ΑΝΩΤΑΤΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΕΙΡΑΙΑ ΤΕΧΝΟΛΟΓΙΚΟΥ ΤΟΜΕΑ

Α.Ε.Ι ΠΕΙΡΑΙΑ Τ.Τ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΕΛΕΓΧΟΣ ΚΑΙ ΜΟΝΤΕΛΟΠΟΙΗΣΗ ΜΗ ΕΠΑΝΔΡΩΜΕΝΟΥ ΙΠΤΑΜΕΝΟΥ ΟΧΗΜΑΤΟΣ

ΣΠΥΡΟΠΟΥΛΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ Α.Μ:41865

ΜΥΛΩΝΑΣ ΕΥΑΓΓΕΛΟΣ Α.Μ:42311

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΝΙΚΟΛΑΟΥ ΓΡΗΓΟΡΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΑΥΤΟΜΑΤΙΣΜΟΥ

ΜΑΡΤΙΟΣ 2017

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΑΥΤΟΜΑΤΙΣΜΟΥ

Π.Ράλλη & Θηβών 250, 12244 Αιγάλεω, Αθήνα - Ελλάδα

Τηλ. 210-5381488

Ευχαριστούμε ιδιαίτερα τον κ. Γρηγόρη Νικολάου για την μεγάλη βοήθεια που μας έδωσε κατά τη διάρκεια αυτής της εργασίας, και τους καθηγητές Sertac Karaman και Fabian Reither από το MIT για το πολύτιμο υλικό που διένειμαν ελεύθερα.

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο / Η κάτωθι υπογεγραμμένος / η Μυλωνάς Ευάγγελος,
του Ιωάννης, με αριθμό μητρώου 42311 φοιτητής / τρια του Τμήματος **Μηχανικών Αυτοματισμού Τ.Ε.** του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

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

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

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

Ο Δηλών

Ημερομηνία
14/3/2017

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο / Η κάτωθι υπογεγραμμένος / η Χαραλαμπίνος Σπυρίδων

του Μιχαήλ, με αριθμό μητρώου 41865 φοιτητής / τρια του Τμήματος **Μηχανικών Αυτοματισμού Τ.Ε.** του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

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

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

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

Ο Δηλών

Ημερομηνία

14 / 3 / 2017

v.mylonas1993@gmail.com
kostas.spy93@gmail.com

Περίληψη

Στη σημερινή εποχή, τα μη επανδρωμένα ιπτάμενα οχήματα (UAV) είναι ένα σημαντικό κομμάτι της σύγχρονης τεχνολογίας και η χρήση τους είναι αρκετά διαδεδομένη, μιας και έχουν πολλά πλεονεκτήματα. Στόχος αυτής της εργασίας είναι η μελέτη και η σχεδιαση αλγόριθμων ελέγχου του quadcopter και υλοποιηση τους για απομακρυσμένο χειρισμό και αυτονομία του οχήματος με διάφορες μεθόδους ελέγχου, χρησιμοποιώντας για ανάδραση την κάμερα και τα IMU αισθητήρια που διαθέτει.

Αρχικά, αναφέρονται οι αρχές λειτουργίας ενός quadcopter με τα μαθηματικά μοντέλα που το διέπουν, καθώς γίνεται και η θεωρητική προσέγγιση στον έλεγχο τόσο σε γενικά πλαίσια όσο και στο συγκεκριμένο μοντέλο. Στη συνέχεια, το θεωρητικό αυτό μοντέλο υλοποιείται σε περιβάλλον εξομοίωσης με τη χρήση του προγράμματος Matlab.

Για την υλοποίηση του σε πραγματικές συνθήκες, χρειάστηκε η εγκατάσταση του υλικολογισμικού, η σύνδεσή του με υπολογιστή που διαθέτει λογισμικό Linux και η εγγραφή κώδικα σε γλώσσα C για τον πλήρη έλεγχο του quadcopter. Οι μέθοδοι ελέγχου που χρησιμοποιήθηκαν είναι απλή αιώρηση πάνω από συγκεκριμένο σημείο, χειρισμό από τον χρήστη μέσω πληκτρολογίου, ακολουθία προκαθορισμένης διαδρομής και ανίχνευση συγκεκριμένων χρωμάτων στο έδαφος με εκτέλεση συγκεκριμένων ενεργείων βάσει αυτών.

Τέλος, γίνεται ανασκόπηση των πειραμάτων και των προβλημάτων που συναντήθηκαν και μελλοντικές βελτιώσεις που μπορούν να υλοποιηθούν.

Abstract

Nowadays, the idea of Unmanned Aerial Vehicles (UAVs) is an important aspect of modern technology and its use is widely rife due to its many advantages that offers. The goal of this project is the study and design of control algorithms, used for remote controlling and autonomy of Rolling Spider with various control methods, using its own vertical camera and its IMU sensors for feedback.

First of all, we explain the operation principles of a quadcopter, the mathematical models that refer to it and the theoretical approach of control, not only about this specific model, but also in general terms. Next, we implement this theoretical approach that we developed in a simulation environment, using Matlab.

In order to entirely control all of this in real life, a firmware installation inside the quadcopter, a connection to a computer using Linux software and implementation of C code to control it, is needed. The methods that are used in this study are simple hover above a certain point, control using keyboard, precomputed path following and inspection of landmarks to act accordingly.

In conclusion, we review the results of those experiments, the problems that occurred, possible troubleshooting and future improvements that can be developed.

Περιεχόμενα Σελίδα

Κεφάλαιο 1: Εισαγωγή στα UAV

1.1. Ορισμός και Χρήση UAV

Τα μη επανδρωμένα ιπτάμενα οχήματα γνωστά και ως drones ή UAV (Unmanned Aerial Vehicle) είναι ιπτάμενα οχήματα χωρίς πιλότο εντός του οχήματος. Ο έλεγχος που ασκεί σε αυτά ο άνθρωπος γίνεται εκτός του οχήματος. Μπορούν να ελεγχθούν από κάποιο απομακρυσμένο σημείο (τηλεκατεύθυνση), ή μπορεί να έχουν προγραμματιστεί είτε να ακολουθούν κάποια συγκεκριμένη πορεία είτε να βρίσκουν από μόνα τους την πορεία που θα ακολουθήσουν, έτσι ώστε να μην υπάρχει η ανθρώπινη παρέμβαση όσο αφορά τον έλεγχό τους (αυτόνομα). Αυτό το γεγονός, δηλαδή η επίτευξη του ελέγχου των ιπτάμενων οχημάτων χωρίς την ανάγκη παρουσίας κάποιου ανθρώπου εντός του οχήματος, καθιστά αυτού του είδους τα οχήματα υπερβολικά χρήσιμα. Για το λόγο αυτό υπάρχει ευρεία χρήση τέτοιων οχημάτων σε μέρη που είναι αφιλόξενα για τον άνθρωπο ή σε μέρη και καταστάσεις όπου η ανθρώπινη ζωή μπορεί να τεθεί σε κίνδυνο. Επίσης, γίνεται συχνή χρήση σε πολλές στρατωτικές επιχειρήσεις όπως αναγνώριση, κατασκοπεία και μεταφορά βομβών. Βέβαια υπάρχει και η χρήση αυτού του είδους των οχημάτων για επιστημονικούς σκοπούς ή ακόμη και για εμπορική ή επαγγελματική χρήση.

Εικόνα 1.1.1 Μεταφορά πακέτου Ά Βοηθειών

Εικόνα 1.1.2 Χρηση UAV σε στρατιωτικές επιχειρήσεις

1.2. Κατηγορίες UAV

Υπάρχουν διάφορες κατηγοριοποιήσεις μη επανδρωμένων ιπτάμενων οχημάτων. Αυτά ποικίλουν ανάλογα με το μέγεθός τους που μπορεί να είναι μικρότερο από την παλάμη ενός χεριού (micro UAV ή MAV) και έχουν βάρος κάτω του ενός κιλού, στα Miniature UAV ή SUAS όπου το βάρος τους φτάνει μέχρι και είκοσι-πέντε κιλά και σε αυτά όπου το μεγέθός τους μπορεί να είναι αντίστοιχο με αυτό ενός καθιερωμένου επανδρωμένου αεροσκάφους. Επίσης, τα UAV χωρίζονται ανάλογα με το τρόπο τον οποίο παράγεται η ώθηση του οχήματος. Υπάρχουν δύο μεγάλες κατηγορίες σε αυτή τη περίπτωση. Η κατηγορία των ιπτάμενων οχημάτων με σταθερά πτερύγια (Εικόνα 1.2.1) και η κατηγορία των ιπτάμενων οχημάτων περιστρεφόμενου ρότορα (Εικόνα 1.2.2). Στη πρώτη κατηγορία ανήκουν τα αεροσκάφη στα οποία η ώθησή τους γίνεται χάρη στην κίνησή τους στον οριζόντιο άξονα. Συνήθως η μορφή αυτών των αεροσκαφών μοίαζει με αυτή των αεροπλάνων. Στη δεύτερη κατηγορία ανήκουν τα αεροσκάφη στα οποία η ώθηση γίνεται χάρη στην περιστροφή των ελίκων στον οριζόντιο άξονα, δηλαδή η ώθηση γίνεται στον κάθετο άξονα. Για το λόγο αυτό πλεονεκτούν σε σχέση με την προηγούμενη κατηγορία γιατί δεν χρειάζονται διάδρομο προσγείωσης και απογείωσης. Επίσης σε αυτή την κατηγορία τα ιπτάμενα οχήματα ποικίλουν ανάλογα με τον αριθμό των κινητήρων-ελίκων που διαθέτουν και χωρίζονται σε ελικόπτερα (με δύο έλικες), σε tricopter (με τρείς έλικες), σε quadcopter ή quadrotor (με τέσσερις έλικες), σε hexacopter (με έξι έλικες), σε octacopter (με οκτώ έλικες) κ.λ.π. Για αυτό το λόγο υπάρχουν και χρησιμοποιούνται οι γενικοί όροι multicopter ή multirotor για να προσδιορίσουν αυτού του είδους τα οχήματα. Στη παρούσα εργασία χρησιμοποιείται το quadcopter.

Εικόνα 1.2.1: UAV σταθερών πτερυγίων

Εικόνα 1.2.2: UAV περιστρεφόμενου ρότορα

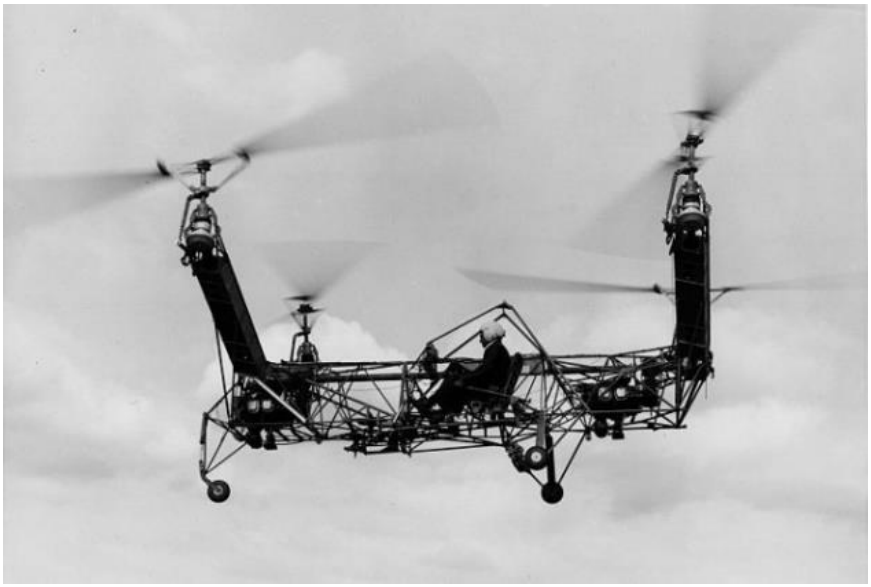
Τα quadcopter ή τετρακόπτερα ανήκουν στην κατηγορία των μη επανδρωμένων αεροσκαφών περιστρεφόμενου ρότορα και διαθέτουν τέσσερις κινητήρες πάνω στους οποίους υπάρχουν προσαρμοσμένοι έλικες. Ο έλεγχός τους μπορεί να γίνει εύκολα ελέγχοντας τις ταχύτητες περιστροφής των τεσσάρων κινητήρων τους.

1.3. Ιστορική Αναδρομή

Κατά τη διάρκεια του 20ού αιώνα από τις στρατιωτικές κυρίως υπηρεσίες εφευρέθηκαν πολλές τεχνολογίες όπως το GPS (Global Position System) και πρωτόκολλα που είναι η βάση του σημερινού Internet. Μια από αυτές τις τεχνολογίες ήταν και τα μη επανδρωμένα αεροσκάφη. Στην αρχή η χρήση τους ήταν μόνο για στρατιωτικές επιχειρήσεις στο πόλεμο. Ο στρατός της Αυστρίας χρησιμοποίησε για πρώτη φορά τέτοιου είδους οχήματα στη μάχη το 1849 όταν επιτέθηκε στη Βενετία της Ιταλίας με μη επανδρωμένα αεροσκάφη γεμάτα εκρηκτική ύλη. Αν και δεν λειτούργησε εξ΄ολοκλήρου το συγκεκριμένο εγχείρημα, λόγω του ανέμου μερικά από τα αεροσκάφη έπεσαν σε Αυστριακό έδαφος, ήταν μια καινοτομία με πολλές προοπτικές. Με τη λήξη του πρώτου παγκόσμιου πολέμου έκαναν την εμφάνισή τους τα πρώτα μη επανδρωμένα αεροπλάνα. Το Hewitt Sperry Automated Airplane, που κατασκευάστηκε από τον Elmer Sperry στην εταιρία Sperry Gyroscope, ήταν ο πρόγονος της σημερινής τορπίλης. Ο Αμερικάνικος στρατός πολύ σύντομα κατασκεύασε–μετέτρεψε το Standard E-1 σε μη επανδρωμένο. Ένα από αυτά, το Laryx, μπορούσε να πλοηγηθεί χωρίς πιλοτο αφού εκτοξευόταν από το πολεμικό πλοίο. Ακολούθησαν πολλά εγχειρήματα κατασκευής τέτοιου είδους αεροσκαφών κυρίως από το Βρετανικό και από τον Αμερικανικό στρατό, όμως ήταν μεμονωμένα. Το πρώτο μαζικής παραγωγής αεροσκάφος ήταν το Reginald Denny, το οποίο είχε το όνομα ενός ηθοποιού του HollyWood. Το ενδιαφέρον του στράφηκε στα μη επανδρωμένα τηλεκατευθυνόμενα αεροσκάφη δημιουργώντας την εταιρία Reginald Denny, όπου αργότερα ενίσχυσε τον Αμερικανικό στρατό στο δεύτερο παγκόσμιο πόλεμο με το Radio Plane το οποίο κατασκεύασε. Βέβαια, ο ίδιος στρατός κατασκεύασε μια τεράστια ποικιλία από μη επανδρωμένα αεροσκάφη τα οποία χρησιμοποιήθηκαν για κατασκοπεία κατά τη διάρκεια του ψυχρού πολέμου.

Τα quadcopter ήταν από τα πρώτα οχήματα που κατάφεραν να απογειωθούν και να προσγειωθούν κατακόρυφα. Πιο νωρίς ήταν τα ελικόπτερα χρησιμοποιώντας έλικα στην ουρά τους για να ισορροπήσουν την ροπή που προκαλεί ο κεντρικός έλικας, που ήταν δύσκολο για τους πιλότους τους να τα χειριστούν και μη αποτελεσματικό. Οι μηχανικοί της εποχής έτσι κατασκεύασαν τα quadcopter για να λύσουν το συγκεκριμένο πρόβλημα και να βοηθήσουν τους πιλότους. Το πρώτο quadcopter ήταν το Omnichen 2 το οποίο κατασκευάστηκε από τον Etienne Omnichen το 1920. Κατάφερε να κάνει χίλιες επιτυχημένες πτήσεις κάνοντας ρεκόρ ύψους 360 μέτρων. Το Convertawings Model A Quadcopter, σχεδιάστηκε από τον Dr. George E. Bothezat, το 1956. Ήταν ο πρώτος που χρησιμοποίησε προωθητή, ή ώθηση προς τα εμπρός,

για τον έλεγχο ενός αεροσκάφους σε roll, pitch και yaw. Ακολούθησε το 1958 το Curtis Wright V27, που κατασκευάστηκε από την εταιρία Curtis.



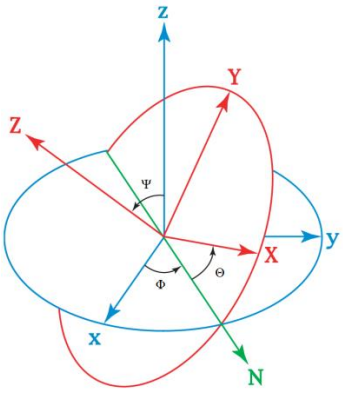
Εικόνα 1.3.1 Converawings Model A Quadcopter

Κεφάλαιο 2: Θεωρητική Προσέγγιση

2.1. Αρχές Λειτουργίας

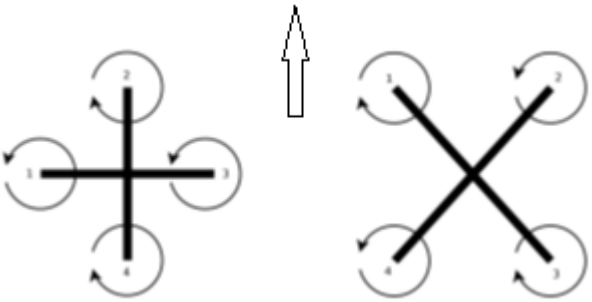
Το quadcopter είναι ένα σύστημα έξι βαθμών ελευθερίας (6 DOF). Οι κινήσεις του περιγράφονται από δώδεκα μεταβλητές κατάστασης. Οι έξι πρώτες αφορούν την κίνηση του quadcopter στον τρισδιάστατο χώρο και τους ρυθμούς μεταβολής τους. Περιγράφονται από τις μεταβλητές X, Y, Z που δηλώνουν την μετατόπιση του στο χώρο, και τις μεταβλητές u, v, w που αφορούν τις μεταβολές της μετατόπισης, δηλαδη την ταχύτητα στον κάθε άξονα.

Οι επόμενες έξι αφορούν τον προσανατολισμό και τους ρυθμούς μεταβολής τους και περιγράφονται από τις μεταβλητές Φ, Θ, Ψ που αντιπροσωπεύουν τις γωνίες Euler (Εικόνα 2.1.1) και τις μεταβολές αυτών, δηλαδή τις γωνιακές ταχύτητες τις οποίες περιγράφουν οι μεταβλητές p, q, r. Το Φ αφορά το roll, που είναι η γωνία περιστροφής του quadcopter γύρω από τον άξονα X. Το Θ αφορά το pitch, δηλαδή την γωνία περιστροφής του γύρω από τον άξονα Y. Τέλος, το Ψ αφορά το yaw, δηλαδή την γωνία περιστροφής γύρω από τον άξονα Z.



Εικόνα 2.1.1 Γωνίες Euler

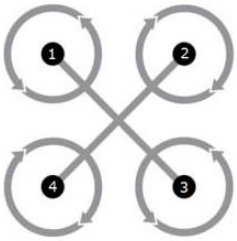
Τα τετρακόπτερα μπορούν να ίπτανται σε δύο σχηματισμούς ανάλογα με τη κατεύθυνση της πτήσης: σε σχηματισμό '+' (Εικόνα 2.1.2) και '×' (Εικόνα 2.1.3). Οι διαφορές τους φαίνονται στο σχήμα.



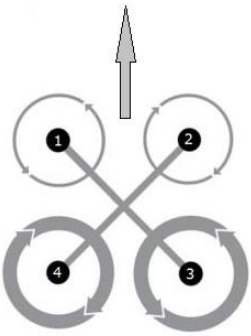
Εικόνα 2.1.2 Εικόνα 2.1.3

Στη περίπτωσή μας, το quadcopter ίπταται σε σχηματισμό '×'. Για την ορθή αιώρησή του απαιτείται το άθροισμα των ροπών των δύο αξόνων να είναι ίσο με το μηδέν. Αυτό επιτυγχάνεται όταν όλοι οι έλικες των κινητήρων έχουν ίσο μετρο γωνιακής ταχύτητας αλλά αντίθετη φορά ανά ζευγάρι. Το quadcopter κάνει σταθερή αιώρηση πάνω από ένα σημείο (hover) όταν το άθροισμα της ώθησης που παράγουν οι τέσσερις κινητήρες ισούται με την δύναμη που ασκεί η βαρύτητα πάνω στο quadcopter (Εικόνα 2.1.4). Στη περίπτωση που το μέτρο της γωνιακής ταχύτητας των τεσσάρων κινητήρων αυξηθεί, τότε αυξάνεται το ύψος του, ενώ στην αντίθετη περίπτωση το ύψος μειώνεται.

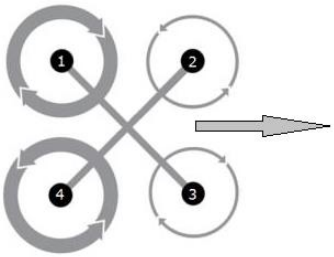
Οι ταχύτητες περιστροφής των τεσσάρων κινητήρων είναι ανεξάρτητες. Χάρη σε αυτό, είναι δυνατόν να ελέγχθούν οι γωνίες Euler. Το pitch μπορεί να ελεγχθεί με αύξηση της γωνιακής ταχύτητας των κινητήρων 3, 4 για κίνηση προς τα εμπρός ή αύξηση της γωνιακής ταχύτητας των κινητήρων 1, 2 για κίνηση προς τα πίσω (Εικόνα 2.1.5). Το roll μπορεί να ελεγχθεί με αύξηση της γωνιακής ταχύτητας των κινητήρων 1, 4 για κίνηση προς τα δεξιά ή με αύξηση της γωνιακής ταχύτητας των κινητήρων 2, 3 για κίνηση προς τα αριστερά (Εικόνα 2.1.6). Τέλος, το yaw μπορεί να ελεγχθεί με αύξηση της γωνιακής ταχύτητας των κινητήρων 1, 3 για περιστροφή του γύρω από τον άξονα z ή με αύξηση της γωνιακής ταχύτητας των κινητήρων 2, 4 για αντίθετη περιστροφή (Εικόνα 2.1.7).



