ταχύτητα μπορούμε να την δούμε στο 2ο GPS το οποίο παραμένει "καθαρό" από τον θόρυβο. Στη συνέχεια η κύρια λωρίδα εμφανίζει ένα υψηλό σφάλμα

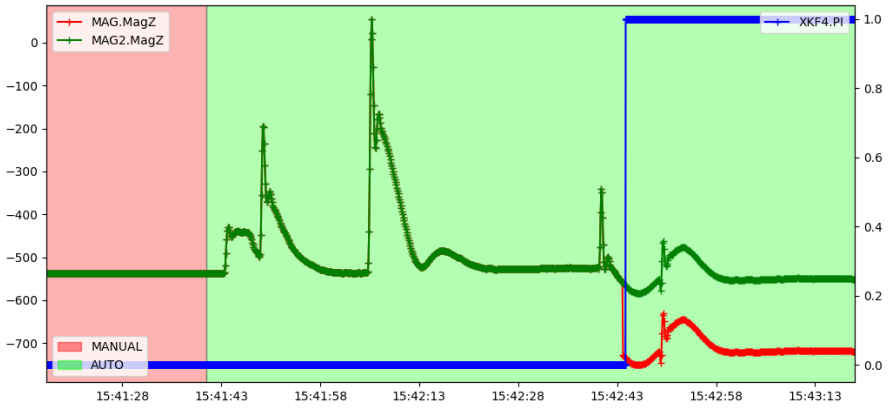
και εμφανίζεται ένας διακόπτης λωρίδας όταν το σφάλμα αυτό ξεπεράσει το προκαθορισμένο όριο που έχει τεθεί.



Εικόνα 9. Έξοδοι των GPS

Magnetometer

Όπως και στα προηγούμενα παραδείγματα έτσι και εδώ έχουμε 2 αισθητήρες και επομένως 2 λωρίδες. Προσομοιώνουμε ένα σφάλμα κατά την διάρκεια της πτήσης στην κύρια λωρίδα, αλλάζοντας την μετατόπιση του στον άξονα Z. Η αλλαγή αυτή παρατηρείται και από το 2ο μαγνητόμετρο. Στη συνέχεια η κύρια λωρίδα αναφέρει ένα σταθερά υψηλό σφάλμα το οποίο φέρνει την διακοπή της όταν φτάσει στο προκαθορισμένο όριο.



Εικόνα 10. Γραφική απεικόνιση εναλλαγής λωρίδας με 2 GPS

Καταγραφή χρόνου πτήσης

Το ArduPilot περιλαμβάνει έναν καταγραφέα χρόνου πτήσης που καταγράφει τον συνολικό χρόνο πτήσης της πλακέτας, τον συνολικό χρόνο λειτουργίας και τον αριθμό των φορών που έχει επανεκκινηθεί η πλακέτα. Αυτά αποθηκεύονται σε παραμέτρους με δυνατότητα επαναφοράς από τον χρήστη, πράγμα που σημαίνει ότι δεν προστατεύονται από παραβιάσεις.

Λεπτομέρειες παραμέτρων:

STAT_BOOTCNT κρατά τον αριθμό των φορών που έχει εκκινηθεί η πλακέτα.

STAT_FLTTIME κρατά τον συνολικό αριθμό δευτερολέπτων που έχει πετάξει το όχημα (σε άθροισμα με τις προηγούμενες πτήσεις).

STAT_RUNTIME κρατά τον συνολικό αριθμό δευτερολέπτων που έχει ενεργοποιηθεί η πλακέτα (συμπεριλαμβανομένων όλων των προηγούμενων πτήσεων)

Επαναφορά Μετρητών
Οι παράμετροι STAT δεν θα διαγραφούν κατά την αναβάθμιση/υποβάθμιση εκδόσεων υλικολογισμικού ενός συγκεκριμένου τύπου οχήματος.

Οι παράμετροι δεν προστατεύονται επομένως μπορούν να αλλάξουν από τον χρήστη.

Επιπλέον μηδενίζονται όταν:

1. Ένας διαφορετικός κώδικας οχήματος γράφεται στην πλακέτα, όπου διαγράφει όλες τις παραμέτρους.

2. Όταν ο χρήστης εκτελέσει μια επαναφορά παραμέτρων.

Εικόνα 11. Περιβάλλον Mission Planner

1.5 Επιλογές πτήσης

Η Εντολή FLIGHT_OPTIONS είναι μια bitmask που επιτρέπει την διαμόρφωση πολλών αλλαγών στην συμπεριφορά του οχήματος.

TRIM_ARSPD_MAX Μετράει την ταχύτητα του αέρα σε cm/s στις αυτόματες λειτουργίες.

ARSPD_FBW_MAX Μέγιστη απαιτούμενη ταχύτητα αέρα στις αυτόματες λειτουργίες. Θα πρέπει να ρυθμιστεί ελαφρώς μικρότερη από το επίπεδο ταχύτητας πτήσης στο THR_MAX και επίσης σε τιμή τουλάχιστον 50% πάνω από το ARSPD_FBW_MIN για να επιτρέπεται ο ακριβής έλεγχος ύψους TECS.

ARSPD_FBW_MIN Ελάχιστη απαιτούμενη ταχύτητα αέρα στις αυτόματες λειτουργίες. Θα πρέπει να ρυθμιστεί σε τιμή υψηλότερη κατά 20% από την ταχύτητα ακινητοποίησης της στάθμης πτήσης.

FBWA και **FBWB** (Fly By Wire A/B) Mode Αποτελούν λειτουργίες υποβοήθειας στην πτήση. Στην B το όχημα διατηρεί και το υψόμετρό του.

CRUISE Mode Αποτελεί λειτουργία υποβοήθειας. Έχει παρόμοια λειτουργία με το FBWB αλλά διατηρεί και την κατεύθυνση του οχήματος.

FLIGHT_OPTIONS	Λειτουργία
0	Επιτρέπει το input του πηδαλίου μόνο σε λειτουργίες MANUAL, STABILIZE και ACRO. Σε άλλες λειτουργίες ελέγχεται από την παράμετρο STICK_MIXING.
1	Αναγκάζει τον κεντρικό μοχλό κίνησης σε TRIM_ARSPD_CM αντί για το μέσο των ARSPD_FBW_MAX και ARSPD_FBW_MIN όταν χρησιμοποιείται αισθητήρας ταχύτητας αέρα σε λειτουργίες FBWB και CRUISE.
2	Απενεργοποίηση του ελέγχου συμπεριφοράς για τις λειτουργίες AUTOTAKEOFF και TAKEOFF που πρέπει να είναι μικρότερες από +-30 μοίρες στροφή και 45 μοίρες βήμα.
3	Επιβάλλει με την χρήση της TRIM_ARSPD_CM μια ταχύτητα αέρα ως στόχο.
4	Δίνει εντολή για ένα επιθυμητό υψόμετρο με την εντολή ALT_HOLD_RTL πριν επιστρέψει προς την θέση HOME, εξαιτίας της παραμέτρου RTL.
5	Ενεργοποίηση του Yaw Damping Controller (αποσβεστήρας εκτροπής) σε λειτουργία ACRO
6	Περιορίζει την κλιμάκωση της ταχύτητας κατά τις αυτόματες απογειώσεις σε 1 ή λιγότερο για να αποφευχθούν οι ταλαντώσεις όταν δεν χρησιμοποιείται αισθητήρας ταχύτητας αέρα.

Προεπιλεγμένα αυτές οι λειτουργίες δεν είναι ενεργοποιημένες (“0”). Η ρύθμιση του bit θα ενεργοποιήσει αυτήν την λειτουργία.

Όριο χαμηλού υψομέτρου Fly-By-Wire
Με αυτήν την επιλογή θέτουμε ένα κατώφλι υψόμετρου στο όχημά μας. Χρησιμοποιείται για την λειτουργία FBWB και είμαστε σε θέση να την χρησιμοποιήσουμε ακόμη και αν δεν έχουμε στην διάθεσή μας έναν αισθητήρα ταχύτητας αέρα.
Σε περίπτωση που το όχημα κατέβει κάτω από το επιθυμητό ελάχιστο υψόμετρο τότε το όχημά μας επιστρέφει αυτόματα σε αυτό το υψόμετρο και μετά επανέρχεται ο έλεγχός του σε εμάς.

1.6 Χρήση για εκπαίδευση

Αυτή η λειτουργία αναφέρεται σε όσους δεν είναι έμπειροι στην πτήση. Χρησιμεύει σε πτήσεις FPV (First Person View). Με την χρήση του FBWB προσδίδει μεγάλη ευχέρεια στην πλοήγηση αποφεύγοντας συγκρούσεις.

Ορισμός ορίου υψομέτρου:
Τα οχήματά μας δεν είναι σε θέση να φτάσουν ένα επιθυμητό υψόμετρο αμέσως. Επομένως η επιλογή του ιδανικού υψόμετρου πρέπει να γίνει με προσοχή. Συνήθως για σχετικά επίπεδες επιφάνειες το ορίζουμε στο 3.000 (30 μέτρα δηλαδή).

1.7 Προσγείωση

Κατά την διάρκεια της προσγείωσης θα χρειαστεί να απενεργοποιήσουμε τις λειτουργίες του οχήματος και να αλλάξουμε το flightmode (λειτουργία πτήσης) σε οποιαδήποτε άλλη λειτουργία πέραν της FBWB.

Δέκτες FPORT

Το πρωτόκολλο επικοινωνίας FPort συνδυάζει πληροφορίες ελέγχου SBus RC αμφίδρομα, από και προς τον αυτόματο πιλότο μέσω ενός καλωδίου υψηλής ταχύτητας baud.

Πολλοί δέκτες της σειράς FRSky X και της σειράς R έχουν αυτήν την δυνατότητα, είτε εγγενώς είτε με αναβάθμιση του υλικού (firmware).
Η σύνδεση που χρησιμοποιούμε διαφέρει ανάλογα με τον τύπο του αυτόματου πιλότου καθώς υπάρχει διαφοροποίηση στο UART (Universal Asynchronous Receiver Transmitter).

1.8 Πρωτόκολλο τηλεμετρίας

Το ArduPilot στέλνει δεδομένα τηλεμετρίας στον δέκτη μέσω Fport χρησιμοποιώντας το πρωτόκολλο τηλεμετρίας Passthrough. Για να εμφανιστούν αισθητήρες στο OpenTX, πρέπει να εγκαταστήσουμε και να ενεργοποιήσουμε ένα script που μπορεί να εμφανίζει την τηλεμετρία Passthrough, όπως το Yaapu FrSky Telemetry Script για OpenTX. Το OpenTX αποτελεί ένα υλικολογισμικό ανοιχτού κώδικα για ραδιοπομπούς RC. Σε περίπτωση που δεν κάνουμε τις παραπάνω διαδικασίες ο μόνος αισθητήρας που θα μπορεί να χρησιμοποιηθεί είναι το GPS.

Διάγραμμα Σύνδεσης

Ορισμένοι δέκτες FrSky έχουν τη δυνατότητα να μεταδίδουν δεδομένα τηλεμετρίας από το όχημα στον πομπό. Αυτο αρχικά επιτεύχθηκε μέσω του SPort και αργότερα μέσω ενός άλλου πρόσθετου πρωτοκόλλου που ονομάζεται FPort το οποίο λαμβάνει υπόψιν του τα δεδομένα RC στον αυτόματο πιλότο. Για την σύνδεση του δέκτη FRSky χρησιμοποιείται οποιοδήποτε UART στον αυτόματο πιλότο. Ωστόσο τόσο το SPort όσο και το FPort είναι πρωτόκολλα διπλής κατεύθυνσης. Δηλαδή απαιτούν αμφίδρομη ανταλλαγή σημάτων, προκειμένου να συνδεθούν με το UART του αυτόματου πιλότου.

1.9Αναλυτικότερη περιγραφή του Extended Kalman Filter

Ο αλγόριθμος EKF (Extended Kalman Filter) που εφαρμόστηκε, υπολογίζει συνολικά 22 καταστάσεις με τις υποκείμενες εξισώσεις που προκύπτουν χρησιμοποιώντας τα ακόλουθα:

Στη συνέχεια θα αναλύσουμε την λειτουργία του αλγορίθμου περιφραστικά με μη μαθηματικό τρόπο:

1. Οι γωνιακοί ρυθμοί των IMU λαμβάνονται υπόψιν για τον υπολογισμό της γωνιακής θέσης του οχήματος.
2. Οι επιταχύνσεις των IMU μετατρέπονται χρησιμοποιώντας τους άξονες X,Y,Z αντιστοιχίζοντάς τους στον Βορρά, στην Ανατολή και στον κάθετο άξονα ως προς την Γη.

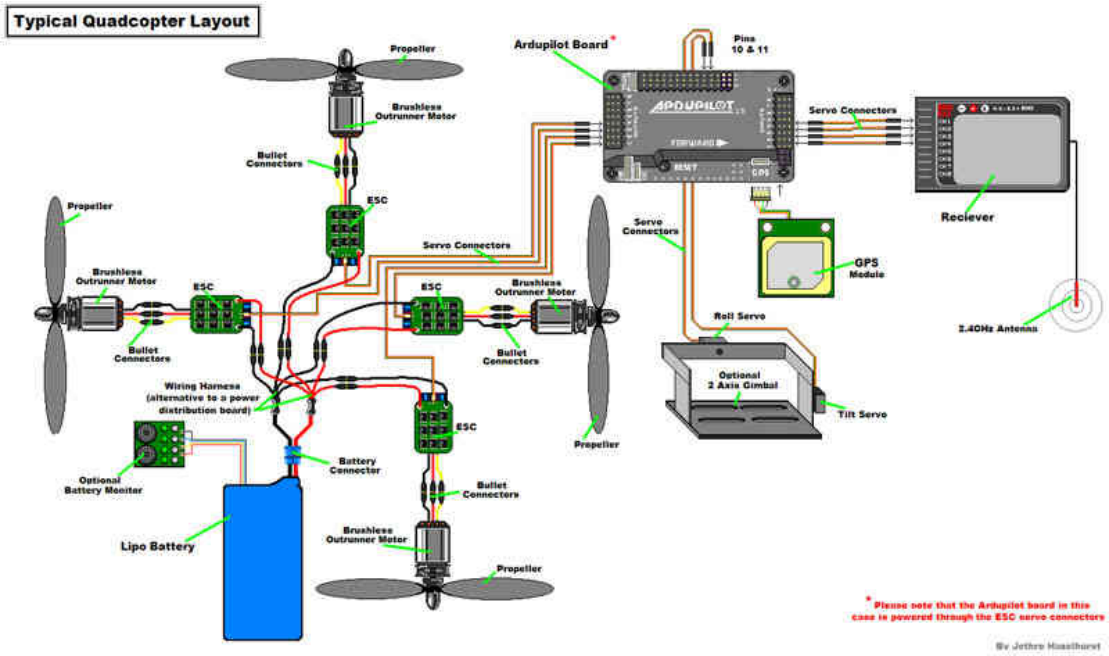
3. Οι επιταχύνσεις λαμβάνονται υπόψιν για τον υπολογισμό της ταχύτητας.
4. Αυτες οι διαδικασίες που προαναφέραμε ονομάζονται **Πρόβλεψη Κατάστασης (State Prediction)**. Σε αυτήν την κατάσταση ορίζονται οι παράμετροι που έχουμε προαναφέρει όπως το βήμα, την ταχύτητα αέρα, το υψόμετρο κ.α.
5. Έχοντας υπόψιν έναν αναμενόμενο θόρυβο στο γυροσκόπιο και το επιταχυνσιόμετρο. Οι εντολές **EKF_GYRO_NOISE** και **EKF_ACC_NOISE** χρησιμοποιούνται για την εκτίμηση της τιμής του σφάλματος, δεχόμενες τα δεδομένα από τα IMU. Όσο μεγαλύτερες είναι αυτές οι παράμετροι τόσο πιο γρήγορα γίνεται ο υπολογισμός της τιμής του σφάλματος. Έτσι καλούμαστε να αποφύγουμε την αύξηση αυτών των τιμών με χρήση άλλων δεδομένων. Αυτά τα δεδομένα καταγράφονται σε έναν πίνακα που ονομάζεται Πίνακας συνδιακύμανσης κατάστασης (State Covariance Matrix).

Τα 1-5 βήματα επαναλαμβάνονται κάθε φορά που έχουμε στην διάθεσή μας καινούργια μέτρηση από τα IMU.

6. Όταν έχουμε μια μέτρηση GPS το φίλτρο υπολογίζει την προβλεπόμενη θέση, σύμφωνα με το βήμα 4 σε σχέση με το GPS. Αυτή η διαφορά ονομάζεται **Καινοτομία** (Innovation).
7. Η καινοτομία, ο πίνακας διακύμανσης κατάστασης και το σφάλμα μέτρησης GPS που καθορίζεται από την εντολή **EKF_POSNE_NOISE** συνδυάζονται για την διόρθωση των μετρήσεων σε κάθε κατάσταση του φίλτρου. Αυτή η διαδικασία ονομάζεται **Διόρθωση Κατάστασης** (State Correction). Αυτό αποτελεί ασφαλώς μια “έξυπνη” λειτουργία του αλγορίθμου καθώς είναι σε θέση να συνδυάσει δεδομένα και να εξάγει συμπεράσματα για την εγκυρότητα της λειτουργίας του φίλτρου.
8. Λαμβάνοντας μια καινούργια μέτρηση, το μέγεθος της αβεβαιότητας σε κάθε μια από τις καταστάσεις που έχουν ήδη ενημερωθεί μειώνεται. Το φίλτρο υπολογίζει την μείωση της αβεβαιότητας αυτής μέσω της λειτουργίας της Διόρθωσης Κατάστασης, ενημερώνει τον Πίνακα Συνδιακύμανσης Κατάστασης και επιστρέφει στη συνέχεια στο βήμα 1.

https://github.com/priseborough/InertialNav/blob/master/derivations/GenerateEquations22states.m
Σε αυτό το link μπορούμε να δούμε αναλυτικά τα βήματα 1-8.

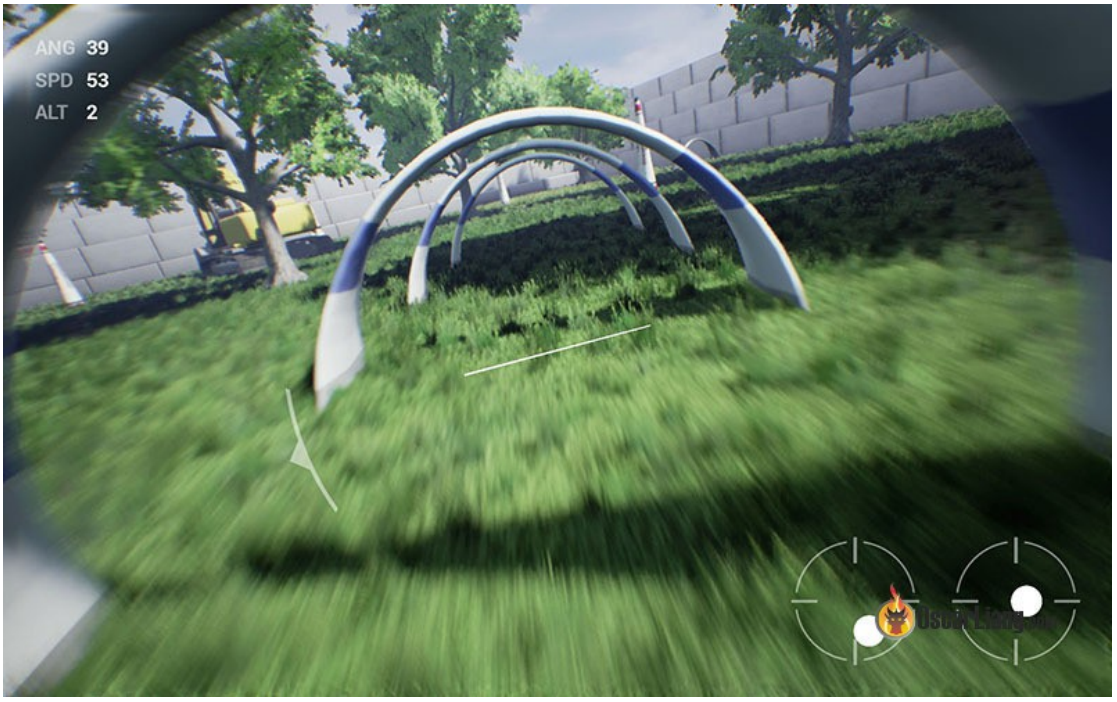
1.10 Το Ardupilot και τα επιμέρους τμήματά του



Εικόνα 12. Συνδεσμολογία Ardupilot

Η πλακέτα του Ardupilot προκειμένου να τεθεί σε λειτουργία ακολουθεί την συνδεσμολογία που φαίνεται στην παραπάνω εικόνα 12. Πρέπει να έχουμε στην διάθεσή μας την σύνδεση με μια μπαταρία για την παροχή ενέργειας. Συνδέουμε το GPS και τους υπόλοιπους αισθητήρες με βάση την αντίστοιχη σήμανση που διαθέτει η πλακέτα μας επάνω. Όλα αυτά καταλήγουν ασφαλώς στα μοτέρ στα οποία έχουμε προσαρμόσει έλικες.

Κεφάλαιο 2 Simulation



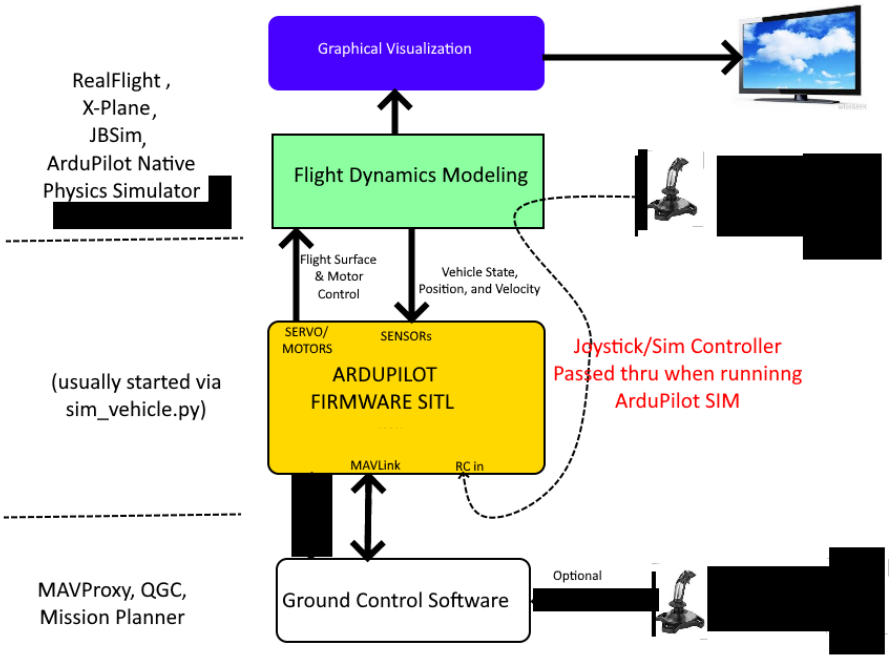
Όπως είδαμε και στα προηγούμενα κεφάλαια ο εξοπλισμός μας υπάρχει περίπτωση να διατρέξει κίνδυνο. Οι κακές καιρικές συνθήκες, κάποιο τυχόν λάθος μας ή και άλλοι αστάθμητοι παράγοντες μπορούν να οδηγήσουν σε “τραυματισμό” του εξοπλισμού μας έως και την καταστροφή του. Αυτό σε πολλές περιπτώσεις συμβαίνει σε ερασιτεχνικό εξοπλισμό ή ακόμη και επαγγελματικό. Γι’αυτόν τον λόγο έχει αναπτυχθεί ένα πλήθος προγραμμάτων τα οποία μας επιτρέπουν να προσομοιώσουμε την πτήση ενός UAV ώστε να ελέγξουμε το σχέδιο πτήσης που σκοπεύουμε να ακολουθήσουμε και όπως και να ελέγξουμε τους αλγόριθμους που σκοπεύουμε να χρησιμοποιήσουμε. Αυτές οι προσομοιώσεις μας δίνουν την δυνατότητα να ελέγξουμε και άλλες παραμέτρους πέρα από την ίδια την λειτουργία του UAV μας. Μπορούμε να δημιουργήσουμε “εμπόδια”, δέντρα, κτίρια ακόμη και απότομες αλλαγές σε καιρικές συνθήκες προκειμένου να ελέγξουμε αλγοριθμικά την λειτουργία του UAV μας και την απόκρισή του σε αυτά.

2.1 Συνοπτικά

Η προσομοίωση υλοποιείται χρησιμοποιώντας ένα μοντέλο δυναμικής πτήσης (FDM) του οχήματος για την προσομοίωση των νόμων της μηχανικής που εμπλέκονται στην κίνηση του οχήματος. Λαμβάνει εισόδους από ένα πρόγραμμα SITL (Software in the Loop) που εκτελεί το υλικολογισμικό ArduPilot (που είναι οι έξοδοι σερβο/μοτέρ του υλικολογισμικού) και εξάγει την κατάσταση, την θέση, τις ταχύτητες του οχήματος, κ.λ.π. που προκύπτουν από αυτές τις εισόδους πίσω στην προσομοίωση υλικολογισμικού. Ακριβώς όπως θα έκαναν οι αισθητήρες στην πραγματική πτήση.

Ο πιλοτικός έλεγχος υλοποιείται είτε μέσω joystick, είτε με ένα ειδικό κιβώτιο ελεγκτή προσομοίωσης (όπως Interlink), είτε με εντολές MAVLink από ένα πρόγραμμα Σταθμού Ελέγχου Εδάφους (GCS), όπως το MAVProxy ή το Mission Planner.

Στο παρακάτω σχήμα αναλύεται η λογική λειτουργίας της προσομοίωσης.



Εικόνα 13. Λογική λειτουργίας της προσομοίωσης

Το ArduPilot παρέχει ένα “ενσωματωμένο” πρόγραμμα προσομοιωτή υλικολογισμικού + FDM (με βάση την sim_vehicle.py), που συχνά αναφέρεται απλώς ως **SITL**, το οποίο συνήθως χρησιμοποιείται με τον GCS του, το MAVProxy. Ο προσομοιωτής υλικολογισμικού και το FDM του ArduPilot μπορούν επίσης να χρησιμοποιηθούν από το Mission Planner για προσομοιώσεις SITL. Επιπλέον, το στοιχείο προσομοιωτή υλικολογισμικού του ArduPilot μπορεί να συνδεθεί με άλλα προγράμματα FDM/Graphics που χρησιμοποιούνται συνήθως ως αυτόνομοι προσομοιωτές πτήσης/οχήματος, προκειμένου να αποκτήσουμε πιο ρεαλιστικά μοντέλα φυσικής ή/και γραφικά οχημάτων υψηλής ευκρίνειας, όπως το RealFlight ή το X-Plane.

Παρακάτω θα αναφέρουμε συνοπτικά κάποια προγράμματα προσομοίωσης:
Το **SITL** (Software In The Loop) είναι ο προσομοιωτής που χρησιμοποιείται πιο συχνά από τους προγραμματιστές. Είναι ένας απλός προσομοιωτής που είναι χτισμένος σε όλες τις εκδόσεις SITL του ArduPilot. Χρησιμοποιείται από τον autotester. Υπάρχουν και άλλοι προσομοιωτές οι οποίοι έχουν χτιστεί με βάση το SITL.

Κάποιοι από αυτούς είναι οι παρακάτω:

- Mission Planner
- Gazebo
- Xplane-10 Soaring
- Xplane-10

- Real Flight
- Morse
- Replay
- JSBSim
- AirSim
- Silent Wings Soaring
- Matlab and Simulink
- JSON Interface
- Webots

2.2 SITL (Software In The Loop)

Το SITL δίνει την δυνατότητα να προσομοιώσουμε Plane ή και Copter (πολλών μοτέρ), χωρίς την χρήση hardware. Βασίζεται σε compiler C++ για την λειτουργία του και διαθέτει ένα ενσωματωμένο executable αρχείο το οποίο μας επιτρέπει να ελέγξουμε τον κώδικα.

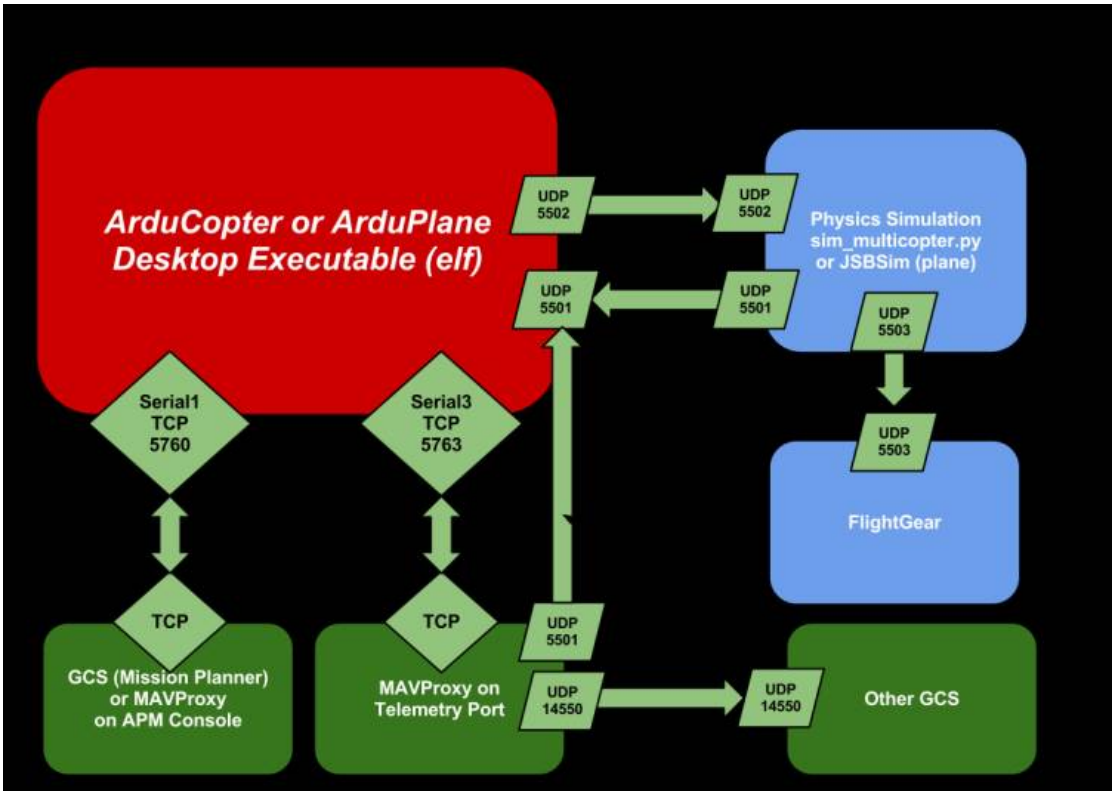
Το Ardupilot αποτελεί ασφαλώς μια πλατφόρμα εύκολη στην μεταφορά και στην συνεργασία με διαφορετικά περιβάλλοντα. Επομένως μπορούμε να το χρησιμοποιήσουμε στον Η/Υ μας και έτσι να προσομοιώσουμε τους αλγόριθμους πτήσης που σκοπεύουμε να δοκιμάσουμε.