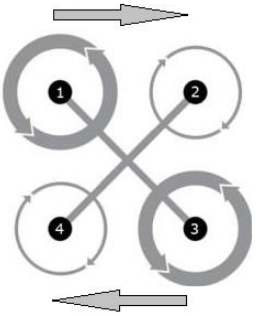
Εικόνα 2.1.4



Εικόνα 2.1.5



Εικόνα 2.1.6



Εικόνα 2.1.7

2.2. Μαθηματική Μοντελοποίηση Quadcopter

Σύμφωνα με τις εξισώσεις κίνησης έχουμε:

$$\vec{F} = m\frac{d^2\vec{x}}{dt} \quad (1)$$

$$\vec{T} = \frac{d\vec{H}}{dt} \quad (2)$$

όπου $\vec{F}$ είναι η δύναμη και $\vec{T}$ είναι η ροπή που ασκείται στο quadcopter, $\vec{x}$ είναι η μετατόπιση και $\vec{H}$ είναι η ορμή που παράγεται από την γωνιακή ταχύτητα.

Οι εξισώσεις που προκύπτουν είναι:

$$\frac{1}{m}\overrightarrow{F_b} = \frac{d^2\vec{x}}{dt} + \overrightarrow{\omega_b} * \overrightarrow{u_b} \quad (3)$$

$$\overrightarrow{T_b} = \frac{d\overrightarrow{H_b}}{dt} + \overrightarrow{\omega_b} * \overrightarrow{H_b} \quad (4)$$

όπου $\overrightarrow{u_b} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$, $\overrightarrow{\omega_b} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$ είναι οι ταχύτητες στους x, y, z άξονες και οι γωνιακές ταχύτητες στο quadcopter.

Αφού η κίνηση γίνεται στο τρισδιάστατο χώρο, συνεπάγεται:

$$\frac{1}{m}\begin{bmatrix} Fx \\ Fy \\ Fz \end{bmatrix} = \begin{bmatrix} \dot{U} \\ \dot{V} \\ \dot{W} \end{bmatrix} + \begin{bmatrix} 0 & -R & Q \\ R & 0 & -P \\ -Q & P & 0 \end{bmatrix}\begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} \dot{U} + QW - RV \\ \dot{V} + RU - PW \\ \dot{W} + PV - QU \end{bmatrix} \quad (5)$$

Θέτοντας Θ, Φ, Ψ τις γωνίες των περιστροφών γύρω από τους άξονες x, y, z αντίστοχα, ισχύει ότι οι πίνακες περιστροφής για τον κάθε άξονα είναι:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\Theta & sin\Theta \\ 0 & -sin\Theta & cos\Theta \end{bmatrix}, R_y = \begin{bmatrix} cos\Phi & 0 & -sin\Phi \\ 0 & 1 & 0 \\ sin\Phi & 0 & cos\Phi \end{bmatrix}, R_z = \begin{bmatrix} cos\Psi & sin\Psi & 0 \\ -sin\Psi & cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Για τον καθορισμό της μετατόπισης και του προσανατολισμού, και κατ' επέκταση της γραμμικής και γωνιακής ταχύτητας και επιτάχυνσης, γίνεται καθορισμός σημείων αναφοράς ανάλογα με το τι θεωρείται σταθερό. Ο καθορισμός αυτός γίνεται είτε θεωρώντας ως σημείο αναφοράς τη Γή, που ονομάζεται σταθερό πλαίσιο, είτε θεωρώντας ως σημείο αναφοράς το κέντρο της μάζας του quadcopter, που ονομάζεται πλαίσιο αναφοράς. Για να επιτευχθεί η μεταφορά απο το ένα πλαίσιο στο άλλο, δημιουργούνται οι πίνακες D και E.

Ο πίνακας D περιέχει τους πίνακες περιστροφής του κάθε άξονα και καθιστά εφικτή τη μεταφορά από το σταθερό πλαίσιο (Γη), στο πλαίσιο αναφοράς (κέντρο μάζας quadcopter):

$$D = R_x R_y R_z \quad =>$$

$$D=\begin{bmatrix} cos\Theta \cos\Psi & cos\Theta\, sin\Psi & -sin\Theta \\ sin\Phi\, sin\Theta\, cos\Psi - cos\Phi\, sin\Psi & sin\Phi\, sin\Theta\, sin\Psi + cos\Phi\, cos\Psi & sin\Phi\, cos\Theta \\ cos\Phi\, sin\Theta\, cos\Psi + sin\Phi\, sin\Psi & cos\Phi\, sin\Theta\, sin\Psi - sin\Phi\, cos\Psi & cos\Phi\, cos\Theta \end{bmatrix} \quad (6)$$

Δηλαδή:

$$D\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}u\\ v\\ w\end{bmatrix} \quad (7)$$

όπου τα x, y, z δηλώνουν τις συντεταγμένες του quadcopter.

Αντίστοιχα ο πίνακας E χρησιμοποιείται για τη μεταφορά των γωνιών από το σταθερό πλαίσιο στο πλαίσιο αναφοράς:

$$E = \begin{bmatrix} 1 & 0 & -sin\Phi \\ 0 & cos\Theta & sin\Theta cos\Phi \\ 0 & -sin\Theta & cos\Theta cos\Phi \end{bmatrix} \quad (8)$$

Δηλαδή:

$$E\begin{bmatrix}\dot{\Theta}\\ \dot{\Phi}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}p\\ q\\ r\end{bmatrix} \quad (9)$$

Οι δυνάμεις οι οποίες ασκούνται στο quadcopter είναι η βαρύτητα και η ώθηση των κινητήρων. Η βαρύτητα δρά στην θετική κατεύθυνση του άξονα z, σύμφωνα με τον σταθερό πλαίσιο, ενώ η ώθηση των κινητήρων στην αρνητική κατεύθυνση του z, σύμφωνα με το πλαίσιο αναφοράς. Για να επιτευχθεί η εξίσωση των δυνάμεων, πρέπει να οριστούν και οι δύο στο πλαίσιο αναφοράς. Με την βοήθεια του πίνακα D, γίνεται η μεταφορά της βαρυτικής δύναμης στο πλαίσιο αναφοράς, δηλαδή:

$$\frac{1}{m}\left(D\begin{bmatrix}0\\0\\mg\end{bmatrix} - \begin{bmatrix}0\\0\\\tau\end{bmatrix}\right) = \begin{bmatrix}\dot{U} + QW - RV\\ \dot{V} + RU - PW\\ \dot{W} + PV - QU\end{bmatrix} \quad (10)$$

όπου τ είναι η συνολική δύναμη ώθησης που παράγεται από τους κινητήρες.

Συνεπάγεται:

$$\dot{U} = \mathrm{RV} - \mathrm{QW} - \mathrm{g}\sin\Phi \tag{11}$$

$$\dot{V} = \mathrm{PW} - \mathrm{RU} - \mathrm{g}\cos\Phi\sin\Theta \tag{12}$$

$$\dot{W} = \mathrm{QU} - \mathrm{PV} + \mathrm{g}\cos\Theta\cos\Phi - \frac{\tau}{m} \tag{13}$$

Η συνολική ώθηση είναι συνάρτηση της ταχύτητας του κάθε κινητήρα. Ορίζεται ως Ω_i η ταχύτητα του i κινητήρα, η οποία εξαρτάται απο το ρεύμα τροφοδοσίας. Οπότε, το τ υπολογίζεται ως:

$$\tau = \mathrm{b}(\Omega_1^2 + \Omega_1^2 + \Omega_1^2 + \Omega_1^2) \tag{14}$$

Στην εξίσωση (4), ισχύει ότι $\overrightarrow{H_b} = \mathrm{I}\ \overrightarrow{\omega_b}$, όπου I είναι ο πίνακας αδράνειας. Αφού το quadcopter είναι συμμετρικό γύρω από τους άξονες x-z και y-z (λόγω συμμετρίας θεωρείται ότι $I_{xx} \approx I_{yy}$), ισχύει:

$$\mathrm{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

Άρα προκύπτει:

$$\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} \dot{P}\, I_{xx} + QR(I_{zz} - I_{yy}) \\ \dot{Q}\, I_{yy} + PR(I_{xx} - I_{zz}) \\ \dot{R}\, I_{zz} + PQ(I_{yy} - I_{xx}) \end{bmatrix} \tag{15}$$

Δηλαδή από την (14) και την (15) συνεπάγεται:

$$T_x = \mathrm{l}\,\mathrm{b}(\Omega_2^2 - \Omega_4^2) \tag{16}$$

$$T_y = \mathrm{l}\,\mathrm{b}(\Omega_1^2 - \Omega_3^2) \tag{17}$$

$$T_z = \mathrm{d}(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \tag{18}$$

όπου l είναι η απόσταση των ελίκων από το κέντρο μάζας και d είναι η ροπή αδράνειας.

Επομένως συμπεραίνεται ότι:

$$\dot{P} = \frac{l\,b}{I_{xx}}(\Omega_2^2 - \Omega_4^2) - \mathrm{QR}\,\frac{I_{zz}-I_{yy}}{I_{xx}} \quad (19)$$

$$\dot{Q} = \frac{l\,b}{I_{yy}}(\Omega_1^2 - \Omega_3^2) - \mathrm{PR}\,\frac{I_{xx}-I_{zz}}{I_{yy}} \quad (20)$$

$$\dot{R} = \frac{d}{I_{zz}}(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \quad (21)$$

Και από πριν ισχύει:

$$\dot{U} = \mathrm{RV} - \mathrm{QW} - g\sin\Phi \quad (11)$$

$$\dot{V} = \mathrm{PW} - \mathrm{RU} - g\cos\Phi\sin\Theta \quad (12)$$

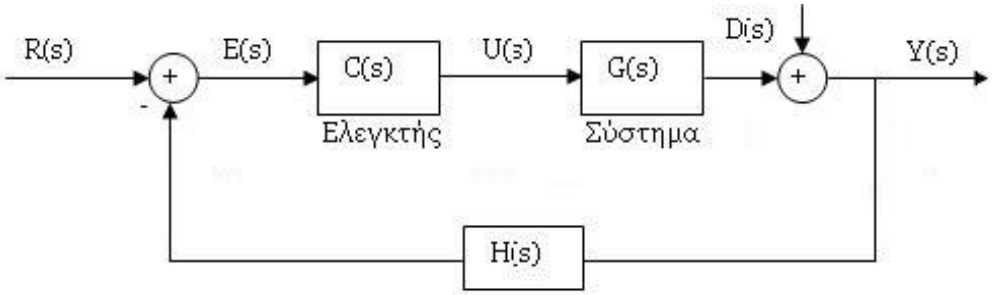
$$\dot{W} = \mathrm{QU} - \mathrm{PV} + g\cos\Theta\cos\Phi - \frac{\tau}{m} \quad (13)$$

2.3. Θεωρία Ελέγχου

Οι κινήσεις που πραγματοποιεί ένα quadcopter καθώς και η επίτευξη της σταθερότητας και της ευστάθειάς του εξαρτώνται από τον ορθό έλεγχο των γωνιακών ταχυτήτων των κινητήρων του. Για να επιτευχθεί αυτός, απαιτείται ένα σύστημα ελέγχου κλειστού βρόγχου. Όπως προαναφέρθηκε, η ταχύτητα του κάθε κινητήρα εξαρτάται από το ρεύμα που τροφοδοτεί τον κινητήρα, οπότε ο έλεγχος αφορά την ισχύ, άρα το ρεύμα που καταναλώνει κάθε χρονική στιγμή ο κινητήρας.

Ένα τυπικό σύστημα κλειστού βρόγχου, στο πεδίο της συχνότητας, αποτελείται από:

- Την είσοδο του συστηματος R(s) (επιθυμητή θέση)
- Τον συγκριτή, ο οποίος υπολογίζει το σφάλμα E(s), δηλαδή τη διαφορά επιθυμητής και πραγματικής εξόδου
- Τον ελεγκτή C(s), ο οποίος διαμορφώνει την είσοδο της διεργασίας, δεδομένου του σφάλματος
- Τη διεργασία-σύστημα G(s) (κινητήρες)
- Τη διαταραχή D(s) του συστήματος που δέχεται από το περιβάλλον
- Την έξοδο του συστήματος Y(s) (τελική θέση)
- Το μετατροπέα H(s) (ανάδραση θέσης)



Εικόνα 2.3.1 Τυπικό σύστημα ελέγχου κλειστού βρόγχου

2.4. Ελεγκτής Συστήματος PID

Ο ελεγκτής του συστήματος C(s) που χρησιμοποιείται στο σύστημα είναι ένας PID ελεγκτής. Αυτού του είδους ο ελεγκτής αποτελείται από το άθροισμα των τριών όρων, του P (Proportional) που είναι ανάλογο του σφάλματος, του I (Integral) που είναι ανάλογο του ολοκληρώματος του σφάλματος και του D (Derivative) που είναι ανάλογο της παραγώγου του σφάλματος. Σκοπός του είναι να ελαχιστοποιήσει ή να μηδενίσει το σφάλμα, έτσι ώστε η πραγματική τιμή της εξόδου να ισούται με την επιθυμητή τιμή.

Η εξίσωση του P δίνεται από τον τύπο: $P = K_p * E(s)$

Η εξίσωση του I δίνεται από τον τύπο: $I = \frac{Ki}{s} * E(s)$

Η εξίσωση του D δίνεται από τον τύπο: $D = K_d * s * E(s)$

Άρα η εξίσωση του PID ελεγκτή δίνεται από τον τύπο:

$$U(s) = \left(K_p + \frac{Ki}{s} + K_d * s\right) * E(s) \quad (21)$$

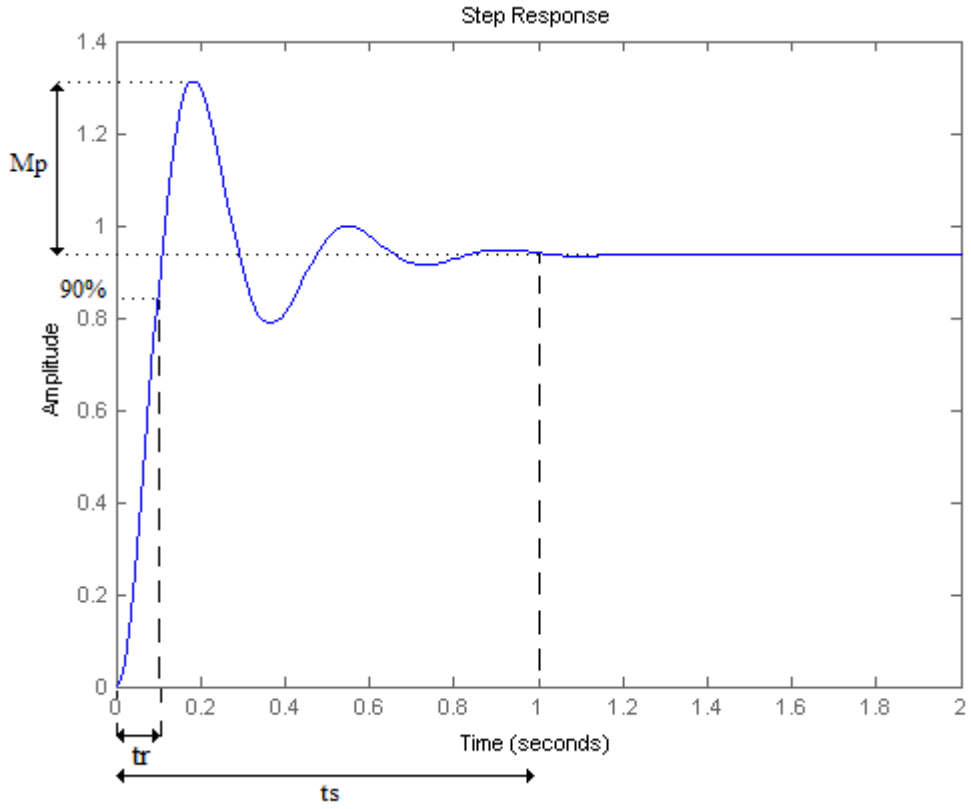
η οποία μπορεί να θεωρηθεί και ως συνάρτηση μεταφοράς του ελεγκτή.

Με την σωστή ρύθμιση των παραμέτρων του ελεγκτή K_p, K_i, K_d μπορεί να επιτευχθεί μεγάλη μείωση, έως και μηδενισμός του μόνιμου σφάλματος.

Εμπειρικά, δίνεται ο πίνακας με την επίδραση που έχει η κάθε παράμετρος στο σύστημα.

Παράμετρος Ελεγκτή	**Μόνιμο Σφάλμα**	**Χρόνος Ανύψωσης tr**	**Χρόνος Αποκατάστασης ts**	**Υπερύψωση Mp**
$\mathbf{K_p}$	Μείωση	Μείωση	Μικρή αλλαγή	Αύξηση
$\mathbf{K_i}$	Μηδενισμός	Μείωση	Αύξηση	Αύξηση
$\mathbf{K_d}$	Μικρή αλλαγή	Μικρή αλλαγή	Μείωση	Μείωση

Πίνακας 1: Επίδραση παραμέτρων PID στο σύστημα



Εικόνα 2.4.1 Απόκριση Συστήματος

2.5. Μέθοδοι Υπολογισμού Παραμέτρων PID

Οι μέθοδοι υπολογισμού των παραμέτρων του PID ποικίλουν ανάλογα με το αν είναι γνωστό ή άγνωστο το μαθηματικό μοντέλο της διεργασίας, την εμπειρία του μηχανικού και την πολυπλοκότητα του συστήματος.

Οι μέθοδοι αυτοί είναι:

- **Ziegler-Nichols**

  Η μέθοδος αυτή στηρίζεται στις εξισώσεις που ανέπτυξαν εμπειρικά οι Ziegler-Nichols χρησιμοποιώντας το K_{crit} και το T_0.

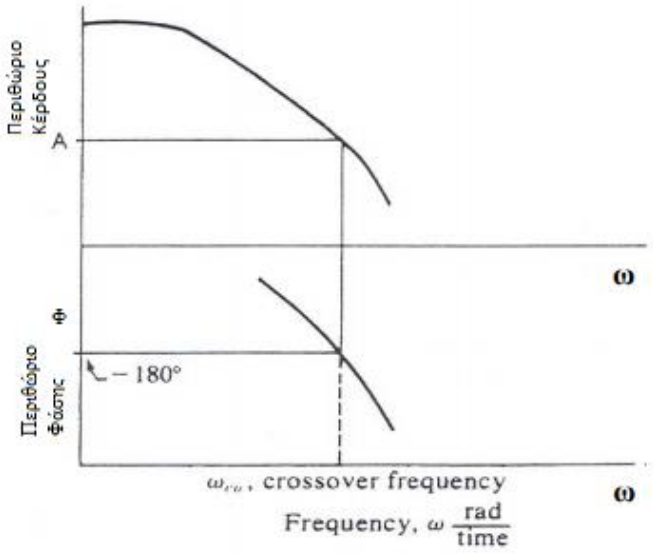
  - Με γνωστό μοντέλο διεργασίας

    i. Ημιπειραματική Μέθοδος

    Οι τιμές των παραμέτρων του ελεγκτή PID τοποθετούνται στην χαμηλότερη δυνατή τιμή τους και αυξάνεται σταδιακά το K_p μέχρι να παρατηρηθεί ταλάντωση σταθερού εύρους στην έξοδο από την οποία υπολογίζεται η συχνότητα T_0. Η τιμή αυτή του K ονομάζεται K_{crit}. Στη συνέχεια, τα K_i και K_d υπολογίζονται με τη βοήθεια του Πίνακα 2.

    ii. Διαγράμματα Bode

    Στη περίπτωση αυτή, το K_{crit} και το T_0 υπολογίζονται απο τα διαγράμματα κέρδους και φάσης στο πεδίο της συχνότητας (διαγράμματα Bode).



Εικόνα 2.5.1 Διαγράμματα Bode

Χρησιμοποιούνται οι εξισώσεις:

$$K_{crit} = 1/A \ , \ T_0 = 2\pi/\omega_0$$

Έπειτα, υπολογίζονται οι παραμέτροι σύμφωνα με τις εξισώσεις του Πίνακα 2.

iii. Αναλυτική Μέθοδος

Στην περίπτωση αυτή, οι τιμές υπολογίζονται αλγεβρικά από την σχέση:

$$-180° = |\underline{G(s)}\ |_{s=j\omega} = |\underline{G(j\omega)}$$

Επιλέγονται τιμές του ω μέχρι να βρεθεί το ω_0 που να ικανοποιεί την σχέση.

Ύστερα, για $\omega = \omega_0$, υπολογίζεται το A από την σχέση:

$$20\log A = 20\log |G(s)|_{s=j\omega}$$

Τέλος, υπολογίζονται τα K_{crit} και T_0 από τις εξισώσεις:

$$K_{crit} = 1/A \ , \ T_0 = 2\pi/\omega_0$$

και οι παράμετροι, στη συνέχεια, από τον Πίνακα 2.

P	$K_p = 0.5 * K_{crit}$	
PI	$K_p = 0.5 * K_{crit}$ $K_i \leq 1.2K_p / T_0$	$K_c = 0.5*K_{crit}$ $T_i = T_0 / 1.2$
PID	$K_p = 0.6 * K_{crit}$ $K_i = 2K_p / T_0$ $K_d \geq 0.125*K_p*T_0$	$K_c = 0.6*K_{crit}$ $T_i = T_0 / 2$ $T_d = T_0 / 8$

Πίνακας 2: Εξισώσεις γνωστού μοντέλου διεργασίας Ziegler-Nichols

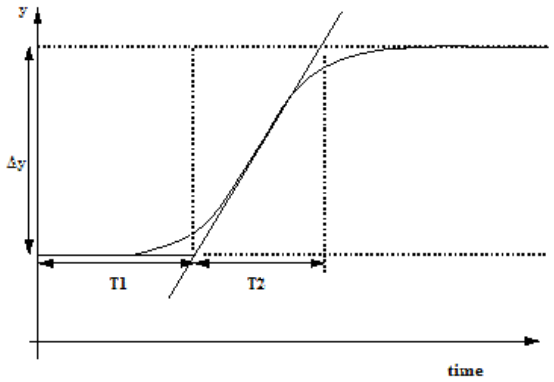
- Με άγνωστο μοντέλο διεργασίας

Εφόσον το μοντέλο της διεργασίας είναι άγνωστο, πρέπει πρώτα να υπολογιστεί η συνάρτηση της διεργασίας G(s). Αυτό γίνεται από την καμπύλη απόκρισης στο πεδίο του χρόνου μετα την έξοδο Y(s), σε βηματική μεταβολή μοναδιαίας τιμής της εισόδου R(s). Σύμφωνα με τον τύπο:

$$G(s) = \frac{K}{1+s*T_2} * e^{-sT_1}$$

υπολογίζονται, κατά προσέγγιση, οι παράμετροι της άγνωστης συνάρτησης μεταφοράς G(s) από την χρονική απόκριση ανοιχτού βρόγχου.

Οι τιμές K, T_1 και T_2 βρίσκονται από τη γραφική παράσταση της απόκρισης του συστήματος στη συγκεκριμένη βηματική είσοδο, όπως φαίνεται στην Εικόνα 4.2.2.



Εικόνα 2.5.2 Βηματική απόκριση ανοιχτού βρόγχου

Στη συνέχεια υπολογίζεται το K από τον τύπο: $K = \Delta y / \Delta x$

Και η κλίση της καμπύλης απόκρισης S από τον τύπο: $S = K / T_2$

Οι παράμετροι του ελεγκτή υπολογίζονται από τον Πίνακα 3.

	K_c	**T_i**	**T_d**
P	$1 / (S*T_1)$		
PI	$0.9 / (S*T_1)$	$3*T_1$	
PID	$1.2 / (S*T_1)$	$2*T_1$	$T_1 / 2$

Πίνακας 3: Εξισώσεις άγνωστου μοντέλου διεργασίας Ziegler-Nichols

- **Cohen-Coon**

  Η μέθοδος αυτή λειτουργεί σε γνωστό και άγνωστο μοντέλο διεργασίας. Έχει παρόμοια λογική με την μέθοδο των Ziegler-Nichols αλλά είναι καλύτερη σε περιπτώσεις που η γρήγορη απόκριση είναι αναγκαία. Χρησιμοποιείται μια προσομοιωμένη συνάρτηση μεταφοράς πρώτου βαθμού της μορφής:

$$G(s) = \frac{K}{1+s*\tau} * e^{-st_d}$$

όπου t_d είναι η σταθερά απόκρισης και τ είναι η σταθερά της διεργασίας. Αυτά υπολογίζονται, μαζί με την παράμετρο K, από την καμπύλη απόκρισης.

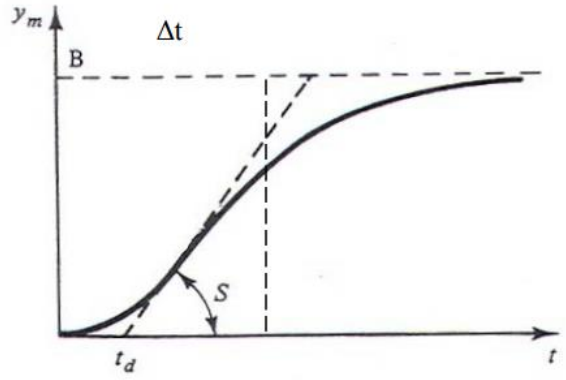
Ισχύει ότι:

$K = B / A$ όπου B είναι η τελική τιμή της εξόδου και A το σήμα εισόδου.

$S = \tan^{-1}(B / \Delta t)$

$\tau = B / S = \Delta C / T = \Delta B / \Delta t$

$A = t_d / \tau$ όπου t_d είναι η σταθερά καθυστέρησης.



Εικόνα 2.5.3 Βηματική απόκριση για μέθοδο Cohen-Coon.

Τέλος, οι παράμετροι του ελεγκτή υπολογίζονται βάσει του Πίνακα 4.

	K_c	T_i	T_d
P	$\frac{1}{K}\left(\frac{1}{A} + 0.33\right)$		
PI	$\frac{1}{K}\left(\frac{0.9}{A} + 0.082\right)$	$\tau\left(\frac{3.33\alpha + 0.33\alpha^2}{1 + 2.2\alpha}\right)$	
PID	$\frac{1}{K}\left(\frac{1.35}{A} + 0.27\right)$	$\tau\left(\frac{2.5\alpha + 0.5\alpha^2}{1 + 0.6\alpha}\right)$	$\tau\left(\frac{0.37\alpha}{1 + 0.2\alpha}\right)$

Πίνακας 4: Εξισώσεις Cohen-Coon

- **<u>Εναλλακτικές Μέθοδοι</u>**

  Υπάρχουν και άλλες μέθοδοι παρόμοιες με αυτές των Ziegler-Nichols και Cohen-Coon όπως η μέθοδος Tyreus-Lyuben όπου έχει την ίδια διαδικασία με των Ziegler-Nichols αλλά χρησιμοποιεί διαφορετικές εξισώσεις για πιο συντηρητικές τιμές (Πίνακας 5).

	K_c	T_i	T_d
PI	$0.31*K_{crit}$	$2.2*T_0$	
PID	$0.45*K_{crit}$	$2.2*T_0$	$T_0 / 6.3$

<u>Πίνακας 5: Εξισώσεις Tyreus-Lyuben</u>

Στη συγκεκριμένη εργασία, χρησιμοποιήθηκε μια καθαρά πειραματική μέθοδος. Οι τιμές των παραμέτρων των ελεγκτών τροποποιήθηκαν εμπειρικά βάσει του Πίνακα 1. Καθώς το σύστημα υλοποιήθηκε και σε εξομοίωση, υπήρχε η δυνατότητα υλοποίησης πολλών πειραμάτων, χωρίς την επιρροή ανεπιθύμητων συμπεριφορών στο υλικό του quadcopter.

Κεφάλαιο 3: Quadcopter

3.1. Χαρακτηριστικά Quadcopter (Hardware)

Το quadcopter που χρησιμοποιείται στη συγκεκριμένη εργασία είναι το Rolling Spider της εταιρίας Parrot, το οποίο ίπταται σε σχηματισμό '×'. Το κύριο σώμα περιλαμβάνει το σκελετό, τους κινητήρες και τα ηλεκτρονικά του στοιχεία (μικροελεγκτής, αισθητήρια).

Ανήκει στην κατηγορία micro UAV αφού η συνολική μάζα του είναι 68 γρ. και αποτελείται από:

- **Μικροελεγκτή Parrot Chipset SIP6**

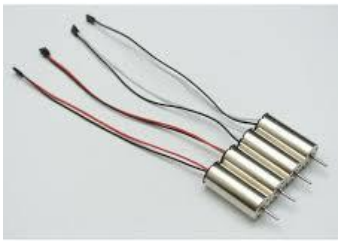
  Τεχνολογίας ARM A9 που συγχρονίζει στα 800 Mhz.

  Η μητρική πλακέτα έχει 1 GB RAM/256 MB DDR.

Εικόνα 3.1.1

- **Τέσσερις κινητήρες**

  33g Thrust ανά κινητήρα.



Εικόνα 3.1.2

- **Μπαταρία λιθίου (li-po)**

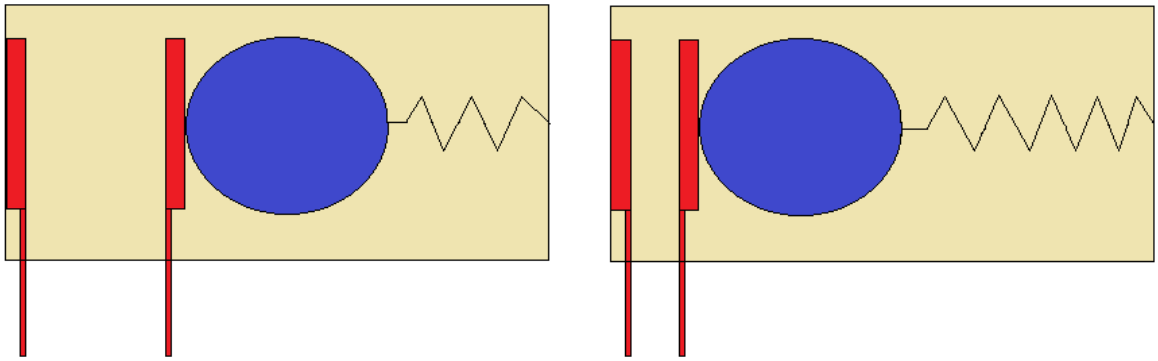
  Επαναφορτιζόμενη στα 3.7 Volt, 550 mAh.

  Στη περίπτωσή μας δίνει ικανότητα πτήσης 5 λεπτά.

Εικόνα 3.1.3

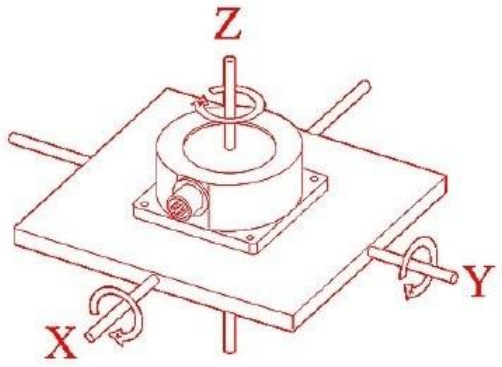
- **IMU (Inertial Measurement Unit): 6-αξόνων επιταχυνσιόμετρο-γυροσκόπιο**

  Το επιταχυνσιόμετρο (accelerometer) είναι μια ηλεκτρονική συσκευή η οποία μετρά επιτάχυνση. Έχει πολλαπλές χρήσεις στη βιομηχανία και στον επιστημονικό χώρο, και είναι σημαντικό στοιχείο σε συστήματα πλοήγησης τα οποία χρησιμοποιούνται σε αεροσκάφη. Στην περίπτωση των UAV και συγκεκριμένα στα quadcopters, η χρήση ενός επιταχυνσιόμετρου βοηθά στην σταθεροποίηση του την ώρα της πτήσης (hover). Με απότομη αλλαγή της μέτρησής του γίνεται αντιληπτή η σύγκρουσή του με κάποιο εμπόδιο. Ο τρόπος που λειτουργούν τα επιταχυνσιόμετρα είναι υπολογίζοντας, όχι την αλλαγή της ταχύτητας, αλλά την δύναμη η οποία ασκείται. Στο εσωτερικό τους, υπάρχει ένα σωματίδιο συνδεδεμένο με ενα ποτενσιόμετρο ή δύο πυκνωτές, όπως στο σχημα (Εικόνα 3.1.4). Όσο αυξάνεται η επιτάχυνση, τόση δύναμη ασκείται στο σωματίδιο και εν συνεχεία πιέζει τους πυκνωτές. Η διαφορά της απόστασης μεταξύ των πυκνωτών δείχνει και την δύναμη που ασκείται.



Εικόνα 3.1.4 Περιγραφή λειτουργίας επιταχυνσιομέτρου

Το γυροσκόπιο (gyroscope) είναι μια συσκευή που χρησιμοποιεί την βαρύτητα της Γης για να υπολογίσει τον προσανατολισμό. Η χρήση του γυροσκοπίου είναι απαραίτητη στην εύρεση του προσανατολισμού ενός UAV. Ο σχεδιασμός του περιέχει ενα ελεύθερα-περιστρεφόμενο δίσκο, προσαρμοσμένο πάνω σε περιστρεφόμενους άξονες στο κέντρο ενος μεγάλου, πιο σταθερού δίσκου που ονομάζεται σφόνδυλος. Όταν αλλάζει ο προσανατολισμός και επομένως και οι άξονες, ο σφόνδυλος παραμένει σταθερός δείχνοντας την βαρυτική έλξη (Εικόνα 3.1.5).

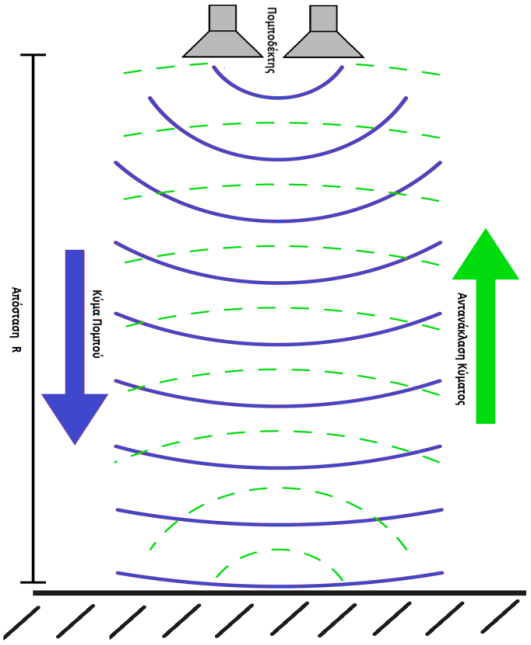


Εικόνα 3.1.5 Περιγραφή λειτουργίας γυροσκοπίου

Για το συγκεκριμένο UAV, χρησιμοποιούμε εναν συνδιασμό επιταχυνσιόμετρου και γυροσκόπιου έξι αξόνων (6-axis accel-gyroscope). Ο λόγος που είναι σημαντική η χρήση του είναι ότι αποφεύγονται ανεπιθύμητες κινήσεις στις τρείς διαστάσεις του χώρου, καθώς επιτυγχάνεται μεγαλύτερη ακρίβεια.

- **<u>Αισθητήριο Υπερήχων (Sonar)</u>**

  Το Sonar (ακρωνύμιο των λέξεων “SOund Navigation And Ranging), είναι ενα αισθητήριο που χρησιμοποιεί την διάδοση υπερηχητικών κυμάτων για ανίχνευση, εντοπισμό και αναγνώριση διαφόρων αντικειμένων. Το αισθητήριο αυτό χρησιμοποιείται για να μετρά την απόσταση του quadcopter απο το έδαφος ή την πιθανή ανίχνευση ενός άλλου αντικειμένου μεγαλύτερου ύψους (Εικόνα 3.1.6), όπου στην συνέχεια θα δώσει σήμα για αύξηση του ύψους του quadcopter για την αποφυγή του. Ένα μεγάλο μειονέκτημα αυτού του αισθητηρίου είναι η πιθανή απόκλιση από την πραγματική απόσταση λόγω του θορύβου, η οποία πολλές φορές δημιουργείται λόγω του αέρα που παραποιεί τα ηχητικά κυματα.



Εικόνα 3.1.6 Περιγραφή λειτουργίας Sonar

- **Αισθητήριο πίεσης (Pressure Sensor)**

  Μετρά την πίεση του αέρα γυρω από το quadcopter. Όσο πιο ψηλά ανεβαίνει, τόσο περισσότερο μειώνεται η πίεση του αέρα. Επομένως είναι σημαντικό να το γνωρίζουμε, γιατί μπορούμε να καταλάβουμε ακριβέστερα το υψόμετρο, βάσει της διαφοράς της μέτρησης από την αρχική του μέτρηση πρίν απογειωθεί. Μια άλλη χρήση του που όμως χρησιμοποιείται περισσότερο σε UAV σταθερών πτερυγίων (fixed-wing aircraft), είναι για την εύρεση της πυκνότητας και της ταχύτητας του αέρα ώστε να υπολογίζει την άνωση στα φτερά από την ροή του αέρα.

- **Κάμερα**

  Ένα από τα πιο σημαντικά αισθητήρια για τον καθορισμό ελέγχου του quadcopter είναι η κάμερα προσανατολισμένη προς τα κάτω. Διαθέτει ανάλυση 300.000 pixels και μπορεί να βγάζει φωτογραφίες με συχνότητα έως και 60 Hz, δηλαδή 60 φωτογραφίες/δευτερόλεπτο. Η χρήση της κατά την πτήση παίζει πολύ σημαντικό ρόλο, γιατί μέσω της επεξεργασίας της κάθε εικόνας μπορούμε να πάρουμε πολλές πληροφορίες. Μια απο αυτές είναι και ο αλγόριθμος υπολογισμού μετατόπισης, ή optical flow, όπου μέσω των διαφορών των pixels από διαδοχικές εικόνες μπορούμε να καταλάβουμε την μετατόπιση του quadcopter, την ταχύτητα ή και την επιτάχυνση του. Περαιτέρω ανάλυση του optical flow θα γίνει και στα επόμενα κεφάλαια. Ακόμα, μπορούμε με χρήση διαφόρων άλλων αλγόριθμων να

ανιχνεύσουμε συγκεκριμένα χρώματα, αντικείμενα, σχετικές μεταβολές, πληροφορίες γενικότερα που μπορούν να βοηθήσουν στην μοντελοποίηση και τον έλεγχο του quadcopter.

- **Bluetooth**

  Με την χρήση του Bluetooth επιτυγχάνουμε ασύρματη επικοινωνία μικρών αποστάσεων. Αυτό γίνεται μέσω μετάδοσης μικροκυμάτων σε φάσμα συχνοτήτων 2.4 GHz και επιτρέπει την απευθείας σύνδεση μεταξύ συσκευών. Η επικοινωνία μέσω Bluetooth είναι αναγκαία για την απευθείας εντολή ελέγχου με χρήση κινητού τηλεφώνου, ή μέσω του ηλεκτρονικού υπολογιστή χρησιμοποιώντας λογισμικό Linux που θα εξηγηθεί και αργότερα.

3.2. Λειτουργικό Σύστημα (Software)

- **Linux**

  Το λειτουργικό σύστημα (software) που χρησιμοποιείται στη συγκεκριμένη εργασία είναι το Ubuntu 14.04 το οποίο ειναι μια έκδοση Linux. Χρησιμοποιείται αυτό το λογισμικό επειδή το Parrot Rolling Spider υποστηρίζει Linux, οπότε είναι απαραίτητο για την επικοινωνία μεταξύ υπολογιστή και quadcopter. Η διαδικασία υλοποιείται μέσω του terminal αφού γίνει η εγκατάσταση του κατάλληλου firmware στο quadcopter. Για να επιτευχθεί η επικοινωνία χρησιμοποιούνται οι ακόλουθες βιβλιοθήκες:

  - **Lftp** (για μεταφορά αρχείων σε UNIX συστήματα)
  - **Bluez-compat** (για επίτευξη επικοινωνίας Bluetooth)
  - **Expect** (για επικοινωνία διαδραστικών προγραμμάτων)

  Επίσης εντός του λογισμικού γίνεται η εγκατάσταση του προγράμματος **Matlab&Simulink 2015a**.

- **Matlab & Simulink**

  Το Matlab είναι ένα πρόγραμμα αριθμητικής υπολογιστικής και χρησιμοποιείται στην επίλυση μαθηματικών προβλημάτων, εμφάνιση γραφικών παραστάσεων, επεξεργασία πολυμέσων, δημιουργία προσομοιώσεων και συστημάτων ελέγχου μέσω μιας γλώσσας προγραμματισμού τέταρτης γενιάς. Στη παρούσα εργασία γίνεται χρήση του Simulink, ως μια πρόσθετη εργαλειοθήκη, η οποία επιτρέπει τη προσομοίωση και την απεικόνιση δυναμικών συστημάτων μέσω δομικών διαγραμμάτων (blocks).

Κεφάλαιο 4: Μοντελοποίηση και Εξομοίωση στο Matlab

4.1. Μοντελοποίηση Συστήματος Εξομοίωσης

Το σύστημα ελέγχου του quadcopter υλοποιήθηκε μέσω του προγράμματος Matlab και συγκεκριμένα με τη βοήθεια του Simulink. Σκοπός της ενέργειας αυτής είναι να γίνει η εξομοίωση όλων των μαθηματικών μοντέλων μέσω της μορφής των δομικών διαγραμμάτων (blocks). Με τον τρόπο αυτό γίνεται εφικτή η διάσπαση του συνολικού συστήματος, το οποίο είναι αρκετά πολύπλοκο, σε μικρότερα απλούστερα υποσυστήματα άρα γίνεται ευκολότερη η κατανόησή του. Η εξομοίωση έχει πολλά πλεονεκτήματα σε σχέση με την απευθείας δοκιμή των μαθηματικών μοντέλων που χρησιμοποιούνται στο quadcopter. Αυτά είναι η αποφυγή ανεπιθύμητων συμπεριφορών σε περίπτωση λάθος υπολογισμού, εξοικονόμηση ενέργειας και χρόνου, καθώς και η απευθείας λήψη γραφικών παραστάσεων που αφορούν την συμπεριφορά του σε κάθε μεταβολή των παραμέτρων του συστήματος και η ευκολότερη επίλυση των προβλημάτων που μπορεί να προκύψουν.

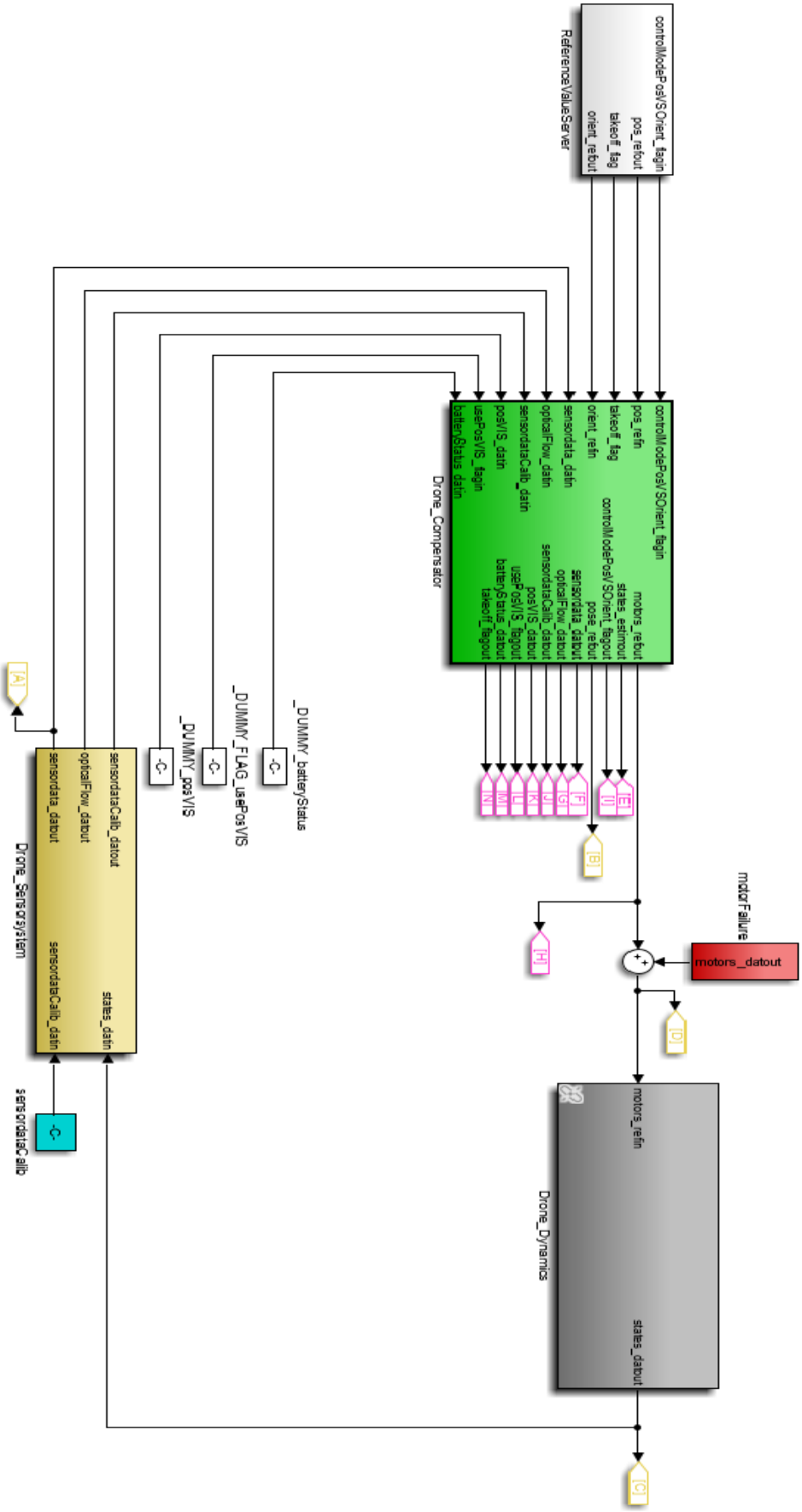
Μέτα την εξομοίωση, το Simulink έχει την δυνατότητα να μετατρέψει το σύστημα ή υποσυστήματα αυτού σε κώδικα C, ο οποίος είναι αναγνωρίσιμος και εκτελέσιμος από το firmware του quadcopter. Έτσι, με αποστολή αυτού του κώδικα με τη χρήση Bluetooth, τα αποτελέσματα αυτής της ενέργειας είναι εφικτό να παρατηρηθούν και στη πράξη, πέρα από την εξομοίωση.