Αυτή η προσομοίωση όμως μπορεί να δημιουργήσει ένα βασικό πρόβλημα. Σε μια πραγματική πτήση τον βασικότερο ίσως ρόλο τον παίζουν οι αισθητήρες του UAV μας. Οι τιμές που θα πάρουμε από αυτούς μας δίνουν την δυνατότητα να ελέγξουμε τους αλγόριθμους που έχουμε χρησιμοποιήσει. Είμαστε λοιπόν σε θέση να προσομοιώσουμε και την λειτουργία των αισθητήρων προκειμένου να έχουμε το input (είσοδο) των δεδομένων που χρειαζόμαστε. Τέτοιες προσομοιώσεις μπορεί να είναι ένα πλήθος βασικών αισθητήρων χρήσης σε πτήσεις, κάμερες, ιχνηλάτηση (tracking) μέσω κεραιών κ.α.

Η ανάπτυξή του σε γλώσσα C++ προσδίδει μεγάλα προτερήματα στο πρόγραμμά μας καθώς υπάρχει αντίστοιχη “υποστήριξη” σε διαδραστικούς debuggers και εργαλεία στατικής και δυναμικής ανάλυσης. Το πρόγραμμα SITL λοιπόν είναι σε θέση να “τρέξει” σε περιβάλλοντα Windows και Linux.

Ενδεικτικά παραθέτουμε πληροφορίες για την χρήση του SITL. Με το SITL υπάρχει διαθέσιμο ένα Script “`sim_vehicle.py” όπου τρέχοντάς το σετάρει το περιβάλλον της προσομοίωσης. Σε περίπτωση που θέλουμε βοήθεια στην χρήση του πληκτρολογούμε sim_vehicle.py --help .

2.2.1 Αρχιτεκτονική SITL



Εικόνα 14. Αρχιτεκτονική SITL

Στην παραπάνω εικόνα τα ports που χρησιμοποιούνται είναι τα 5501/5502, για την σύνδεση μεταξύ Ardupilot και την προσομοίωση. Μπορούν να χρησιμοποιηθούν και άλλες και είναι τυχαία επιλογή.

2.2.2 Τύποι frames

Οι διαθέσιμοι τύποι οχημάτων είναι Plane, Rover, Copter. Για να επιλέξουμε κάποιο όχημα πληκτρολογούμε sim_vehicle.py -v ArduPlane --console --map όπου η παράμετρος -v ορίζει την επιλογή οχήματος που κάνουμε, στην προκειμένη περίπτωση το ArduPlane. Σε περίπτωση που θέλουμε να αλλάξουμε όχημα πληκτρολογούμε sim_vehicle.py -v ArduPlane -f quadplane --console --map όπου με την παράμετρο -f και το όνομα του αντίστοιχου frame που θέλουμε να επιλέξουμε ορίζουμε την αλλαγή.

2.2.3 Ορίζουμε αρχική θέση οχήματος

Για να πετάξουμε το όχημα θα χρειαστούμε αρχικά να του ορίσουμε μια αρχική θέση. Η αρχική αυτή θέση του οχήματος (συνήθως αναφέρεται και ως Home) αποτελεί τόσο την θέση που θα επιστρέψει το όχημα εφόσον ολοκληρώσει την αποστολή του όσο και μια ασφάλεια σε περίπτωση που υπάρξει πρόβλημα (χαμηλή μπαταρία, βρει κάποιο μη προσπελάσιμο εμπόδιο κ.α.) ώστε να επιστρέψει εκεί.

Για να ξεκινήσουμε γράφουμε:
cd ArduCopter
sim_vehicle.py -L locationname--console --map

Σε περίπτωση που θέλουμε on screen display γράφουμε:

sim_vehicle.py -v ArduPlane --console --map --osd

δηλαδή στην ουσία προσθέτουμε την παράμετρο --osd.

Το SITL έχει προρυθμισμένες κάποιες παραμέτρους ώστε να είναι σε θέση να λειτουργήσει. Αυτές τις παραμέτρους θα πρέπει να τις αλλάξουμε ώστε να προσαρμοστούν στις ανάγκες μας. Αυτές οι παράμετροι αναφέρονται στα στοιχεία του οχήματός μας και στο περιβάλλον.

Η εντολή ώστε να τις καλέσουμε είναι η:
param load ../Tools/autotest/default_params/vehiclename.parm
όπου vehicletype είναι ο τύπος του οχήματος, δηλαδή plane,copter ή rover.

Σε περίπτωση που θελήσουμε να φορτώσουμε ξανά τις αρχικές παραμέτρους πληκτρολογούμε:
sim_vehicle.py προσθέτοντας την flag -w .

Για να αποθηκεύσουμε τις παραμέτρους πληκτρολογούμε:
param save ./myparams.parm

2.2.4 Ρύθμιση παραμέτρων

Για να δούμε το σύνολο των παραμέτρων γράφουμε την εντολή:
param show

Ώστε να ρυθμίσουμε κάποια παράμετρο χρησιμοποιούμε την εντολή:
param set PARAMETERNAME VALUE

Για τον έλεγχο του Remote Control πρέπει να ορίσουμε την παράμετρο SIM_RC_FAIL ίση με 1 , επομένως:
param set SIM_RC_FAIL 1

αντιστοίχως για το GPS:
param set SIM_GPS_DISABLE 1

2.2.5 Έλεγχος του GPS-for-Yaw

Με χρήση 2 προσομοιώσεων Ublox GPS:

param set EK3_SRC1_YAW 2
param set GPS_AUTO_CONFIG 0
param set GPS_TYPE 17
param set GPS_TYPE2 18
param set GPS_POS1_Y -0.2
param set GPS_POS2_Y 0.2
param set SIM_GPS_POS_Y -0.2
param set SIM_GPS2_POS_Y 0.2
param set SIM_GPS2_DISABLE 0
param set SIM_GPS2_HDG 1

Στη συνέχεια κάνουμε επανεκκίνηση στο SITL και εμφανίζεται το παρακάτω μήνυμα:
status GPS2_RAW:

Για έλεγχο σε ένα μονό NMEA GPS
param set EK3_SRC1_YAW 2
param set GPS_TYPE 5
param set SIM_GPS_TYPE 5
param set SIM_GPS_HDG 1

Στη συνέχεια κάνουμε επανεκκίνηση στο SITL και εμφανίζεται το παρακάτω μήνυμα:
status GPS_RAW_INT:

Όπως προαναφέραμε, είμαστε σε θέση να θέσουμε παραμέτρους και στο περιβάλλον. Με την παρακάτω εντολή βλέπουμε τις διαθέσιμες επιλογές μας όσων αφορά τον αέρα:
param show sim_wind*
και με τις εντολές:
param set SIM_WIND_DIR value
param set SIM_WIND_SPD value
ορίζουμε την κατεύθυνση (direction) και την ταχύτητα (speed) αντίστοιχα αλλάζοντας την value με μια τιμή. Η κατεύθυνση ασφαλώς ορίζεται σε μοίρες.

2.3 Προσομοίωση Gimbal

Μια ακόμα παράμετρος που μπορούμε να προσομοιώσουμε με χρήση SITL είναι το Gimbal για χρήση σε UAV με κάμερα. Στην ουσία με αυτήν την προσομοίωση μας επιστρέφονται κάποιες τιμές στον δείκτη MOUNT_STATUS. Τον ορίζουμε ως:
param set MNT1_TYPE 1
παρακάτω ορίζουμε το Remote Control και την έξοδο που θα πάρουμε από αυτό καθώς και τον τύπο του μοτέρ που είναι το:
Pan-Tilt Multi Servo
param set SERVO6_FUNCTION 6
ακόμη παρακάτω ορίζουμε το Roll Servo που είναι το άλλο μοτέρ που χρησιμοποιείται σε αυτήν την λειτουργία και το Remote Control καθώς και επίσης την έξοδό μας:
param set SERVO7_FUNCTION 8
Για να σταματήσουμε την διαδικασία πληκτρολογούμε:
sim_vehicle.py -M
Με την flag -M σταματάμε την διαδικασία.

Με παρόμοιο τρόπο μπορούμε να ορίσουμε τα:
Rangefinder
Optical Flow Sensor
RPM Sensor
Wheel Encoders
Range Beacons

2.4 UARTs και κονσόλα

Ένα μεγάλο σύνολο των εφαρμογών του Ardupilot βασίζεται στα UARTs (Universal Asynchronous Receiver Transmitter). Χρησιμοποιούνται για debug, για τηλεμετρία, για GPS και άλλα.

Διαθέτει 8 UARTs τα οποία δεν έχουν οριστεί κάπως στο HAL όμως άλλα ports θεωρούν αυτόματα ότι έχουν επιλεχθεί αυτά. Οι επιλογές της γραμμής εντολών για την χρήση με το sim_vehicle.py της σειριακής θύρας θα πρέπει να προηγούνται από το -A για να περάσουν στο δυαδικό αρχείο του οχήματος. Επίσης θα πρέπει να συμπεριλάβουμε το πρωτόκολλο uart. Επομένως ένα παράδειγμα χρήσης είναι το:
sim_vehicle.py --console --map -A --serial5=uart:/dev/ttyS15:115200

Parameters Prefixed	**sim_vehicle command line**	**Defined Role**	**Default Connection**
SERIAL0_	- -uartA= or - -serial0=	Console	tcp:localhost:5760:wait
SERIAL1_	- -uartA= or - -serial1=	MAVLink	tcp:localhost:5762
SERIAL2_	- -uartA= or - -serial2=	MAVLink	tcp:localhost:5763
SERIAL3_	- -uartA= or - -serial3=	GPS	Simulated GPS
SERIAL4_	- -uartA= or - -serial4=	GPS	Simulated GPS
SERIAL5_	- -uartA= or - -serial5=		
SERIAL6_	- -uartA= or - -serial6=		
SERIAL7_	- -uartA= or - -serial7=		

Αυτές λοιπόν είναι οι προεγκατεστημένες παράμετροι τις οποίες μπορούμε να αλλάξουμε.

2.5 Κονσόλα Debug

Πέρα από τις UARTs που διαθέτουμε, μια εξ αυτών χρησιμοποιείται στο debug. Συνήθως η θύρα για debug συνδέεται με την θύρα USB και στην συνέχεια συνδέεται με την κονσόλα απευθείας. Η port του USB είναι η 5760. Μπορούμε να αλλάξουμε την USB θύρα με την hal.console->printf() εντολή.

2.6 Συναρτήσεις UART

Κάθε UART διαθέτει έναν αριθμό βασικών λειτουργιών IO. Οι βασικές λειτουργίες είναι:
printf - Εκτύπωση
printf_P - Εκτύπωση με χρήση συμβολοσειράς προγράμματος
println - Εκτύπωση γραμμής και τροφοδοσία γραμμής
write - Εγγραφή κάποιον bytes
read - Ανάγνωση κάποιον bytes
available - Έλεγχος εάν υπάρχουν διαθέσιμα bytes που περιμένουν
txspace - Έλεγχος του χώρου που υπάρχει στον buffer
get_flow_control - Έλεγχος εάν το UART έχει δυνατότητες ελέγχου ροής

2.7 Matlab και Simulink εισαγωγικά

Το **Matlab (Matrix Laboratory)** αποτελεί μια **γλώσσα προγραμματισμού** και ένα **περιβάλλον υπολογισμών** από την εταιρεία Mathworks. Αυτό το περιβάλλον δίνει την δυνατότητα υπολογισμών σε πίνακες, γραφική απεικόνιση συναρτήσεων και δεδομένων, εκτέλεσης λειτουργιών αλγορίθμων, δημιουργία γραφικού περιβάλλοντος χρηστών (User Interface) και αλληλεπίδρασης με άλλες γλώσσες προγραμματισμού. Αν και η αρχική του σχεδίαση ήταν αποκλειστικά για υπολογιστική χρήση, ένα toolbox έδωσε την δυνατότητα χρήσης της υπολογιστικής άλγεβρας. Ένα επιπλέον toolbox παρέχει την δυνατότητα δημιουργίας γραφικού περιβάλλοντος, προσφέροντας δυνατότητες προσομοίωσης και μοντελοποίησης σχεδίων και ενσωματωμένων συστημάτων. Πλέον αποτελεί ένα πρόγραμμα το οποίο χρησιμοποιείται από ένα πλήθος ειδικοτήτων όπως ο τομέας των μηχανικών, των επιστημόνων και των οικονομικών.

Όπως αναφέραμε και παραπάνω το Matlab ως περιβάλλον υπολογισμών είναι σε θέση να λειτουργήσει συνεργατικά με άλλες γλώσσες προγραμματισμού. Τέτοιες γλώσσες είναι η C, η Fortran και η Python. Γλώσσες που επίσης χρησιμοποιούνται σε μεγάλο βαθμό τα τελευταία χρόνια σε πλήθος επιστημονικών κλάδων. Βιβλιοθήκες γραμμένες σε Perl, Java, ActiveX και .NET μπορούν να κληθούν απευθείας εντός του κώδικά μας.

2.7.1 Simulink

Το **Simulink** αποτελεί ένα **περιβάλλον** εντός του Matlab, για **γραφικό προγραμματισμό** και **μοντελοποίηση** όπως και για **προσομοίωση** και **ανάλυση δυναμικών συστημάτων**.
Το Simulink λοιπόν μας δίνει την δυνατότητα να σχεδιάσουμε ένα σενάριο προσομοίωσης για την πτήση του UAV μας. Σε αυτήν την προσομοίωση μπορούμε να παραμετροποιήσουμε και να σχεδιάσουμε το μοντέλο του UAV μας χρησιμοποιώντας το σε διαφορετικές συνθήκες πτήσης.

Για παράδειγμα μπορούμε να σχεδιάσουμε ένα σενάριο κατά το οποίο το UAV μας πετάει ανάμεσα σε 2 τετράγωνα σε μια πόλη,προσομοιάζοντας τα κτίρια κ.α.

Η προσομοίωση με χρήση ρεαλιστικών σεναρίων UAV και μοντέλων αισθητήρων αποτελεί βασικό προσόν στις δοκιμές που κάνουμε στους αλγόριθμους που δημιουργούμε ή χρησιμοποιούμε διότι:

- Σε ένα τετραγωνικό περιβάλλον προσομοίωσης, το περιβάλλον, το/α οχήμα/τά μας και άλλα αντικείμενα εμφανίζονται ως απλά κουτιά. Αυτή η προσομοίωση χρησιμοποιείται για εύκολο και γρήγορο έλεγχο αισθητήρων ή και κάποιων αλγόριθμων στους οποίους δεν θα υπάρξει κάποια διαφοροποίηση από μια πλήρη και ακριβή προσομοίωση. Για να τρέξουμε αυτό το σενάριο γίνεται με το UavScenario.
- Στην Unreal Engine, ένα περιβάλλον προσομοίωσης, οι προσομοιώσεις γίνονται render μέσω της Unreal Engine μιας μηχανής γραφικών της εταιρείας Epic Games που δραστηριοποιείται στον τομέα του Gaming. Χρησιμοποιώντας αυτό το περιβάλλον έχουμε την δυνατότητα να οπτικοποιήσουμε το σενάριό μας. Ρεαλιστικό περιβάλλον, ραντάρ υψηλής αξιοπιστίας, κάμερα, και δεδομένα αισθητήρων παρέχονται σε αυτήν την προσομοίωση. Αυτό το περιβάλλον παρέχεται μόνο για χρήση σε Windows.

2.7.2 UAV Toolbox

Το UAV Toolbox παρέχει εργαλεία και εφαρμογές για σχεδιασμό, προσομοίωση, έλεγχο και ανάπτυξη UAVs και εφαρμογών UAVs. Δίνει την δυνατότητα σχεδιασμού αλγορίθμων αυτόνομης πτήσης, αποστολών UAV και flight controllers. Η εφαρμογή Flight Log Analyzer μας επιτρέπει διαδραστικά να αναλύσουμε τρισδιάστατες διαδρομές πτήσεων, δεδομένα χρήσιμα στην τηλεμετρία και δεδομένα αισθητήρων από συλλογή διαφόρων πτήσεων.

Για χρήση σε H/Y και hardware-in-the-loop (HIL) έλεγχο αλγόριθμων αυτόματων πτήσεων και flight controllers μπορούμε να παράξουμε και να προσομοιώσουμε UAV σενάρια. Μπορούμε να προσομοιώσουμε επίσης την χρήση κάμερας, lidar, IMU (Inertial Measurement Unit). Ο GPS αισθητήρας μας παράγει ένα 3D φωτορεαλιστικό περιβάλλον ή και ένα 2.5D περιβάλλον προσομοίωσης.

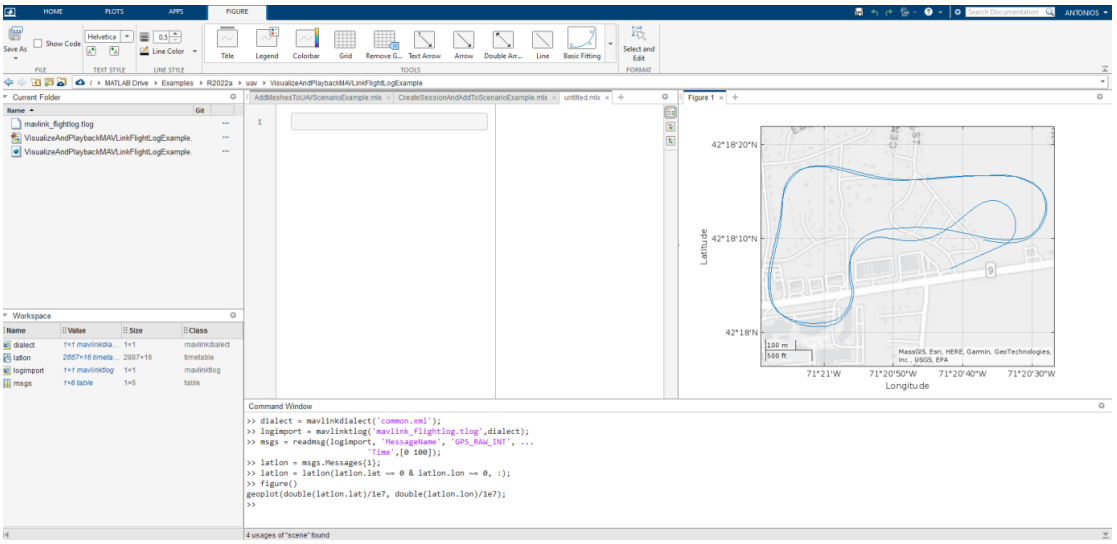
Το UAV Toolbox παρέχει παραδείγματα εφαρμογών αναφοράς για κοινές χρήσεις UAV, όπως η αυτόνομη παράδοση πακέτου με χρήση multirotor UAVs. Το toolbox υποστηρίζει την δημιουργία κώδικα C/C++ για γρήγορη δημιουργία πρωτοτύπων, δοκιμές HIL και αυτόνομη ανάπτυξη σε υλικό.

2.7.3 Οπτικοποίηση και αναπαραγωγή MAVLink Flight Log

Στο παρακάτω παράδειγμα θα δούμε πως να φορτώσουμε τα αρχεία μιας τηλεμετρίας (TLOG) που περιέχει πακέτα MAVLink, στο Matlab. Τα στοιχεία αυτά των αρχείων μας βοηθάνε να σχεδιάσουμε την γραφική απεικόνισή μας. Στη συνέχεια μας εμφανίζονται απεικονίσεις των MAVLink επικοινωνιών. Είναι στην ουσία UAV που σχεδιάζουν βάση του tlog την πτήση.

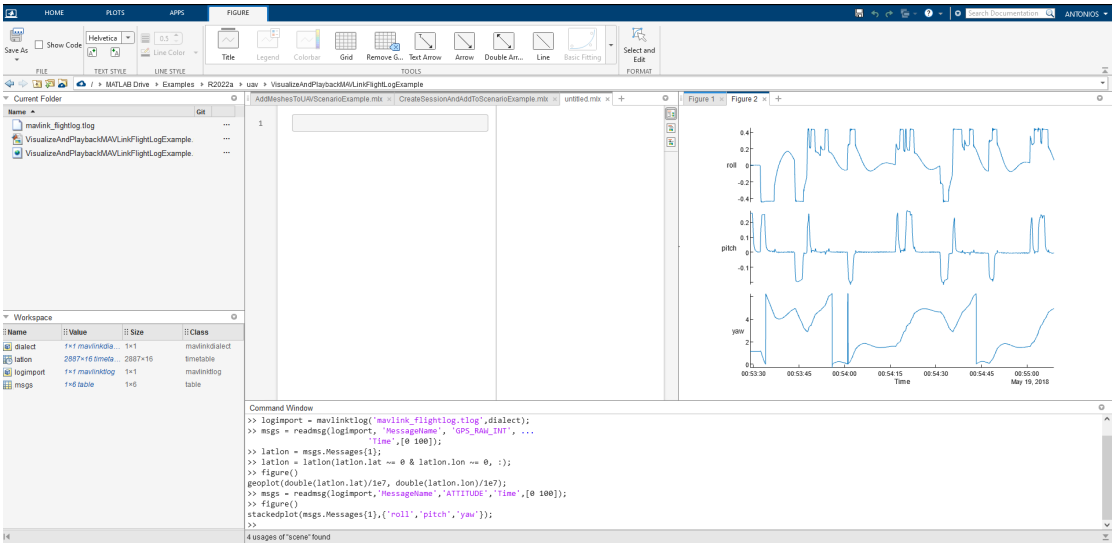
Φορτώνουμε το MAVLink TLOG:

Δημιουργούμε ένα αντικείμενο, το mavlinkdialect χρησιμοποιώντας την “διάλεκτο” common.xml . Και στη συνέχεια χρησιμοποιούμε την mavlinklog προκειμένου να φορτώσουμε τα δεδομένα του TLOG.



Εικόνα 15. Φορτώνουμε (Load) τα δεδομένα του TLOG

Στη συνέχεια εξάγουμε τα attitude messages και ορίζουμε το όνομα του message. Έπειτα χρησιμοποιώντας την stackedplot σχεδιάζουμε τα δεδομένα roll, pitch και yaw.

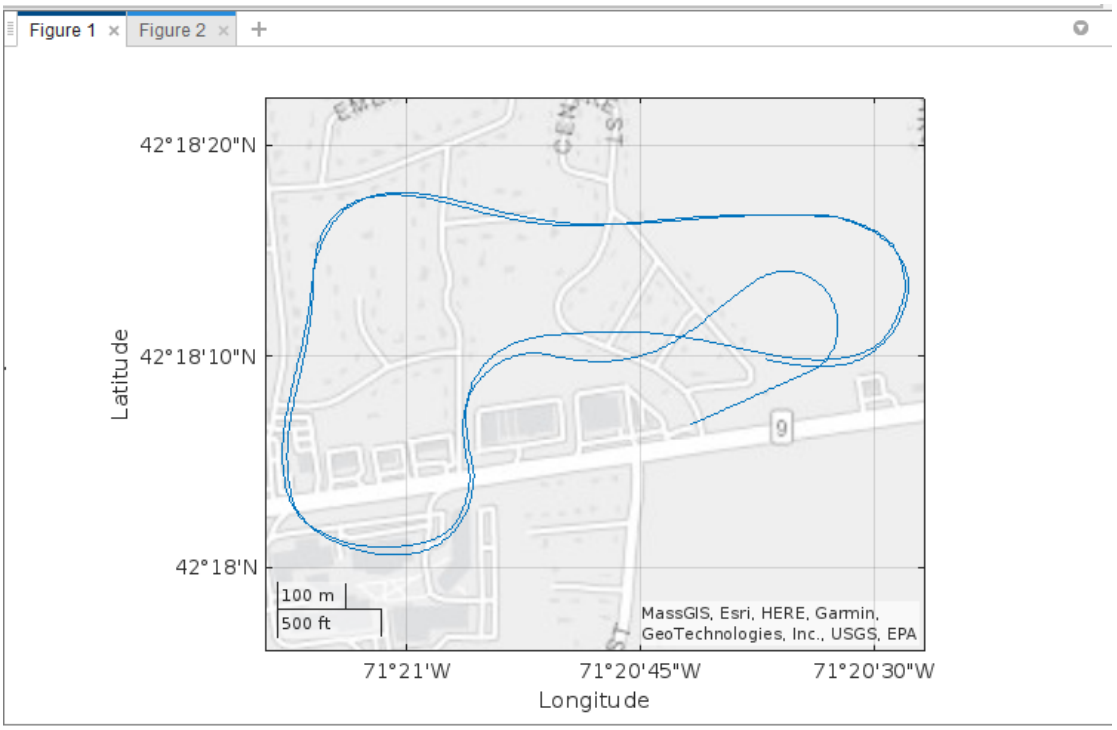


Εικόνα 16. Σχεδίαση γραφικής παράστασης με την stackedplot

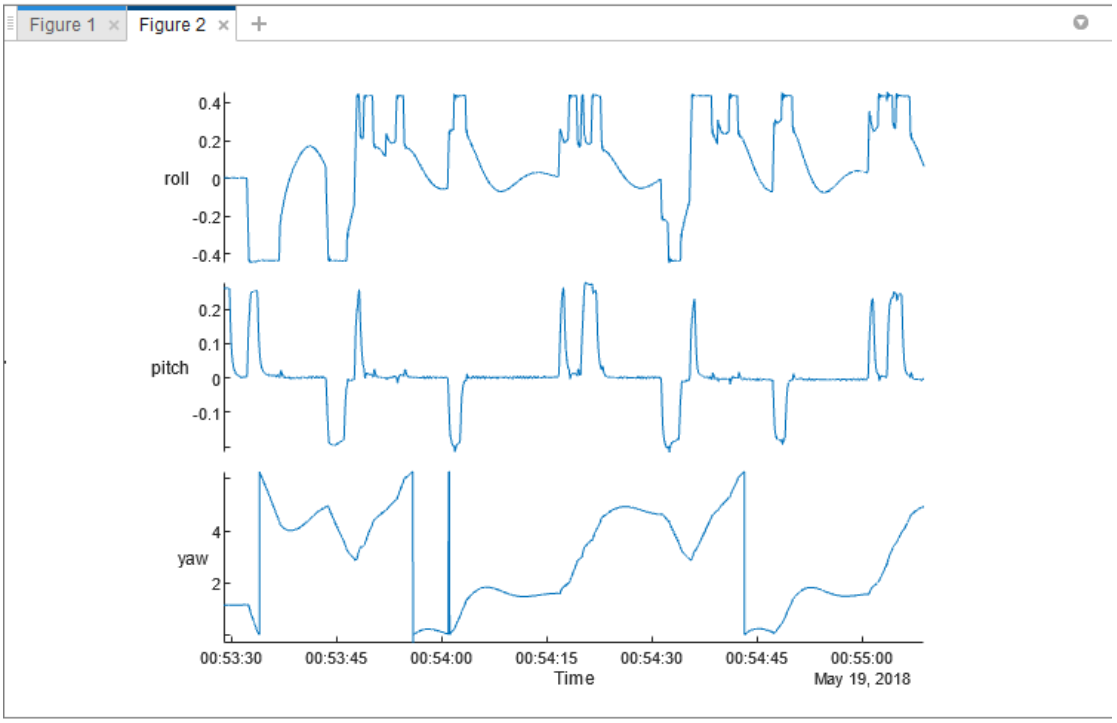
```
Command Window
>> dialect = mavlinkdialect('common.xml');
>> logimport = mavlinktlog('mavlink_flightlog.tlog',dialect);
>> msgs = readmsg(logimport, 'MessageName', 'GPS_RAW_INT', ...
                          'Time',[0 100]);
>> latlon = msgs.Messages{1};
>> latlon = latlon(latlon.lat ~= 0 & latlon.lon ~= 0, :);
>> figure()
geoplot(double(latlon.lat)/1e7, double(latlon.lon)/1e7);
>> msgs = readmsg(logimport,'MessageName','ATTITUDE','Time',[0 100]);
>> figure()
stackedplot(msgs.Messages{1},{'roll','pitch','yaw'});
>>
```

Εικόνα 17. Command Window

Στις παρακάτω εικόνες βλέπουμε τα 2 διαδοχικά plots.



Εικόνα 18. Το plot της εικόνας 15



Εικόνα 19. Το plot της εικόνας 16

2.7.4 Αναπαραγωγή εισόδων MAVLink Log

Δημιουργούμε ένα **interface** επικοινωνίας MAVLink και εμφανίζουμε τα μηνύματα από το tlog στην θύρα UDP που έχει οριστεί από τον χρήστη. Ορίζουμε έναν πομπό και έναν δέκτη για την μεταφορά αυτών των μηνυμάτων. Αυτή η επικοινωνία αποτελεί μια ακριβή προσομοίωση hardware. Γίνεται με τον ίδιο τρόπο επικοινωνίας χρησιμοποιώντας τα πρωτόκολλα επικοινωνίας MAVLink.

```
sender = mavlinkio(dialect,'SystemID',1,'ComponentID',1,...
'AutopilotType',"MAV_AUTOPILOT_GENERIC",...
'ComponentType',"MAV_TYPE_QUADROTOR");
connect(sender,'UDP');
destinationPort = 14550;
destinationHost = '127.0.0.1';
receiver = mavlinkio(dialect);
connect(receiver,'UDP','LocalPort',destinationPort);
subscriber =
mavlinksub(receiver,'ATTITUDE','NewMessageFcn',@(~,msg)disp(msg.Payload));
```

Στέλνουμε 3 μηνύματα με συχνότητα 50Hz.

```
>> sender = mavlinkio(dialect,'SystemID',1,'ComponentID',1,...
                  'AutopilotType',"MAV_AUTOPILOT_GENERIC",...
                  'ComponentType',"MAV_TYPE_QUADROTOR");
>> connect(sender,'UDP');
>> destinationPort = 14550;
>> destinationHost = '127.0.0.1';
>> receiver = mavlinkio(dialect);
>> connect(receiver,'UDP','LocalPort',destinationPort);
>> subscriber = mavlinksub(receiver,'ATTITUDE','NewMessageFcn',@(~,msg)disp(msg.Payload));
>>
>> payloads = table2struct(msgs.Messages{1});
>> attitudeDefinition = msginfo(dialect, 'ATTITUDE');
>> for msgIdx = 1:100
    sendudpmsg(sender,struct('MsgID', attitudeDefinition.MessageID, 'Payload', payloads(msgIdx)),destinationHost,destinationPort);
    pause(1/50);
end
    time_boot_ms: 0
            roll: 0.0025
           pitch: 0.2619
             yaw: 1.1516
       rollspeed: 0
      pitchspeed: 0
        yawspeed: 0

    time_boot_ms: 0
            roll: 0.0023
           pitch: 0.2617
4 usages of "scene" found
```

Εικόνα 20. Αποστολή μηνυμάτων με συχνότητα 50Hz

payloads = table2struct(msgs.Messages{1});
attitudeDefinition = msginfo(dialect, 'ATTITUDE');
for msgIdx = 1:3
sendudpmsg(sender,struct('MsgID', attitudeDefinition.MessageID, 'Payload',
payloads(msgIdx)),destinationHost,destinationPort);
pause(1/50);
end

```
>> for msgIdx = 1:3
    sendudpmsg(sender,struct('MsgID', attitudeDefinition.MessageID, 'Payload', payloads(msgIdx)),destinationHost,destinationPort);
    pause(1/50);
end
    time_boot_ms: 0
            roll: 0.0025
           pitch: 0.2619
             yaw: 1.1516
       rollspeed: 0
      pitchspeed: 0
        yawspeed: 0

    time_boot_ms: 0
            roll: 0.0027
           pitch: 0.2619
             yaw: 1.1514
       rollspeed: 0
      pitchspeed: 0
        yawspeed: 0

    time_boot_ms: 0
            roll: 0.0023
           pitch: 0.2617
             yaw: 1.1516
       rollspeed: 0
      pitchspeed: 0
        yawspeed: 0
```

Εικόνα 21. Έξοδος της for που χρησιμοποιήσαμε

Για να αποσυνδεθούμε από τα interfaces του MAVLink χρησιμοποιούμε τις εντολές:
disconnect(receiver)
disconnect(sender)

2.7.5 Οπτικοποίηση προσαρμοσμένου Flight Log

Διαμορφώνουμε το αντικείμενο **flightLogSignalMapping** για την οπτικοποίηση (visualize) δεδομένων από ένα αρχείο flight log.

Φόρτωση του custom Flight Log:

Σε αυτό το παράδειγμα θεωρούμε ότι τα δεδομένα πτήσης έχουν ήδη αναλυθεί στο Matlab, ως εκ τούτου είναι ήδη σε μορφή .mat. Θα δούμε πως μπορούμε να διαμορφώσουμε το αντικείμενο **flightLogSignalMapping** ώστε να μπορεί να χειριστεί τα δεδομένα log που έχουμε αποθηκεύσει στο .mat αρχείο και να τα οπτικοποιήσει. Το αντικείμενο **customFlightData.mat** αποθηκεύει μια δομή (structure) που περιλαμβάνει 3 πεδία.

- Η **Fs** είναι η δειγματοληπτική συχνότητα σημάτων από τα σήματα που βρίσκονται αποθηκευμένα στο .mat αρχείο μας.
- Τα **IMU** και **Trajectory** είναι πίνακες που περιλαμβάνουν πληροφορίες πραγματικών πτήσεων.
- Τα **IMU Data** και **Trajectory Data** αντίθετα περιλαμβάνουν δεδομένα τα οποία προέρχονται από προσομοιώσεις πτήσεων σε ένα προεπιλεγμένο ορθογώνιο μονοπάτι.

```
customData = load("customFlightData.mat");
logData = customData.logData

logData =

struct with fields:

IMU: [2785×9 double]
Fs: 100
Trajectory: [2785×10 double]
```

Το πεδίο IMU στο logdata είναι ένας πίνακας n-by-9, όπου οι πρώτες 3 στήλες είναι οι ενδείξεις του επιταχυνσιόμετρου σε m/s2.

Οι επόμενες 3 στήλες είναι οι ενδείξεις του γυροσκόπιου σε rad/s, και οι 3 τελευταίες στήλες είναι οι ενδείξεις του μαγνητόμετρου σε μT.

```
logData.IMU(1:5, :)

ans =

0.8208 0.7968 10.7424 0.0862 0.0873 0.0862 327.6000 297.6000 283.8000
0.8016 0.8160 10.7904 0.0883 0.0873 0.0862 327.6000 297.6000 283.8000
0.7680 0.7680 10.7568 0.0862 0.0851 0.0851 327.6000 297.6000 283.8000
0.8208 0.7536 10.7520 0.0873 0.0883 0.0819 327.6000 297.6000 283.8000
0.7872 0.7728 10.7328 0.0873 0.0862 0.0830 327.6000 297.6000 283.8000
```

Το πεδίο Trajectory στο logData είναι ένας πίνακας n-by-9, με τις πρώτες 3 στήλες να είναι οι συντεταγμένες XYZ NED σε m. Οι επόμενες 3 στήλες είναι η ταχύτητα προς την κατεύθυνση XYZ NED σε m/s και οι τελευταίες 4 στήλες είναι τα quaternions που περιγράφουν την περιστροφή του UAV από το πλαίσιο αδράνειας NED στο πλαίσιο σώματος. Κάθε σειρά είναι ένα μόνο σημείο της τροχιάς με όλες αυτές τις παραμέτρους καθορισμένες.

```
logData.Trajectory(1:5,:)

ans =

0.0200 0 -4.0000 2.0000 0 -0.0036 1.0000 0 0 -0.0000
0.0400 0 -4.0001 2.0000 0 -0.0072 1.0000 0 0 -0.0000
0.0600 0 -4.0002 2.0000 0 -0.0108 1.0000 0 0 -0.0000
0.0800 0 -4.0003 2.0000 0 -0.0143 1.0000 0 0 -0.0000
0.1000 0 -4.0004 2.0000 0 -0.0179 1.0000 0 0 -0.0001
```

2.7.6 Οπτικοποίηση του custom Flight Log με χρήση προκαθορισμένης μορφής σήματος και γραφική απεικόνιση

Δημιουργούμε ένα flightLogSignalMapping αντικείμενο χωρίς ορισμό input argument, δεδομένου ότι το custom log format που χρησιμοποιούμε δεν ακολουθεί ulog ή tlog ορισμούς.

```
customPlotter = flightLogSignalMapping;
info(customPlotter, "Signal")

ans =

18×4 table

SignalName  IsMapped  SignalFields  FieldUnits
_____________________  ________  ____________________________________________________________
_____________________  ________  ____________________________________________________________
_____________________  ________  ____________________________________________________________
_____________________  ________  ____________________________________________________________
_  ____________________  ________  ____________________________________________________________

"Accel#" false "AccelX, AccelY, AccelZ" "m/s^2, m/s^2, m/s^2"
"Airspeed#" false "PressDiff, IndicatedAirSpeed, Temperature" "Pa, m/s, degreeC"
"AttitudeEuler" false "Roll, Pitch, Yaw" "rad, rad, rad"
"AttitudeRate" false "BodyRotationRateX, BodyRotationRateY, BodyRotationRateZ"
"rad/s, rad/s, rad/s"
"AttitudeTargetEuler" false "RollTarget, PitchTarget, YawTarget" "rad, rad, rad"
"Barometer#" false "PressAbs, PressAltitude, Temperature" "Pa, m, degreeC"
"Battery" false "Voltage_1, Voltage_2, Voltage_3, Voltage_4, Voltage_5, Voltage_6,
Voltage_7, Voltage_8, Voltage_9, Voltage_10, Voltage_11, Voltage_12, Voltage_13,
Voltage_14, Voltage_15, Voltage_16, RemainingCapacity" "v, v, v, v, v, v, v, v, v, v, v,
v, v, v, v, v, %"
"GPS#" false "Latitude, Longitude, Altitude, GroundSpeed, CourseAngle,
SatellitesVisible" "degree, degree, m, m/s, degree, N/A"
"Gyro#" false "GyroX, GyroY, GyroZ" "rad/s, rad/s, rad/s"
"LocalENU" false "X, Y, Z" "m, m, m"
"LocalENUTarget" false "XTarget, YTarget, ZTarget" "m, m, m"
"LocalENUVel" false "VX, VY, VZ" "m/s, m/s, m/s"
"LocalENUVelTarget" false "VXTarget, VYTarget, VZTarget" "m/s, m/s, m/s"
```

"LocalNED" false "X, Y, Z" "m, m, m"
"LocalNEDTarget" false "XTarget, YTarget, ZTarget" "m, m, m"
"LocalNEDVel" false "VX, VY, VZ" "m/s, m/s, m/s"
"LocalNEDVelTarget" false "VXTarget, VYTarget, VZTarget" "m/s, m/s, m/s"
"Mag#" false "MagX, MagY, MagZ" "Gs, Gs, Gs"

info(customPlotter,"Plot")

ans =

10×4 table

PlotName ReadyToPlot MissingSignals RequiredSignals
_____ _____ _____ _____

"Attitude" false "AttitudeEuler, AttitudeRate, Gyro#" "AttitudeEuler, AttitudeRate, Gyro#"
"AttitudeControl" false "AttitudeEuler, AttitudeTargetEuler" "AttitudeEuler, AttitudeTargetEuler"
"Battery" false "Battery" "Battery"
"Compass" false "AttitudeEuler, Mag#, GPS#" "AttitudeEuler, Mag#, GPS#"
"GPS2D" false "GPS#" "GPS#"
"Height" false "Barometer#, GPS#, LocalNED" "Barometer#, GPS#, LocalNED"
"Speed" false "GPS#, Airspeed#" "GPS#, Airspeed#"
"Trajectory" false "LocalNED, LocalNEDTarget" "LocalNED, LocalNEDTarget"
"TrajectoryTracking" false "LocalNED, LocalNEDTarget" "LocalNED, LocalNEDTarget"
"TrajectoryVelTracking" false "LocalNEDVel, LocalNEDVelTarget" "LocalNEDVel, LocalNEDVelTarget"

Το αντικείμενο flightLogSignalMapping πρέπει να γνωρίζει πώς αποθηκεύονται τα δεδομένα στο αρχείο καταγραφής πτήσης ώστε να μπορέσει να οπτικοποιήσει τα δεδομένα. Για να συσχετίσουμε τα ονόματα των σημάτων με λαβές συναρτήσεων που έχουν πρόσβαση στις σχετικές πληροφορίες στα logData, πρέπει να αντιστοιχίσουμε τα σήματα χρησιμοποιώντας το mapSignal. Κάθε σήμα ορίζεται ως ένα χρονικό σημείο σε ένα διάνυσμα (timestamp) και ένας πίνακας τιμών σήματος.

Για να αντιστοιχίσουμε το σήμα Gyro#, ορίζουμε μια συνάρτηση timeAccess σύμφωνα με την συχνότητα δειγματοληψίας των δεδομένων του αισθητήρα.

timeAccess = @(x)seconds(1/x.Fs*(1:size(x.IMU)));

Στην συνέχεια ελέγχουμε ποιά πεδία πρέπει να οριστούν.

info(customPlotter,"Signal","Gyro#")

ans =

1×4 table

SignalName IsMapped SignalFields FieldUnits
_____ _____ _____ _____

"Gyro#" false "GyroX, GyroY, GyroZ" "rad/s, rad/s, rad/s"

Το σήμα GyroX χρειάζεται 3 στήλες που περιέχουν τα στοιχεία του γυροσκόπιου για τους άξονες XYZ..

```
gyroAccess = @(x)x.IMU(:,4:6);
mapSignal(customPlotter,"Gyro",timeAccess,gyroAccess);
```

Ομοίως, αντιστοιχίζουμε και τα άλλα προκαθορισμένα σήματα για δεδομένα που υπάρχουν στο αρχείο του flight log. Καθορίζουμε τα function handles των τιμών για τα δεδομένα. Στη συνέχεια αντιστοιχίζουμε τα σήματα χρησιμοποιώντας την ίδια διανυσματική συνάρτηση timestamp, timeAccess.

Τα δεδομένα IMU αποθηκεύουν τα δεδομένα του accelerometer και του magnetometer:

```
accelAccess = @(x)x.IMU(:,1:3);
magAccess = @(x)x.IMU(:,7:9)*1e-2;
```

Την πτήση trajectory σε τοπικές συντεταγμένες NED και τις συντεταγμένες σε συνάρτηση των αξόνων XYZ:

```
nedAccess = @(x)x.Trajectory(:, 1:3);
```

Την ταχύτητα σε συνάρτηση με τους άξονες XYZ:

```
nedVelAccess = @(x)x.Trajectory(:, 4:6);
```

Περιστροφές Roll Pitch Yaw που μετατράπηκαν από τα quaternion

```
attitudeAccess = @(x)flip(quat2eul(x.Trajectory(:, 7:10)),2);
```

Διαμόρφωση του flightLogSignalMapping για προσαρμοσμένα δεδομένα

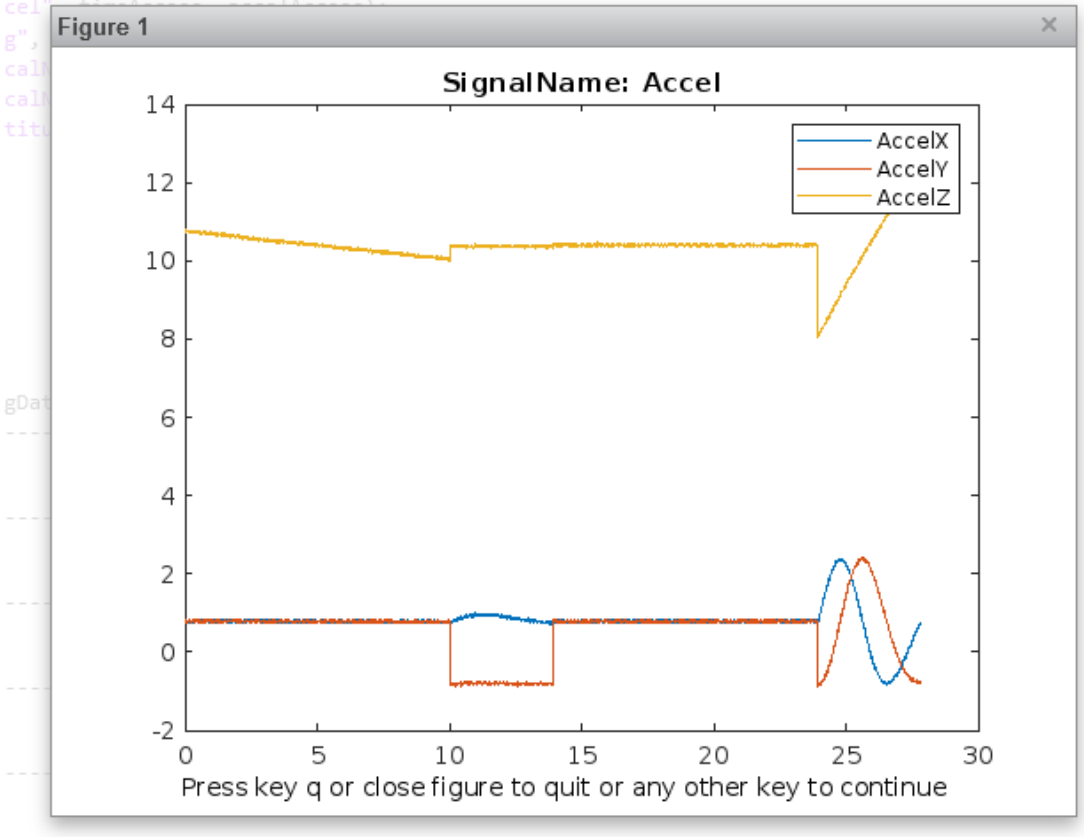
```
mapSignal(customPlotter, "Accel", timeAccess, accelAccess);
mapSignal(customPlotter, "Mag", timeAccess, magAccess);
mapSignal(customPlotter, "LocalNED", timeAccess, nedAccess);
mapSignal(customPlotter, "LocalNEDVel", timeAccess, nedVelAccess);
mapSignal(customPlotter, "AttitudeEuler", timeAccess, attitudeAccess);
```

Μόλις αντιστοιχιστούν όλα τα σήματα, το customPlotter είναι έτοιμο να δημιουργήσει γραφικά με βάση τα δεδομένα σήματος που είναι αποθηκευμένα στο αρχείο καταγραφής. Για να ελέγξουμε γρήγορα εάν τα σήματα έχουν αντιστοιχιστεί σωστά, καλούμε τις συναρτήσεις checkSignal και καθορίζουμε τα logData:

```
checkSignal(customPlotter,logData);
-----------------------------------------
SignalName: Gyro
Pass
-----------------------------------------
SignalName: Accel
Pass
-----------------------------------------
SignalName: Mag
Pass
-----------------------------------------
SignalName: LocalNED
Pass
-----------------------------------------
SignalName: LocalNEDVel
Pass
-----------------------------------------
SignalName: AttitudeEuler
Pass
```

Για να λάβουμε μια προεπισκόπηση ενός αντιστοιχισμένου σήματος, διαλέγουμε την επιλογή προεπισκόπησης στο checkSignal:

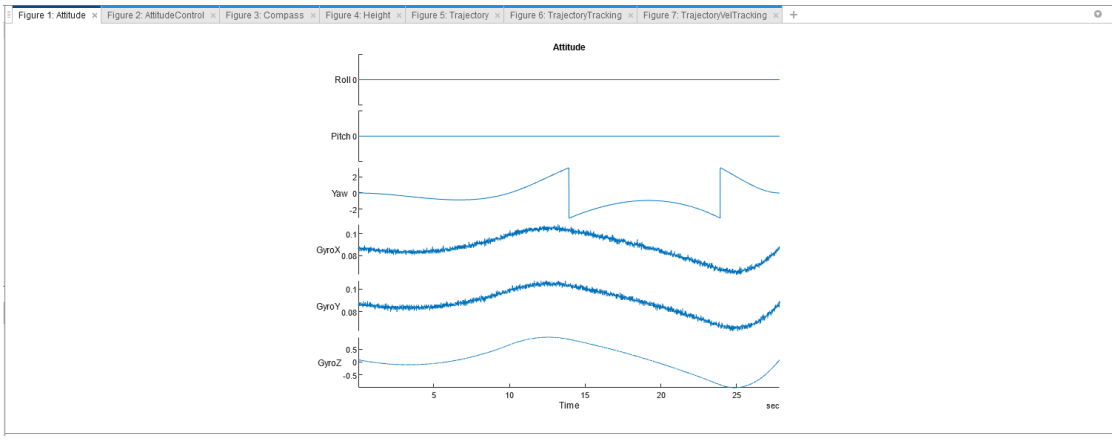
```
checkSignal(customPlotter,logData,'Preview',"on",'Signal',"Accel");
-----------------------------------------
SignalName: Accel
Pass
Press a key to continue or 'q' to quit. Figure needs to be in focus.
```



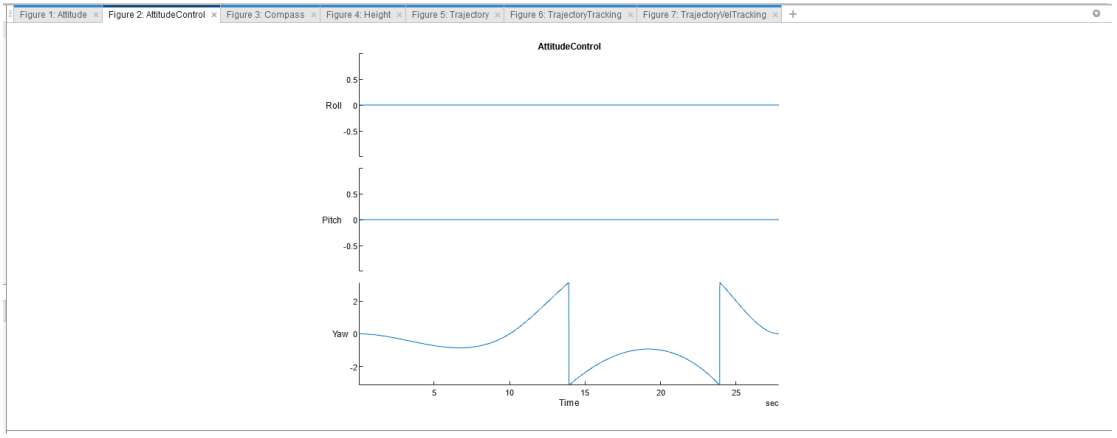
Εικόνα 22. Γραφική παράσταση του Accel

Για να οπτικοποιήσουμε τα δεδομένα του αρχείου καταγραφής πτήσης, καλούμε την show και καθορίζουμε τα logData. Όλα τα διαθέσιμα διαγράμματα με βάση τα χαρτογραφημένα σήματα φαίνονται στα παρακάτω σχήματα.

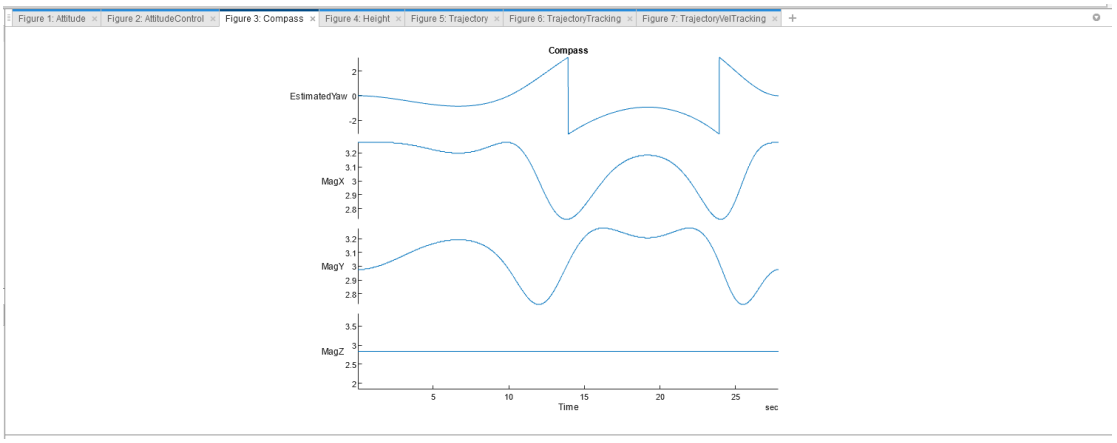
predefinedPlots = show(customPlotter,logData);



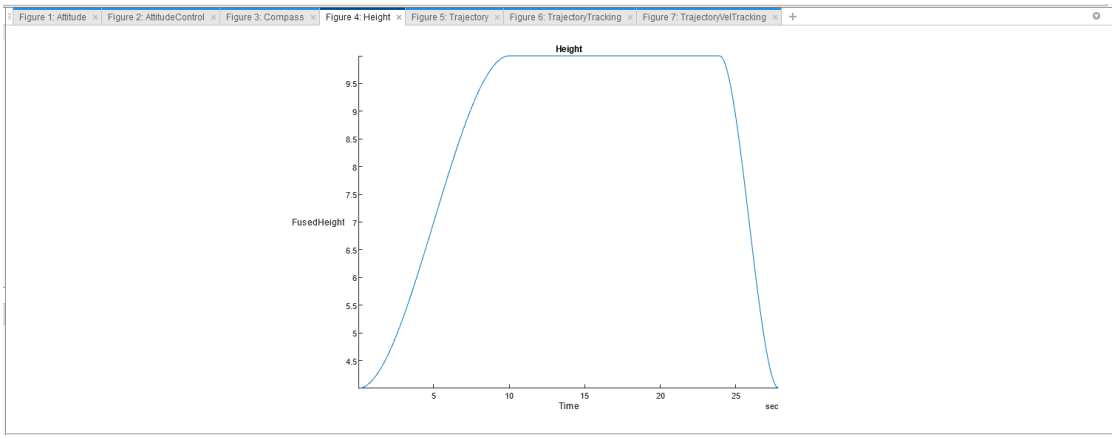
Εικόνα 23. Έξοδος της show για τα logData



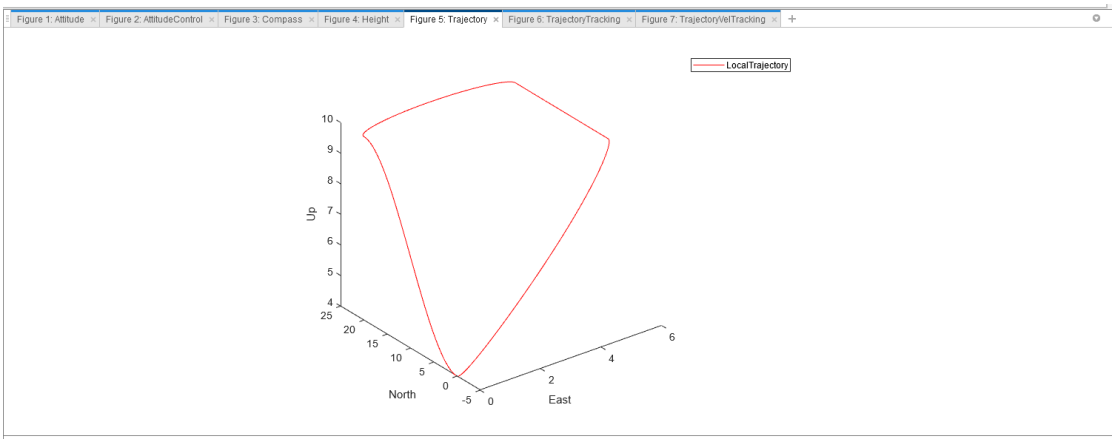
Εικόνα 24. Έξοδος της show για τα logData



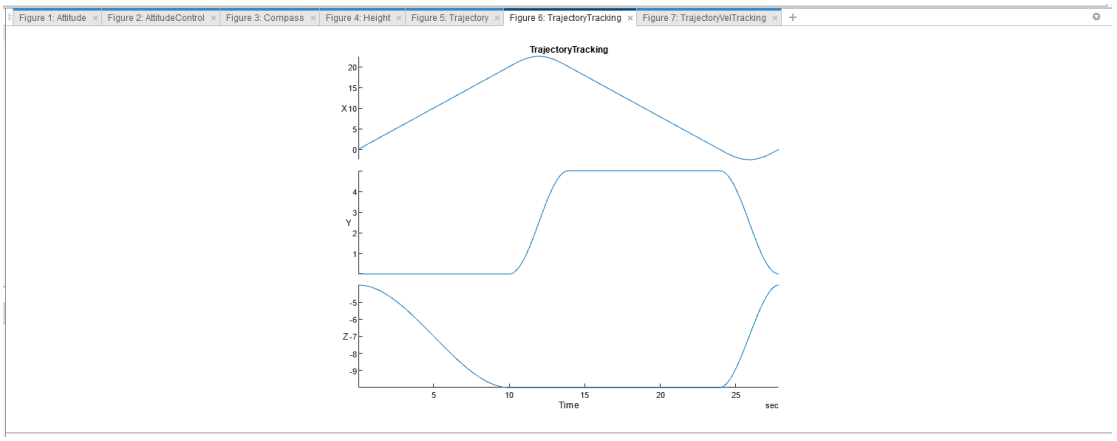
Εικόνα 25. Έξοδος της show για τα logData



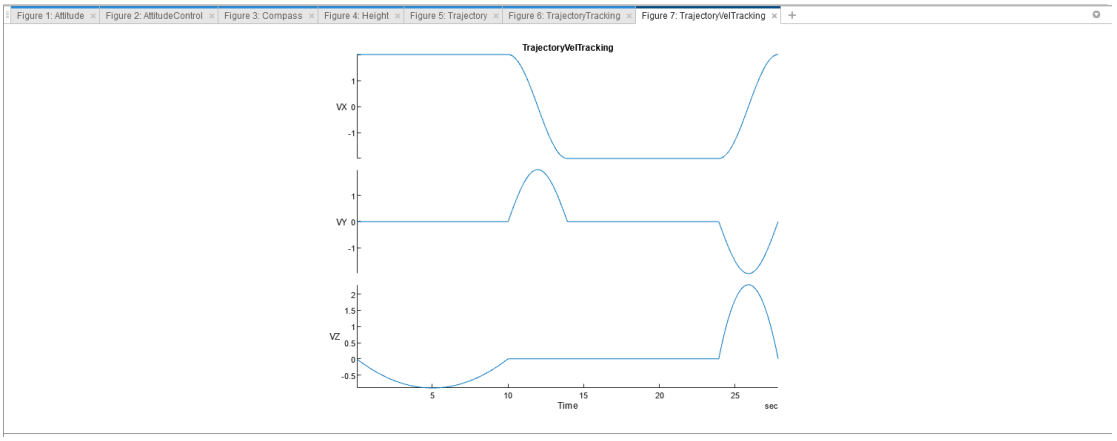
Εικόνα 26. Έξοδος της show για τα logData.Έξοδος της show για τα logData



Εικόνα 27. Έξοδος της show για τα logData



Εικόνα 28.Έξοδος της show για τα logData



Εικόνα 29. Έξοδος της show για τα logData

2.7.7 Οπτικοποίηση και custom Flight Log - Custom Plot

Για ανάλυση του αρχείου καταγραφής λεπτομερειών mod, ορίζουμε περισσότερα σήματα και προσθέτουμε περισσότερα διαγράμματα εκτός από τα προκαθορισμένα που είναι αποθηκευμένα στο flightLogSignalMapping.