Με τον τρόπο αυτό, γίνεται η επικοινωνία του υπολογιστή με το quadcopter είτε για απομακρυσμένο έλεγχο του από κάποιο χειριστή, είτε για την μετάδοση ενός αλγορίθμου που καθιστά εφικτή την αυτοματοποιημένη πλοήγηση του χωρίς την ανθρώπινη παρέμβαση.

Το συνολικό σύστημα εξομοίωσης αποτελείται από ένα κλειστό βρόγχο, σκοπός του οποίου είναι να κάνει εφικτή την ορθή αιώρηση του quadcopter.

Το μοντέλο που χρησιμοποιήθηκε σαν βάση της εργασίας αυτής, αρχικά αναπτύχθηκε από τους Sertac Karaman και Fabian Reither από το MIT, διανέμεται με άδεια ανοιχτής πρόσβασης* και στη συνέχεια τροποποιήθηκε για τις ανάγκες των στόχων της εργασίας.

* https://github.com/Parrot-Developers/RollingSpiderEdu

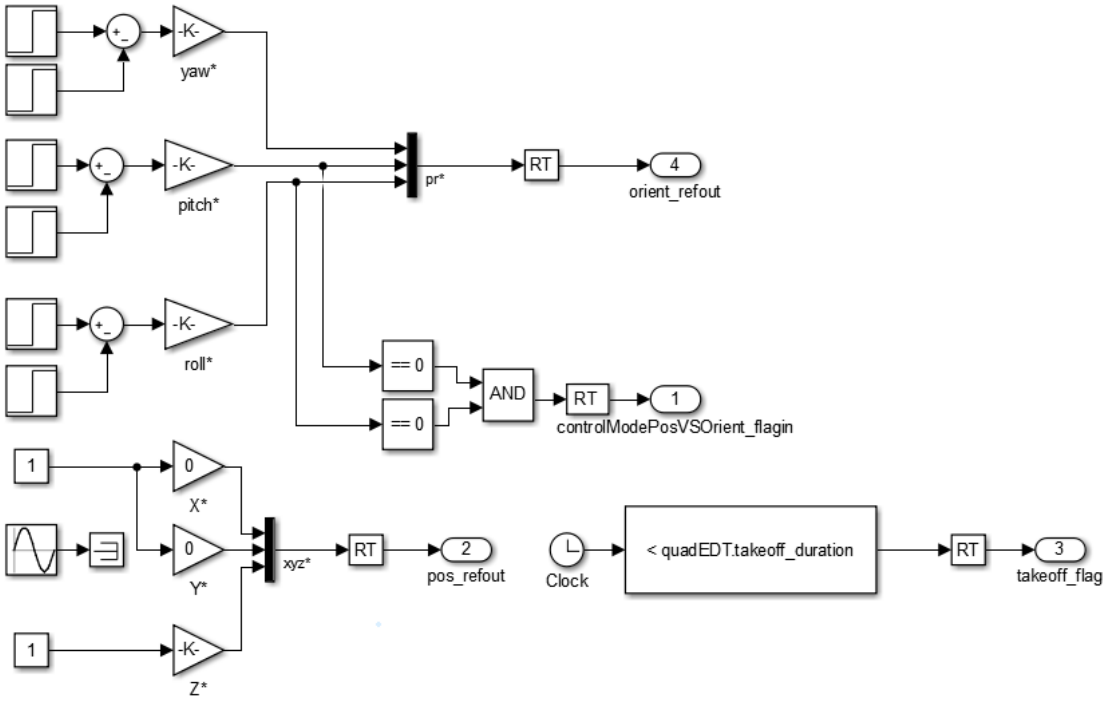


Εικόνα 4.1.1 Δομικά Διαγράμματα Εξομοίωσης Quadcopter

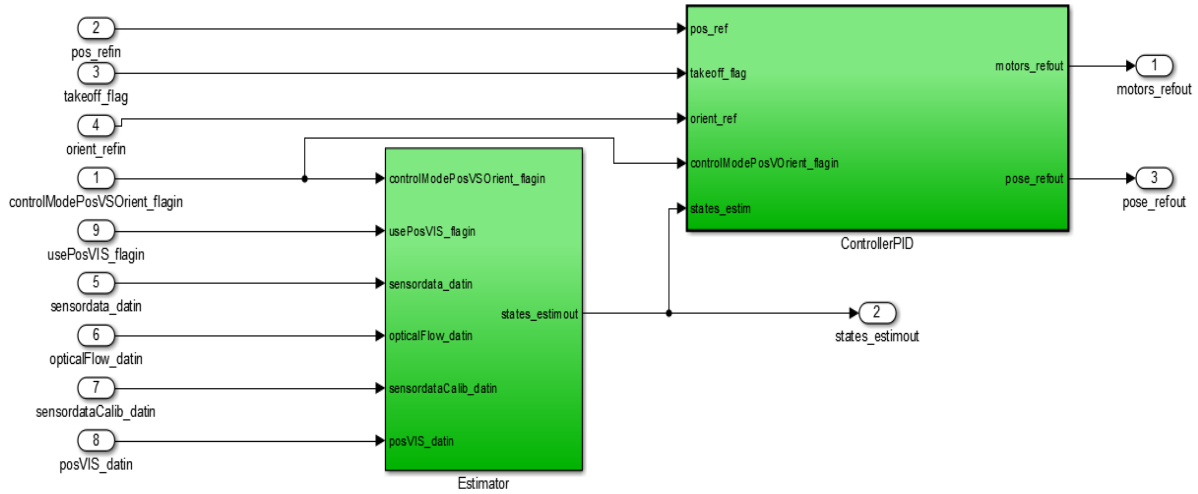
4.2. Περιγραφή Υποσυστημάτων

- Είσοδος Συστήματος (ReferenceValueServer)

  Το πρώτο δομικό διάγραμμα που ονομάζεται ReferenceValueServer, αποτελεί την είσοδο του συστήματος. Σκοπός του είναι να δώσει τιμές στις μεταβλητές κατάστασης, δηλαδή στις γωνίες Euler και στην μετατόπιση του στο χώρο, ώστε να γίνει εξομοίωση των αποκρίσεών του.



Εικόνα 4.2.1 ReferenceValueServer



Εικόνα 4.2.2 Drone_Compensator

- Συγκριτής και Ελεγκτής Συστήματος (Drone_Compensator)

  Το επόμενο κατά σειρά δομικό διάγραμμα που ονομάζεται Drone_Compensator, αποτελείται από τον συγκριτή ή εκτιμητή (Estimator) και τον ελεγκτή PID (ControllerPID) όπως φαίνεται και στην Εικόνα 4.2.2.

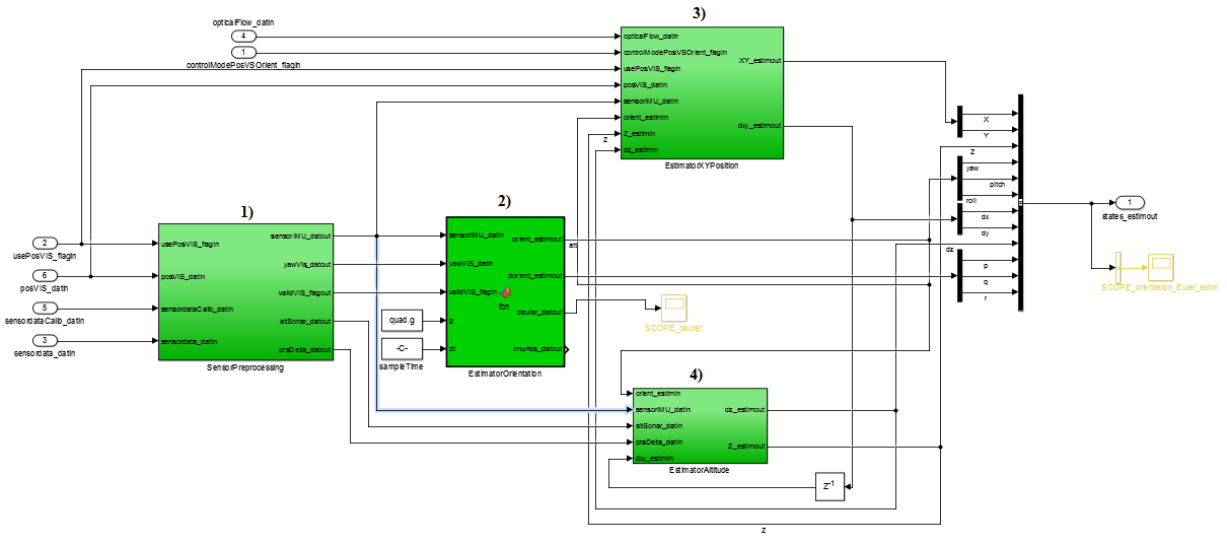
  - Ο Estimator δέχεται ως είσοδο τις τιμές των αισθητηρίων από την ανάδραση και για έξοδο δίνει ένα πίνακα δώδεκα στοιχείων (states_estimout) που περιέχει τις μεταβλητές κατάστασης μετά από επεξεργασία. Συγκεκριμένα, ο Estimator (Εικόνα 4.2.3) αποτελείται από:

    i. SensorPreprocessing: Είναι υπεύθυνο για την επεξεργασία συγκεκριμένων σημάτων των αισθητηρίων για να είναι ευκολότερη η ανάλυση και η επεξεργασία τους στα επόμενα δομικά διαγράμματα. (π.χ. προσθήκη κέρδους στο IMU (6-axis accel-gyro) και προσθήκη φίλτρου IIR στο σήμα του γυροσκόπιου και φίλτρου FIR στο σήμα του επιταχυνσιόμετρου).

    ii. EstimatorOrientation: Αυτό το δομικό διάγραμμα αποτελείται από κώδικα στο Matlab που είναι υπεύθυνος για την διόρθωση σημάτων που δεν περιέχουν κάποια πληροφορία και μετασχηματισμός των γωνιών Euler από το πλαίσιο αναφοράς στο σταθερό πλαίσιο.

    iii. EstimatorXYPosition: Το block αυτό δέχεται ως είσοδο δεδομένα από τον optical flow (αλγόριθμος που χρησιμοποιεί την κάμερα για να υπολογίσει ταχύτητα), σήματα από τα αισθητήρια όπως ύψος Z και σήματα από το IMU και υπολογίζει την ταχύτητα και την μετατόπιση. Τα σήματα αυτά, πριν την έξοδο, περνούν μέσα από ενα φίλτρο Kalman το οποίο είναι υπεύθυνο για την εξάλειψη του θορύβου (Παράρτημα 1 – Φίλτρο Kalman). Επίσης, είναι υπεύθυνο για τον προκαθορισμό του αρχικού σημείου αναφοράς, ανάλογα από το να ανιχνεύσει landmarks ή όχι.

    iv. EstimatorAltitude: Κάνει εκτίμηση του ύψους Z βάσει των αισθητηρίων IMU, πίεσης και sonar περνώντας τα σήματα των εξόδων τους μέσα από ένα φίλτρο Kalman.



Εικόνα 4.2.3 Estimator

- Ο PID του συστήματος υλοποιείται μέσα στο δομικό διάγραμμα ControllerPID. Σκοπός του είναι να διατηρεί την ευστάθεια του συστήματος, δηλαδή να μπορεί το quadcopter να αιωρείται και να πραγματοποιεί τις οποιεσδήποτε κινήσεις του αναθέτονται χωρίς ανεπιθύμητες συμπεριφορές. Έχει εισόδους την έξοδο του Estimator που είναι οι δώδεκα μεταβλητές κατάστασης και τις εισόδους του συστήματος που δίνει το δομικό διάγραμμα ReferenceValueServer. Σαν έξοδο, έχει τις ταχύτητες των κινητήρων, δηλαδή την ισχύ με την οποία θα τροφοδοτηθούν οι κινητήρες.

  Αναλυτικά, χρησιμοποιούνται κυρίως ελεγκτές PD, δηλαδή δεν χρησιμοποιείται ο όρος I (Integral) που είναι η ολοκλήρωση του σφάλματος. Ο λόγος που δεν χρειάζεται αυτός ο όρος είναι γιατί με την προσθήκη του αυξάνεται ο χρόνος αποκατάστασης, καθώς και το ύψος της ταλάντωσης της απόκρισης. Σε ένα τέτοιο σύστημα αυτό δημιουργεί πρόβλημα για το λόγο ότι η απόκριση πρέπει να καταλήγει κοντά στην επιθυμητή τιμή γρήγορα, έστω και με την ύπαρξη ενός μικρού μόνιμου σφάλματος. Οι τέσσερις αυτοί ελεγκτές αφορούν:

  i. <u>Το ύψος (Altitude)</u>

     Ο ελεγκτής έχει ως είσοδο την επιθυμητή τιμή του ύψους που δέχεται από το ReferenceValueServer και το συγκρίνει με την πραγματική τιμή του ύψους που δέχεται από τα αισθητήρια. Η διαφορά τους είναι το σφάλμα, το οποίο συγκρίνεται μαζί με τον ρυθμό μεταβολής του ύψους dz από τα αισθητήρια και την προσθήκη κερδών K_c=0.8 και K_d=0.3. Στη συνέχεια, αν το quadcopter έχει απογειωθεί, τότε το ύψος που θα φτάσει είναι αυτό που δίνεται από το

ReferenceValueServer, δηλαδή η επιθυμητή τιμή. Σε αυτό, βέβαια, γίνεται και η προσθήκη της δύναμης της βαρύτητας για ισορροπήσει, καθώς και ο κόρος που λειτουργεί για την αποτροπή ανεπιθύμητων τιμών.

ii. Τη περιστροφή γύρω από τον άξονα Z (Yaw)

Με την ίδια λογική, αυτός ο ελεγκτής δέχεται ως είσοδο την επιθυμητή τιμή του yaw από το ReferenceValueServer και την πραγματική τιμή του yaw από τα αισθητήρια. Από την σύγκριση των δυο προκύπτει το σφάλμα πολλαπλασιασμένο με το κέρδος K_c=0.04. Ο ρυθμός μεταβολής του μαζί με την προσθήκη κέρδους K_d=0.0012 αφαιρείται.

iii. Την μετατόπισή του στο επίπεδο XY βάσει της περιστροφής του στον άξονα Z

Ομοίως, ο ελεγκτής αυτός δέχεται ως είσοδο την επιθυμητή τιμή του XY από το ReferenceValueServer και την πραγματική τιμή του XY από τα αισθητήρια. Από την σύγκριση των δυο προκύπτει το σφάλμα το οποίο είναι πολλαπλασιασμένο με τον πίνακα που προκύπτει από την πραγματική τιμή του yaw.

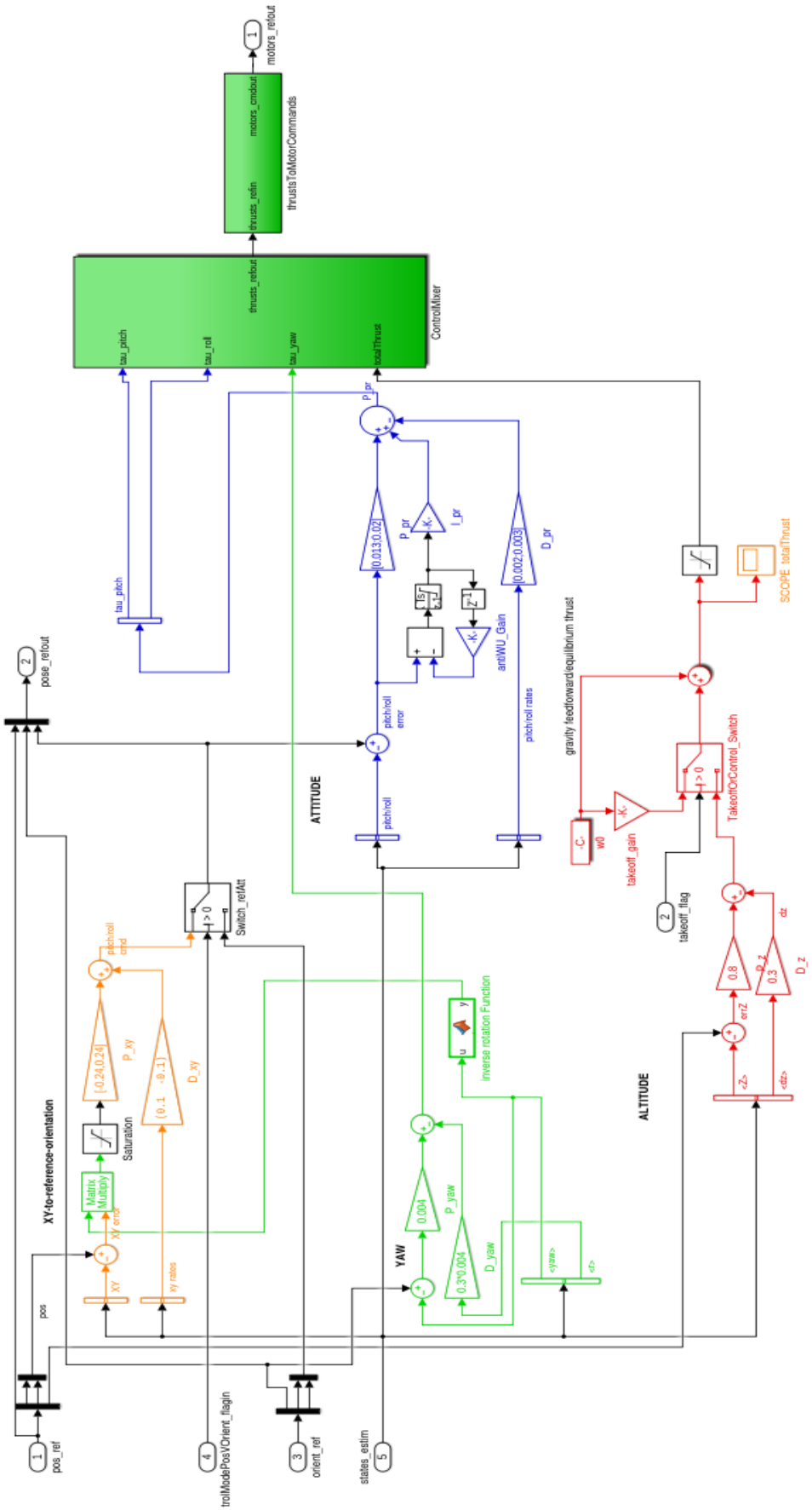
$$\text{yaw} = \begin{bmatrix} cos\Psi & sin\Psi \\ -sin\Psi & cos\Psi \end{bmatrix}$$

Έπειτα, περνάει από εναν κόρο ο οποίος δίνει μέγιστη και ελάχιστη τιμή στη μεταβολή του XY και πολλαπλασιάζεται με το κέρδος K_{cx}=-0.24 και K_{cy}=0.24. Ο ρυθμός μεταβολής του προστίθεται μαζί με τα κέρδη K_{dx}=0.1 και K_{dy}=-0.1.

iv. Τον προσανατολισμό του βάσει της μετατόπισης στο επίπεδο XY (Attitude)

Ο ελεγκτής αυτός δεν είναι PD, όπως και οι προηγούμενοι, αλλά PID. Διαθέτει δηλαδή και τον παράγοντα της ολοκλήρωσης του σφάλματος I. Αυτό γίνεται γιατί σε αυτή την περίπτωση είναι ανεπιθύμητο ένα μόνιμο σφάλμα, έστω και αρκετά μικρό και είναι προτιμότερη η καθυστέρηση της επίτευξης της τελικής τιμής. Στην παράμετρο του I χρησιμοποιείται το Z^{-1} για να μπορέσει να δεχθεί σαν είσοδο την τελευταία έξοδο.

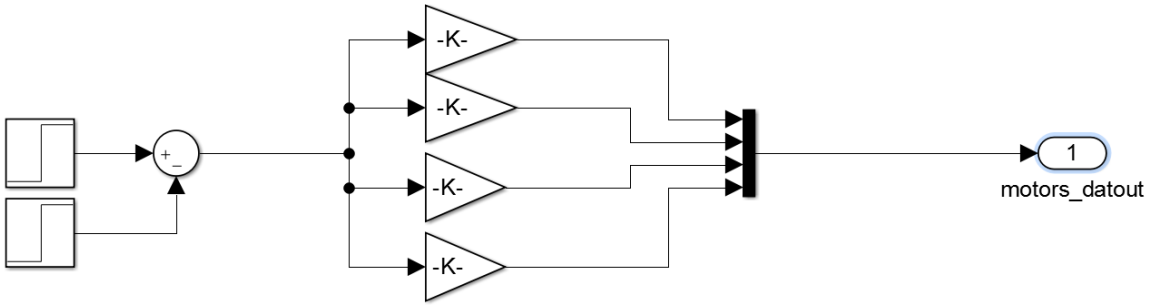
Τέλος, οι έξοδοι των ελεγκτών tau_pitch, tau_roll, tau_yaw και totalThrust καταλήγουν στο ControlMixer. Αυτό το block, βγάζει σήμα εξόδου thrust_refout που περιέχει την ώση που πρέπει να παράξει ο κάθε κινητήρας και μέσα στο thrustsToMotorCommands γίνεται η προσθήκη κερδών, κόρου και φορά στρέψης των κινητήρων.



Εικόνα 4.2.4 Controller PID

- Διαταραχή Συστήματος (motorFailure)

  Εδώ γίνεται η εξομοίωση μιας διαταραχής στο σύστημα σε χρόνο και μέγεθος που επιλέγει ο χρήστης μέσω βηματικών εισόδων για να παρατηρηθεί η ανταπόκριση του quadcopter στις ανάλογες συνθήκες. Η διαταραχή αυτή αφορά τον κάθε κινητήρα ξεχωριστά.