Ορίζουμε ένα function handle που φιλτράρει επιταχύνσεις μεγαλύτερες από 1:

```
accelThreshold = @(x)(vecnorm(accelAccess(x)')>11)';
mapSignal(customPlotter, "HighAccel", timeAccess,accelThreshold, "AccelGreaterThan11", "N/A");
```

Καλούμε την updatePlot για να προσθέσουμε προσαρμοσμένα διαγράμματα. Καθορίζουμε το αντικείμενο σχεδίου του flightLog και ένα όνομα για την γραφική παράσταση ως πρώτο και ως δεύτερο όρισμα. Για να καθορίσουμε μια χρονική σειρά δεδομένων, χρησιμοποιούμε το "Timeseries" ως τρίτο όρισμα και στην συνέχεια καταχωρούμε τα δεδομένα:

```
updatePlot(customPlotter, "AnalyzeAccel","Timeseries",["HighAccel.AccelGreaterThan11", "LocalNEDVel.VX", "LocalNEDVel.VY", "LocalNEDVel.VZ"]);
```

Ορίζουμε ένα custom function handle για την δημιουργία ενός figure handle. Αυτή η συνάρτηση δημιουργεί ένα περιοδόγραμμα χρησιμοποιώντας το FFT και άλλες συναρτήσεις στα δεδομένα επιτάχυνσης και τα σχεδιάζει. Η συνάρτηση επιστρέφει μια handle συνάρτησης:

```
updatePlot(customPlotter, "plotFFTAccel",@(acc)plotFFTAccel(acc),"Accel");
```

Ελέγχουμε ότι το customPlotter περιέχει τώρα ένα νέο σήμα και δύο νέες γραφικές χρησιμοποιώντας την info:

```
info(customPlotter, "Signal")

ans =

19×4 table

SignalName  IsMapped  SignalFields  FieldUnits
__________________  ________
_____________________________________________________________________
_____________________________________________________________________
_____________________________________________________________________
_  ____________________________________________________

"Accel" true "AccelX, AccelY, AccelZ" "m/s^2, m/s^2, m/s^2"
"AttitudeEuler" true "Roll, Pitch, Yaw" "rad, rad, rad"
"Gyro" true "GyroX, GyroY, GyroZ" "rad/s, rad/s, rad/s"
"HighAccel" true "AccelGreaterThan11" "N/A"
"LocalNED" true "X, Y, Z" "m, m, m"
"LocalNEDVel" true "VX, VY, VZ" "m/s, m/s, m/s"
"Mag" true "MagX, MagY, MagZ" "Gs, Gs, Gs"
"Airspeed#" false "PressDiff, IndicatedAirSpeed, Temperature" "Pa, m/s, degreeC"
```

"AttitudeRate" false "BodyRotationRateX, BodyRotationRateY, BodyRotationRateZ" "rad/s, rad/s, rad/s"
"AttitudeTargetEuler" false "RollTarget, PitchTarget, YawTarget" "rad, rad, rad"
"Barometer#" false "PressAbs, PressAltitude, Temperature" "Pa, m, degreeC"
"Battery" false "Voltage_1, Voltage_2, Voltage_3, Voltage_4, Voltage_5, Voltage_6, Voltage_7, Voltage_8, Voltage_9, Voltage_10, Voltage_11, Voltage_12, Voltage_13, Voltage_14, Voltage_15, Voltage_16, RemainingCapacity" "v, v, v, v, v, v, v, v, v, v, v, v, v, v, v, v, %"
"GPS#" false "Latitude, Longitude, Altitude, GroundSpeed, CourseAngle, SatellitesVisible" "degree, degree, m, m/s, degree, N/A"
"LocalENU" false "X, Y, Z" "m, m, m"
"LocalENUTarget" false "XTarget, YTarget, ZTarget" "m, m, m"
"LocalENUVel" false "VX, VY, VZ" "m/s, m/s, m/s"
"LocalENUVelTarget" false "VXTarget, VYTarget, VZTarget" "m/s, m/s, m/s"
"LocalNEDTarget" false "XTarget, YTarget, ZTarget" "m, m, m"
"LocalNEDVelTarget" false "VXTarget, VYTarget, VZTarget" "m/s, m/s, m/s"

info(customPlotter, "Plot")

ans =

12×4 table

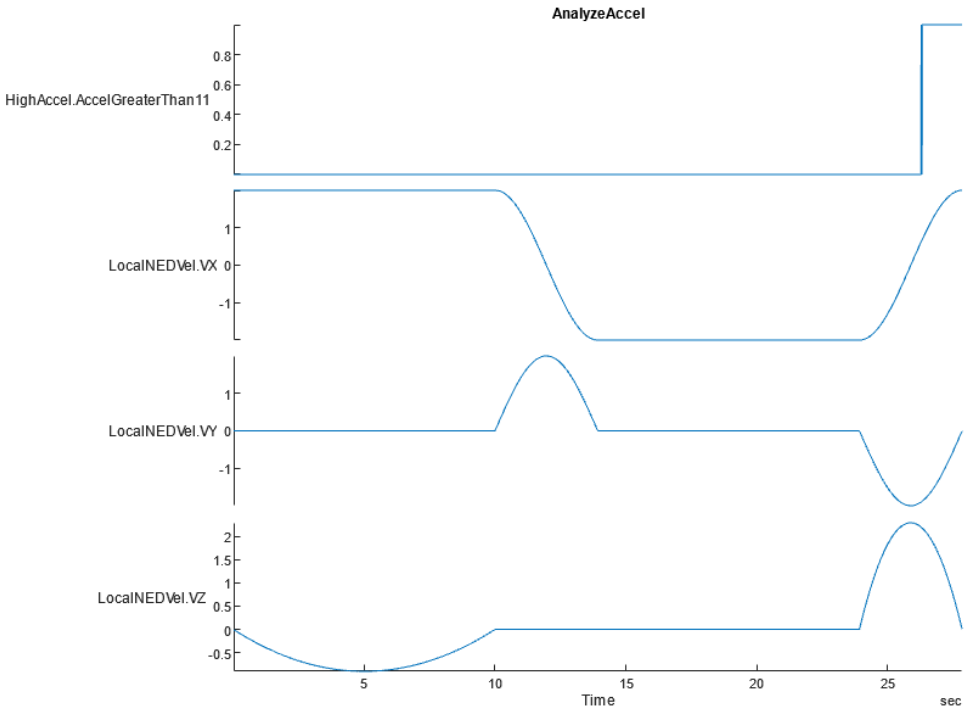
PlotName ReadyToPlot MissingSignals RequiredSignals
_____ _____ _____ _____

"AnalyzeAccel" true "" "HighAccel, LocalNEDVel"
"Attitude" true "AttitudeRate" "AttitudeEuler, AttitudeRate, Gyro#"
"AttitudeControl" true "AttitudeTargetEuler" "AttitudeEuler, AttitudeTargetEuler"
"Compass" true "GPS#" "AttitudeEuler, Mag#, GPS#"
"Height" true "Barometer#, GPS#" "Barometer#, GPS#, LocalNED"
"Trajectory" true "LocalNEDTarget" "LocalNED, LocalNEDTarget"
"TrajectoryTracking" true "LocalNEDTarget" "LocalNED, LocalNEDTarget"
"TrajectoryVelTracking" true "LocalNEDVelTarget" "LocalNEDVel, LocalNEDVelTarget"
"plotFFTAccel" true "" "Accel"
"Battery" false "Battery" "Battery"
"GPS2D" false "GPS#" "GPS#"
"Speed" false "GPS#, Airspeed#" "GPS#, Airspeed#"

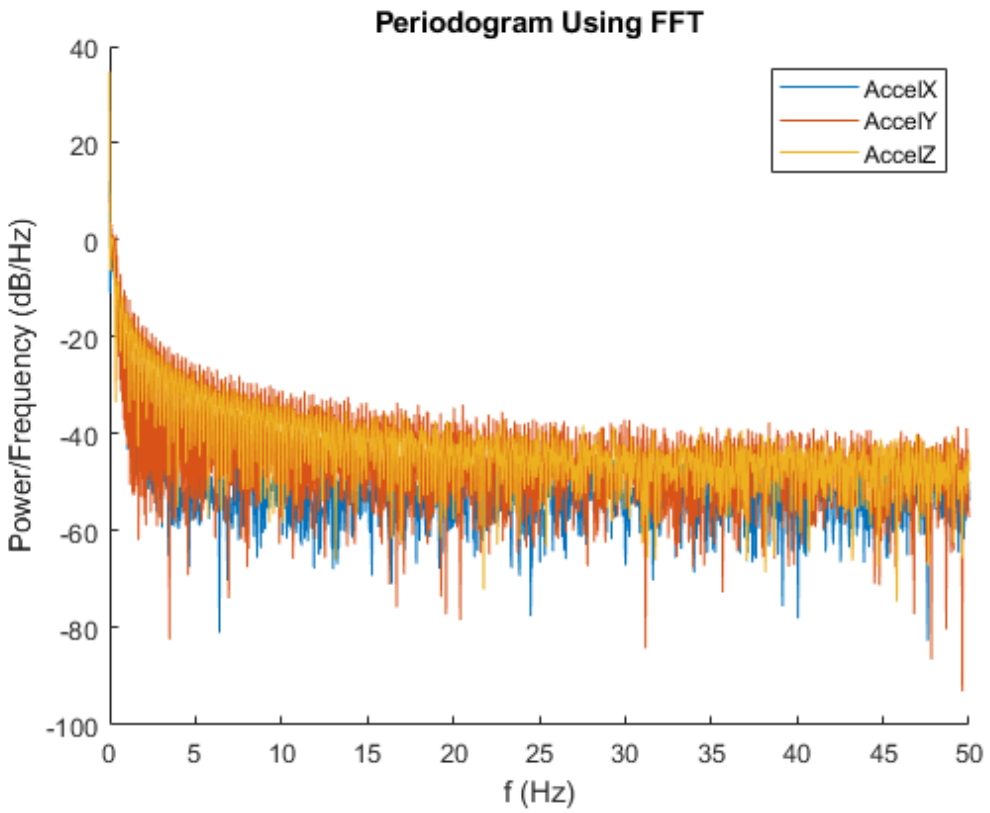
Καθορίζουμε ποια ονόματα γραφικών θέλουμε να σχεδιάσουμε. Καλούμε την call χρησιμοποιώντας "PlotsToShow" ώστε να οπτικοποιήσουμε την ανάλυση των δεδομένων επιτάχυνσης:

accelAnalysisProfile = ["AnalyzeAccel", "plotFFTAccel"];

```
accelAnalysisPlots = show(customPlotter, logData, "PlotsToShow",
accelAnalysisProfile);
```



Εικόνα 30. Γραφική με την χρήση της PlotsToShow για τα δεδομένα της επιτάχυνσης (acceleration)



Εικόνα 31. Περιοδόγραμμα με χρήση FFT (Fast Fourier Transform)

Παραπάνω είδαμε πως μπορούμε να χρησιμοποιήσουμε το αντικείμενο flightLogSignalMapping για να δούμε τα προκαθορισμένα σήματα και τις γραφικές, καθώς και να προσαρμόσουμε τις γραφικές μας για ανάλυση του αρχείου flightLog.

Analyze Acceleration Data Function Definition

```
function h = plotFFTAccel(acc)
    h = figure("Name", "AccelFFT");
    ax = newplot(h);
    v = acc.Values{1};
    Fs = v.Properties.SampleRate;
    N = floor(length(v.AccelX)/2)*2;
    hold(ax, "on");
    for idx = 1:3
        x = v{1:N, idx};
        xdft = fft(x);
        xdft = xdft(1:N/2+1);
        psdx = (1/(Fs*N)) * abs(xdft).^2;
        psdx(2:end-1) = 2*psdx(2:end-1);
        freq = 0:Fs/length(x):Fs/2;
        plot(ax, freq, 10*log10(psdx));
    end
    hold(ax, "off");
```

```
    title("Periodogram Using FFT");
    xlabel("f (Hz)");
    ylabel("Power/Frequency (dB/Hz)");
    legend("AccelX", "AccelY", "AccelZ");
end
```

2.8 Σχεδιασμός Σεναρίου UAV

Ο σχεδιασμός σεναρίου UAV (UAV Scenario Designer) δίνει την δυνατότητα δημιουργίας ενός σεναρίου με χρήση εδάφους πλατφορμών. Επίσης δίνει την δυνατότητα χρήσης αισθητήρων και του σχεδιασμού των τροχιών του UAV μας. Συνοπτικά μας δίνει τις παρακάτω δυνατότητες:

- Εισαγωγή, εξαγωγή και δημιουργία UAV Σεναρίων
- Εισαγωγή εδάφους από την Digital Terrain Elevation Data (DTED)
- Προσθήκη και επεξεργασία των εδαφών, των πλατφορμών και των αισθητήρων που εισήχθησαν στο σενάριό μας
- Προσθήκη custom εδαφών και αντικειμένων τύπου STL
- Δημιουργία και επεξεργασία των τροχιών ανά έδαφος
- Προσομοίωση του σεναρίου

2.8.1 Άνοιγμα του UAV Scenario Designer App

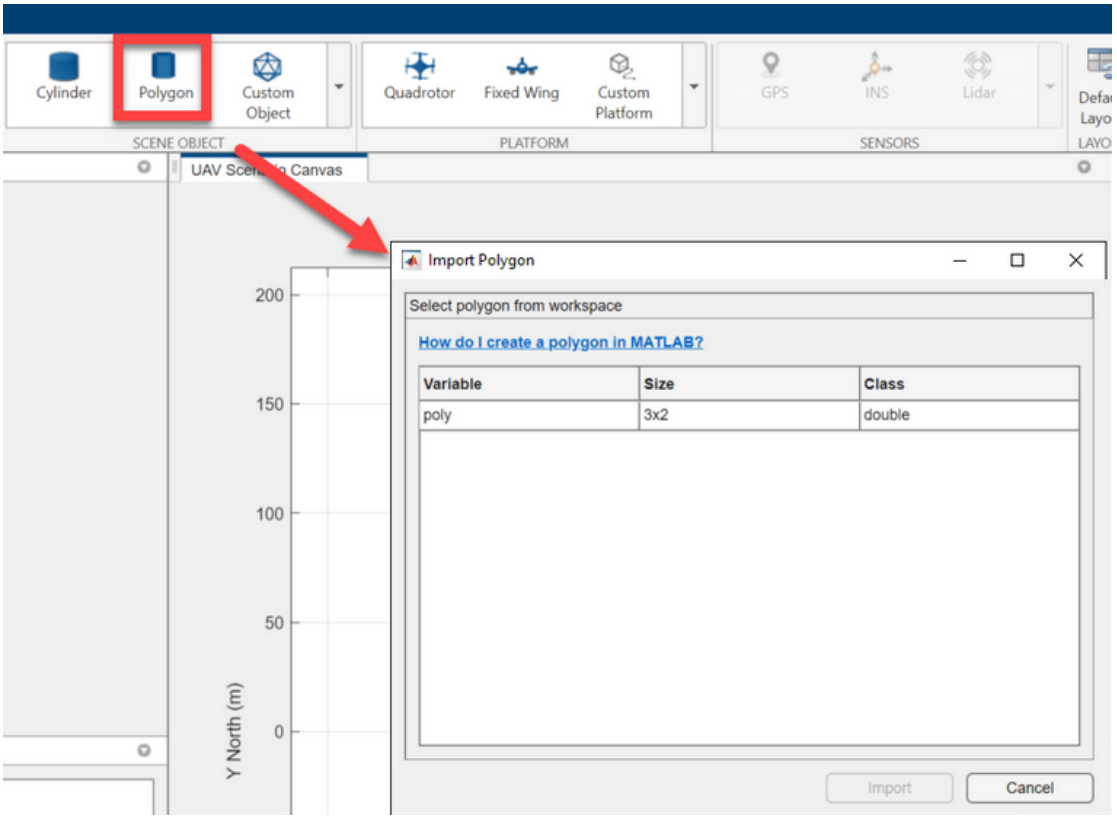
Πατάμε στην επιλογή Apps>>UAV Scenario Designer
ή μέσω του Matlab Command Prompt γράφοντας uavScenarioDesigner

Παραδείγματα:
Ανοίγουμε το UAV Scenario Designer App με την εντολή
>> uavScenarioDesigner

Ορίζουμε ένα πολύγωνο με 3 κορυφές
>> poly = [0 0; 1 1; 2 0];

Εναλλακτικός τρόπος εισαγωγής πολυγώνου:

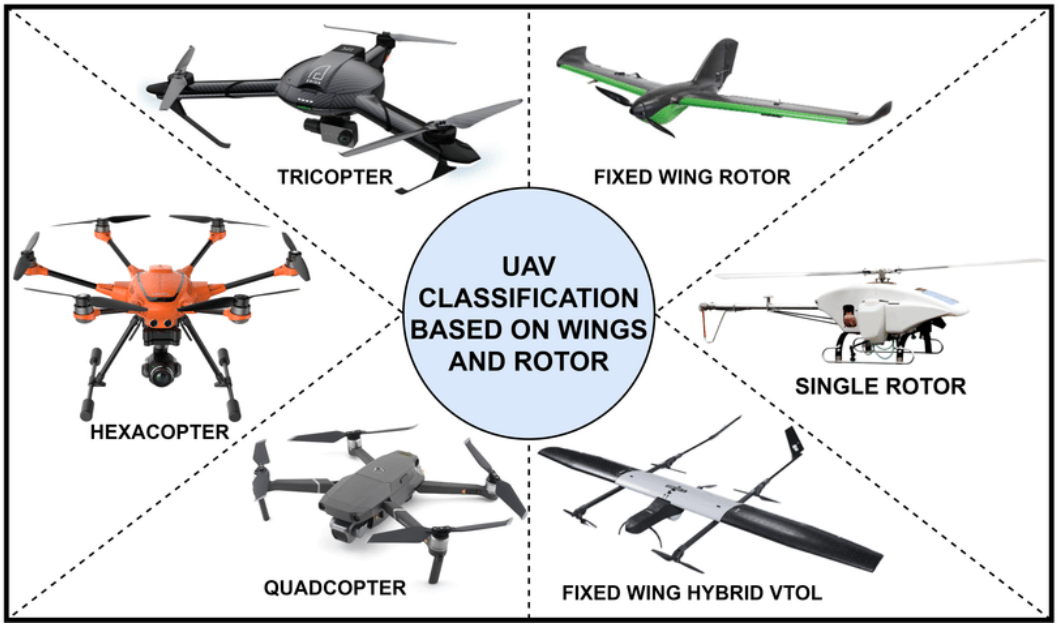
Στο **UAV Scenario Designer** επιλέγουμε από την κατηγορία **Scene Object** το **Polygon**



Εικόνα 32. Σχεδίαση Πολυγώνου στο Scenario Designer

Τα τετρακόπτερα και άλλοι τύποι ιπτάμενων UAVs είναι εξαιρετικά δημοφιλείς. Γι αυτόν τον λόγο έχουν αναπτυχθεί σε τέτοιο βαθμό ώστε να είναι σε θέση να πετάξουν και να λειτουργήσουν σχετικά αυτόνομα χωρίς μεγάλο (ή και ανύπαρκτο) βαθμό παρέμβασης από τον άνθρωπο.

Σε αυτήν την ενότητα θα αναλύσουμε τους 6 βαθμούς ελευθερίας του UAV και πως θα πρέπει να κινηθούμε ώστε να φτάσουμε στο σημείο να πετάξει.



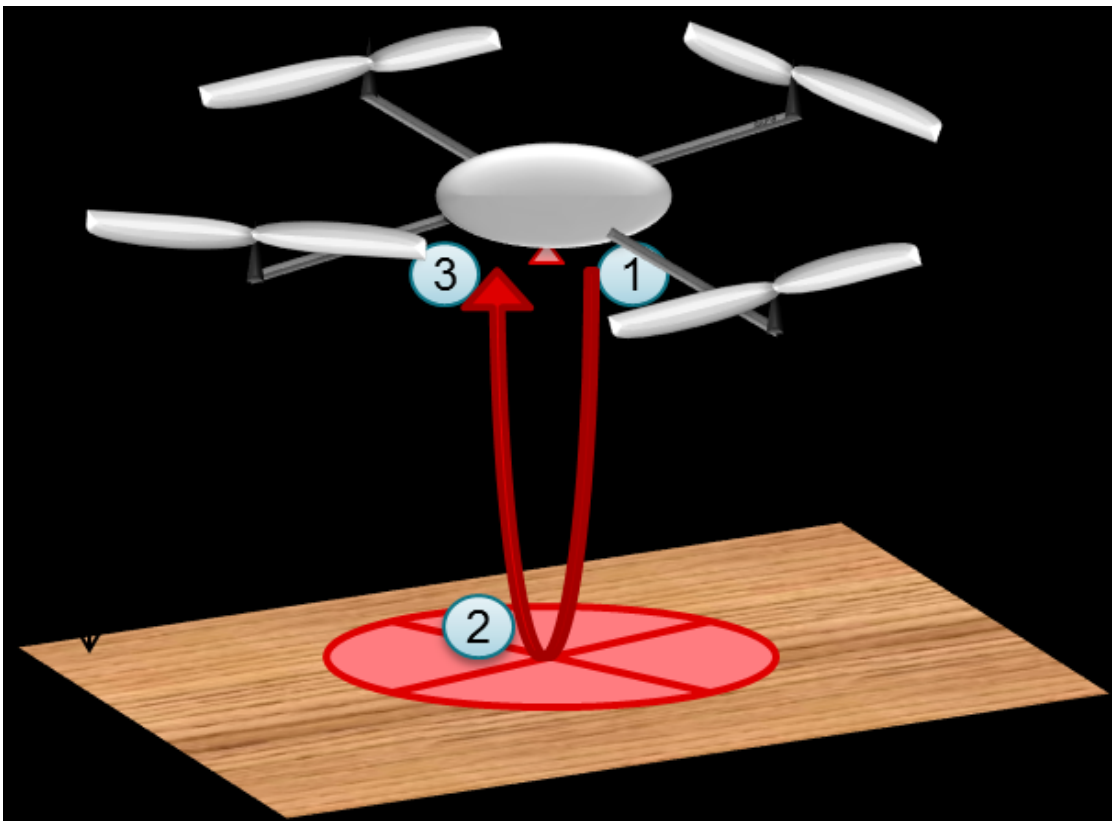
Εικόνα 33. Τύποι UAVs

Υπάρχουν διάφοροι τύποι UAVs που παίρνουν το όνομά τους βάση του σχεδιασμού τους ως προς τα μοτέρ. Με άλλα λόγια ονομάζονται με βάση τον τρόπο που θα πετάξουν και θα κινηθούν. Τα πιο συνηθισμένα που συναντάμε στην καθημερινότητα, δεδομένης και της δημοφιλίας που γνωρίζουν το τελευταίο διάστημα, είναι τα τετρακόπτερα (quadcopters). Έχουν πάρει το όνομά τους ασφαλώς από τα 4 μοτέρ που χρησιμοποιούν ώστε να κινηθούν.

Εικόνα 34. Τετρακόπτερο (Quadcopter)

Αντίστοιχα αν χρησιμοποιηθούν 6 μοτέρ ονομάζονται εξακόπτερα, με 8 οκτακόπτερα κ.α. Όλα αυτά ανήκουν στην γενικότερη κατηγορία που ονομάζεται **rotorcrafts**. Σε αυτήν την κατηγορία ανήκουν και τα ελικόπτερα κ.α.

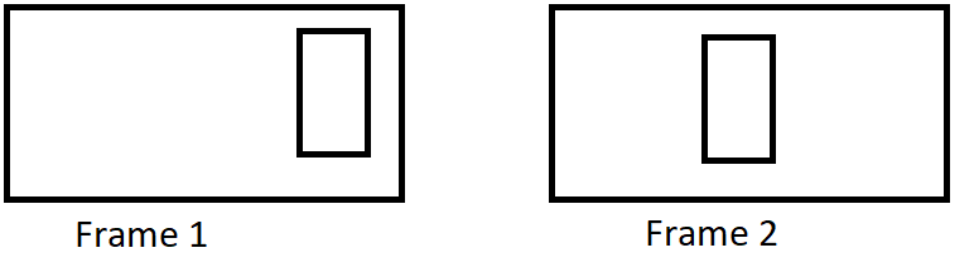
Βλέποντας αναλυτικά τα κομμάτια του, έχουμε τον **Ultrasonic Sensor** που λειτουργεί παράγοντας υπερηχητικά κύματα τα οποία επιστρέφουν στο όχημά μας γνωστοποιώντας μας την θέση άλλων αντικειμένων, όπως ακριβώς λειτουργεί το σόναρ για παράδειγμα στα οχήματα που κινούνται στην θάλασσα.



Εικόνα 35. Αναπαράσταση της λειτουργίας του Ultrasonic Sensor

2.8.2 Camera

Το UAV μας συνήθως χρησιμοποιεί μια κάμερα. Μια κάμερα, ακόμα και στραμμένη κάθετα προς το έδαφος, είναι σε θέση να μας δώσει πληροφορίες. Μπορεί να μας προσφέρει στοιχεία για την προσγείωση, αλλά ακόμη και στοιχεία για την κίνηση του drone.



Εικόνα 36. Παράδειγμα 2 διαδοχικών Frames της camera

Στο παραπάνω σχήμα βλέπουμε την κίνηση ενός uav και διαδοχικά τι “βλέπει” σε 2 ξεχωριστές καταστάσεις η κάμερα. Το αντικείμενο εντός του frame μετακινήθηκε από το αρχικό του σημείο. Επομένως είμαστε σε θέση να υπολογίσουμε εάν αυτή η μετακίνηση προήλθε μόνο από την μετακίνηση του UAV ή μετακινείτε όντως και το ίδιο το αντικείμενο.

2.8.3 Αισθητήρας πίεσης

Ένας ακόμη αισθητήρας που χρησιμοποιείται είναι αυτός της πίεσης. Η πληροφορία που μας δίνει μας βοηθάει στον υπολογισμό του υψόμετρου που βρίσκεται το UAV μας. Σε ένα χαμηλό υψόμετρο η πίεση είναι μεγαλύτερη, ενώ παίρνοντας ύψος σταδιακά μειώνεται.

2.8.4 Inertial Measurement Unit - IMU

Αυτός ο αισθητήρας (αναφέρεται σε συντόμευση ως IMU) μας δίνει την επιτάχυνση (acceleration) και τον γωνιακό ρυθμό κλίσης (angular rate). Από αυτά τα 2 στοιχεία είμαστε σε θέση να υπολογίσουμε το πόσο γρήγορα γυρνάει ένα UAV σε περίπτωση για παράδειγμα που το παρασύρει κάποιο δυνατό κύμα αέρα.

2.8.5 Motors

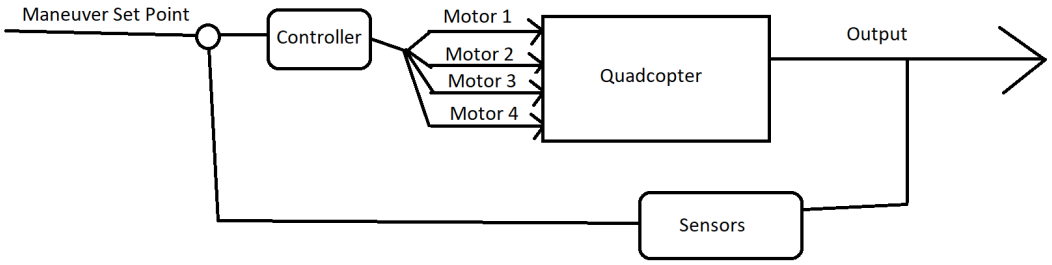
Όπως προαναφέραμε τα quadcopters διαθέτουν 4 μοτέρ. Στην παραπάνω εικόνα βλέπουμε πως βρίσκονται μεταξύ τους στον χαρακτηριστικό σχηματισμό X.. Τα μοτέρ που βρίσκονται στον ίδιο άξονα κινούνται με το ίδιο μέτρο γωνιακής ταχύτητας.

Εικόνα 37. Γωνιακή ταχύτητα των μοτέρ

Αυτό είναι απαραίτητο ώστε να είμαστε σίγουροι πως μπορούμε να ελέγξουμε πλήρως το UAV μας στις κινήσεις που θέλουμε να κάνει.

2.9 Πτήση UAV

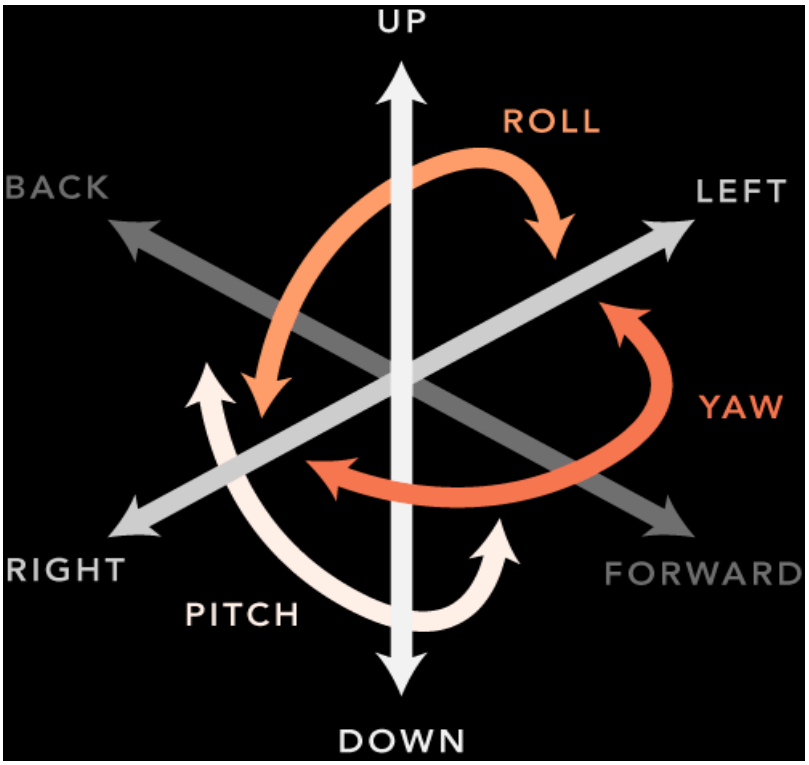
Επομένως έχουμε ένα σύστημα με τους αισθητήρες που προαναφέραμε και 4 μοτέρ. Το πρόβλημα που προκύπτει τώρα είναι πως ακριβώς θα χειριστούμε τα 4 μοτέρ προκειμένου το UAV μας να μπορεί να μετακινηθεί χωρίς πρόβλημα και να χρησιμοποιεί διαφορετικά το κάθε μοτέρ ώστε να μπορεί να πραγματοποιήσει κινήσεις.