Εικόνα 4.2.5 MotorFailure

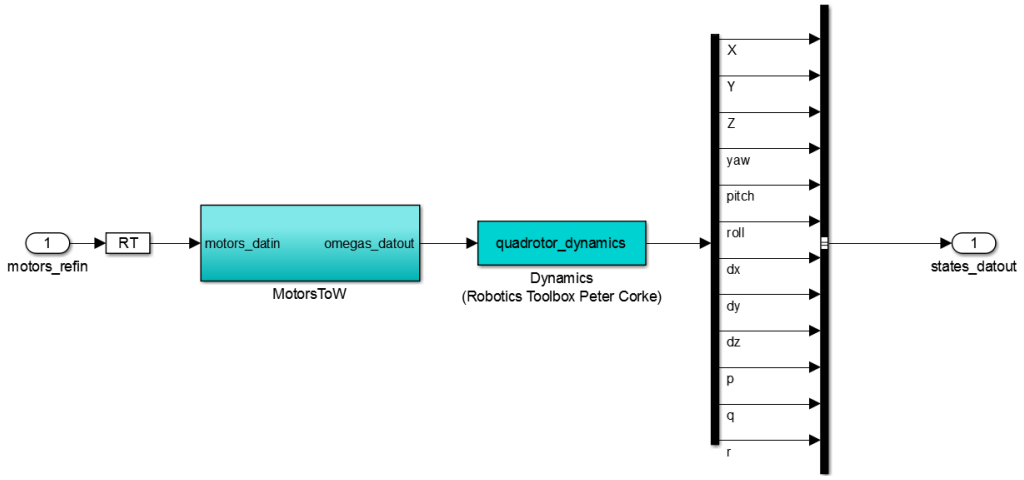
- Δυναμική Συστήματος (Drone_Dynamics)

  Είναι η βασική διεργασία του συστήματος και μέσα σε αυτή εμπεριέχεται η μαθηματική μοντελοποίηση του quadcopter, όπως αναλύθηκε και στο Κεφάλαιο 3. Το block quadrotor_dynamics, το οποίο υλοποιήθηκε βάσει των προδιαγραφών που δίνει ο Peter Corke μέσω του Robotics Toolbox για το Matlab, εμπεριέχει κώδικα με τις μαθηματικές εξισώσεις που διέπουν το σύστημα του Rolling Spider.

  Γενικότερα, στο κώδικα αρχικοποιούνται οι δώδεκα μεταβλητές κατάστασης και καθορίζονται οι αποστάσεις μεταξύ κινητήρων και κέντρου μάζας. Έπειτα, προσδιορίζονται τα πλαίσια μέσα στα οποία μπορεί να πραγματοποιηθεί πτήση, δηλαδή αποτρέπονται καταστάσεις στις οποίες το quadcopter μπορεί να αποπροσανατολιστεί. Επίσης, γίνεται μαθηματική μοντελοποίηση των κινητήρων και του συνολικού συστήματος και υπολογίζεται η δύναμη της ώθησης που ασκούν οι κινητήρες στο κύριο σώμα. (Παράρτημα 2 - Κώδικας quadrotor_dynamics.m)

  Η έξοδος αυτού του δομικού διαγράμματος είναι και η τελική έξοδος του συνολικού συστήματος, δηλαδή περιέχει τις τελικές, πραγματικές τιμές των μεταβλητών κατάστασης.

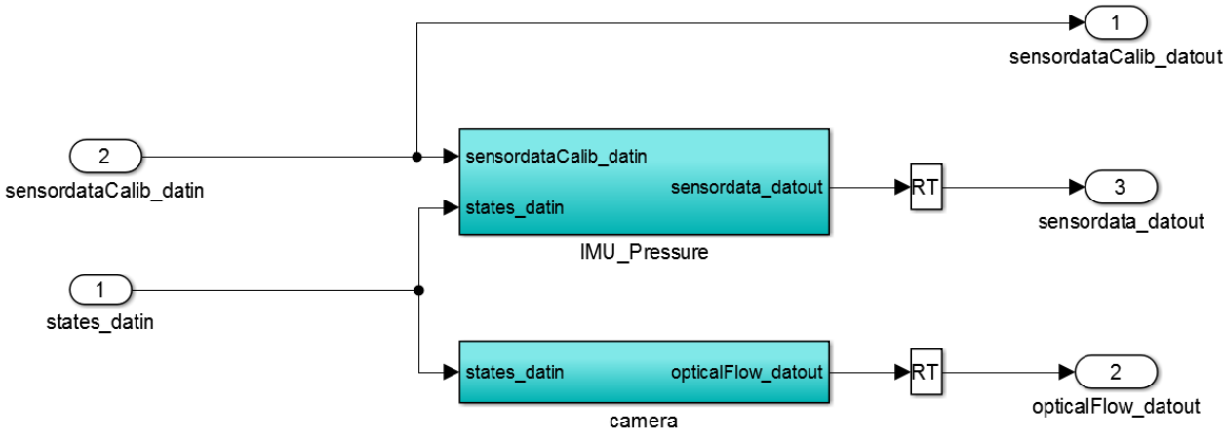
  Από αυτό το σημείο, πρέπει να παρθούν οι απαραίτητες μετρήσεις από τα αισθητήρια ωστέ να εξακριβωθεί εάν η πραγματική έξοδος συμπίπτει με την επιθυμητή. Αυτό υλοποιείται μέσω της ανάδρασης.



Εικόνα 4.2.6 Drone_Dynamics

- Σύστημα Αισθητηρίων (Drone_Sensorsystem)

  Στην ανάδραση, υλοποιείται το σύστημα αισθητηρίων όπου γίνεται η λήψη μετρήσεων από τα αισθητήρια. Η είσοδος αυτού του υποσυστήματος είναι οι μεταβλητές κατάστασης όπως ακριβώς τις δέχεται από την έξοδο του συνολικού συστήματος και ένας πίνακας sensordataCalib όπου γίνονται οι απαραίτητες ρυθμίσεις για την ορθή λειτουργία των αισθητηρίων (calibration). Αποτελείται από δύο δομικά διαγράμματα, ένα για την λήψη μέτρησης των αισθητηρίων, δηλαδή του γυροσκόπιου, επιταχυνσιόμετρου, πίεσης και του sonar (IMU_Pressure), και ένα για την κάμερα (camera).



Εικόνα 4.2.7 Drone_Sensorsystem

  - IMU_Pressure

    Στο block αυτό, γίνεται εξομοίωση του επιταχυνσιόμετρου και του γυροσκόπιου.

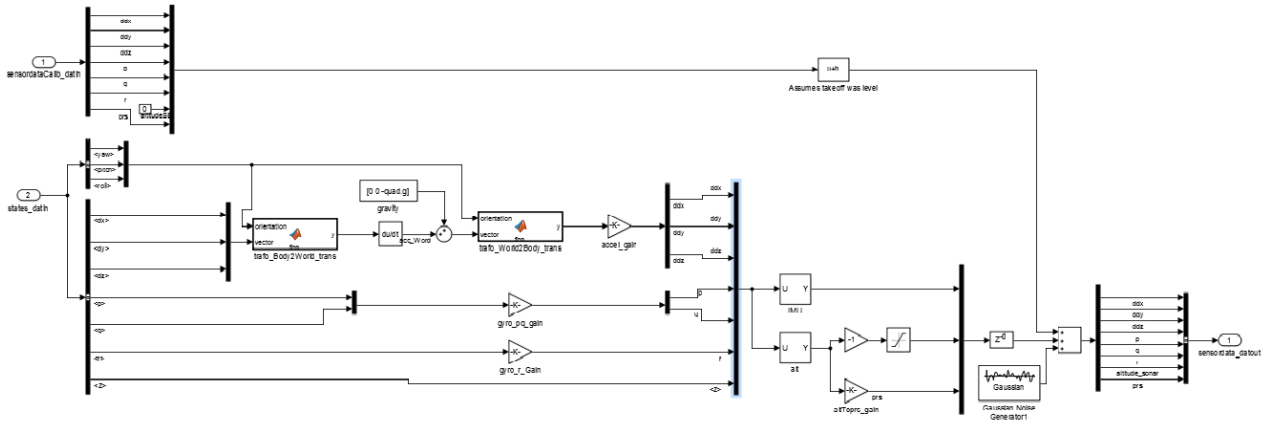
    Για το επιταχυνσιόμετρο, χρησιμοποιεί τις γωνίες Euler καθώς και τις ταχύτητες στους άξονες χ, y, z και έχει ως αποτέλεσμα τις επιταχύνσεις στους άξονες x, y, z, όπως

δηλαδή και την πραγματική λειτουργία του. Αυτό γίνεται μέσω της μεταφοράς αυτών των μεταβλητών κατάστασης από το πλαίσιο αναφοράς στο σταθερό πλαίσιο. Μετά, γίνεται παραγώγιση (δηλαδή υπολογισμός της επιτάχυνσης μέσω της ταχύτητας) και προστίθεται η δύναμη της βαρύτητας. Τέλος, γίνεται ξανά μεταφορά από το σταθερό πλαίσιο στο πλαίσιο αναφοράς (Εξισώσεις 7, 9 – Κεφάλαιο 3).

Για το γυροσκόπιο, απλά γίνεται η προσθήκη κέρδους στους ρυθμούς μεταβολής των γωνιών Euler.

Για το ύψος Z, εξομοιώνει τις μετρήσεις του αισθητηρίου πίεσης και του sonar με τις κατάλληλες προσθήκες κερδών και κόρου.

Τελικά, προστίθονται τα δεδομένα αυτά μαζί με τις ρυθμίσεις των παραμέτρων και μια γεννήτρια θορύβου Gauss.



Εικόνα 4.2.8 IMU_Pressure

- Camera

  Εδώ γίνεται η υλοποίηση του αλγορίθμου optical flow για τον υπολογισμο των συντεταγμένων μέσω εικόνας. Στην εξομοίωση, η λειτουργία του optical flow απέχει πολύ από τη πραγματικότητα (Παράρτημα 1 – Optical Flow)

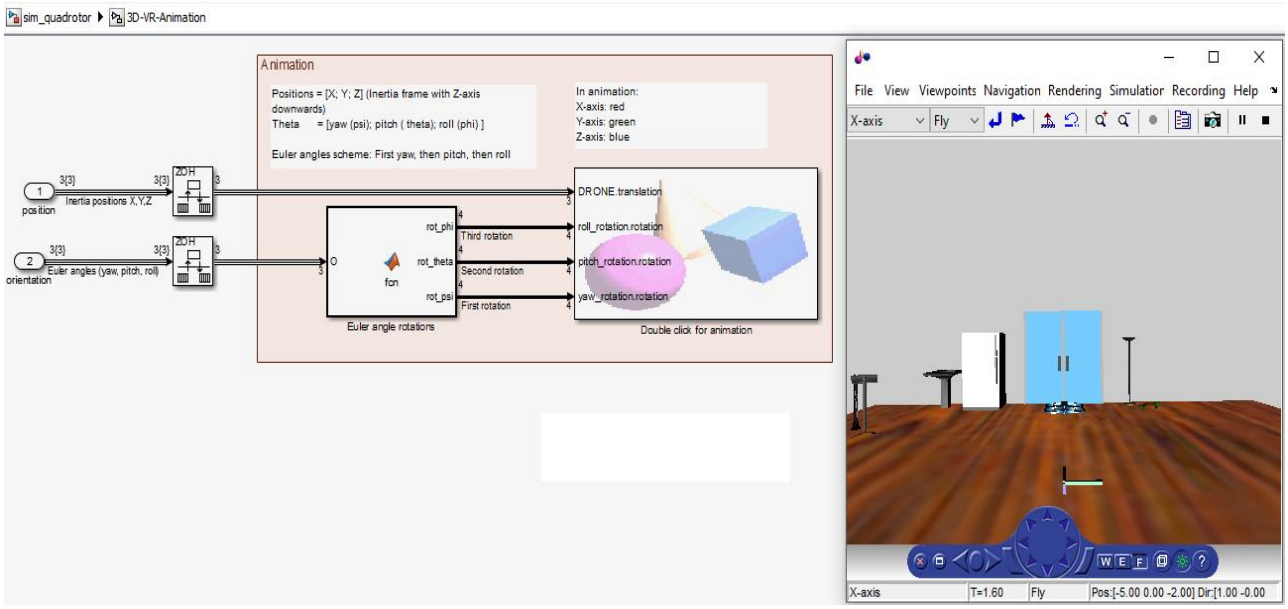
Τέλος, υπάρχουν οι σταθερές _DUMMY_batteryStatus, όπου εξομοιώνει την διάρκεια της μπαταρίας, και οι _DUMMY_FLAG_usePosVIS και _DUMMY_posVIS όπου ελέγχουν τη θέση του quadcopter σε σχέση με τα landmarks.

4.3. Εξομοίωση

Εδώ θα αναλυθεί η διαδικασία της εξομοίωσης που γίνεται μέσω του Simulink. Ο σκοπός αυτής της διεργασίας είναι:

- Η καλύτερη κατανόηση του όλου συστήματος.
- Η εύρεση των κατάλληλων κερδών στο block του Controller_PID, ώστε να επιτευχθεί ο ορθός έλεγχος του quadcopter.
- Η καταγραφή των αποκρίσεων της εξόδου σε διάφορες βηματικές εισόδους για την περαιτέρω ανάλυση της συμπεριφοράς του συστήματος.
- Ο έλεγχος της αντίδρασης του συστήματος σε ακραίες διαταραχές που δέχεται από το περιβάλλον.

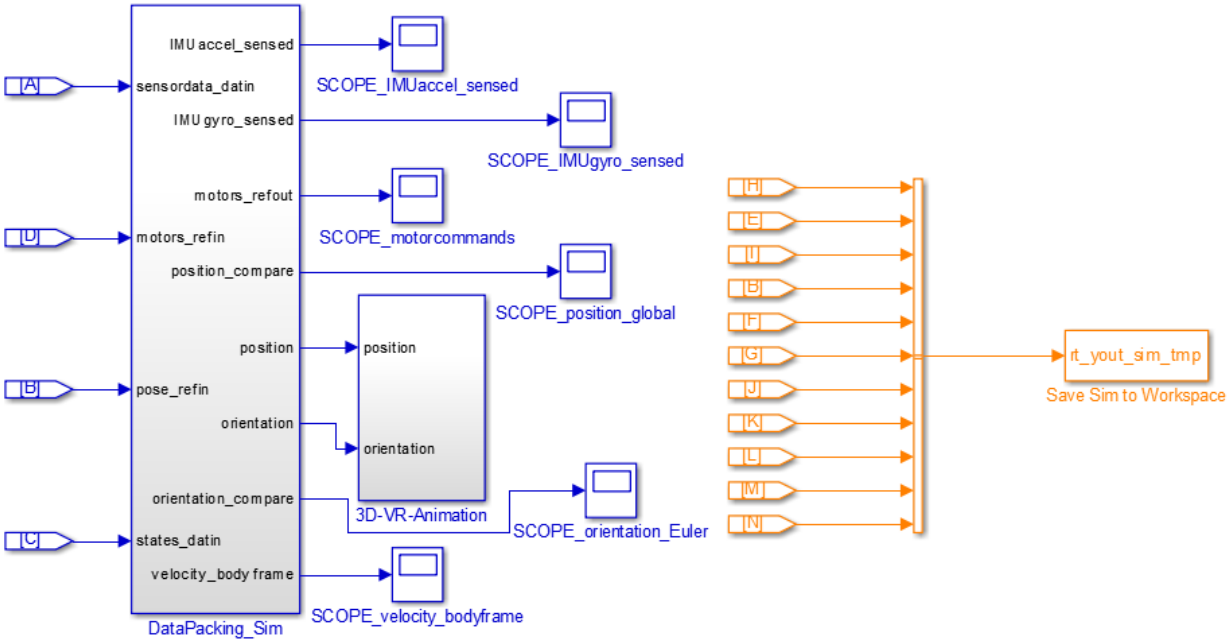
Εδώ είναι σημαντικό να επισημανθεί ότι πέρα από το συνολικό σύστημα που παρουσιάστηκε στο προηγούμενο κεφάλαιο, υπάρχουν ακόμα δυο blocks που σκοπός τους είναι η παρουσίαση των γραφικών παραστάσεων των αποκρίσεων, μεταφορά της συμπεριφοράς του συστήματος σε μια τρισδιάστατη απεικόνιση μέσω του 3D Animation Player και 3D World Editor Toolbox (Εικόνα 4.3.1), και αποθήκευση των τιμών της εξομοίωσης από το Simulink στο workspace.



Εικόνα 4.3.1 Δομικό διάγραμμα και 3D απεικόνιση της εξομοίωσης μέσω του 3D Animation Player και 3D World Editor Toolbox

Γραφικά παρουσιάζονται:

- Οι αποκρίσεις των αισθητηρίων επιταχυνσιόμετρου και γυροσκόπιου όπως αυτά αναλύονται μετά το block Drone_Sensorsystem.
- Η απόκριση των εντολών που δέχονται οι κινητήρες μετά το Drone_Compensator και την διαταραχή motorFailure.
- Η θέση στους άξονες XYZ και οι γωνίες Euler στην τελική έξοδο του συστήματος.
- Η ταχύτητα του quadcopter στην τελική έξοδο του συστήματος.



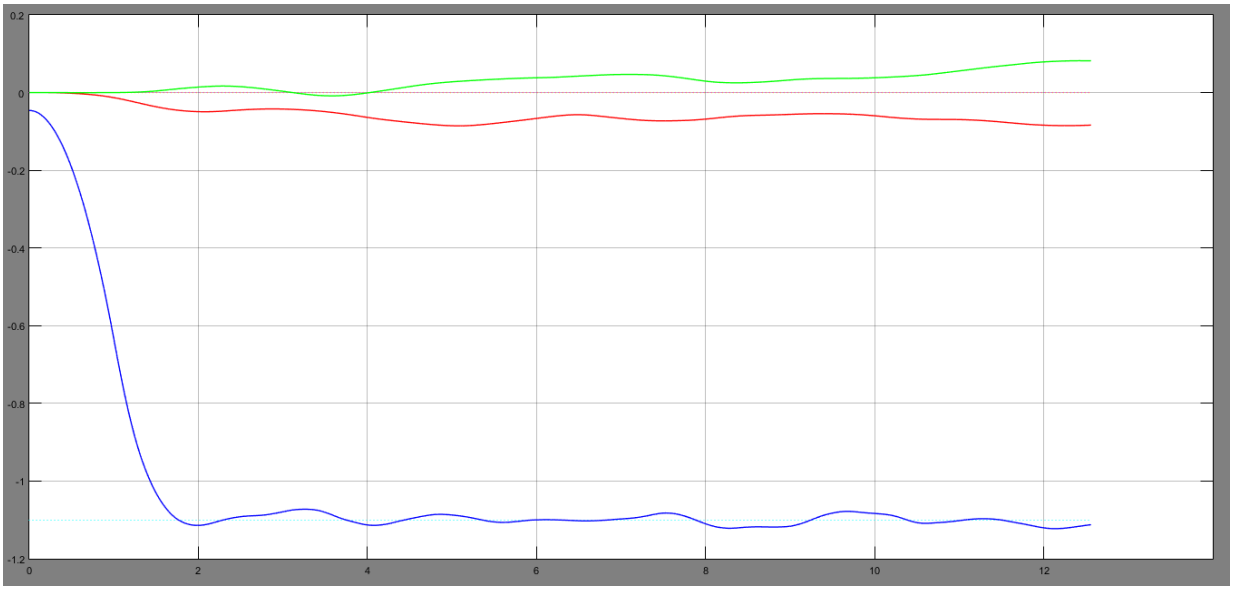
Εικόνα 4.3.2 Block δημιουργίας γραφικών παραστάσεων και αποθήκευσης μεταβλητών στο workspace

4.4. Παράδειγμα Εξομοίωσης Hover

Μετά τη ρύθμιση των κατάλληλων παραμέτρων του PID, υλοποιείται η πρώτη εξομοίωση δίνοντας μηδενικές βηματικές εισόδους στο block ReferenceValueServer. Δηλαδή, γίνεται η προσπάθεια επίτευξης του hover, που σημαίνει σταθεροποίηση του quadcopter σε σταθερό ύψος πάνω από ένα συγκεκριμένο σημείο μετά την απογείωσή του. Άρα μηδενίζουμε τα κέρδη των yaw, pitch, roll και X, Y αλλά όχι του Z που ισούται με -1.1, αλλιώς δεν θα υπήρχε ποτέ απογείωση.

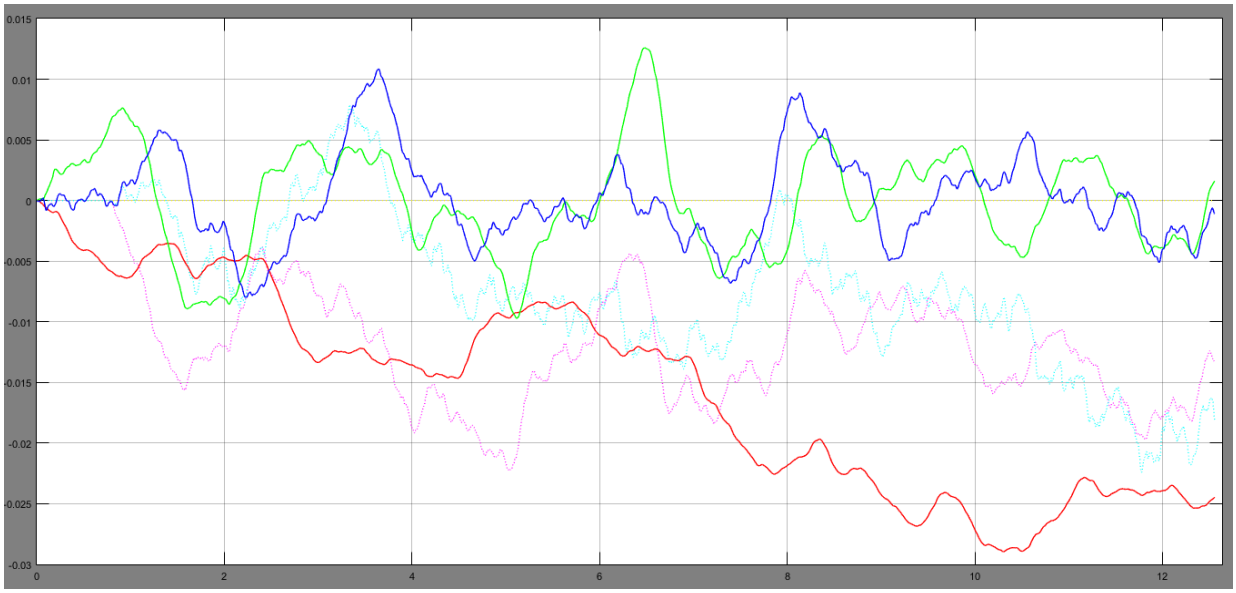
Μετά από την εκτέλεση της εξομοίωση διάρκειας 12 δευτερολέπτων, παρατηρούνται οι εξής αποκρίσεις.

Όπως φαίνεται στη παρακάτω γραφική, οι αποκρίσεις των X, Y κινούται γύρω από το μηδέν, ενώ η απόκριση του άξονα Z σταθεροποιείται γύρω απο το -1.1 σε διάστημα περίπου 1.5 δευτερολέπτου, όπου είναι και ο χρόνος που χρειάζεται το quadcopter για να απογειωθεί.



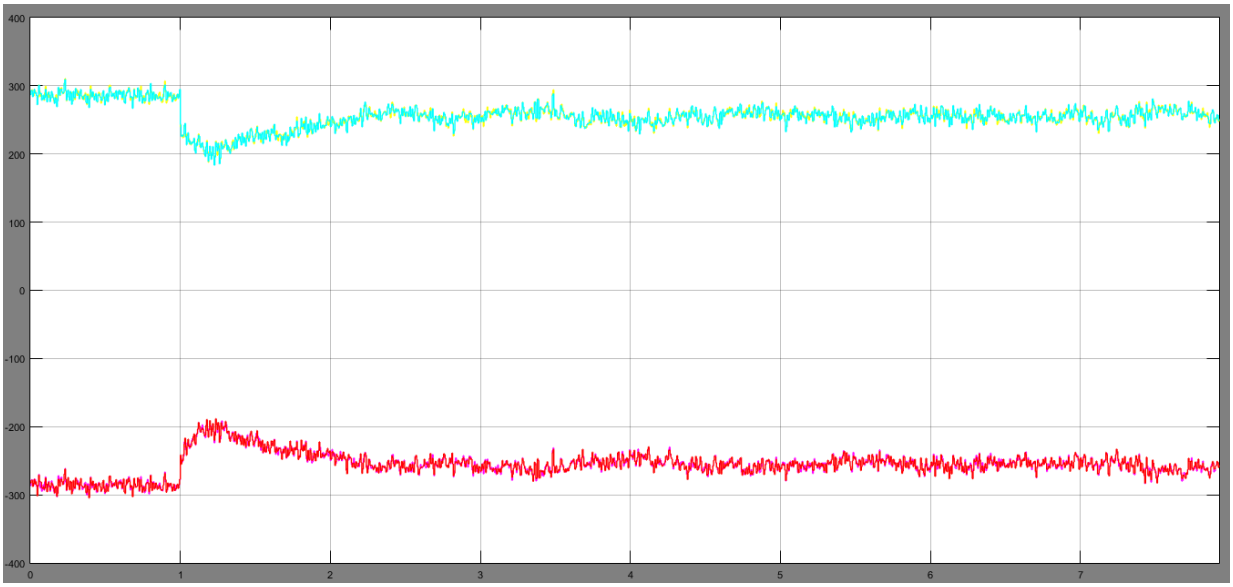
Εικόνα 4.4.1 Γραφική απεικόνιση θέσης στους άξονες XYZ

Αντίστοιχα, παρατηρείται ότι και οι γωνίες Euler κινούνται γύρω από το μηδέν, διότι δεν υπάρχει είσοδος έτσι ώστε να αλλάξει κάτι. Αν και παρατηρείται μεγάλη διακύμανση των τιμών, να σημειωθεί ότι η απόκριση κινείται υπό πολύ μικρή κλίμακα στον άξονα Y.



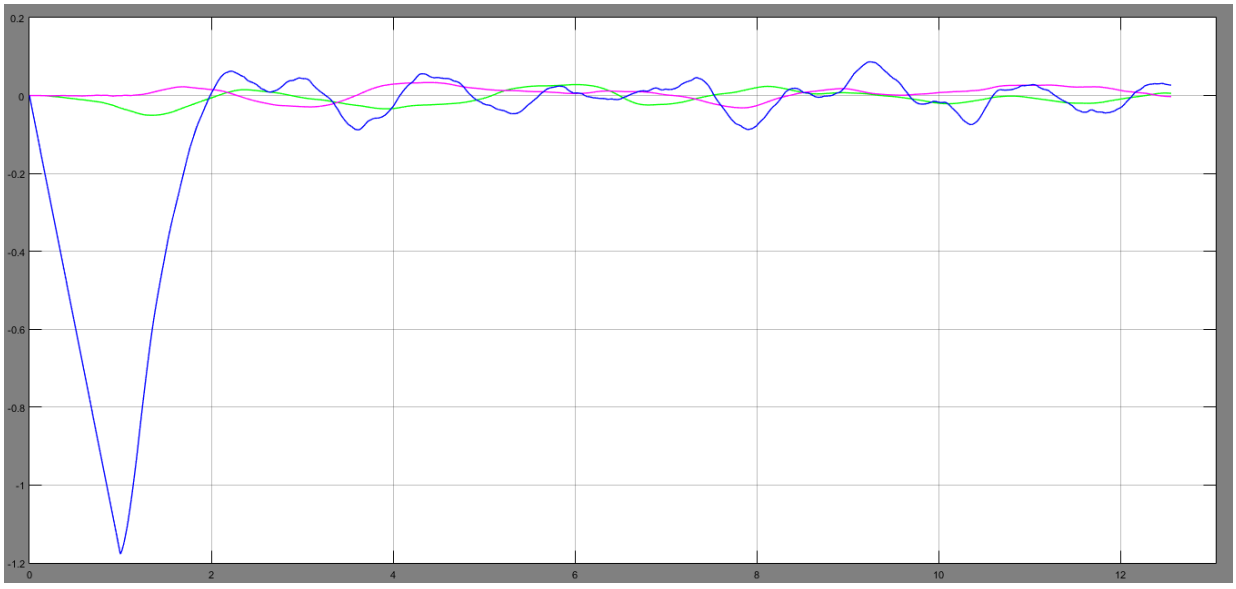
Εικόνα 4.4.2 Γραφική απεικόνιση γωνιών Euler

Οι ταχύτητες των κινητήρων που φαίνονται στη παρακάτω γραφική, έχουν ίδιο μέτρο και αντίθετη φορά ανά ζευγάρι. Για το λόγο αυτό φαίνονται αντικατοπτρισμένες. Αυξάνουν απότομα στροφές στο πρώτο δευτερόλεπτο όπου συμβαίνει και η απογείωση και μειώνονται σταθερά μέχρι το δεύτερο δευτερόλεπτο, όπου σταθεροποιούνται.



Εικόνα 4.4.3 Γραφική απεικόνιση ταχύτητας κινητήρων

Στη παρακάτω γραφική φαίνονται οι ταχύτητες ως πρός τους άξονες XYZ, όπου παρατηρείται σημαντική αλλαγή στον άξονα Z κατά την απογείωσή του και μηδενισμός με το που γίνει η σταθεροποίηση του quadcopter.



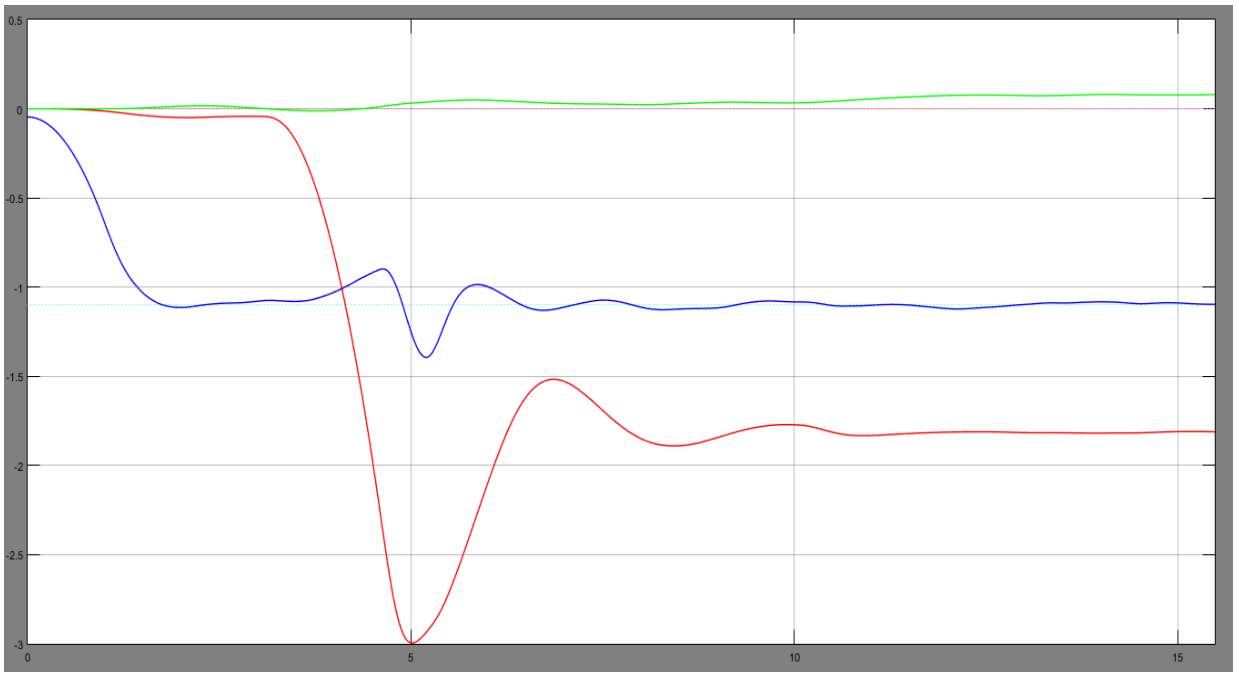
Εικόνα 4.4.4 Γραφική απεικόνιση ταχυτήτων στους άξονες XYZ

4.5. Παράδειγμα Εξομοίωσης με Τυχαίες Βηματικές Εισόδους

Διατηρώντας τις ίδιες παραμέτρους του ελεγκτή PID που χρησιμοποιήθηκαν και στη προηγούμενη εξομοίωση, γίνεται πρόσθεση:

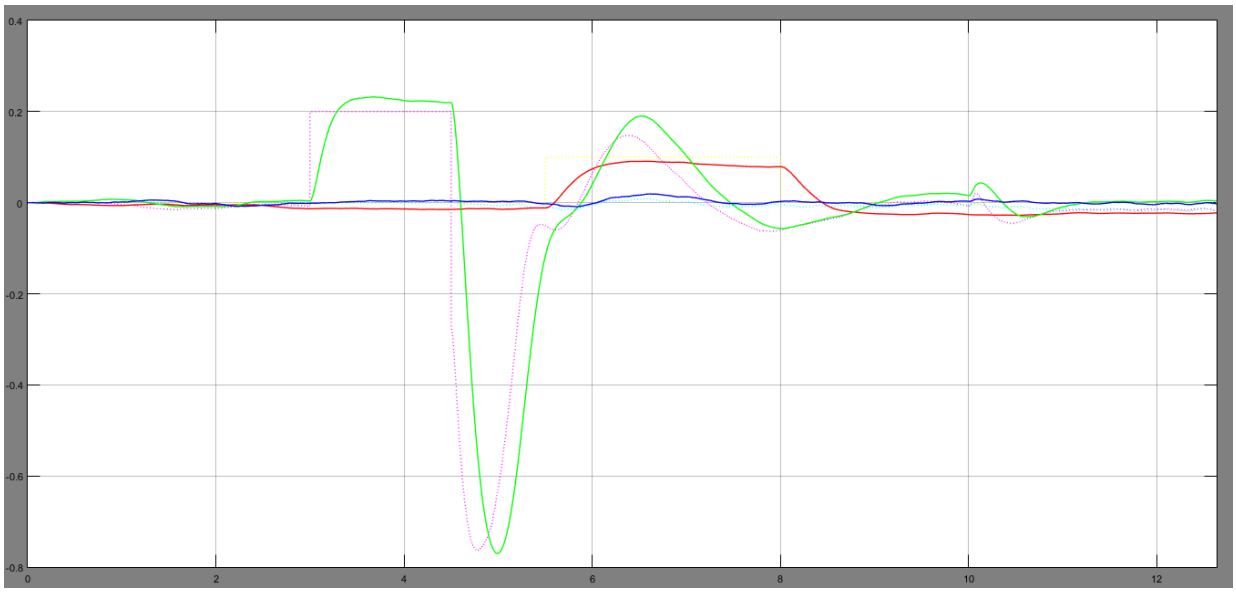
- Βηματική είσοδο στο pitch τη χρονική στιγμή t = 3 sec, διάρκειας Δt = 1.5 sec και κέρδους K = 0.2.
- Βηματική είσοδο στο yaw τη χρονική στιγμή t = 5.5 sec, διάρκειας Δt = 2.5 sec και κέρδους K = 0.1.
- Διαταραχή motorFailure στους κινητήρες 3 (K_3 = -30) και 4 (K_4 = 20) τη χρονική στιγμή t = 10 sec, διάρκειας Δt = 0.05 sec.

Παρατηρείται ότι την χρονική στιγμή t = 3 sec αλλάζει η μετατόπιση του quadcopter στον άξονα X μέχρι τη χρονική στιγμή t = 4.5 sec όπου προσπαθεί να σταθεροποιηθεί. Στη προσπάθεια σταθεροποίησης, παρουσιάζεται και μια διαταραχή στο ύψος, η οποία και εξαλείφεται. Για την αλλαγή του yaw δεν παρατηρείται αλλαγή όπως ήταν αναμενόμενο.



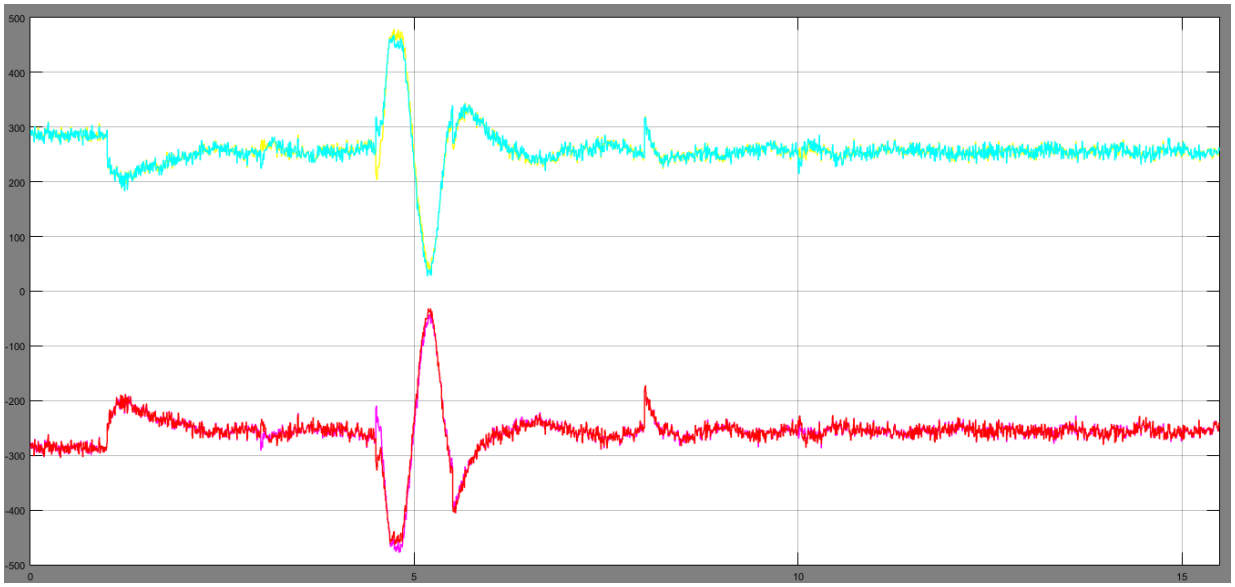
Εικόνα 4.5.1 Γραφική απεικόνιση θέσης στον XYZ

Στην γραφική απεικόνιση των γωνιών Euler φαίνονται ακριβώς οι διαταραχές που δόθηκαν, δηλαδή αλλαγή στο pitch από t = 3 sec έως t = 4.5 sec και έπειτα η προσπάθεια αποκατάστασης του σφάλματος και η αλλαγή στο yaw από t = 5.5 sec έως τα t = 8 sec. Εδώ φαίνεται περισσότερο και η μικρή διαταραχή που υφίστανται στους κινητήρες 3, 4 στα 10 sec, καθώς λόγω της θέσης τους επηρεάζουν εντονότερα το roll.



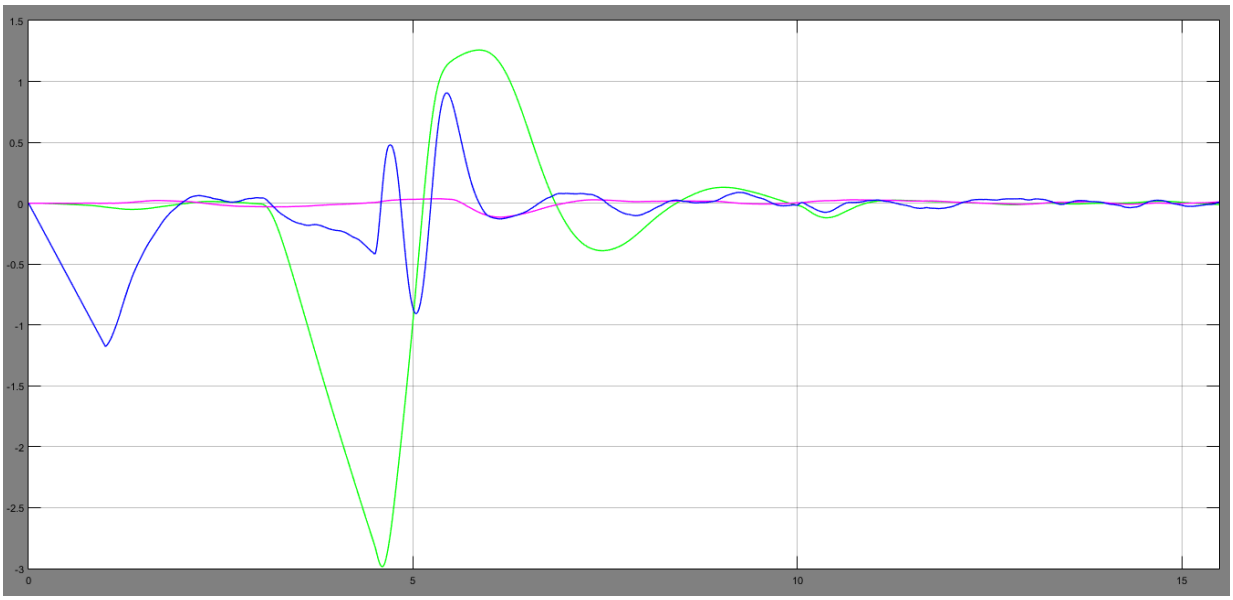
Εικόνα 4.5.2 Γραφική απεικόνιση γωνιών Euler

Αντίστοιχα, οι εντολές που δέχονται οι κινητήρες είναι ανάλογες των εισόδων.



Εικόνα 4.5.3 Γραφική απεικόνιση ταχύτητας κινητήρων

Τέλος, εδώ φαίνονται και οι ταχύτητες στους άξονες XYZ. Μπορεί επίσης να παρατηρηθεί και η μικρή διαταραχή στο ύψος που υπάρχει κατά την αποκατάσταση του pitch.



Εικόνα 4.5.4 Γραφική απεικόνιση ταχυτήτων στους άξονες XYZ

Κεφάλαιο 5: Προαπαιτούμενα Εκτέλεσης Πειραμάτων

5.1. Εγκατάσταση firmware

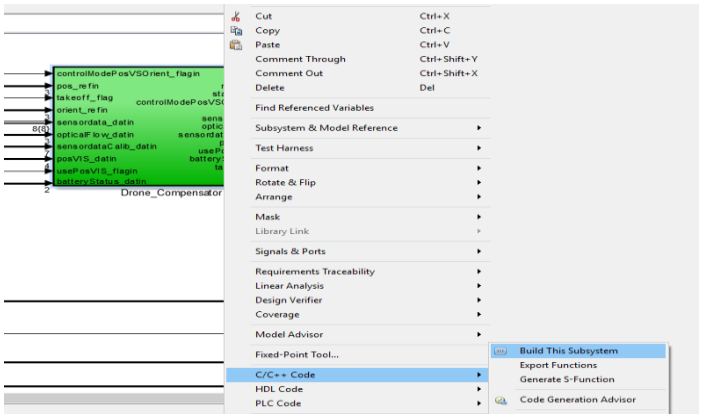
Για την λειτουργία του quadcopter, είναι απαραίτητη η εγκατάσταση ενός custom firmware. Ως firmware ορίζεται το υλικολογισμικό, ή λογισμικό ηλεκτρονικών συσκευών, το οποίο είναι γραμμένο σε γλώσσα μηχανής από την ίδια την κατασκευαστική εταιρία και την αντιλαμβάνεται μόνο το συγκεκριμένο μοντέλο. Πάνω στις προδιαγραφές αυτού του συγκεκριμένου υλικολογισμικού είναι "χτισμένο" όλο του σύστημα στο Simulink. Η διαδικασία εγκατάστασης υλοποιείται μέσω της ενσύρματης σύνδεσης του Rolling Spider με τον υπολογιστή. Αυτό γίνεται από τη γραμμή εντολών (Terminal), μέσω των παρακάτω προγραμμάτων:

- EDUfirmwareUploadSYS.sh (Αντιγραφή του rollingspider.edu.plf στο κύριο φάκελο του drone)
- EDUfirmwareUploadFILES.sh (Αποστολή αρχείων μέσω ftp)
- EDUfirmwareInitialize.sh (Εγκατάσταση και παροχή δικαιωμάτων διαχειριστή)

Τέλος, για την σύνδεση του quadcopter με τον υπολογιστή, η οποία γίνεται μέσω Bluetooth, χρειάστηκε απλά η καταγραφή της MAC Address του quadcopter, που είναι μοναδική για κάθε συσκευή, φυσική διεύθυνση για επικοινωνία στο φυσικό τμήμα του δικτύου.

5.2. Δημιουργία Κώδικα C από το Simulink

Μια από τις αξιοσημείωτες λειτουργίες που μπορεί να προσφέρει το Matlab, και κατ' επέκταση το Simulink, είναι η δημιουργία κώδικα C ή άλλων γλωσσών προγραμματισμού από ένα ή πολλά block του συστήματος που ο ίδιος ο χρήστης έχει υλοποιήσει. Ο λόγος που γίνεται εδώ η χρήση αυτής της λειτουργίας είναι ότι για να ακολουθήσει το quadcopter τον έλεγχο που εξηγήθηκε και αναλύθηκε στα προηγούμενα κεφάλαια, πρέπει να μεταφραστεί σε γλώσσα τέτοια που να μπορεί να αντιληφθεί το υλικολογισμικό που διαθέτει το quadcopter. Η δημιουργία αυτού του κώδικα γίνεται κάνοντας δεξί κλικ στο block Drone_Compensator → C/C++ Code → Build this Subsystem.



Εικόνα 5.2.1 Παράδειγμα Υλοποιήσης Κώδικα C

Μετά την δημιουργία του, ο κώδικας μπορεί να αποσταλλεί με Bluetooth στο quadcopter μέσω του DroneUploadEmbeddedCode.sh. Το πρόγραμμα αυτό ενσωματώνει τον κώδικα C του Simulink με τον κώδικα rsedu_control.c (Παράρτημα 2 - Κώδικας rsedu_control.c) και τον αποστέλλει μαζί με τους κώδικες rsedu_vis_helpers.c, rsedu_vis.c και rsedu_of.c μέσω της βιβλιοθήκης librsedu.so με την χρήση της διεργασίας expect.