Εικόνα 38. Σχηματικό της λειτουργίας του UAV

Στο παραπάνω σχήμα βλέπουμε αναλυτικά το πρόβλημα. Αυτό που καλούμαστε να κάνουμε ουσιαστικά είναι να αναπτύξουμε τον controller, έναν αλγόριθμο δηλαδή, ώστε να ορίσουμε την μετακίνηση των μοτέρ κατάλληλα ώστε να ανταπεξέλθουν στις αλλαγές κίνησης. Μην ξεχνάμε ότι σε μια πραγματική πτήση και όχι σε μια θεωρητική προσομοίωση, οι συνθήκες είναι σε μεγάλο βαθμό απρόβλεπτες ή άγνωστες σε εμάς. Δεν μπορούμε για παράδειγμα να είμαστε σε θέση να προβλέψουμε ξαφνικά κύματα αέρα που θα προκαλέσουν αλλαγές στην ταχύτητά μας ή και στην κατεύθυνση του UAV.

Οι **βαθμοί ελευθερίας** ορίζονται ως το σύνολο των **ανεξάρτητων μεταβλητών** που είναι σε θέση να ορίσουν την **θέση** και την **δυνατότητα κίνησης** του αντικειμένου μας στον χώρο. Έτσι στο UAV μας έχουμε 6 βαθμούς ελευθερίας όπως φαίνονται και στην παρακάτω εικόνα.



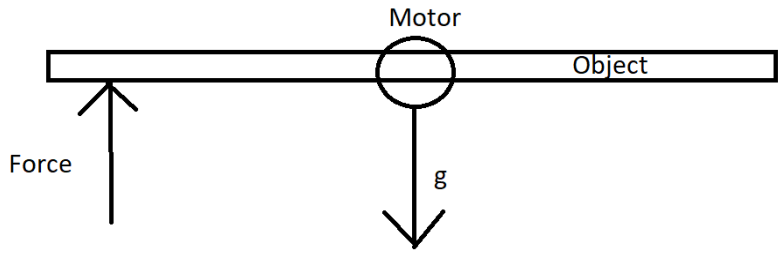
Εικόνα 39. Βαθμοί ελευθερίας του UAV

Θεωρούμε δεδομένο ότι δεν είμαστε σε θέση να ελέγχουμε ξεχωριστά την κάθε κίνηση απόλυτα. Για παράδειγμα δεν μπορούμε να μετακινήσουμε το UAV μας προς τα μπροστά διατηρώντας τις υπόλοιπες τιμές σταθερές. Θα πρέπει να γείρει προς τα μπροστά για να κάνει αυτήν την κίνηση.

Τα στοιχεία που αποτελούν την κίνησή μας είναι τα παρακάτω:

1. Ώθηση(thrust)
2. Κύλιση(roll)
3. Ύψος(pitch)
4. Εκτροπή (yaw)

Αν θεωρήσουμε ένα σώμα το οποίο έχει ένα μοτέρ στο κέντρο βάρους του τότε η ώθηση που δημιουργεί η κίνησή του θα μετακινήσει το σώμα προς τα πάνω χωρίς να στρίψει καθόλου. Παρόλα αυτά αν ασκηθεί μια εξωτερική δύναμη στο άκρο του σώματος είναι σε θέση να το μετακινήσει χωρίς το σώμα να μπορεί να αντισταθεί σε αυτήν.



Εικόνα 40. Παράδειγμα λειτουργίας μοτέρ σε έναν άξονα

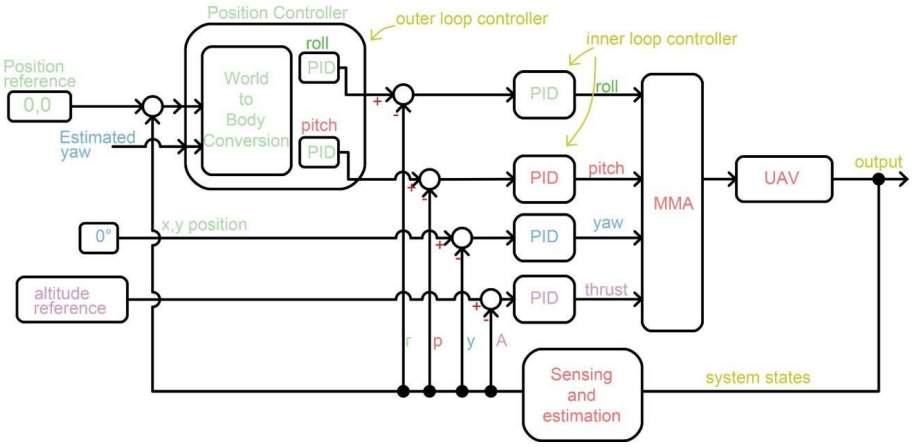
Επομένως μπορούμε να συμπεράνουμε ότι για να μετακινήσουμε το UAV μας προς μια κατεύθυνση θα πρέπει να το γείρουμε κατάλληλα και έπειτα να αυξήσουμε την κίνηση ως προς αυτήν την κατεύθυνση. Για παράδειγμα εάν θέλουμε να προχωρήσει προς τα μπροστά θα πρέπει να μειώσουμε την κίνηση των μπροστά μοτέρ ή/και να αυξήσουμε των πίσω ώστε να γείρει μέχρι ένα σταθερό σημείο έτσι ώστε να πάρει κλίση και να μπορούμε μετά να σταθεροποιήσουμε την κίνηση των μοτέρ προκειμένου να προχωρήσει ευθεία.

Επομένως μπορούμε να πούμε ότι η κίνηση που θα επιτύχει το κάθε μοτέρ είναι συνάρτηση των **thrust**, **roll**, **pitch** και **yaw**.

Αναλυτικότερα:
Η τιμή του thrust όπως προαναφέραμε μας ορίζει το υψόμετρο που βρισκόμαστε. Όσο αυξάνεται η τιμή του έχουμε μεγαλύτερο υψόμετρο ενώ όσο μειώνεται μικρότερο. Παρ'όλα αυτά αυτό ισχύει για τις περιπτώσεις στις οποίες αυτή η δύναμη ενεργεί κάθετα προς το έδαφος. Σε περίπτωση που το όχημά μας γέρνει τότε αυτή η δύναμη ενεργεί υπό γωνία.

Με παρόμοια λογική όπως ορίζει το παραπάνω σχήμα θα πρέπει να ορίσουμε έναν PID Controller ο οποίος θα λαμβάνει υπόψιν του τις αλλαγές τις οποίες θα πρέπει να συμβούν προκειμένου να αλλάξει το υψόμετρο του UAV μας. Αυτό ασφαλώς δεν είναι

τόσο απλό καθώς όπως είπαμε μικρές αλλαγές στην φορά της δύναμης που ασκούν τα μοτέρ θα προκαλέσουν κίνηση σε παραπάνω από έναν άξονες. Με την ίδια λογική θα δημιουργήσουμε άλλους 3 PID Controllers προκειμένου να ελέγξουμε τα pitch, roll και yaw.

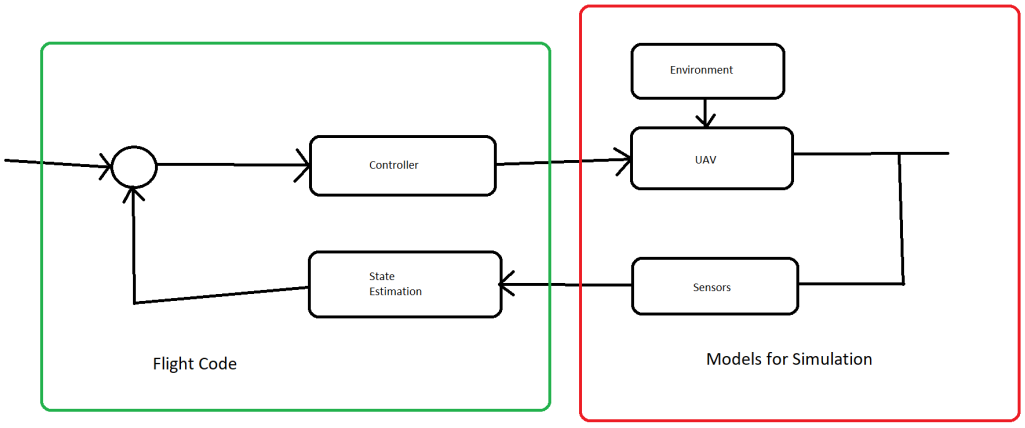


Εικόνα 41. Σχηματικό λειτουργίας του συστήματος με χρήση PIDs

Την κάθε μια από τις παραπάνω εισόδους στα PID που βλέπουμε στην εικόνα 41 θα τις πάρουμε με βάση τις τιμές από τους αισθητήρες. Για παράδειγμα ας θεωρήσουμε ότι η μόνη σταθερά που έχουμε είναι το υψόμετρο. Επομένως η είσοδος για το thrust θα πρέπει να είναι αυτή ακριβώς η ένδειξη που συσχετίζεται με το υψόμετρο. Αυτό όμως όπως έχουμε αναφέρει ήδη δεν είναι δυνατό. Επομένως για να μετακινηθεί το UAV μας θα πρέπει η είσοδος στο κάθε PID να είναι μια συνάρτηση διαφόρων τιμών υπολογίζοντας την θέση του στον άξονα XYZ κάθε στιγμή. Άρα συμπεραίνουμε πως ακόμα και αν θέλουμε κίνηση προς τα μπροστά μας ενδιαφέρουν παραπάνω από μια τιμές των PIDs.

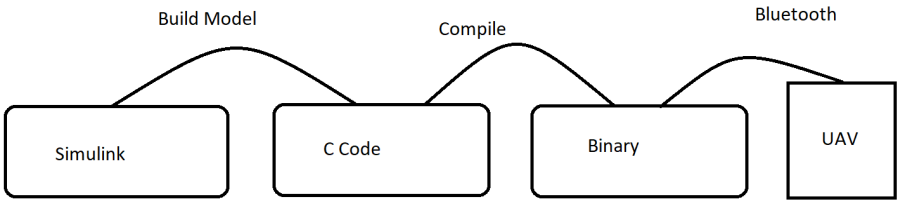
Αυτές τις διαφοροποιήσεις στις θέσεις μπορούμε να τις ορίσουμε μόνο ως προς μια αρχική θέση. Επομένως πρέπει να ορίσουμε στο UAV μας μια αρχική θέση (θέση 0) σύμφωνα με την οποία θα υπολογίζουμε τις διαφορές στην θέση. Ασφαλώς αυτή η θέση μας βολεύει να είναι εντελώς οριζόντια σε σχέση το έδαφος σε μια ακίνητη θέση.

2.9.1 Flight Code

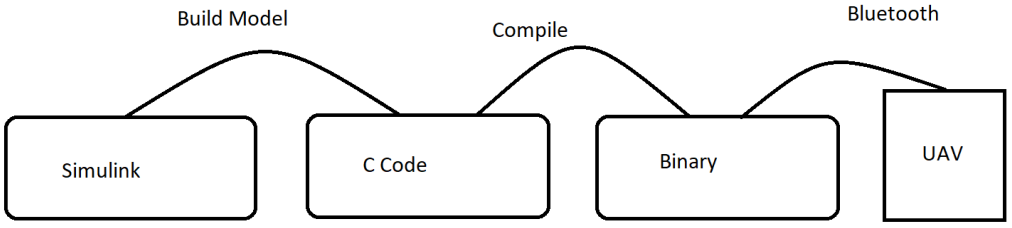


Εικόνα 42. Διαχωρισμός Flight Code και Μοντέλων προς προσομοίωση και η μεταξύ τους λειτουργία

Όσων αφορά την λειτουργία των UAVs υπάρχουν πολλών ειδών κώδικες προκειμένου να ρυθμίσουν και να αυτοματοποιήσουν τις διαδικασίες λειτουργίας τους. Bluetooth, Αισθητήρες, LEDs, Μοτέρ, Διαχείριση μπαταρίας, Διαχείριση μνήμης και Επικοινωνίες είναι κάποια από τα παραδείγματα αυτών των διαδικασιών.



Εικόνα 43. 1ος τρόπος σχεδίασης μοντέλου

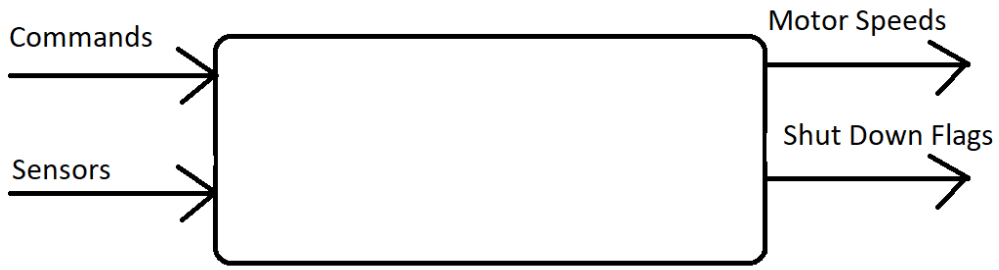


Εικόνα 44. 2ος τρόπος σχεδίασης μοντέλου

Στα παραπάνω σχήματα βλέπουμε τις 2 διαφορετικές επιλογές που μπορούμε να ακολουθήσουμε ως κατεύθυνση όταν προσομοιώνουμε κώδικα για χρήση στο UAV μας. Θα επιλέξουμε την 2η κατηγορία. Πρώτα χτίζουμε το μοντέλο μας στο **Simulink**.

Στη συνέχεια γράφουμε τον κώδικα σε **C** τον κάνουμε **Compile** και μέσω **Bluetooth** έρχεται σε επαφή με το **UAV** μας.

Για να γράψουμε κώδικα και να τον ελέγξουμε θεωρούμε τα προγράμματα που ρυθμίζουν σταθερές (Bluetooth, Αισθητήρες κλπ) αμετάβλητα, προκειμένου να είμαστε σε θέση να ελέγξουμε μικρές και διαχειρίσιμες αλλαγές στον κώδικά μας. Θα ακολουθήσουμε λοιπόν την λογική του παρακάτω σχήματος.



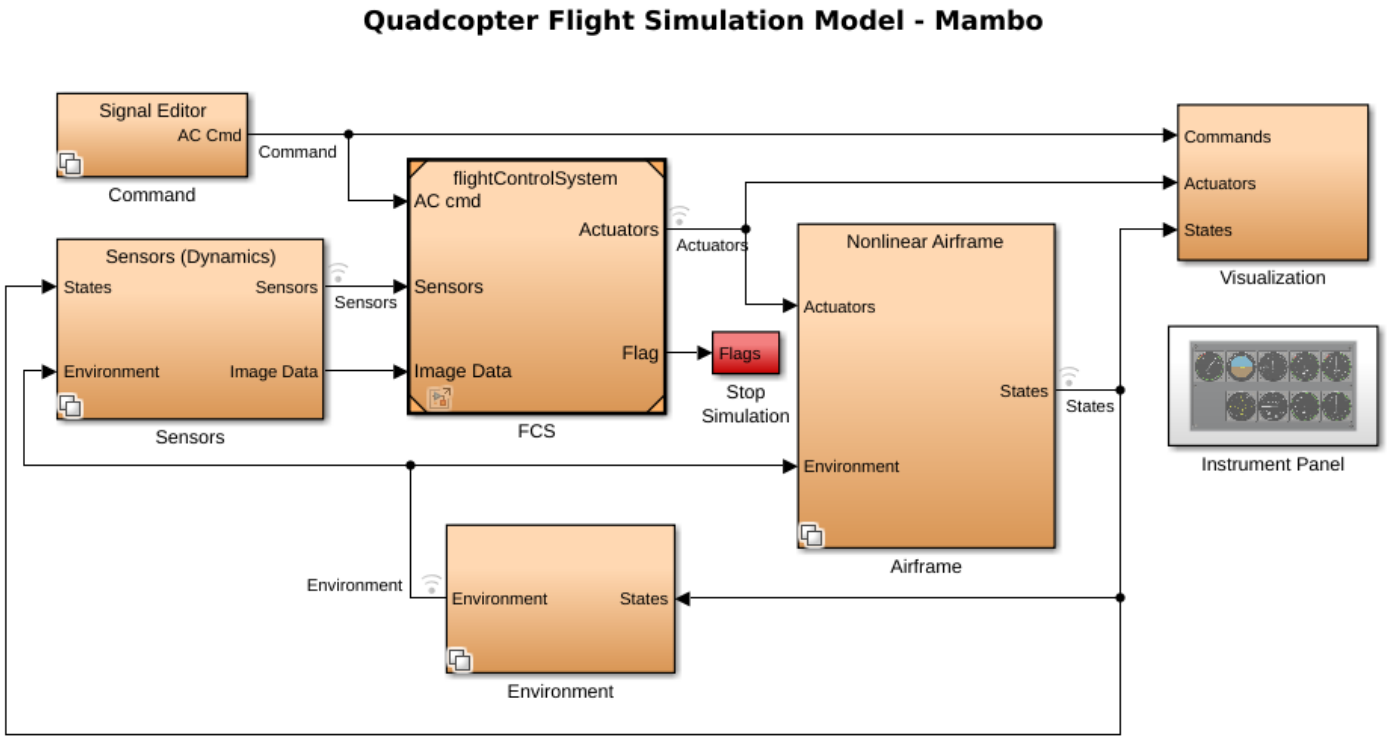
Εικόνα 45. Το UAV μας ως σύστημα

Εισάγουμε λοιπόν στο σύστημά μας τις εντολές με τα δεδομένα από τους αισθητήρες και παίρνουμε ως έξοδο τις τιμές της λειτουργίας των μοτέρ καθώς και κάποια Shut Down Flags που υπάρχουν για λόγους ασφαλείας.

Σε αυτό το σύστημα έχουμε τα κομμάτια που θα κάνουν το UAV μας λειτουργικό. Έχουμε δηλαδή τον **Controller**, το **State Estimator**, το **Fault Protection** και τα **Data Logging**. Το State Estimator χρησιμοποιεί τα δεδομένα των αισθητήρων και τα μετατρέπει σε στοιχεία που μπορούμε να χρησιμοποιήσουμε αλλιώς. Θα μπορούσαμε να πούμε ότι βγάζει το συμπέρασμα βάση των. Για παράδειγμα ο αισθητήρας πίεσης αέρα μας δίνει μια τιμή σε πίεση. Χρειάζεται κάποια μετατροπή προκειμένου να προκύψει το υψόμετρο που έχουμε κ.ο.κ. . Το Fault Protection θέτει σταθερές που δεν πρέπει να παραβιαστούν προκειμένου να υπάρξει ασφάλεια για το όχημά μας και το Data Logging μας δίνει την δυνατότητα να αποθηκεύσουμε κάποιες πληροφορίες που πιθανόν να θέλουμε να συλλέξουμε για μετέπειτα χρήση.

2.9.2 asbQuadcopterStart

Με την παραπάνω εντολή στο matlab ξεκινάει μια έτοιμη προσομοίωση ενός συστήματος.



Εικόνα 46. Μοντέλο προσομοίωσης Mambo

Πάνω αριστερά βλέπουμε το **Signal Editor**. Είναι το Setpoint που θέλουμε το σύστημα να ακολουθήσει.

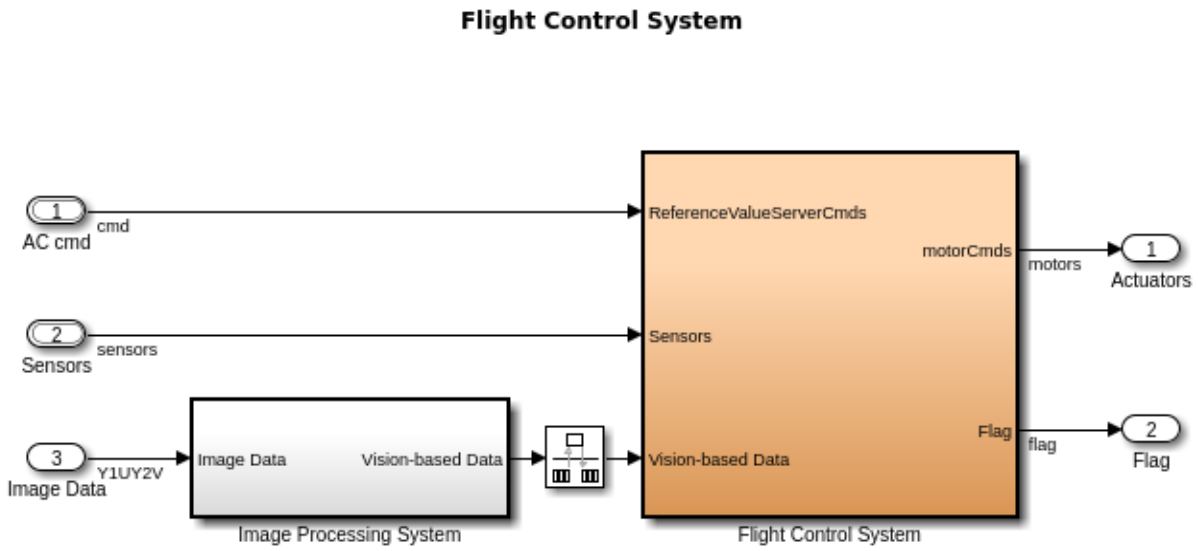
Υπάρχει το κομμάτι του **flightControlSystem** το οποίο είναι ο πυρήνας του συστήματός μας.

Το **Nonlinear Airframe** είναι το κομμάτι που πηγαίνουν οι έξοδοι από το flightControlSystem.

Το **Visualization** κομμάτι είναι αυτό που φτιάχνει τις γραφικές παραστάσεις από τα σήματα που λαμβάνει.

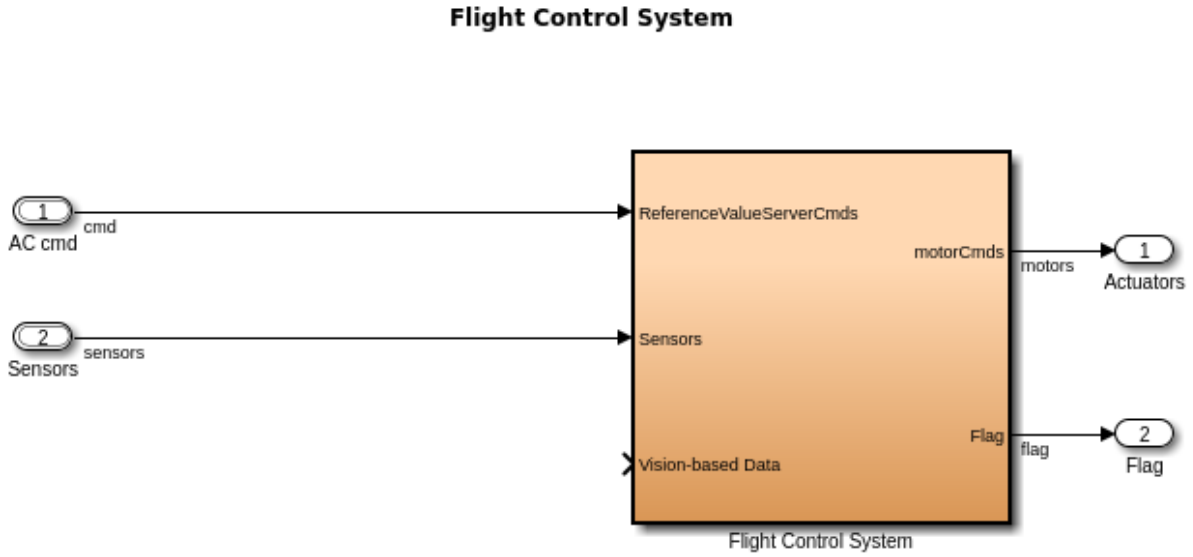
Το **Environment** είναι το τμήμα όπου μοντελοποιούμε τα φυσικά μεγέθη όπως η βαρύτητα και η ατμοσφαιρική πίεση.

Στο τμήμα **Sensors** είναι το κομμάτι όπου προσομοιώνονται αισθητήρες, παράγει επίσης τεχνητό θόρυβο ώστε να προσομοιωθεί η πραγματική λειτουργία των αισθητήρων.

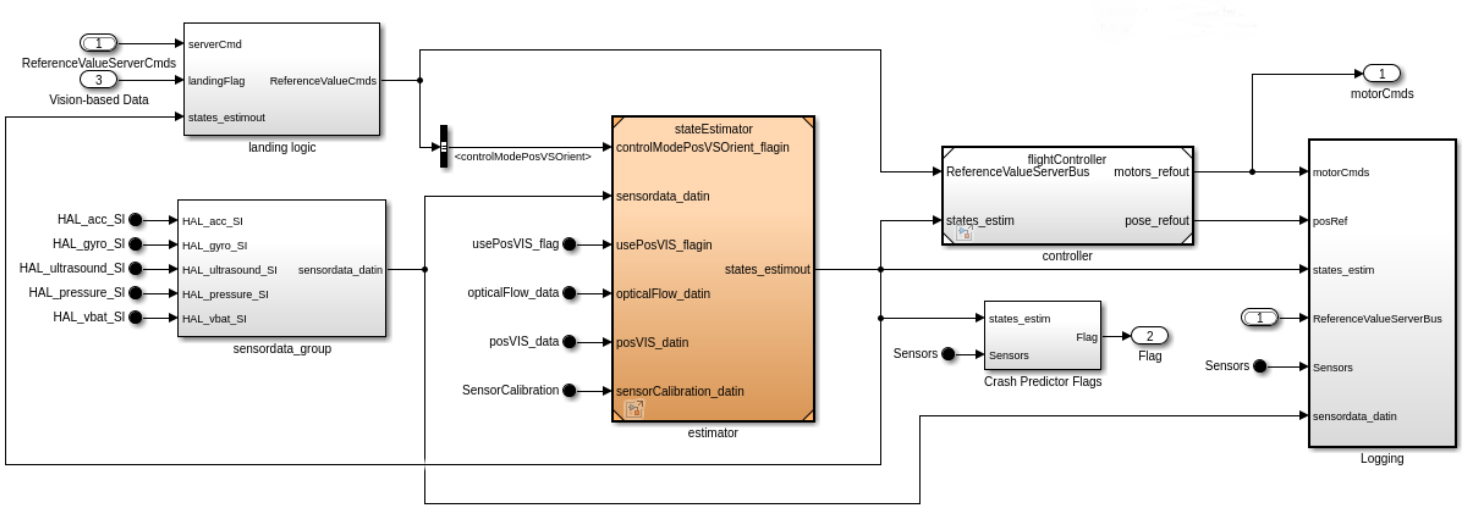


Εικόνα 47. Απεικόνιση του Flight Control System

Εντός του flightControlSystem βλέπουμε το παραπάνω σχήμα. Για να συμβαδίσουμε με τα προηγούμενα λεγόμενά μας θα αφαιρέσουμε το κομμάτι του Image Data από το σύστημα.

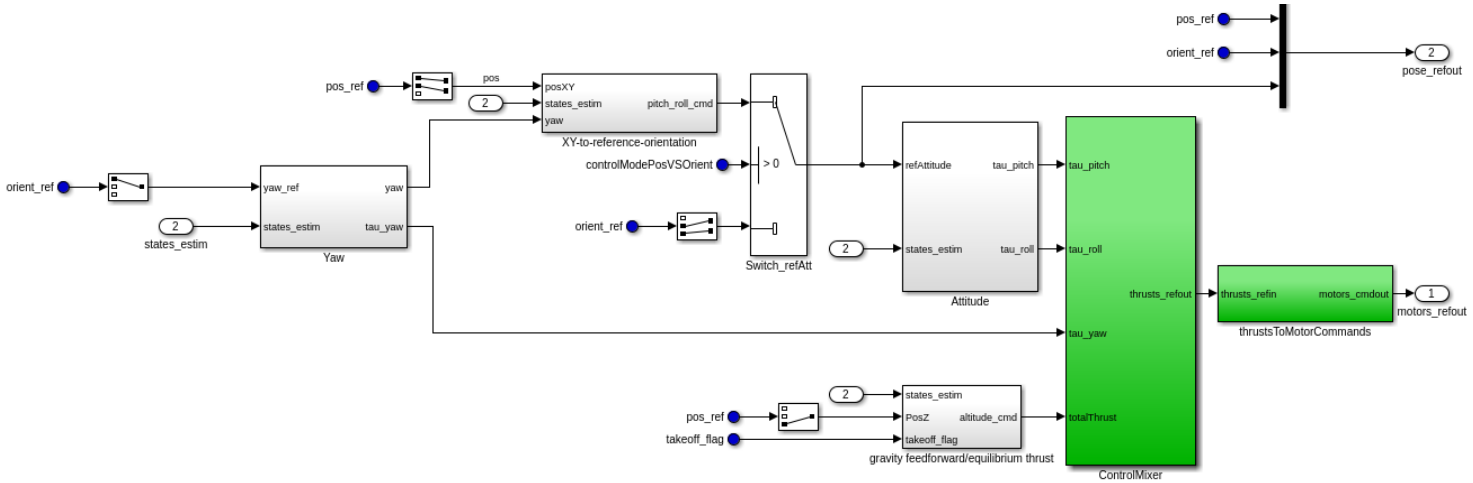


Εικόνα 48. Απεικόνιση του Flight Control System μετά την αφαίρεση του Image Processing System



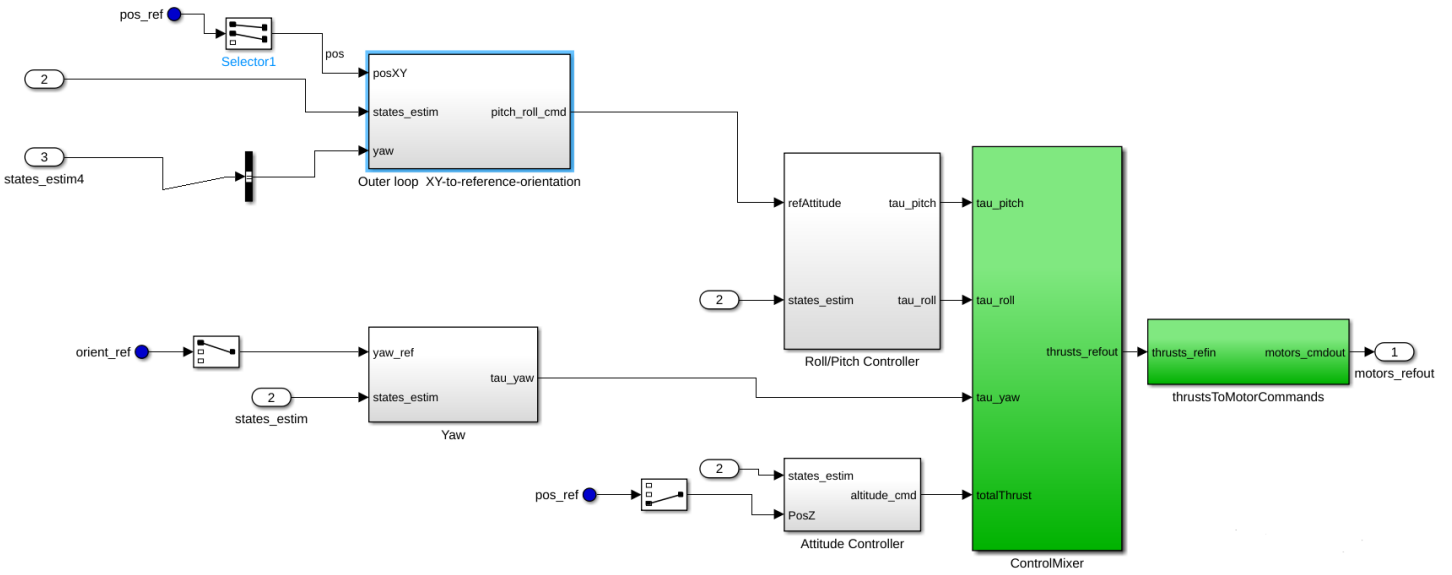
Εικόνα 49. Το σχηματικό του flight Control System

Εντός του flightControlSystem βλέπουμε ένα παρόμοιο σχήμα με αυτό που είχαμε χρησιμοποιήσει νωρίτερα.



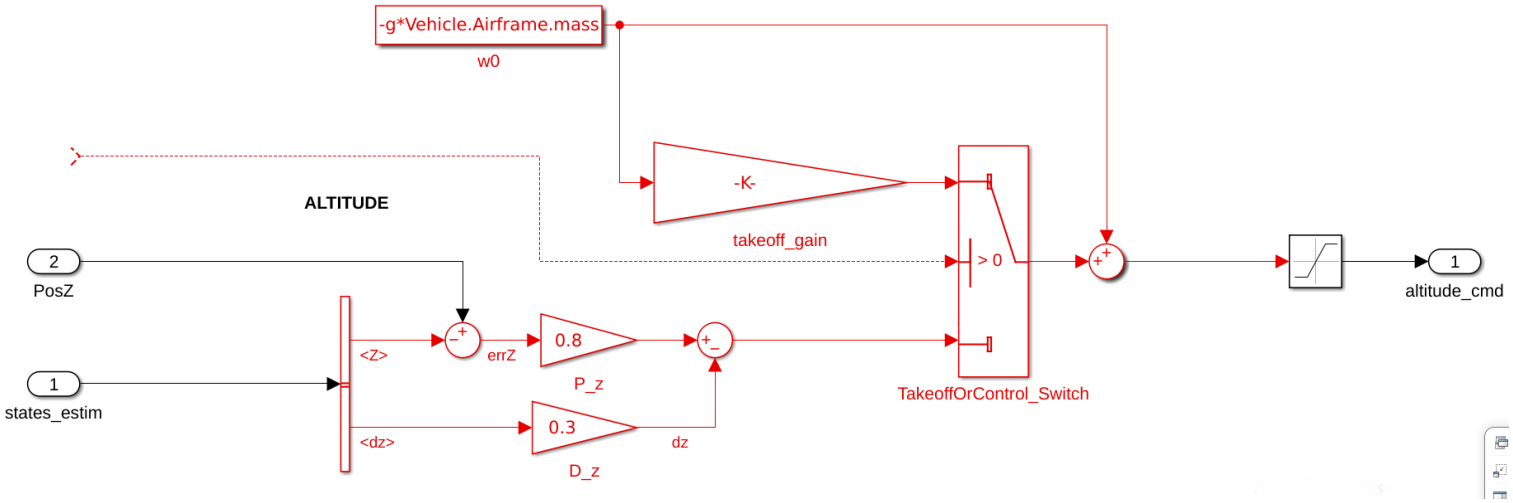
Εικόνα 50. Flags και Control Mixer

Εντός του Flight Controller βλέπουμε το παραπάνω σχήμα. Σε αυτό το σημείο είμαστε σε θέση να πειράξουμε παραμέτρους όπως το take off flag ή το orientation.



Εικόνα 51. Εντός του control mixer

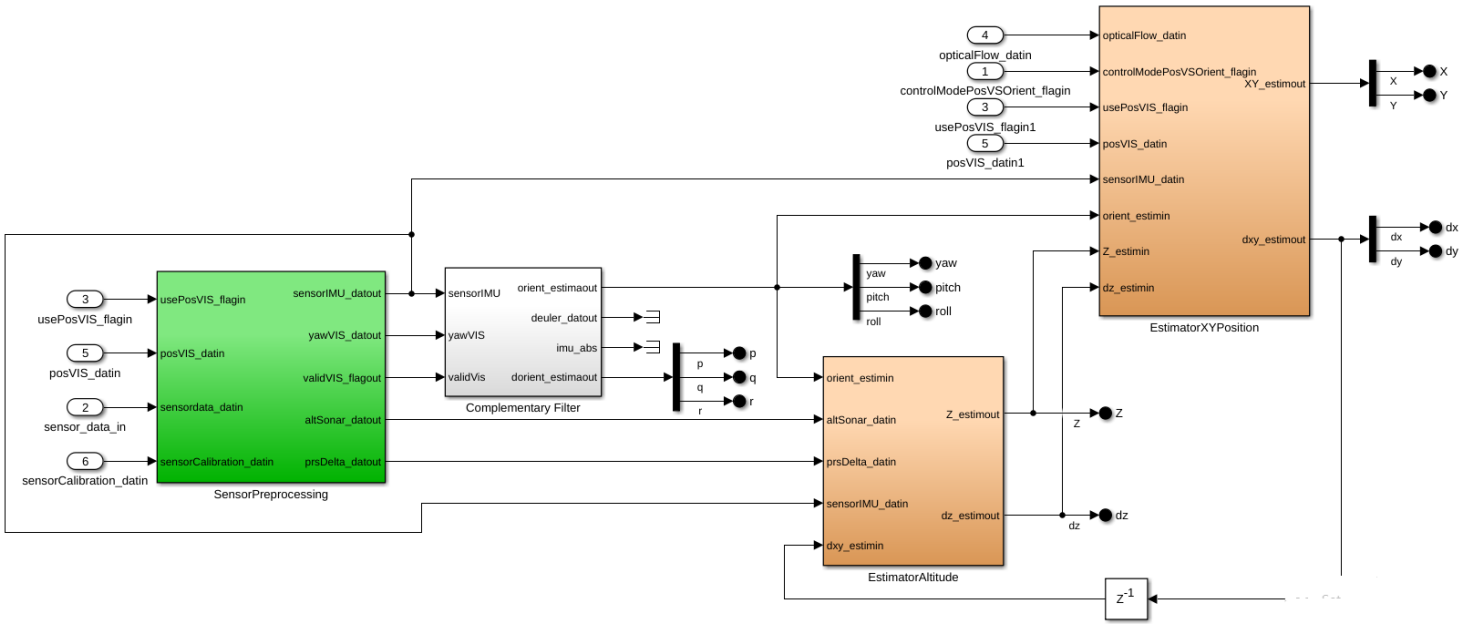
Συνολικά έχουμε 6 PID Controllers ώστε να ελέγξουμε την θέση και το orientation του UAV μας.



Εικόνα 52. Λειτουργία της posZ

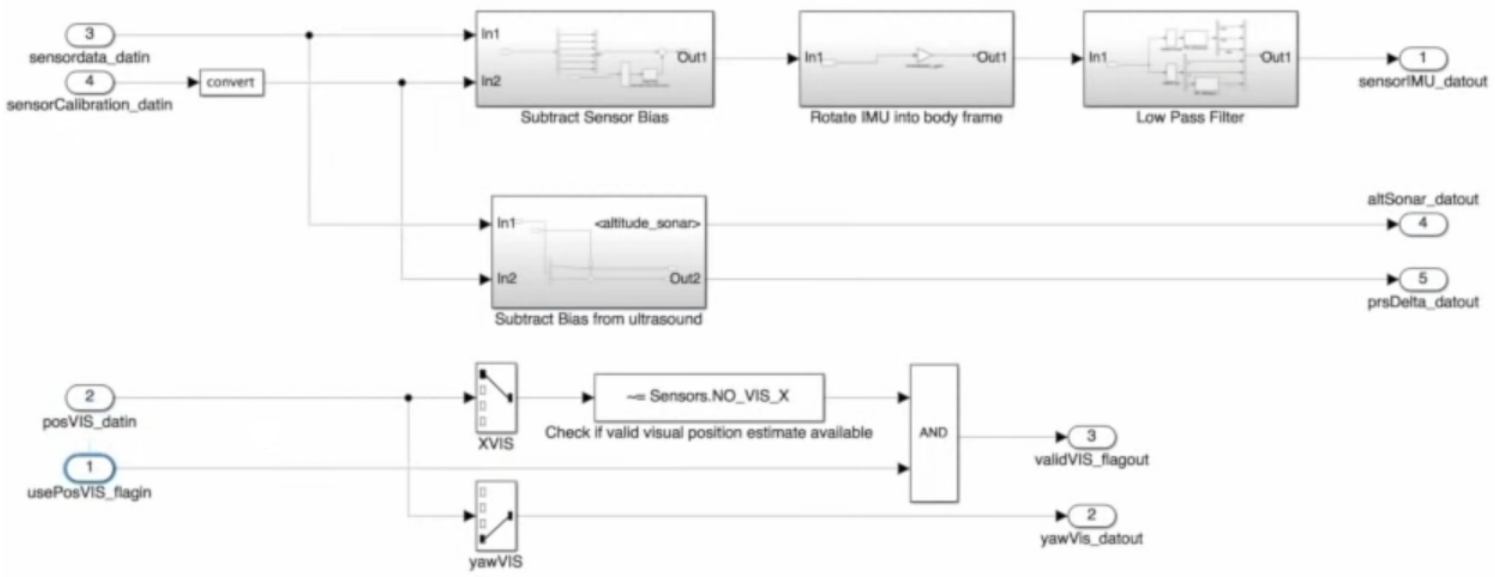
Στο παραπάνω σχηματικό βλέπουμε ότι η θέση (**posZ**) υπολογίζεται σε συνάρτηση με την τιμή του Ultrasound (<Z>). Προέρχεται δηλαδή από την **states_estim** που είναι η προβλεπόμενη κατάσταση. Έτσι διαθέτουμε ένα παραγωμένο σήμα.

Αυτά τα σήματα καταλλήγουν στο **Control Mixer** που είδαμε σε παραπάνω σχήμα τα οποία ορίζουν την ταχύτητα των μοτέρ στο **thrustsToMotorCommands** τμήμα.

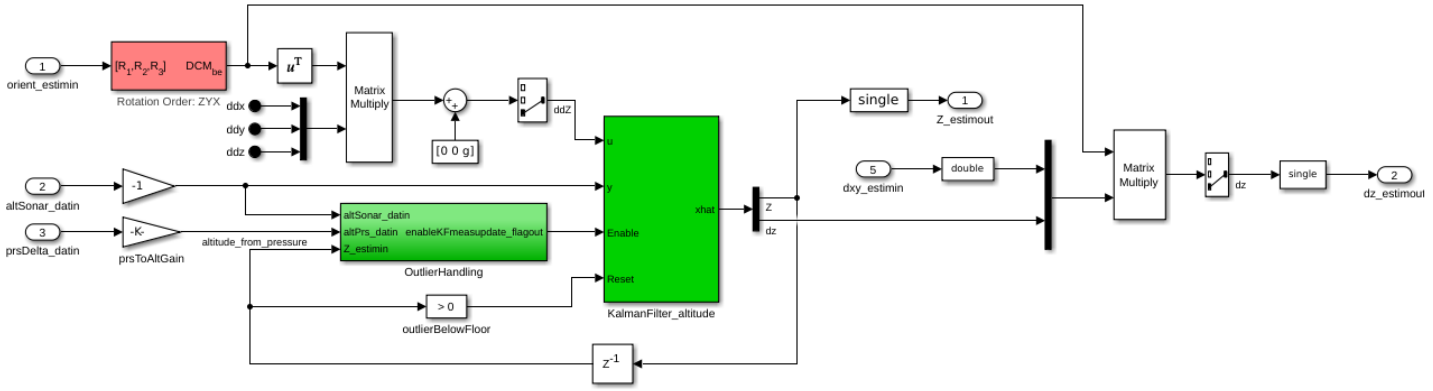


Εικόνα 53. Σχηματικό του State Estimator

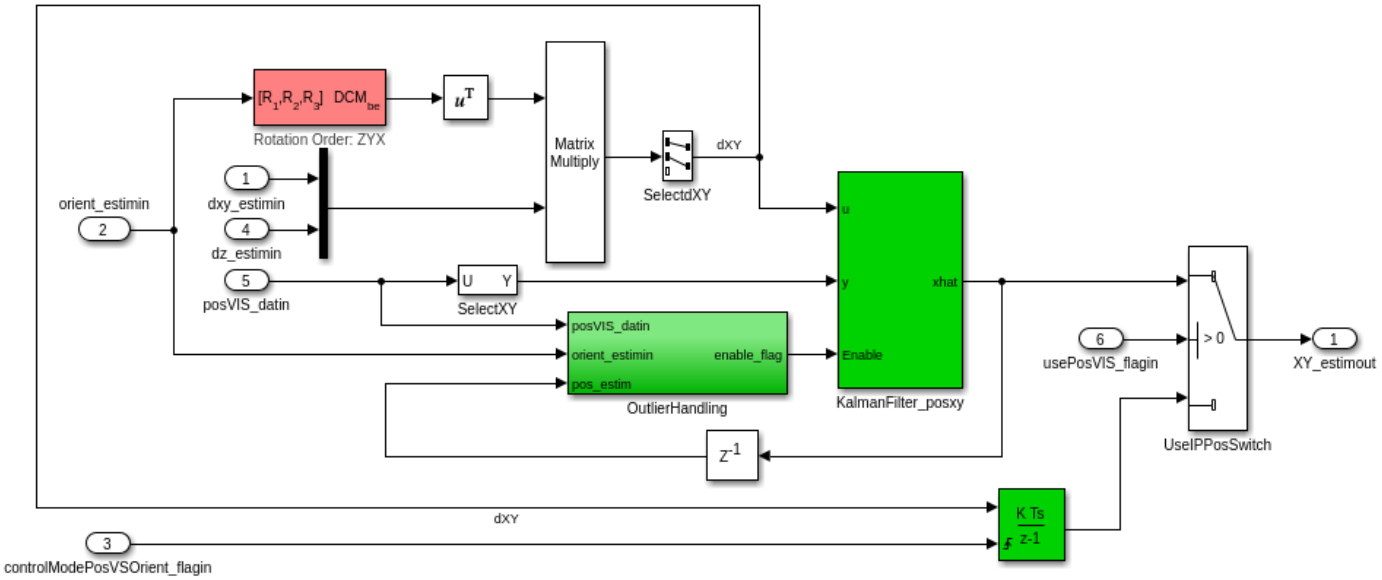
Στο παραπάνω σχήμα βλέπουμε το σχηματικό του **State Estimator**. Βλέπουμε ότι πρώτα χρησιμοποιούμε το **SensorProcessing** που επεξεργαζόμαστε τα δεδομένα από τους αισθητήρες και έπειτα η έξοδος περνάει από τα φίλτρα που έχουμε.
Παρακάτω βλέπουμε το σχηματικό του τμήματος **SensorProcessing** μετά από κάποιες αλλαγές.



Εικόνα 54. Επεξεργασία Δεδομένων από τους Αισθητήρες (Sensor Processing)



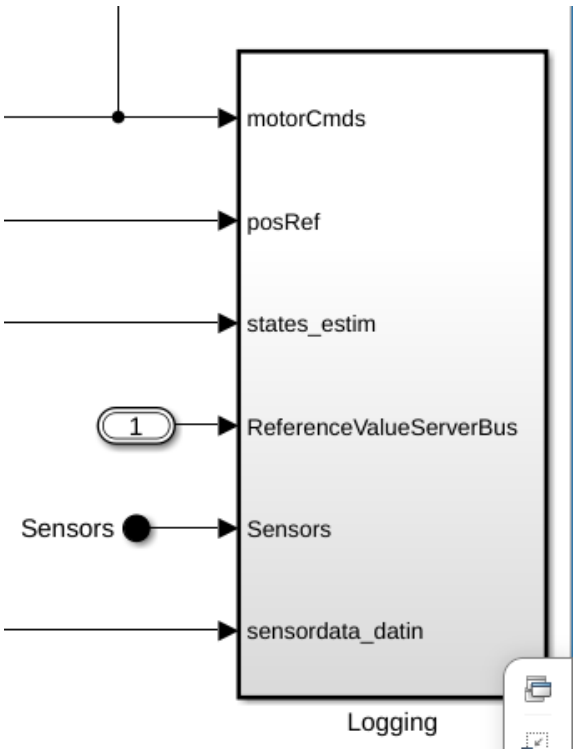
Εικόνα 55. Estimator Altitude



Εικόνα 56. EstimatorXYPosition

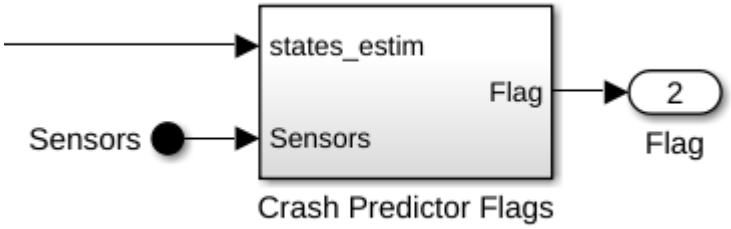
Εντός των σχηματικών των 2 Estimators (**EstimatorAltitude** και **EstimatorXYPosition**) παρατηρούμε ότι χρησιμοποιούνται τα **φίλτρα Kaiman** που είχαμε κάνει αναφορά νωρίτερα στο κείμενό μας.

Ακόμη χρησιμοποιείται ένα **Complementary Filter**. Χρησιμοποιείται ως ένας εύκολος τρόπος να συνδυάσεις δεδομένα από παραπάνω από 1 αισθητήρα ταυτόχρονα.



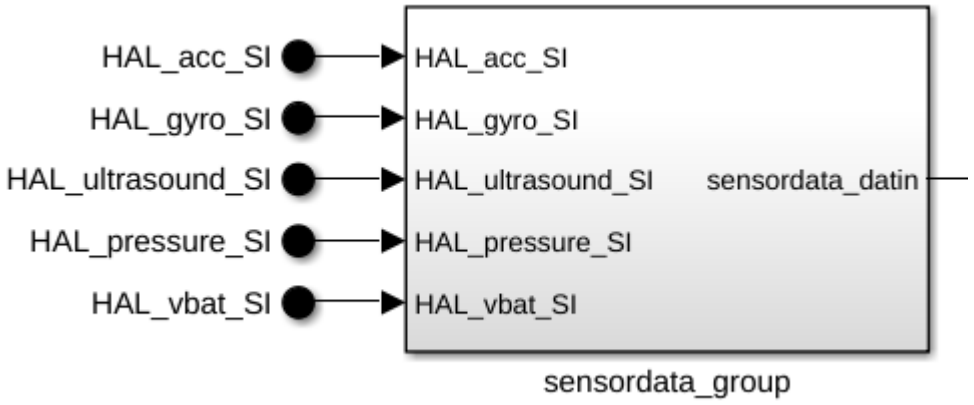
Εικόνα 57. Οι είσοδοι του Logging

Στο υπόλοιπο σχήμα παρατηρούμε το **Logging** που όπως προαναφέραμε είναι το σημείο που αποθηκεύουμε προσωρινά τις πληροφορίες που θέλουμε.



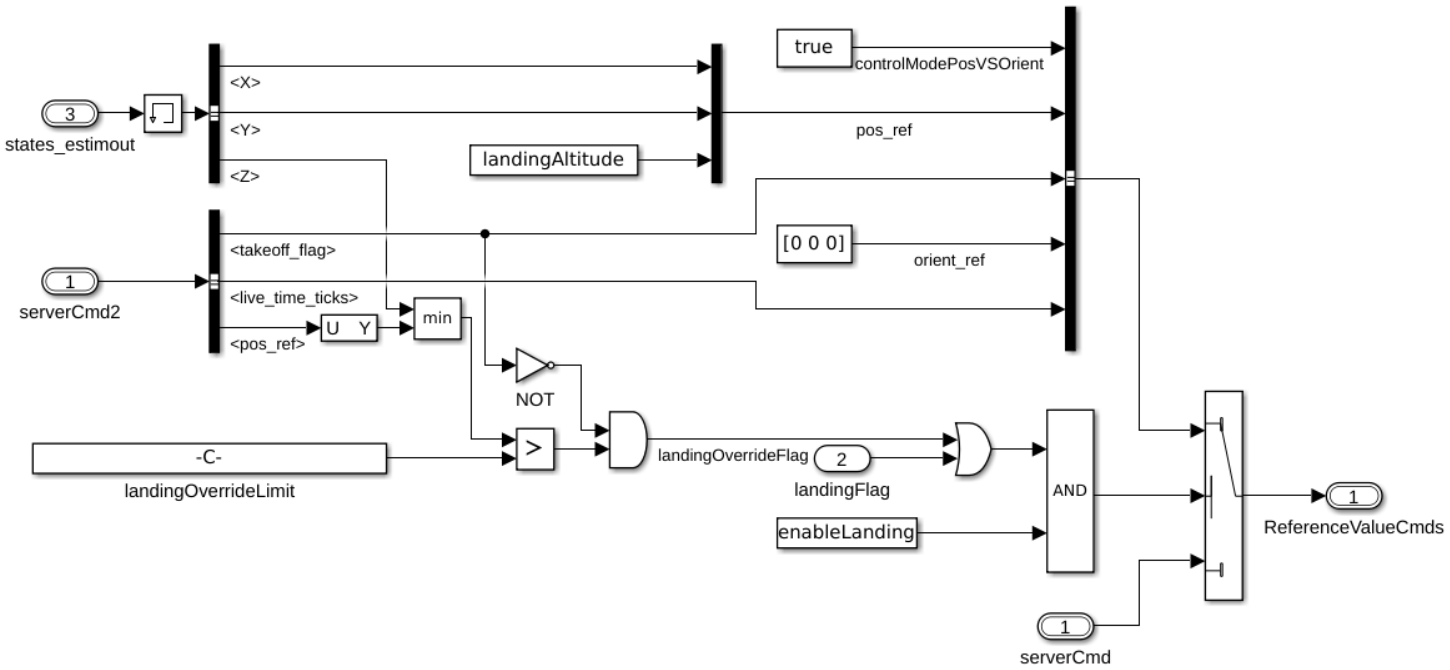
Εικόνα 58. Το Crash Predictor Flags

Ακόμη έχουμε το **Crash Predictor Flags** που διασφαλίζει την ασφάλεια του οχήματός μας, όπως είχαμε αναφέρει και νωρίτερα. Σε περίπτωση που θελήσουμε μπορούμε να προσθέσουμε επιπλέον προστασίες.



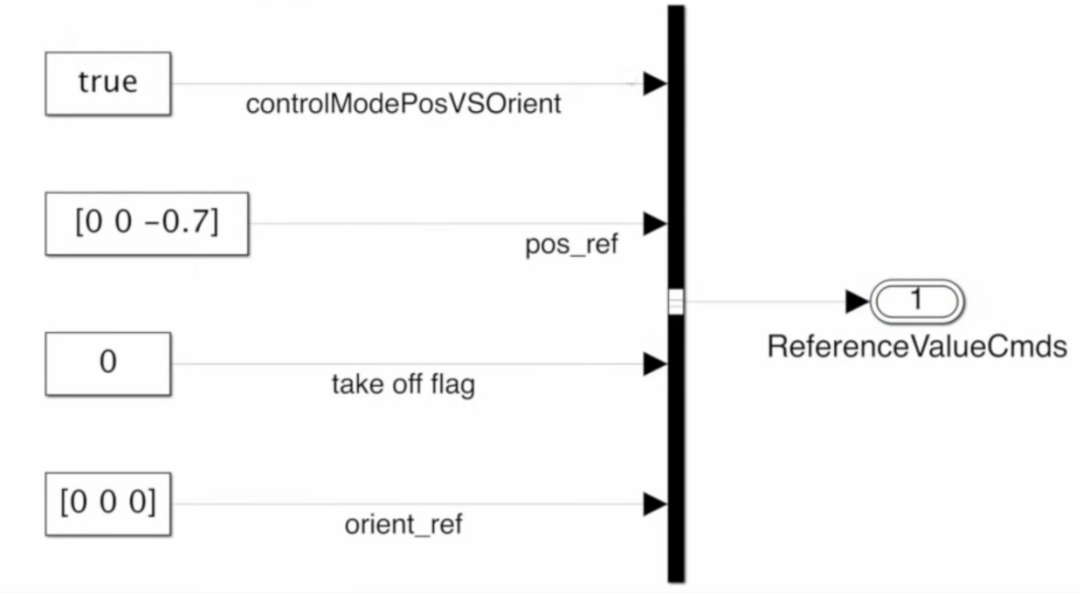
Εικόνα 59. Το sensordata_group

Στην εικόνα 59 βλέπουμε το **sensordata_group** όπου μεταφέρει συγκεκριμένα δεδομένα από τους αισθητήρες εκτός του Sensor Bus για χρήση σε άλλα σημεία του κώδικα.



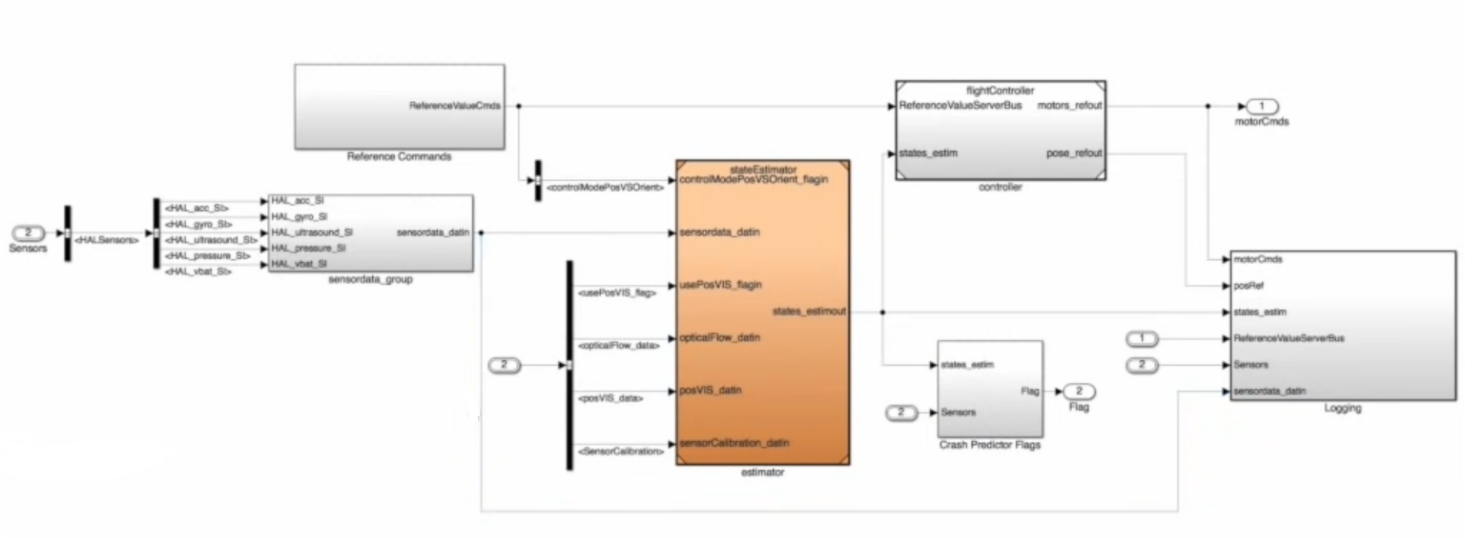
Εικόνα 60. Εντός του sensordata_group

Και τέλος έχουμε το landing logic όπου θα μετατρέψει τα external reference commands σε landing commands αν η σημαία (flag) προσγείωσης το υποδεικνύει.



Εικόνα 61. Είσοδοι στο landing logic υποσύστημα

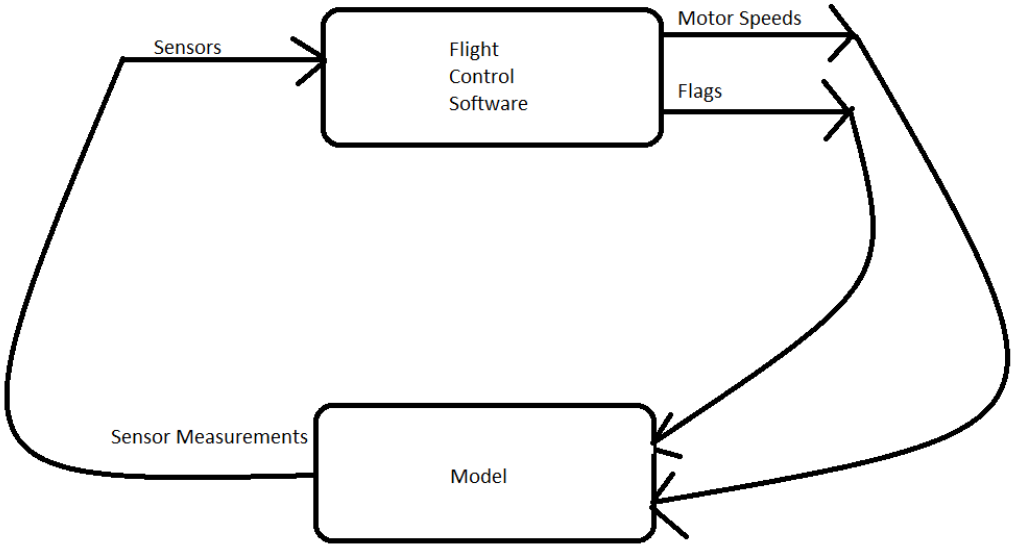
Σε περίπτωση που θέλουμε πιο απλή λειτουργία μπορούμε να αλλάξουμε το landing logic στο παραπάνω σχήμα. Το -0.7 υπάρχει εξαιτίας της θέσης του αισθητήρα στο UAV. Θα πρέπει όμως να αλλάξουμε το αρχικό σχήμα σβήνοντας τις εισόδους στο landing logic υποσύστημα.