5.3. Ανάλυση Λειτουργιών Κώδικα Πτήσης

Το firmware περιέχει τη βασική διεργασία 'SpiderFlight.sh', όπου κατά την πτήση ενεργοποιεί:

- rsedu_control.c (Παράρτημα 2 – Κώδικας rsedu_control.c)

  Η πρώτη διεργασία rsedu_control είναι ο κώδικας που έχει στόχο τον έλεγχο του quadcopter και έχει αποσταλεί στο firmware. Το quadcopter καλεί αυτή τη διεργασία με συχνότητα 200Hz. Αρχικά, γίνεται ο προκαθορισμός των κύκλων μηχανής για κάθε στάδιο της πτήσης και των παραμέτρων του quadcopter σε περίπτωση που δεν βρεθεί το αρχείο paramsEDU.dat, το οποίο βρίσκεται εντός του firmware και περιέχει τις βασικές παραμέτρους λειτουργίας του quadcopter. Έπειτα, ακολουθεί ο κώδικας του Drone_Compensator ο οποίος " χτίζεται " από το Simulink.

paramsEDU.dat (Βρίσκεται εντός του quadcopter στην διεύθυνση /data/edu/params/)

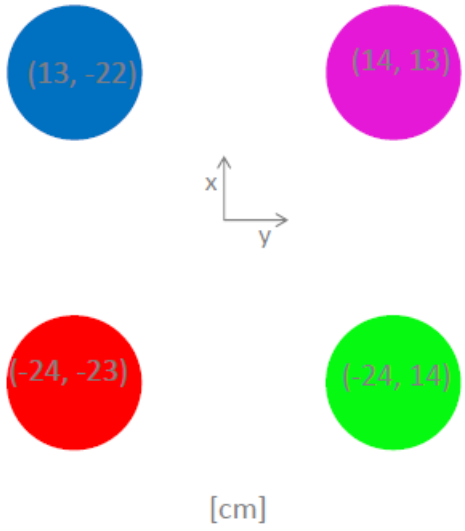
```
FEAT_TIME          = 0;    //Καταγραφή χρόνων για είσοδο ή έξοδο από τους κώδικες
FEAT_OF_ACTIVE     = 1;    //Ενεργοποίηση/Απενεργοποίηση optical flow
FEAT_POSVIS_RUN    = 1;    //Εύρεση θέσης ανάλογα από το αν υπάρχουν landmarks
FEAT_POSVIS_USE    = 0;    //Χρήση POSVIS_RUN για ενίσχυση της εκτίμησης Kalman
FEAT_IMSAVE        = 1;    //1: Αποθήκευση εικόνων, 2: Online μετάδοση εικόνων
FEAT_NOLOOK        = 0;    //Χρήση ή όχι του lookuptable για εύρεση landmarks
FEAT_NOSAFETY      = 0;    //1: Εκτέλεση μη ασφαλούς πτήσης
```

Ο υπόλοιπος κώδικας αφορά τη πτήση η οποία χωρίζεται σε πέντε στάδια:

- Ανάγνωση του paramsEDU.dat, υλοποίηση παραμέτρων για επικοινωνία με image_processing και optical_flow (fifo) και δημιουργία πίνακα με τις αρχικές μετρήσεις των αισθητηρίων.
- Καταγραφή δεδομένων αισθητηρίων για τη βαθμονόμηση τους και ενεργοποίηση των κινητήρων.
- Έλεγχος παραμέτρων για την επίτρεψη της πτήσης, καταγραφή εισόδων και εξόδων από το Drone_Compensator, έλεγχος λειτουργίας optical_flow και image_processing και μοντελοποίηση των αισθητηρίων.
- Ανάγνωση εντολών από το ReferenceValueServer και rsedu_vis. Υλοποίηση τους στο μοντέλο και έλεγχος συνθηκών για τη διακοπή της πτήσης (χαμηλή μπαταρία, πτήση εκτός εμβέλειας, σύγκρουση με άλλο αντικείμενο και μη καταγραφή optical_flow).
- Αποθήκευση δεδομένων και τερματισμός πτήσης.

- rsedu_vis.c (Παράρτημα 2 - Κώδικας rsedu_vis.c)

  Η διεργασία αυτή έχει ως σκοπό την εύρεση της θέσης και του προσανατολισμού του quadcopter στο χώρο βάσει της εικόνας που δέχεται από την κάμερα, έπειτα από επεξεργασία αυτής. Για τη σωστή λειτουργία, μπορεί να γίνει εντοπισμός μιας συγκεκριμένης κατάταξης χρωμάτων (landmarks), με καταγραφή των συντεταγμένων τους στο lookuptable.dat για χρήση στην επόμενη πτήση, και όπου σύμφωνα με αυτή γίνεται η σωστή αντίληψη του χώρου. Διαφορετικά, θεωρεί το σημείο απογείωσης ως αρχή των αξόνων.



Εικόνα 5.3.1 Landmarks

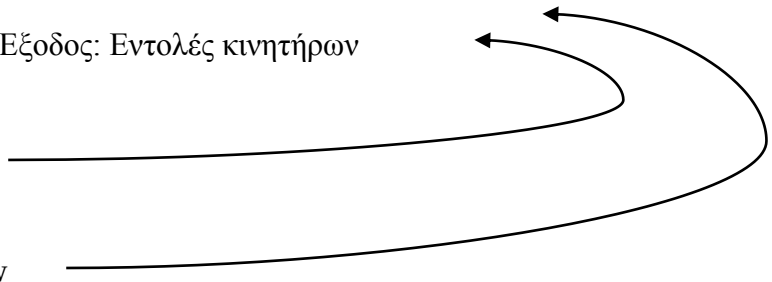
Η κάμερα μετατρέπει κάθε pixel της εικόνας στη μορφή HSV (Hue Saturation Value) και συγκρίνει την απόχρωσή του με την τιμή Η που υπάρχει σε μια βάση δεδομένων. Η μορφή HSV είναι μια κυλινδρική αναπαράσταση όπου τα χρώματα κατατάσσονται στις συντεταγμένες του κυλίνδρου ανάλογα με την απόχρωση, τον κορεσμό και τη φωτεινότητα τους. Στη περίπτωση που κάποιο pixel αντιστοιχεί σε κάποια τιμή Η της βάσης δεδομένων, καταγράφονται οι συντεταγμένες του. Μόλις βρεθεί ο απαιτούμενος αριθμός γειτονικών pixel που αντιστοιχούν στη τιμή Η του χρώματος αυτού, τα αποτελέσματα μεταφέρονται στο κώδικα rsedu_vis_helpers.c, όπου γίνεται η μετατροπή των συντεταγμένων των landmarks από pixel σε συντεταγμένες χώρου. Το quadcopter ορίζει το κέντρο της συγκεκριμένης διάταξης χρωμάτων σαν αρχή των αξόνων. Έπειτα η καινούρια θέση της κάμερας, άρα και του quadcopter, αποστέλλεται στον κώδικα rsedu_control.c. Τέλος, δίνεται η δυνατότητα επιλογής λήψης των δεδομένων είτε σε πραγματικό χρόνο είτε μετά το πέρας της πτήσης.

- rsedu_optical_flow
  Η διεργασία αυτή έχει σαν είσοδο το αποτέλεσμα τον υπολογισμό των ταχυτήτων του optical flow και την αποστολή τους μέσω του FIFO pipe.

Οι δύο τελευταίες διεργασίες καλούνται στα 60Hz και επικοινωνούν με το rsedu_control μέσω FIFO pipes (Εικόνα 5.3.2).
Τα FIFO pipes είναι ένας μηχανισμός για διαδιεργασιακή επικοινωνία. Τα δεδομένα γράφονται στο pipe, ή σωλήνωση, από μια διεργασία και μπορούν να διαβαστούν από μια άλλη. Τα δεδομένα αυτά διαχειρίζονται με την μέθοδο FIFO (First In – First Out). Αυτός ο τρόπος επικοινωνίας είναι αρκετά διαδεδομένος σε λειτουργικά συστήματα για επικοινωνία μεταξύ διαφόρων διεργασιών.

Επικοινωνία διεργασιών με fifo pipes

- rsedu_control.c στα 200Hz
  Είσοδος: Δεδομένα αισθητηρίων, Έξοδος: Εντολές κινητήρων
- rsedu_vis.c στα 60Hz
  Είσοδος: Εικόνα κάμερας
- rsedu_of.c στα 60Hz
  Είσοδος: Υπολογισμός optical flow



5.4. Διαδικασία Σύνδεσης και Πτήσης

Οι εντολές για τη σύνδεση είναι:

i. DroneConnect.sh

   Σύνδεση υπολογιστή με το quadcopter μέσω Bluetooth. Η σύνδεση αυτή γίνεται μέσω της MAC Address του quadcopter και κάνοντας ping στην διεύθυνση IP του για τον έλεγχο της σύνδεσης.

ii. DroneDisconnect.sh

   Λήξη της σύνδεσης του υπολογιστή με το quadcopter.

iii. DroneReboot.sh

   Επανεκκίνηση του quadcopter.

Οι εντολές για τη πτήση είναι:

i. DroneTest.sh

   Δοκιμή της ανταπόκρισης του quadcopter, περιστρέφοντας τους έλικες σε χαμηλό αριθμό στροφών για μικρό χρονικό διάστημα (POWERGAIN = 10).

ii. DroneRun.sh

   Αποστολή σήματος για την εκκίνηση της πτήσης, αυξάνοντας τον αριθμό των στροφών έτσι ώστε να επιτευχθεί αιώρηση του quadcopter (POWERGAIN = 100).

iii. DroneKeyboardPilot.sh

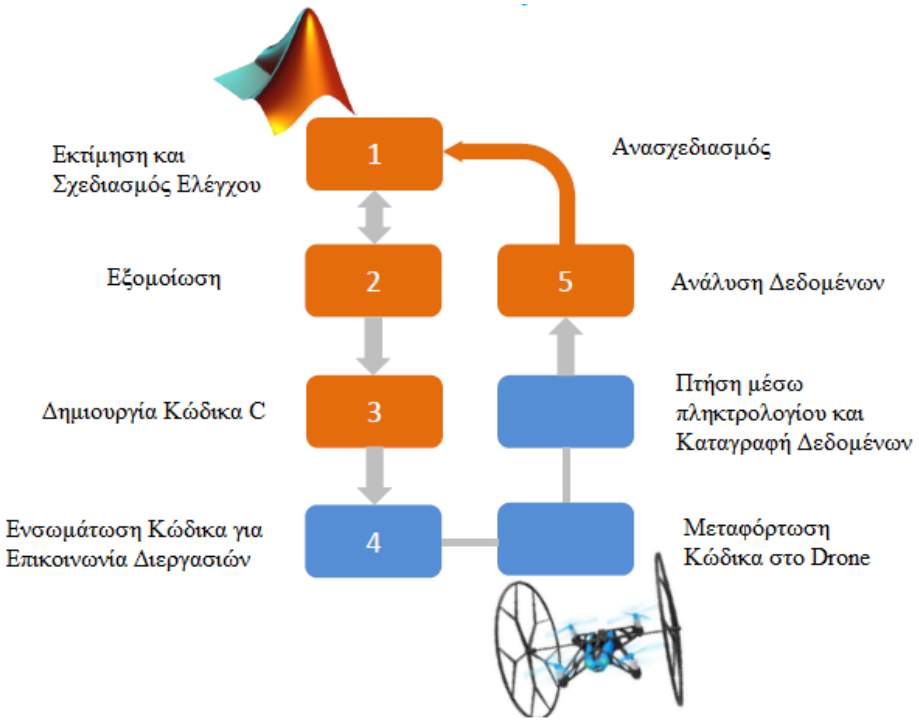
   Αποτελείται από δυο διεργασίες, τη DroneWatchDog.sh η οποία κάνει τον έλεγχο λειτουργίας του DroneTest.sh ή DroneRun.sh, και τη ReferenceValueServer.c (Παράρτημα 2 - Κώδικας ReferenceValueServer.c) η οποία κάνει αποστολή εντολών από το χρήστη μέσω του πληκτρολογίου. (Σημείωση: Τα πλήκτρα για τις εντολές πρέπει να πατηθούν από τον χρήστη μια φορά (tap) και όχι συνεχόμενα.)

Οι δύο τελευταίες εντολές υλοποιούνται σε διαφορετικά παράθυρα της γραμμής εντολών.

```
Waiting for connection to drone...

 d,a : roll
 w,s : pitch
 j,l : yaw
 i,k : alt 0.2
 y,h : alt 0.6
 f : coordinates input
 z : pattern x-y
 x : pattern Euler
 q : reset position
 r : reset, e : exit
```

Εικόνα 5.4.1 Ένδειξη πλήκτρων χειρισμού από το πληκτρολόγιο όπως φαίνεται από το terminal

5.5. Επισκόπηση Συστήματος



Εικόνα 5.5.1 Διάγραμμα Ροής Ακολουθίας Εργασιών *

- Βήμα 1:

  Μοντελοποίηση των μαθηματικών εξισώσεων του quadcopter

  Δημιουργία ελεγκτη PID
- Βήμα 2:

  Εξομοίωση συστήματος

  Καταγραφή αποκρίσεων σε διάφορες εισόδους του συστήματος

  Τροποποίηση παραμέτρων σε τυχόν ανεπιθύμητες συμπεριφορές
- Βήμα 3:

  Δημιουργία Κώδικα C από το υπάρχον σύστημα
- Βήμα 4:

  Δημιουργία τελικού κώδικα και αποστολή του στο quadcopter.

  Πτήση

  Καταγραφή αποτελεσμάτων
- Βήμα 5:

  Ανάλυση αποτελεσμάτων και ανασχεδιασμός του συστήματος

* https://github.com/Parrot-Developers/RollingSpiderEdu/blob/master/MIT_MatlabToolbox/media/GettingStarted.pdf

Κεφάλαιο 6: Πειραματικό Μέρος

6.1. Σταθερή Αιώρηση και Λήψη Δεδομένων μετά το πέρας της πτήσης

Για την υλοποίηση της σταθερής αιώρησης πρώτα γίνεται σύνδεση με την εκτέλεση του DroneConnect.sh. Έπειτα, σε διαφορετικό παράθυρο γραμμής εντολών εκτελείται το DroneKeyboardPilot.sh και στο πρώτο παράθυρο εκτελείται το DroneRun.sh.

Η σταθερή αιώρηση γίνεται είτε με την χρήση μιας συγκεκριμένης διάταξης χρωμάτων (landmarks), είτε χωρίς αυτή.
Στη περίπτωση της λειτουργίας της σταθερής αιώρησης πάνω από τα landmarks, το quadcopter σταθεροποιείται στο κέντρο της συγκεκριμένης διάταξης χρωμάτων. Η εύρεση και η αναγνώριση των χρωμάτων υλοποιείται εντός του κώδικα rsedu_vis.c και με την βοήθεια του lookuptable.dat που εξηγήθηκε προηγουμένως. Η όλη διαδικασία μπορεί να υλοποιηθεί και χωρίς το lookuptable.dat, τροποποιώντας την παράμετρο NO_LOOK = 1 εντός του paramsEDU.dat και μέσω του κώδικα image_module.c όπου η αναγνώριση των landmarks και η καταγραφή των θέσεών τους γίνεται online. Η συγκεκριμένη διαδικασία είναι πιο αργή λόγω των υπολογισμών που κάνει εκείνη την στιγμή. Με πιθανή μετακίνηση όλων των landmarks, το quadcopter προσαρμόζει ανάλογα την θέση του.
Στην περίπτωση που δεν υπάρχουν landmarks, το quadcopter ορίζει το σημείο απογείωσης ως την αρχή των αξόνων.

Μετά τη πτήση του quadcopter, υπάρχει δυνατότητα λήψης διαφόρων δεδομένων όπως γραφικές παραστάσεις, εικόνες και πίνακες για την καλύτερη κατανόηση της συμπεριφοράς του. Τα δεδομένα αυτά δημιουργούνται και αποθηκεύονται μέσα στο σύστημα του quadcopter ώστε μετά τη πτήση να αποσταλλούν στον υπολογιστή. Τα παραπάνω υλοποιούται με τη ρύθμιση της παραμέτρου IM_SAVE = 1 από το paramsEDU.dat, όπου επιτρέπει τη δημιουργία ενός φακέλου /tmp/edu/ για την αποθήκευση των δεδομένων εντός του quadcopter.

Οι εντολές που χρησιμοποιούνται για τη συλλογή δεδομένων μετά τη πτήση είναι:

i. DroneDownloadFlightData.sh

   Λήψη γραφικών παραστάσεων και εμφάνισή τους μέσω του αρχείου FlightAnalyzer.m από το Matlab.

ii. DroneDownloadPTimes.sh

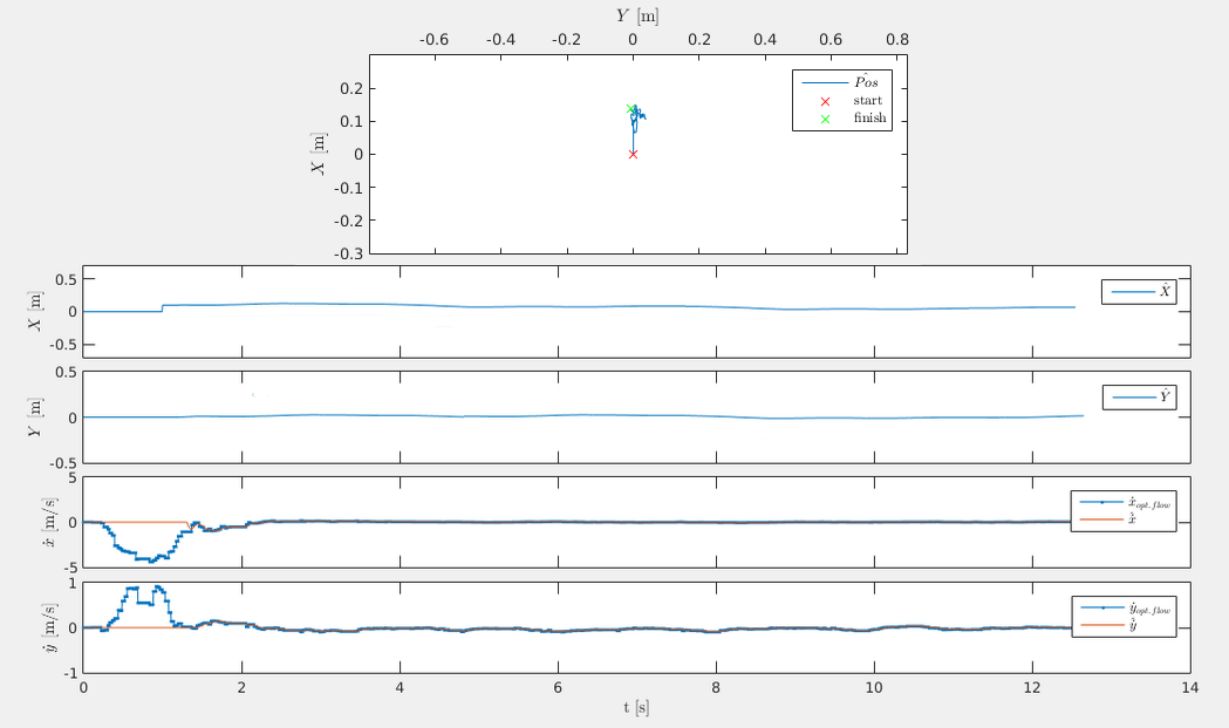
   Λήψη γραφικής παράστασης που αφορά την συσχέτιση των σημάτων control, image processing και optical flow και εμφάνισή τους μέσω του αρχείου PTimesAnalyzer.m από το Matlab.

iii. DroneDownloadImages.sh

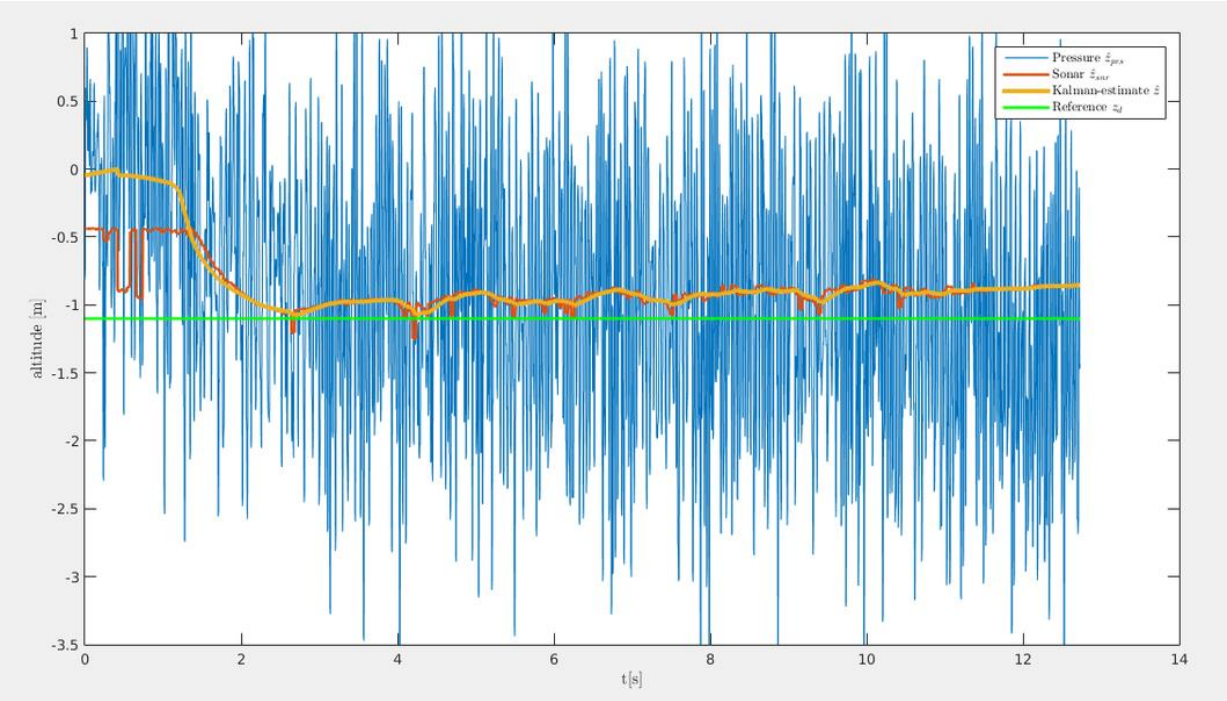
   Λήψη εικόνων από την κάμερα του Rolling Spider.

iv. VisionPrePostProcessor.sh.