Εικόνα 62. Νέο σύστημα μετά τις αλλαγές

Κεφάλαιο 3 Σχεδιασμός Μοντέλου

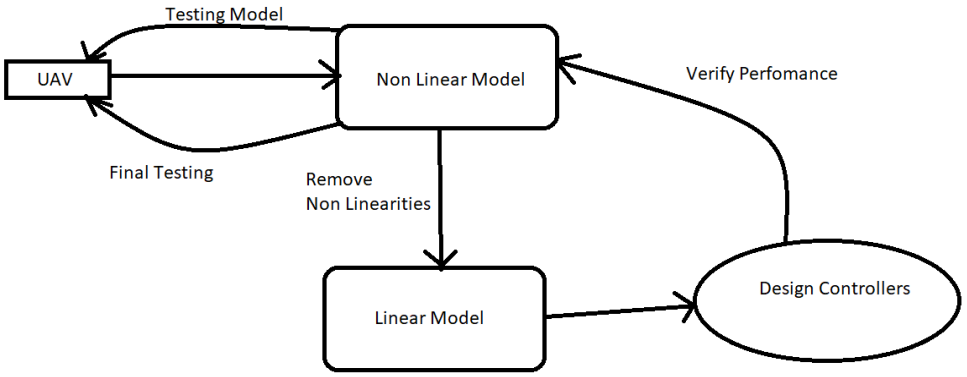
Στο παρακάτω σχήμα βλέπουμε την λογική του σχεδιασμού του μοντέλου μας. Στο τμήμα **Fight Control Software** περιέχονται όλα όσα είδαμε στις προηγούμενες ενότητες τα οποία δέχονται ως είσοδο μόνο τις εισόδους από τους **αισθητήρες** καθώς διαγράψαμε τα **references** που υπήρχαν στον κώδικα του Simulink. Ως έξοδο παράγει την **συνάρτηση ταχύτητας** των μοτέρ καθώς και τα αντίστοιχα **flags** ασφαλείας. Αυτές οι έξοδοι όμως δεν μπορούμε να είμαστε σίγουροι πως ανταποκρίνεται 100% στο πραγματικό περιβάλλον καθώς δεν γίνεται να το προσομοιώσουμε απόλυτα. Για αυτόν τον λόγο οι έξοδοι από το Flight Control System θα πρέπει να περάσουν ξανά από κάποιον **έλεγχο** προκειμένου να εισαχθούν τα νέα δεδομένα ξανά ως είσοδος στο σύστημά μας. Σε περίπτωση που είχαμε ένα ιδανικό μοντέλο αναπαράστασης του πραγματικού κόσμου η έξοδος της προσομοίωσης και η έξοδος από μια πραγματική πτήση θα ήταν ταυτόσημες, κάτι το οποίο δεν μπορεί να συμβεί.



Εικόνα 63. Λογική σχεδίασης μοντέλου

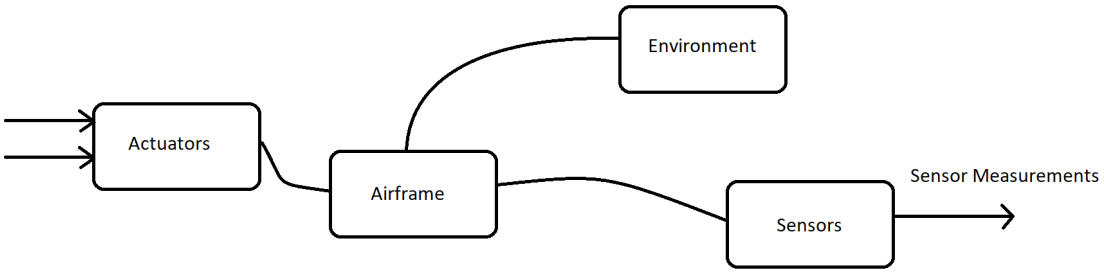
Καλούμαστε λοιπόν να σχεδιάσουμε ένα Linear Model σε αντίθεση με το Non Linear Model που δίνεται στο παράδειγμα στις προηγούμενες ενότητες. Παρότι το Non Linear Model μας δίνει καλύτερη απόκριση της πραγματικότητας στο Linear είμαστε σε θέση να σχεδιάσουμε τους Controllers μας όπως θέλουμε. Ασφαλώς παρέχει αρκετή ακρίβεια στην αναπαράσταση της πραγματικότητας ώστε να μπορεί να χρησιμοποιηθεί. Έτσι καταλήγουμε σε έναν **συνδυασμό** των 2 μοντέλων.

Αρχικά χρησιμοποιούμε ένα **Non Linear Model**. Ελέγχουμε την απόκριση που έχει απέναντι σε μια πραγματική πτήση και με βάση αυτή σχεδιάζουμε ένα **Linear Model** διαγράφοντας τα Non Linear τμήματα του προηγούμενου μοντέλου μας. Έπειτα με βάση τα δεδομένα που έχουμε σχεδιάζουμε τους **Controllers** μας.



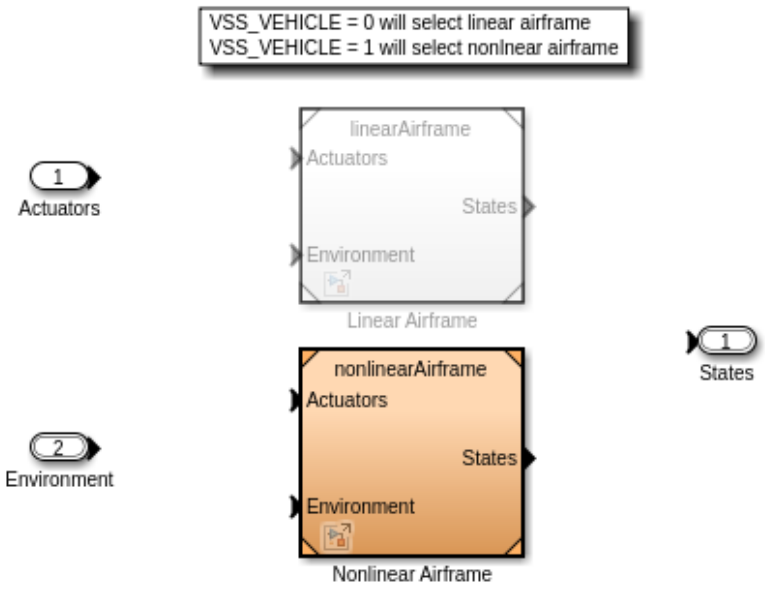
Εικόνα 64. Λογική Γραμμικών και Μη Γραμμικών Μοντέλων στο ίδιο σύστημα

Στο παρακάτω σχήμα βλέπουμε ξανά πως μπορούμε να σπάσουμε σε μικρότερα **υποσυστήματα** τον σχεδιασμό του κάθε τμήματος. Για μεγάλα projects μπορούμε για παράδειγμα να αναθέσουμε σε ξεχωριστές ειδικότητες τον σχεδιασμό του κάθε αντικειμένου επιτυγχάνοντας μεγαλύτερη σταθερότητα στον συνολικό σχεδιασμό, αποφυγή λαθών, δυνατότητα για μικρότερες και ελεγχόμενες αλλαγές αλλά και μικρότερο συνολικό χρόνο πραγμάτωσης του έργου.



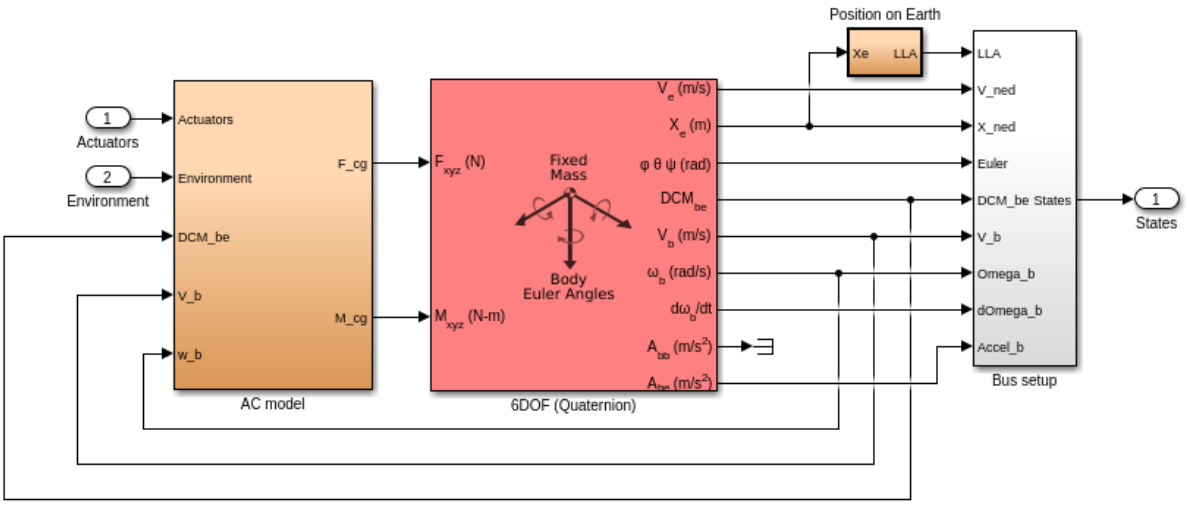
Εικόνα 65. Υποσυστήματα και η λειτουργία τους

Περνώντας στο Simulink βλέπουμε στην παρακάτω εικόνα τον σχεδιασμό εντός του Aiframe τμήματος. Εκεί μπορούμε ανάλογα με την σύνδεση να επιλέξουμε μεταξύ των 2 μοντέλων (Linear και Non Linear).



Εικόνα 66. Linear και Nonlinear Airframes

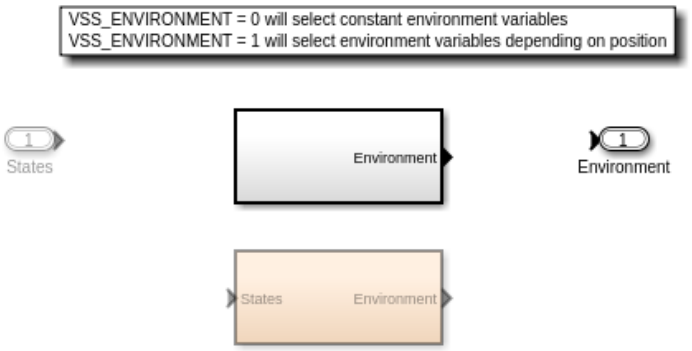
Θα ακολουθήσει ανάλυση του NonLinear Airframe.



Εικόνα 67. Εντός του Non Linear Airframe

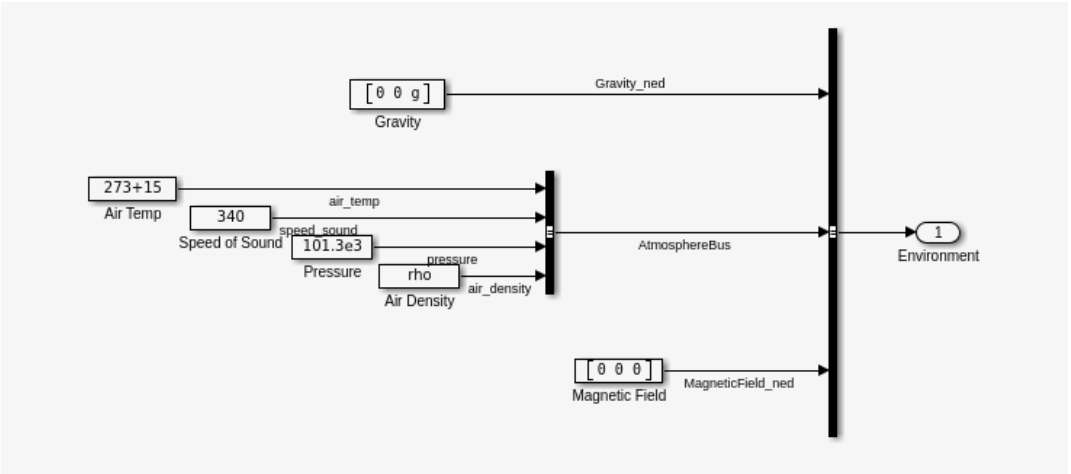
Βλέπουμε στα αριστερά το **AC Model**. Είναι το σημείο όπου δέχεται εισόδους από τους **actuators** και την είσοδο **Environment**. Δηλαδή είναι το σημείο στο οποίο υπολογίζουμε τον βαθμό που οι εξωγενείς παράγοντες επηρεάζουν την λειτουργία. Στη συνέχεια βλέπουμε με το ροζ χρώμα το 6DOF όπου είναι το σημείο που υπολογίζεται η θέση του UAV στον χώρο και μας δίνει ως εξόδους στοιχεία της κίνησής του.

Στη συνέχεια βλέπουμε εντός του Environment υποσυστήματος.



Εικόνα 68. Εντός του Environment υποσυστήματος

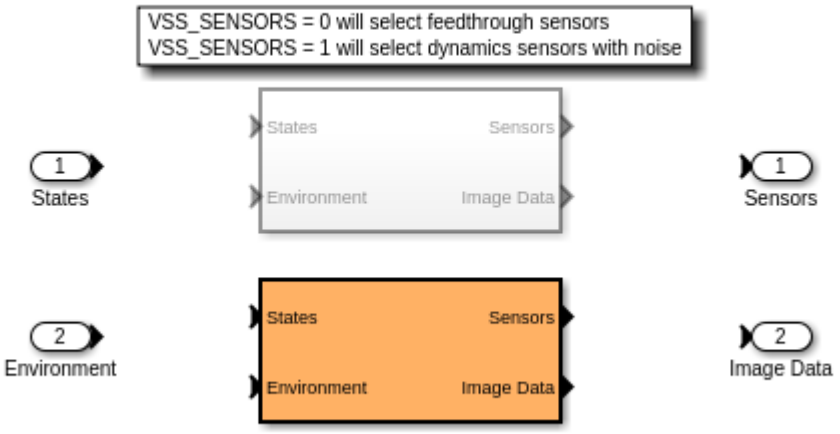
Βλέπουμε 2 επιλογές για **Environment**. Η πάνω είναι η **συνεχής (Consant)** επιλογή όπου οι σταθερές του περιβάλλοντος παραμένουν αναλλοίωτες και η κάτω η **μεταβλητή (Variable)** όπου οι σταθερές του περιβάλλοντος αλλάζουν. Θα χρησιμοποιήσουμε την **Consant** επιλογή.



Εικόνα 69. Εντός του Constant Environment

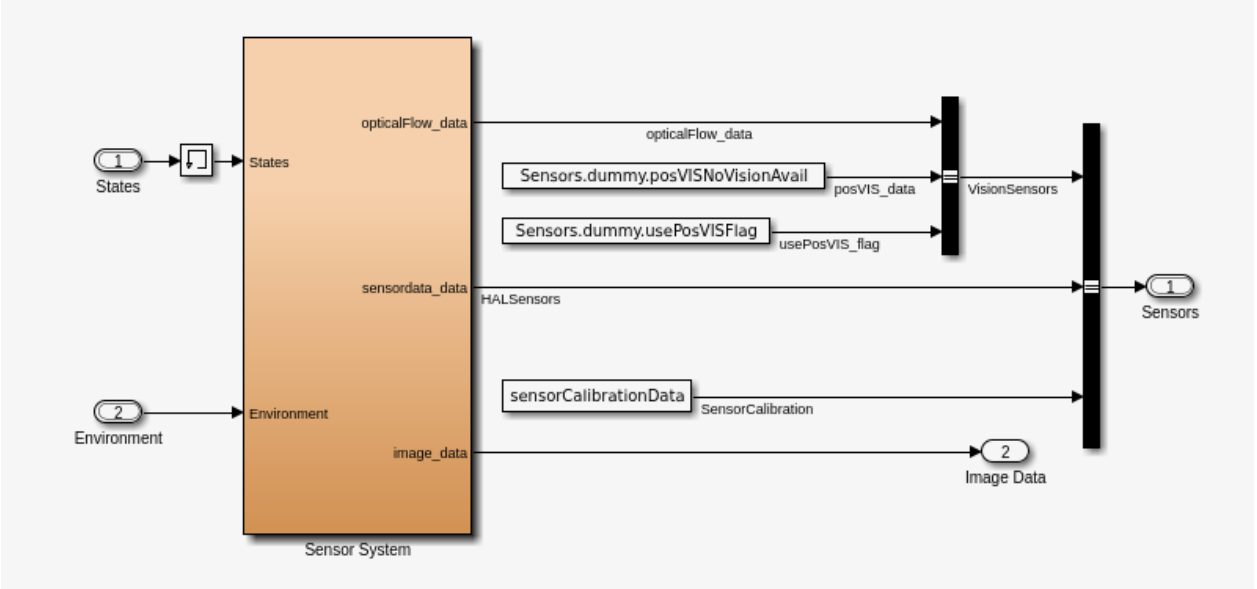
Μέσα στην **Constant** βλέπουμε το παραπάνω σχήμα με κάποιες σταθερές τιμές. Σε περίπτωση που θέλουμε να πετάξουμε το UAV μας σε μεγαλύτερο ύψος θα παρατηρήσουμε μια αλλαγή στο **Pressure** και στο **Air Density**. Αυτό ασφαλώς θα γίνει μέχρι κάποιο σημείο πέρα από το οποίο δεν θα μας επιτρέψει περαιτέρω αλλαγές καθώς θα θέσει σε κίνδυνο την ασφάλεια του UAV μας.

Τέλος θα δούμε το υποσύστημα των **Sensors**. Σε αυτό το υποσύστημα βλέπουμε το παρακάτω σχήμα.



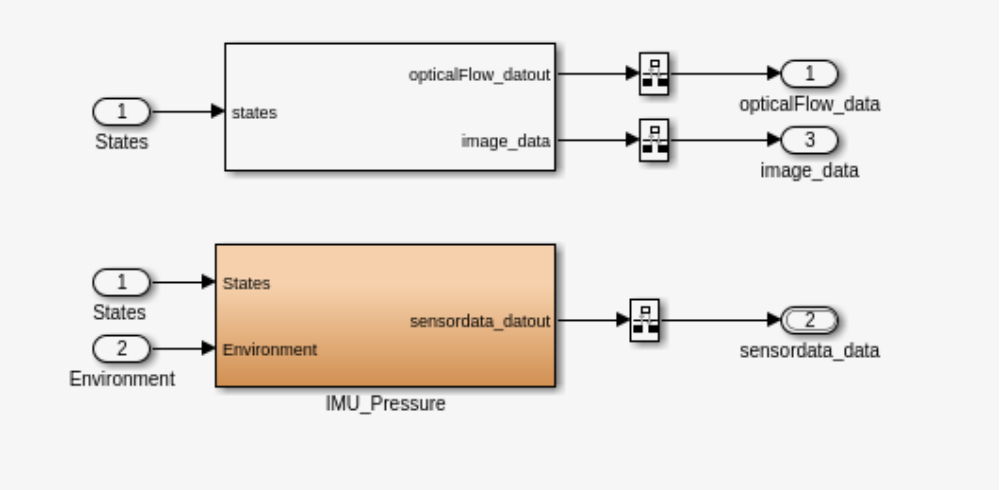
Εικόνα 70. Εντός του Sensors υποσυστήματος

Και εδώ βλέπουμε 2 διαφορετικά τμήματα. Το πάνω είναι το **Feedthrough** και το κάτω το **Dynamics**. Επιλέγοντας το δυναμικό σύστημα βλέπουμε το παρακάτω σχήμα.



Εικόνα 71. Εντός του Dynamics υποσυστήματος

Βλέπουμε το **SensorCalibrationData** που αποτελεί ένα **hardcoded** τμήμα. Στα αριστερά βλέπουμε το Sensor System το οποίο όπως βλέπουμε και στην παρακάτω εικόνα περιέχει τα στοιχεία Camera και IMU_Presure.



Εικόνα 72. Εντός του Sensor System

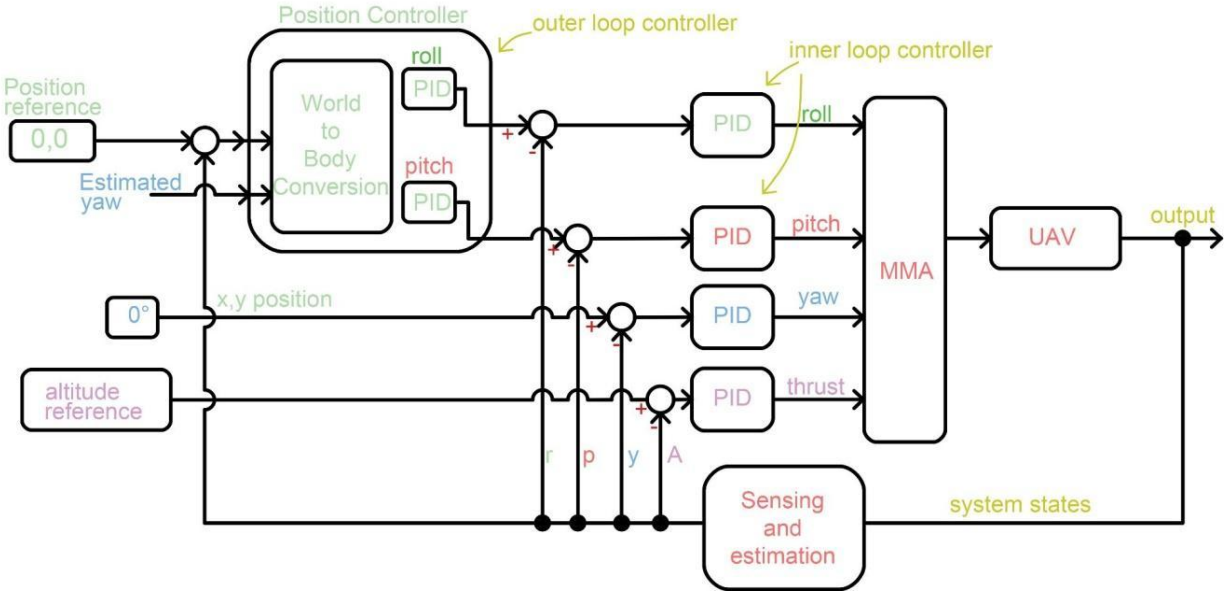
3.1 Flags και κανόνες ασφαλείας

Όπως είδαμε νωρίτερα υπάρχουν έτοιμα τμήματα κώδικα τα οποία περιέχονται στο σύστημά μας. Μπορούμε να πειράξουμε πολλά από αυτά σε περίπτωση που θέλουμε. Για παράδειγμα μπορούμε να πειράξουμε την αποδεκτή τιμή στο γυροσκόπιο αλλά μετά θα πρέπει να αλλάξουμε την αποδεκτή τιμή στο Shutdown Flag προκειμένου να μην συνεχίσει έως ότου να χτυπήσει κάπου. Βλέπουμε έτσι ότι συνδυαστικά η λογική που δουλεύουν οι αισθητήρες και τα flags μπορεί να μας δώσει πληροφορίες για τον κώδικά μας χωρίς όμως να δημιουργηθεί κάποιο ατύχημα.

3.2 Λειτουργία με ένα Linear μοντέλο

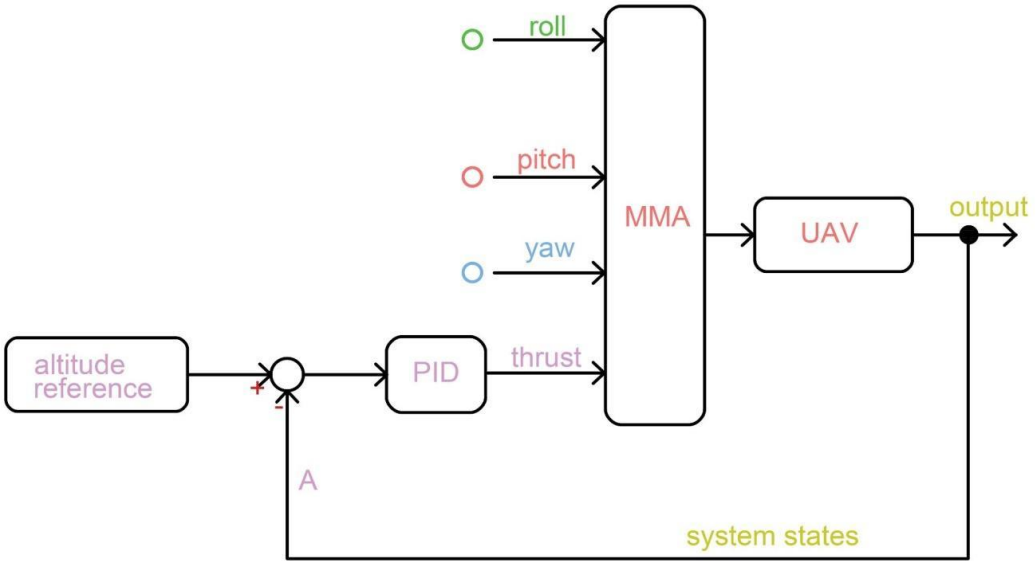
Τα υποσυστήματα **Sensors**, **Environment**, **flightControlSystem** και **Airframe** αποτελούν **Non Linear** κομμάτια. Πράγμα που σημαίνει ότι θα πρέπει να γίνει μια μετατροπή ώστε να μπορούν να συμβαδίσουν με το Linear μοντέλο μας.

Στην πραγματικότητα θα αφαιρέσουμε πολλά τμήματα του κώδικα στο **Simulink** και στην συνέχεια θα ρυθμίσουμε τους **PID Controllers** ώστε να λειτουργήσει ως **Linear** το σύστημά μας. Αυτή η διαδικασία ονομάζεται **Linearize**.

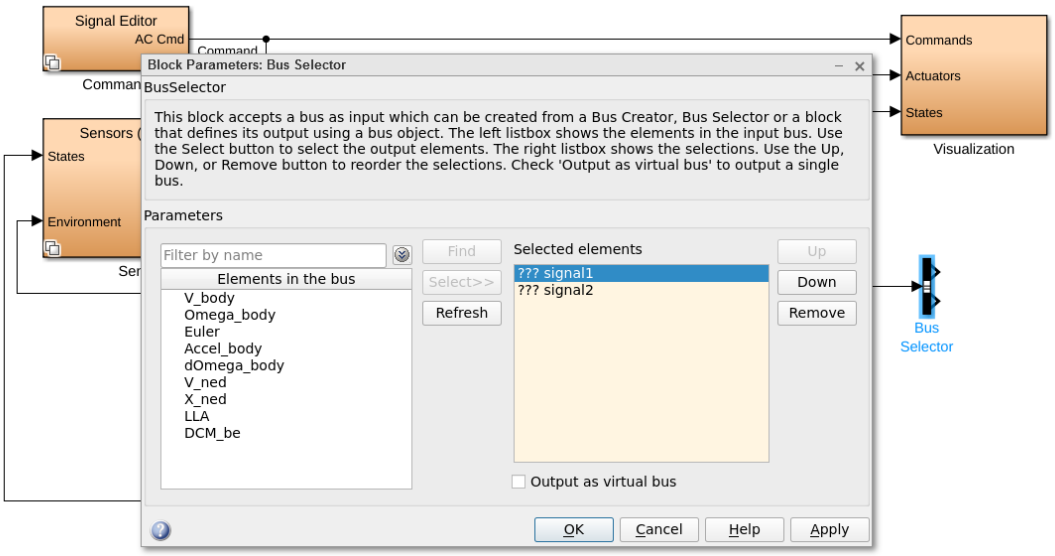


Εικόνα 73. Πλήρες σχηματικό με τα 6 PIDs

Όπως βλέπουμε στο παραπάνω σχήμα που είχαμε χρησιμοποιήσει και νωρίτερα έχουμε **4 PID στα inner loop controllers** και **2 PID στο outer loop controller**. Για να ελέγξουμε αρχικά το σύστημά μας θα πρέπει να αφαιρέσουμε το **outer loop controller** εντελώς και να θεωρήσουμε μηδενικά τα **roll**, **pitch** και **yaw**. Στη συνέχεια θα πρέπει να αφαιρέσουμε το **Sensing and Estimation** τμήμα. Με αυτόν τον τρόπο θεωρούμε ότι ο θόρυβος και άλλοι εξωγενείς παράγοντες δεν θα μας επηρεάσουν στην προσομοίωση. Ουσιαστικά θεωρούμε πως το UAV μας γνωρίζει την ακριβή του θέση στον τρισδιάστατο χώρο με ακρίβεια. Σε περίπτωση που η λειτουργία του UAV μας δεν είναι ομαλή το επαναφέρουμε στην προηγούμενη κατάσταση και συνεχίζουμε έτσι τις δοκιμές. Σε περίπτωση που η λειτουργία μας για το thrust είναι ομαλή συνεχίζουμε στο yaw, μετέπειτα στο roll και τέλος στο pitch.



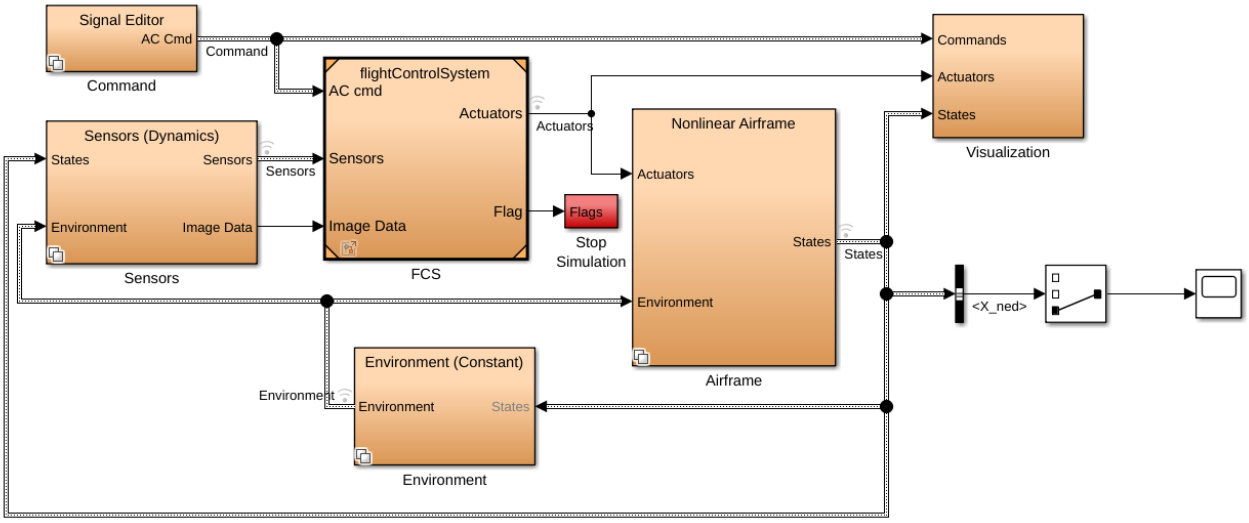
Εικόνα 74. Ρύθμιση του PID για το thrust



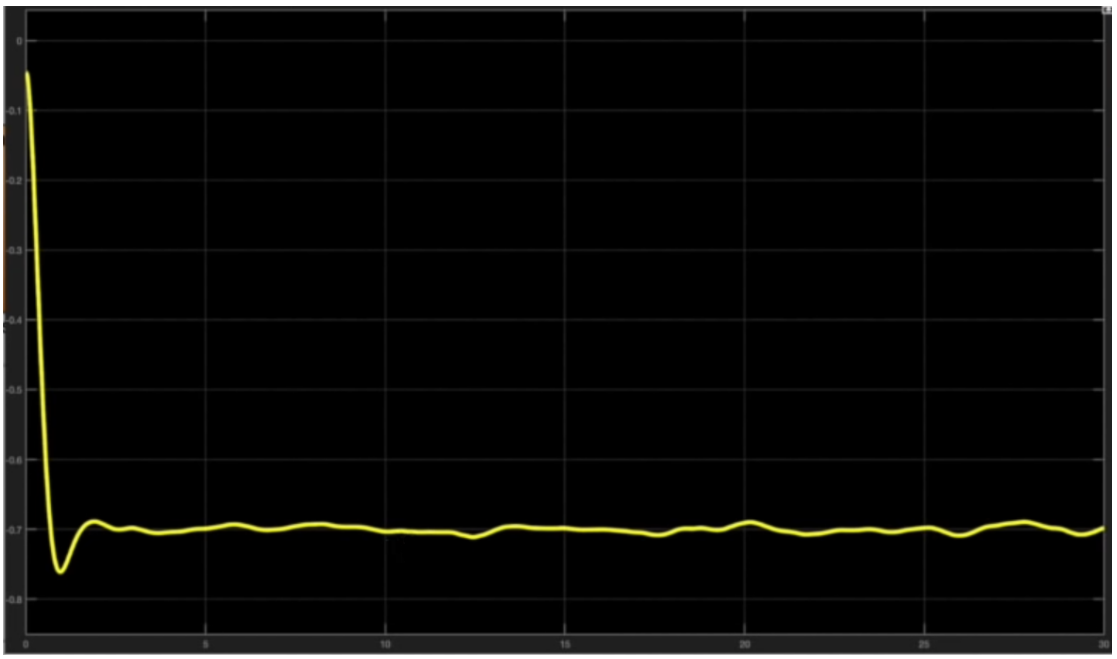
Εικόνα 75. Τοποθέτηση Bus Selector στο Simulink

Πηγαίνουμε λοιπόν στο Simulink και δημιουργούμε έναν **Bus Selector**. Στην επεξεργασία του αφαιρούμε τα 2 σήματα που έχει και προσθέτουμε την **X_ned**.

Προσθέτουμε έναν **Selector** στο σχήμα μας και συνδέουμε πάνω του την έξοδο **X_ned** από τον **Bus Selector**. Στην συνέχεια προσθέτουμε ένα **Scope** ώστε να ελέγξουμε την γραφική παράσταση. Το σχήμα μας πριν το τρέξουμε θα πρέπει να έχει την παρακάτω μορφή.



Εικόνα 76. Σχηματικό για τον έλεγχο με Scope

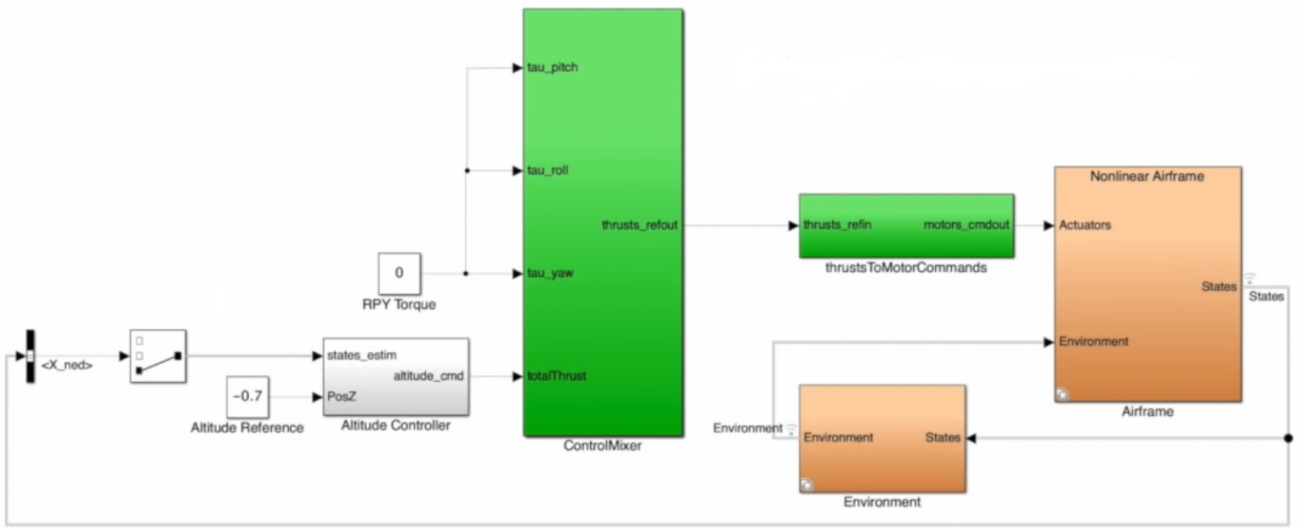


Εικόνα 77. Έξοδος του Scope

Αφού πατήσουμε **Run** βλέπουμε την παραπάνω έξοδο από το Scope. Όπως βλέπουμε αρχικά είχε μια μικρή καθυστέρηση στην ευθυγράμμιση αλλά παρ'όλα αυτά διατηρεί μια σχετικά σταθερή θέση στο -0.7 όπως περιμέναμε.

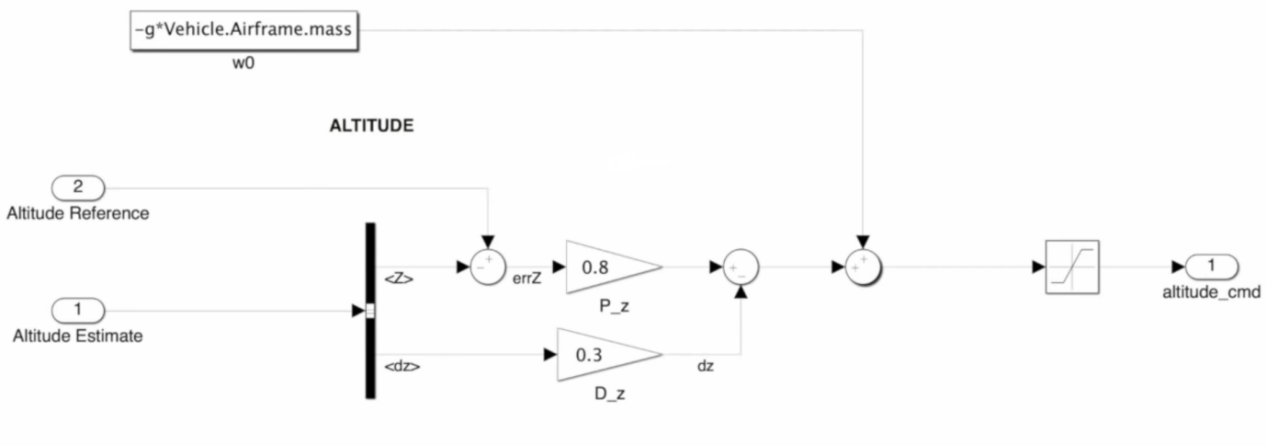
Στην συνέχεια αρχίζουμε να διαγράφουμε κάποια τμήματα του συστήματός μας. Διαγράφουμε όσα προσθέσαμε μόλις μαζί με το **visualize**. Έπειτα μεταφερόμαστε στο παρακάτω path

και παίρνουμε τα: **Altitude Reference**, **ControlMixer** και **thrustsToMotorCommands**. Αυτά τα μεταφέρουμε στην 1η οθόνη του συστήματός μας και τα συνδέουμε όπως φαίνεται στην φωτογραφία παρακάτω, διαγράφοντας το υπόλοιπο **FCS**.



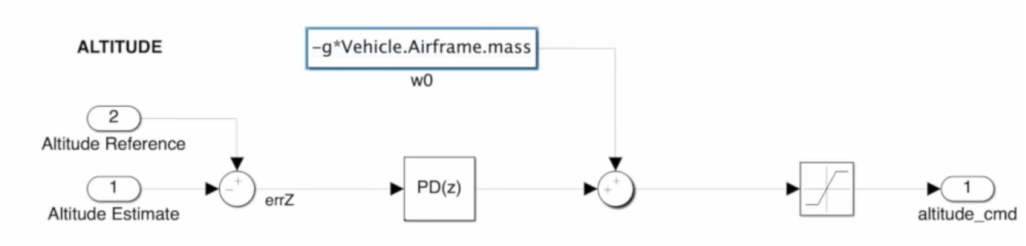
Εικόνα 78. Σχηματικό μετά τις αλλαγές

Το σχήμα μας λοιπόν θα πρέπει να έχει την παραπάνω μορφή. Έχουμε θέσει τα yaw, roll και pitch στο 0 και έχουμε συνδέσει μόνο τον Altitude Controller στο thrust.



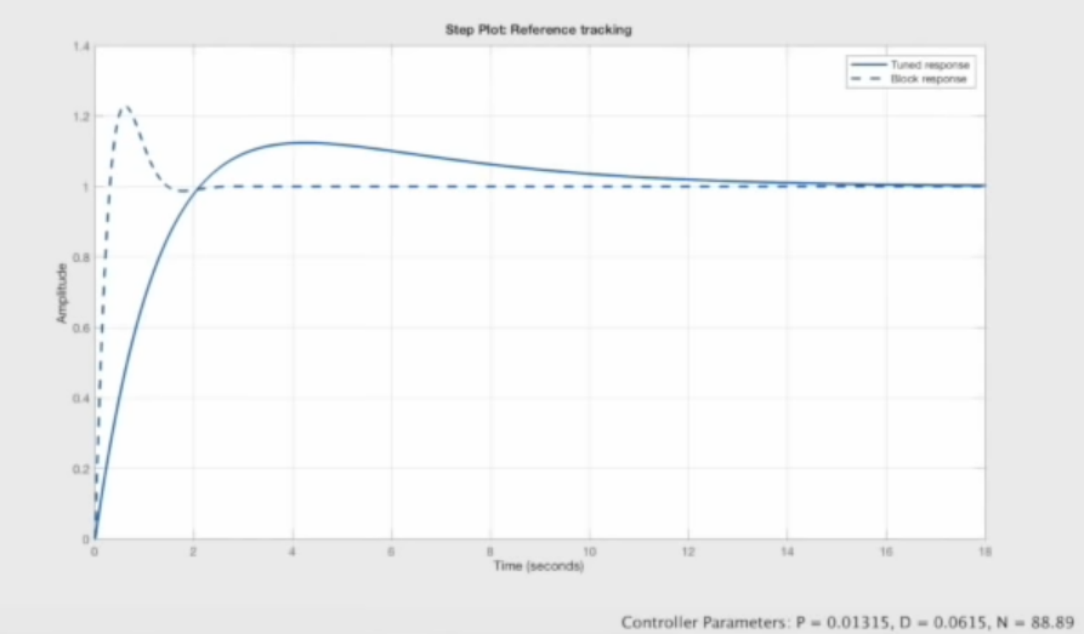
Εικόνα 79. Σχηματικό υπολογισμού του Altitude

Εντός του **Altitude Controller** το αρχικό σχήμα είναι κάπως έτσι. Θα προσθέσουμε έναν **PID Controller**. Θέτουμε την λειτουργία του από το μενού “Επεξεργασία” σε **PD** και τις τιμές του ως: **source - internal**, **Proportional (P) - 0.8 και Derivative (D) - 0.3** . Στη συνέχεια διαγράφουμε τα υπάρχοντα στοιχεία PD από το σχήμα μας και τα αντικαθιστούμε με το PD που δημιουργήσαμε. Τώρα είναι έτοιμο να κάνει **Tune**.



Εικόνα 80. Σχηματικό υπολογισμού του Altitude με χρήση PD

Αυτή είναι η έξοδος που έχουμε:

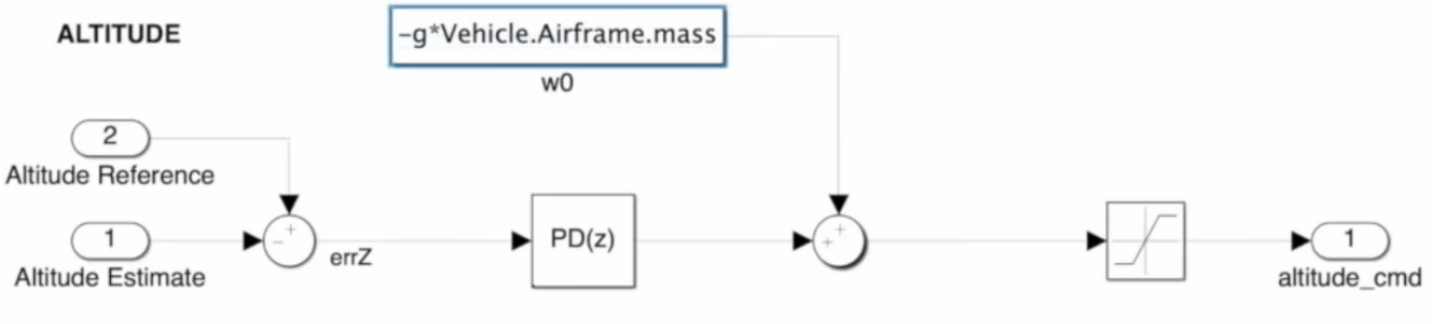


Εικόνα 81. Έξοδος από το σχηματικό της εικόνας 80

```
val =

  A =
              ub,vb,wb   xe,ye,ze
   ub,vb,wb          1          0
   xe,ye,ze      0.005          1

  B =
              PID Controll
   ub,vb,wb        0.07937
   xe,ye,ze      0.0001984

  C =
                  ub,vb,wb   xe,ye,ze
   PID Controll          0          1

  D =
                  PID Controll
   PID Controll               0

Sample time: 0.005 seconds
Discrete-time state-space model.
```

Εικόνα 82. Έξοδος από το σχηματικό της εικόνας 80 στο αρχείο Plant

Παρατηρούμε ότι η διακεκομμένη γραμμή είναι παρόμοια με την έξοδο που είδαμε πριν από το Scope. Είναι κάπως πιο σταθερή καθώς έχουμε **αφαιρέσει το κομμάτι του θορύβου** από τις εισόδους μας. Αλλάζοντας την τιμή στο Proportional (P) που είδαμε σε 0.3 για παράδειγμα, θα παρατηρήσουμε αρκετά κοντινή εικόνα με αυτήν της εξόδου του Scope.
Αυτός ο κώδικας ασφαλώς μπορεί να βρει εμπόδια στην πραγματική πτήση. Ας δούμε για παράδειγμα ξανά ένα σχήμα και να το αναλύσουμε.



Εικόνα 83. Σχηματικό υπολογισμού του Altitude με χρήση PD

Το τμήμα **-g*Vehicle.Airframe.mass** ορίζει την κατάλληλη δύναμη ώστε να απογειωθεί το UAV μας. Αυτό όμως από hardware σε hardware μπορεί να διαφέρει. Είναι άλλωστε αποτελέσματα τα οποία διαφοροποιούνται με μικρές αλλαγές στα γραμμάρια. Στο συγκεκριμένο σχήμα εάν θεωρήσουμε ότι το -g*Vehicle.Airframe.mass είναι αρκετό ώστε να ξεπεράσει την βαρύτητα και θέσουμε τις τιμές του PD στο 0 θα παρατηρήσουμε ότι τα μοτέρ μας δουλεύουν αλλά το UAV μας παραμένει στο έδαφος. Αυξάνοντάς την τότε απογειώνεται αλλά δεν μπορεί αυτή η αλλαγή από μόνη της να διατηρήσει σταθερή θέση. Χρειάζεται το **PD Controller** ώστε να διατηρήσει αυτήν την θέση και να μην παρασυρθεί.

4. Συμπεράσματα

Βλέπουμε πως η διαφορά ανάμεσα στα μοντέλα και στην πραγματικότητα είναι πολύ μεγάλη και θα συνεχίσει να είναι καθώς είναι αδύνατο να προβλέψουμε οποιαδήποτε ξαφνική μεταβολή μπορεί να συμβεί στο περιβάλλον.

Μέσα από αυτήν την προσομοίωση στόχος μας ήτανε να ελέγξουμε εάν η Linear λογική σχεδίασης μοντέλων είναι καλύτερη για την προσομοίωση σε σχέση με την Non Linear. Καταλήγουμε λοιπόν στα παρακάτω συμπεράσματα.

Non Linear Model: Ακολουθώντας αυτόν τον τρόπο σχεδίασης έχουμε πολύ καλύτερο έλεγχο στις αλλαγές που σκοπεύουμε να κάνουμε σε μεγαλύτερη κλίμακα. Όταν έχουμε ένα περίπλοκο σύστημα στο οποίο καλούμαστε να κάνουμε μια μικρή αλλαγή και γνωρίζουμε ότι είναι ελεγχόμενη είναι προτιμότερη η χρήση τέτοιων μοντέλων.
Μοντελοποιώντας διαδοχικά τα επιμέρους τμήματα του UAV μας καλούμαστε να ορίσουμε τιμές και συμπεριφορές πολύ στοχευμένα για το κάθε κομμάτι που δουλεύουμε εκείνη την στιγμή. Επομένως μια ρεαλιστική απεικόνιση από το πρώτο μας βήμα θα μας δυσκολέψει αισθητά χωρίς κάποιο σοβαρό αντίκρισμα στο αποτέλεσμά μας.

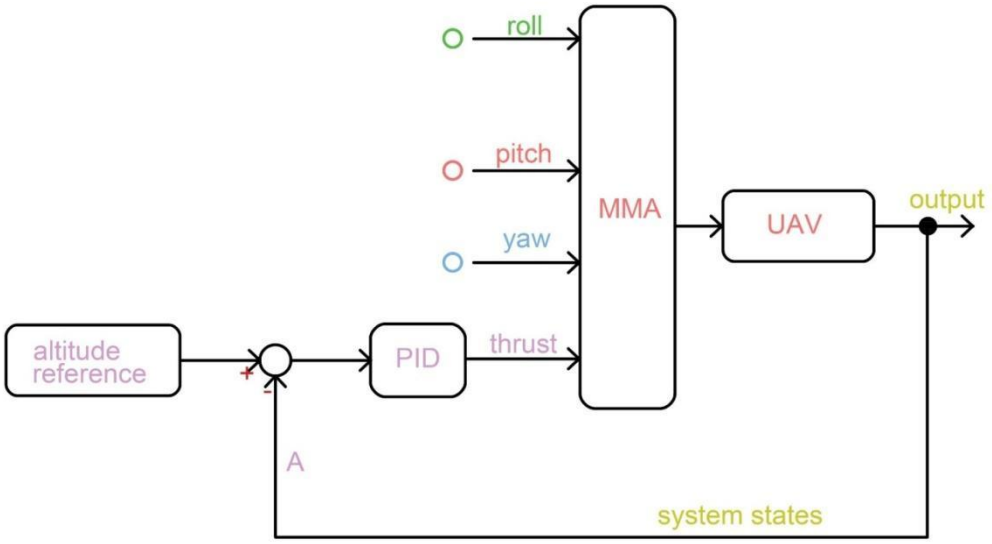
Linear Model: Χρησιμοποιώντας αυτόν τον τρόπο σχεδίασης είμαστε σε θέση να θεωρούμε ως δεδομένες κάποιες τιμές (όπως μας βολεύει την εκάστοτε φορά) και να επικεντρωθούμε στο τμήμα που θέλουμε να αλλάξουμε συγκεκριμένα χωρίς άλλες παρεμβολές. Αυτό δεν είμαστε σε θέση να το πετύχουμε στο Non Linear Model καθώς αποτελείται σε μεγάλο βαθμό από μη ελεγχόμενες παραμέτρους. Για παράδειγμα μπορεί να θέλουμε να ελέγξουμε το thrust σε ένα σύστημα θέτοντας τα roll, pitch και yaw στο 0 ή σε χαμηλές τιμές προκειμένου να μην επηρεάσουν το thrust (εάν θελήσουμε να το δοκιμάσουμε και εκτός προσομοίωσης). Αμφισβητούμε εντελώς δηλαδή τον θόρυβο τον αισθητήρων ή τις αλλαγές στην ατμοσφαιρική πίεση την ταχύτητα του ανέμου κ.α.

Εφόσον το τμήμα που θέλουμε να αλλάξουμε μας δίνει τα αναμενόμενα αποτελέσματα και η ελεγχόμενη αλλαγή μας θεωρείται επιτυχημένη τότε μπορούμε να προσθέσουμε οι ίδιοι θόρυβο ή άλλες παραμέτρους που ενδέχεται να επηρεάσουν το αποτέλεσμά μας, είτε όλα μαζί είτε ένα την φορά ώστε να βλέπουμε την ξεχωριστή απόκριση του συστήματός μας και να δίνουμε λύση στο ξεχωριστό πρόβλημα κάθε φορά.

Αναλυτικότερα τα μη γραμμικά μοντέλα αποτελούν μια έμπιστη λύση σε περίπτωση που έχουμε μεγάλη εμπειρία στον τομέα. Είναι μια πιθανή λύση (και κάποιες φορές καλύτερη από την γραμμική) σε περιπτώσεις που έχουμε αναπτύξει από την αρχή το σύστημά μας. Έτσι θα είμαστε σε θέση να γνωρίζουμε σε πολύ μεγάλο βαθμό εάν κάποια αλλαγή θα επηρεάσει την ευστάθειά του. Σε ένα σύστημα πολυπαραγοντικό όπως είναι η πτήση ενός UAV και η δοκιμή του κώδικα που αναπτύσσουμε έχουμε δει ότι οι αστάθμητοι παράγοντες μπορούν να κάνουν το πείραμά μας να αποτύχει και να βγάλουμε λάθος συμπεράσματα. Ακόμη δεν πρέπει να ξεχνάμε ότι αποτελεί προσομοίωση και όχι πραγματική πτήση. Για να είμαστε βέβαιοι θα πρέπει πάντα να ελέγχουμε τον κώδικά μας και σε πραγματική πτήση ώστε να έρχεται σε αντίστιξη με το UAV που έχουμε.

Για παράδειγμα με μια αλλαγή στους έλικες του UAV μας ώστε να έχουμε μεγαλύτερη επιφάνεια η ευστάθεια του κώδικά μας δεν θα παραμείνει σταθερή. Ή ακόμη αλλάζοντας σκελετό ώστε να είναι πιο βαρύ και δυσκολότερο να παρασυρθεί από ρεύματα αέρα αλλάζει εντελώς η σταθερότητά του ακόμα και στην απογείωση.
Αντίστοιχα τα γραμμικά μοντέλα μας δίνουν την σταθερότητα που χρειαζόμαστε ώστε να ελέγξουμε και να προβλέψουμε την συμπεριφορά του συστήματός μας σε περίπλοκες συνθήκες. Με την μεθοδολογία που ακολουθήσαμε μπορούμε διαιρώντας το σύστημα σε μικρότερα υποσυστήματα κ.ο.κ. να ελέγξουμε την κάθε πτυχή του κώδικα. Ακόμη και κάποιο ελάττωμα στο hardware θα είμαστε σε θέση να το ελέγξουμε. Όλα αυτά τα τμήματα δουλεύοντάς τα παράλληλα θα καθιστούσαν εξαιρετικά δύσκολο και αναίτια πολύπλοκο τον έλεγχο και την προσομοίωση.

Επίσης μια αναφορά αξίζει να γίνει στο τι εννοούμε ακριβώς όταν λέμε **Linear** Και **Non Linear Models**. Υπάρχει πάντα η επιλογή της χρήσης κάποιων ενδιάμεσων μοντέλων που συνδυάζουν κάποια υποσυστήματα Non Linear και κάποια Linear τα οποία και πιθανότατα να είναι πιο εύκολα να “πειράξουμε”. Μπορούμε ας πούμε κατανοώντας πλήρως την λειτουργία και την αρχιτεκτονική ενός συστήματος να αναπτύξουμε το Linear κομμάτι του και ελέγχοντάς το όταν πλέον είμαστε σίγουροι για την λειτουργία του να δοκιμάσουμε την απόκρισή του με ένα Non Linear κομμάτι. Στο παράδειγμα ας πούμε που αναφέραμε αφού ελέγξουμε τα thrust, yaw, roll και pitch στη συνέχεια να το συνδέαμε άμεσα με ένα Non Linear κομμάτι κώδικα που θα παράγει θόρυβο στους αισθητήρες μας, ή θα αλλάζει απότομα το έδαφος δημιουργώντας μεγάλα ύψη ή χαράδρες.



Εικόνα 84. Ρύθμιση του PID για το thrust

Όσον αφορά την πλατφόρμα προσομοίωσης που επιλέχθηκε. Το Matlab-Simulink αποτελεί ένα πρόγραμμα, και αντίστοιχα ένα περιβάλλον εντός αυτού, υπολογιστικής αριθμητικής. Λόγω της δημοφιλίας του και της χρήσης του από διαφορετικούς τομείς και ειδικεύσεις, ακόμα και εκτός της τεχνολογικής επιστήμης, έχει αναπτυχθεί ένας μεγάλος αριθμός αντίστοιχων περιβαλλόντων όπως το Simulink. Γι'αυτόν τον λόγο υπάρχει ένας αντίστοιχα μεγάλος αριθμός επιπρόσθετων (add-ons) διαθέσιμων ώστε να τα χρησιμοποιήσουμε, να παρακολουθήσουμε projects άλλων, ακόμη και να μοιραστούμε κάποια από τα δικά μας. Ασφαλώς πολλά από αυτά συμβαίνουν και σε άλλες πλατφόρμες αλλά συνολικά η βοήθεια που προσφέρει στον χρήστη (ανεξαρτήτως επιπέδου γνώσης) είναι πολύ μεγάλη. Διαθέτει μεγάλο σύνολο από ολοκληρωμένους αλγόριθμους ώστε να βοηθηθεί ο χρήστης να ελέγξει το κάθε ξεχωριστό τμήμα του σχηματικού του, καθώς και έναν εύκολο και γρήγορο τρόπο να ελεγχθούν επιμέρους τμήματα με την χρήση των Scopes. Διαθέτει έναν μεγάλο αριθμό έτοιμων λύσεων σε περίπτωση που χρειαστεί. Ακόμη έχει προφανώς άμεση σύνδεση με το Matlab. Λόγω της εξαιρετικά μεγάλης συνδεσιμότητας του προγράμματος και της γλώσσας, μας δίνει μεγάλη προσαρμοστικότητα τόσο σε διαφορετικά λειτουργικά συστήματα, όσο και στην ίδια την πλακέτα του Ardupilot.

Ακόμη μέσα από τον κώδικά του είμαστε σε θέση είτε να χρησιμοποιήσουμε έτοιμα περιβάλλοντα με προσομοιώσεις σε πραγματικές συνθήκες πτήσεις (έντονα καιρικά φαινόμενα, αλλαγές στην κατεύθυνση ή την πυκνότητα του αέρα, αλλαγές στο έδαφος κ.α.). Μπορούμε ακόμη με μεγάλη ευκολία να δημιουργήσουμε προσομοιώσεις συγκεκριμένων διαδρομών αν θέλουμε για παράδειγμα να κατευθύνουμε ένα UAV σε μια υπαρκτή περιοχή. Αυτά τα οφέλη ασφαλώς έρχονται σε συνάρτηση με το γεγονός ότι είναι ένα πρόγραμμα που μεγάλη μερίδα κόσμου το έχει χρησιμοποιήσει. Διατηρεί την προγραμματιστική λογική και γι'αυτόν τον λόγο είναι εύκολο να προσαρμοστεί ακόμη και ένα άτομο που του είναι ξένο ως πρόγραμμα αλλά γνωρίζει προγραμματισμό.

Πηγές

Wikipedia για τα Μη Επανδρωμένα Οχήματα -
https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle

Ardupilot - https://ardupilot.org/
υποενότητες:
https://ardupilot.org/dev/docs/simulation-2.html
https://ardupilot.org/dev/docs/using-sitl-for-ardupilot-testing.html
https://ardupilot.org/dev/docs/learning-ardupilot-uarts-and-the-console.html

Extended Kalman Filter: https://github.com/shazraz/Extended-Kalman-Filter

Σύνδεση Ardupilot και Matlab: https://github.com/Georacer/ardupilog

Matlab/Simulink:
UAVToolbox:https://www.mathworks.com/help/uav/index.html?s_tid=CRUX_lftnav
Developing UAV Applications with Matlab and Simulink:
https://www.mathworks.com/videos/developing-uav-applications-with-matlab-and-simulink-1624305602400.html
Unmanned Aerial Vehicles using Matlab and Simulink:
https://www.mathworks.com/videos/unmanned-aerial-vehicles-using-matlab-and-simulink-1607017122691.html
Visualize Custom Flight Log:
https://www.mathworks.com/help/uav/ug/visualize-custom-flight-log.html

Quadcopter Project:
https://www.mathworks.com/help/aeroblks/quadcopter-project.html

Matlab Support Package for Parrot Drones:
https://www.mathworks.com/help/supportpkg/parrotio/

UAV Scenario Tutorial:
https://www.mathworks.com/help/uav/ug/uav-scenario-tutorial.html

Flight Log Analysis:
https://www.mathworks.com/help/uav/flight-log-analysis.html?s_tid=CRUX_lftnav

UAV Scenario Designer:
https://www.mathworks.com/help/uav/ref/uavscenariodesigner-app.html

Scenario Simulation:
https://www.mathworks.com/help/uav/scenario-simulation.html

Drone Simulation and Control:
https://www.mathworks.com/videos/drone-simulation-and-control-part-1-setting-up-the-control-problem-1539323440930.html?s_tid=srchtitle_Drone%2520Simulation%2520and%2520Control%252C%2520Part%2520_2