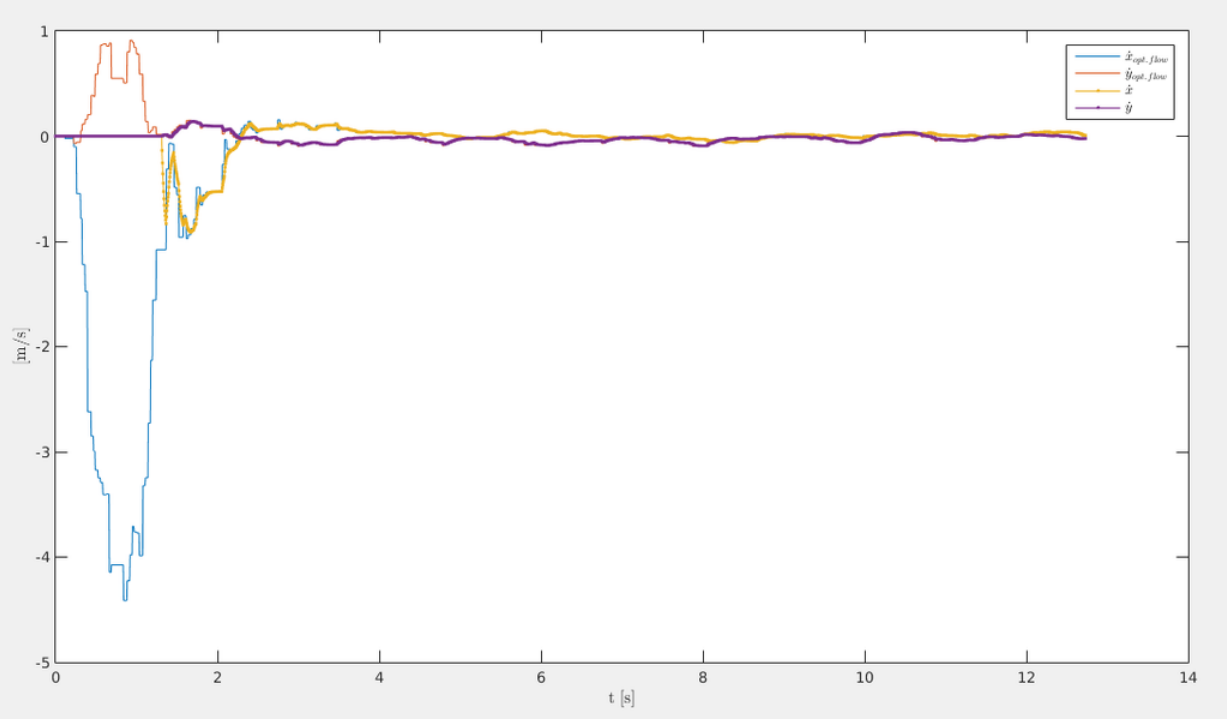
   Εισαγωγή πίνακα διανυσμάτων lookuptable.dat μετά από ανάλυση μέσω του optical flow, μετατροπή των εικόνων που λήφθηκαν σε έγχρωμη και ασπρόμαυρη (ppm) μορφή και καταγραφή θέσεων landmarks στο poses.txt.



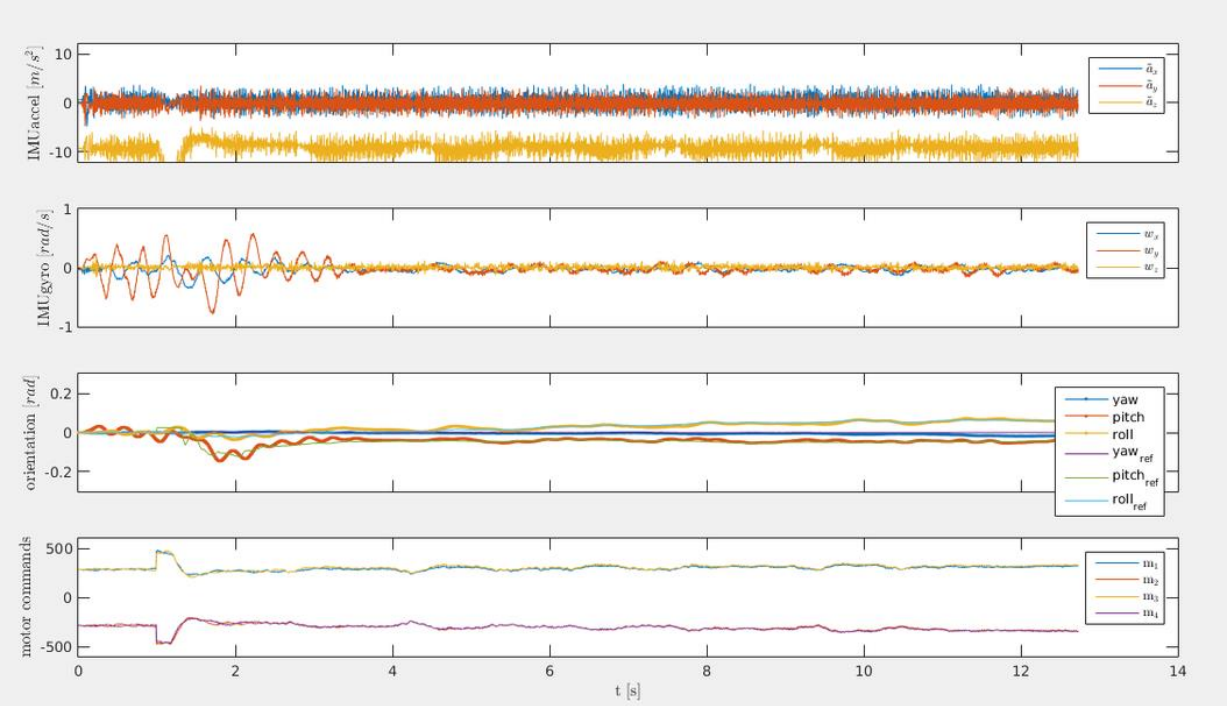
Εικόνα 6.1.1 Θέσεις στον X, Y άξονα και ταχύτητες



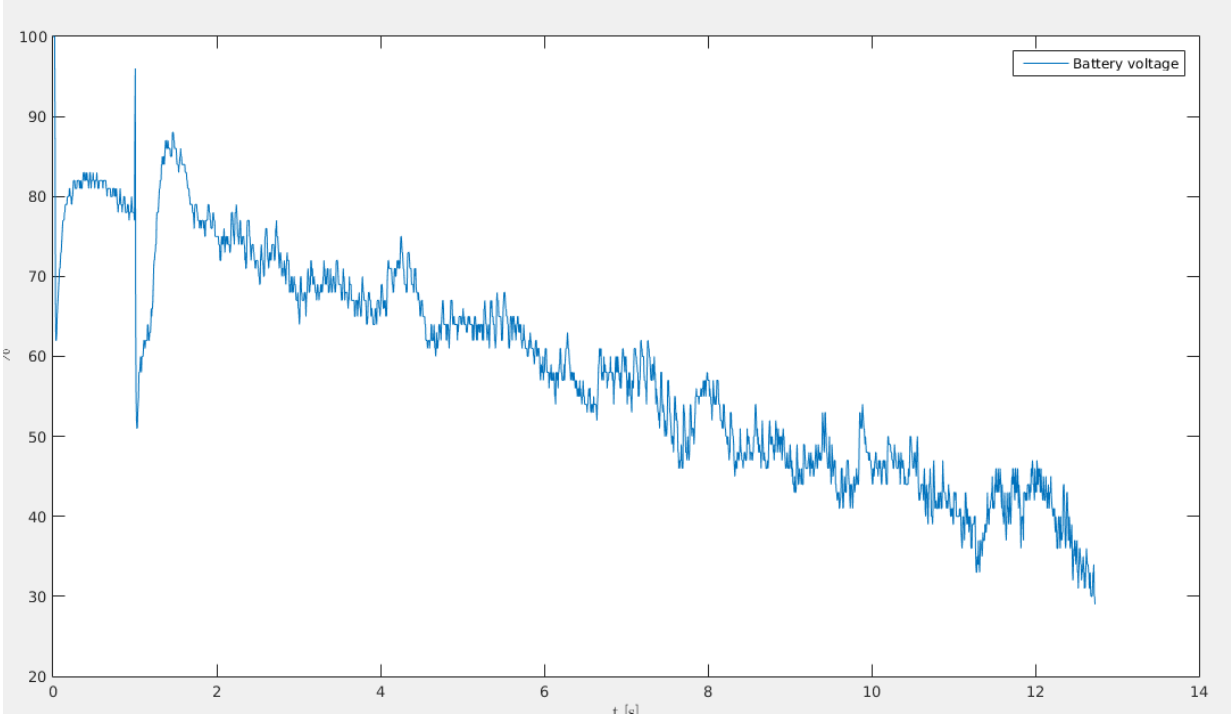
Εικόνα 6.1.2 Ύψος, μετρήσεις αισθητηρίων πίεσης και sonar και εκτίμηση Kalman



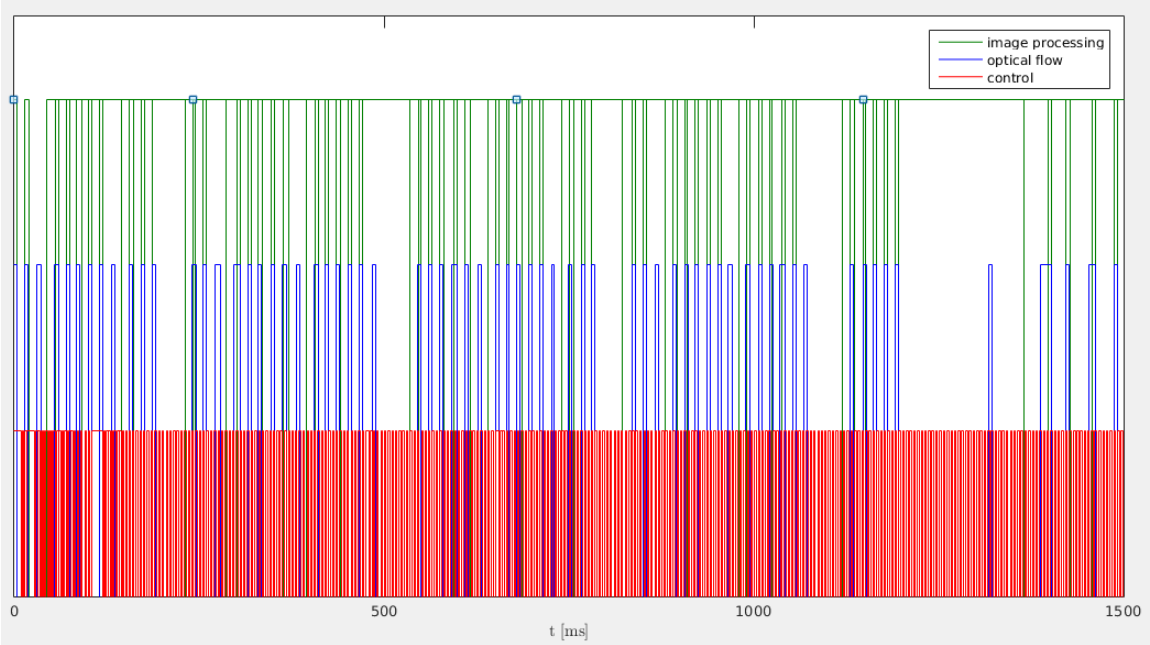
Εικόνα 6.1.3 Ταχύτητες στον X, Y άξονα και εκτίμηση ταχύτητας από τον optical flow



Εικόνα 6.1.4 Αισθητήρια IMU, προσανατολισμός και εντολές κινητήρων

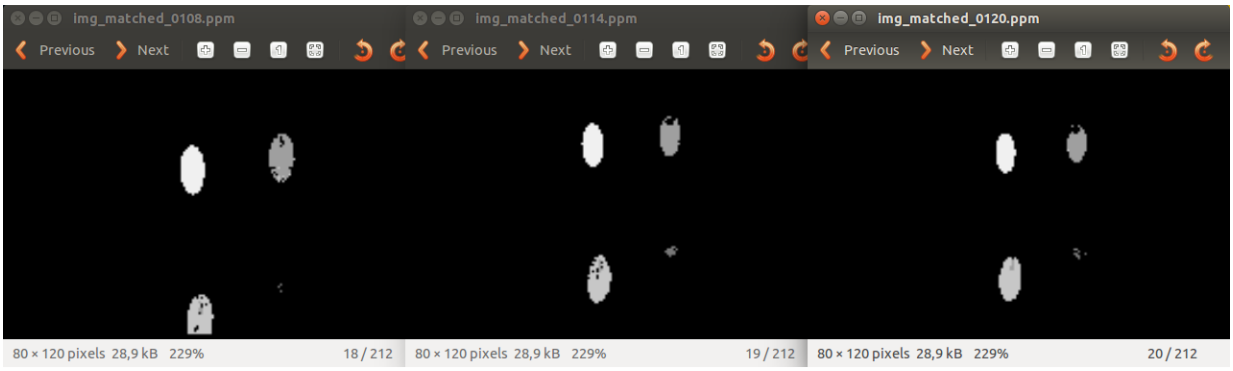


Εικόνα 6.1.5 Τάση μπαταρίας



Εικόνα 6.1.6 Γραφική παράσταση από PTimes

Εικόνα 6.1.7 Παράδειγμα από την κάμερα σε μορφή rgb



Εικόνα 6.1.8 Παράδειγμα από την κάμερα για landmark identification

Οι εικόνες μετατρέπονται σε binary μορφή για να μπορέσει το quadcopter να αναγνωρίσει τα landmarks. Παράλληλα ο κώδικας του VisionPrePostProcessor.sh δημιουργεί τον πίνακα lookuptable.dat βάσει του οποίου γίνεται αποθήκευση των θέσεών τους ώστε στην επόμενη πτήση να γνωρίζει την θέση της κάμεράς του. Παρατηρείται ότι το πράσινο χρώμα δεν ανιχνεύεται με την ίδια ευκολία λόγω της παρόμοιας φωτεινότητας που έχει με το λευκό.

6.2. Σταθερή Αιώρηση και Λήψη Δεδομένων κατά τη διάρκεια της πτήσης

Για την επίτευξη της λήψης δεδομένων κατά τη διάρκεια της πτήσης, απαραίτητη προϋπόθεση είναι η ρύθμιση της παραμέτρου IM_SAVE = 2. Αυτό επιτρέπει τη δημιουργία ενός FIFO pipe με το όνομα /tmp/picture για τη μεταφορά των εικόνων σε αυτό το προορισμό.

Τα βήματα για την υλοποίηση της επικοινωνίας μέσω του FIFO pipe είναι:

- Άνοιγμα παραθύρου γραμμής εντολών και εκτέλεση του προγράμματος

  DroneConnect.sh

  για τη σύνδεση με το drone. (Εικόνα 6.2.1, Παράθυρο γραμμής εντολών No 1)

- Άνοιγμα δεύτερου παραθύρου και χρήση της εντολής:

  telnet 192.168.1.1.

  Το telnet είναι ένα πρωτόκολλο επικοινωνίας, το οποίο μαζί με την IP του drone επιτυγχάνει τη διασύνδεση του υπολογιστή με το drone, έτσι ώστε να επιτρέπεται ο ελεγχός του. Εντός του drone εκτελείται η εντολή:

  while [1]; do cat /tmp/picture | nc 192.168.1.2 1234; done

  Με τη παραπάνω εντολή, για όσο υπάρχει σύνδεση, γίνεται η ανάγνωση των εικόνων από το tmp/picture το οποίο είναι ένα FIFO pipe που έχει δημιουργηθεί από τον κώδικα rsedu_vis.c. Η κάθετη γραμμή (|) που υπάρχει στην παραπάνω εντολή δηλώνει ότι η έξοδος της ‘do cat /tmp/picture’, που είναι οι εικόνες οι οποίες έχουν ληφθεί από την κάμερα του drone, λειτουργεί σαν είσοδος της ‘ nc 192.168.1.2 1234’. Η ‘nc 192.168.1.2 1234’ καταγράφει τις εικόνες στην IP 192.168.1.2 του port 1234. (Εικόνα 6.2.1, Παράθυρο γραμμής εντολών No 2)

- Άνοιγμα τρίτου παραθύρου γραμμής εντολών και εκτέλεση των εντολών:

  mkfifo /tmp/rollingspiderpicture;

  Η εντολή αυτή δημιουργεί ένα FIFO pipe με όνομα /tmp/rollingspiderpicture

  while [1]; do nc –l 1234 > /tmp/rollingspiderpicture; done

  Σε αυτή την εντολή, για όσο υπάρχει σύνδεση, καταγράφονται τα δεδομένα από το port 1234 και εισέρχονται στο /tmp/rollingspiderpicture. (Εικόνα 6.2.1, Παράθυρο γραμμής εντολών No 3)

- Προαιρετικά, σε τέταρτο παράθυρο γραμμής εντολών, με την εκτέλεση της παρακάτω εντολής (Εικόνα 6.2.1, Παράθυρο γραμμής εντολών No 4):

  mplayer –demuxer rawvideo –rawvideo

  w=160:h=120:format=yuy2 –loop 0 /tmp/rollingspiderpicture

  υπάρχει δυνατότητα εμφάνισης των εικόνων σε πραγματικό χρόνο, σε νέο παράθυρο, κατά τη διάρκεια της πτήσης. (Εικόνα 6.2.1, Παράθυρο No 6). Οι εικόνες αυτές προβάλλονται μέσω του mplayer μόνο μια φορά, καθώς δεν αποθηκεύονται σε συγκεκριμένο φάκελο.

- Άνοιγμα πέμπτου παραθύρου γραμμής εντολών και εκτέλεση του προγράμματος:

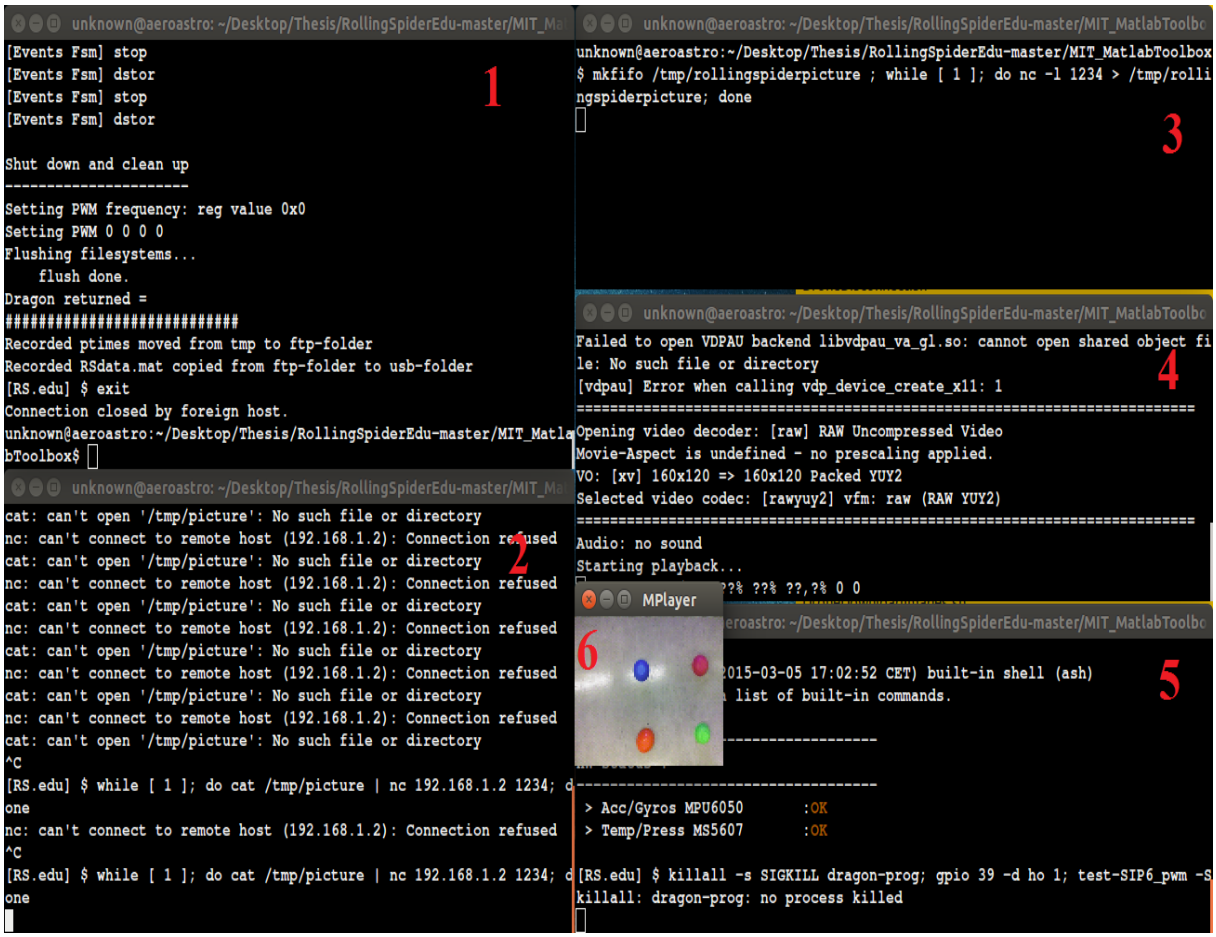
  DroneKeyboardPilot.sh

  για τον έλεγχο του drone. (Εικόνα 6.2.1, Παράθυρο γραμμής εντολών No 5).

- Τέλος, στο πρώτο παράθυρο γραμμής εντολών γίνεται η εκτέλεση του προγράμματος:

  DroneRun.sh

  για την εκκίνηση της πτήσης του quadcopter (Εικόνα 6.2.1, Παράθυρο γραμμής εντολών No1). Κάνοντας κλίκ στο πέμπτο παράθυρο γραμμής εντολών έχουμε πρόσβαση στο ReferenceValueServer.sh, και οτιδήποτε καταγράφεται σε αυτό αποστέλλεται στο quadcopter.



Εικόνα 6.2.1 Αποτελέσματα από την εκτέλεση εντολών και προγραμμάτων στα παράθυρα γραμμής εντολών

6.3. Εκτέλεση Προκαθορισμένης Διαδρομής (Pattern)

Σε αυτό το πείραμα γίνεται η αυτόματη εκτέλεση προκαθορισμένης διαδρομής από το quadcopter με την αποστολή μόνο ενός σήματος από το χρήστη. Δηλαδή, δεν είναι απαραίτητος ο χειρισμός του quadcopter από το χρήστη καθ'όλη τη διάρκεια της πτήσης. Ο κώδικας που υλοποιεί αυτό το πείραμα είναι ο ReferenceValueServer.c, γιατί η εντολοδότηση στην αρχή της διαδικασίας γίνεται από το πληκτρολόγιο, άρα μέσω του DroneKeyboardPilot.sh.

Εντός του ReferenceValueServer.c, γίνεται η ανάγνωση των πλήκτρων μέσω της συνάρτησης getch(). Στο κυρίως πρόγραμμα ορίζονται οι ακραίες τιμές ύψους και γωνιών και εμφανίζεται ένα υπόμνημα με τις εντολές που είναι δυνατό να δώσει ο χρήστης. Στη συνέχεια του κώδικα, κατά τη διάρκεια της πτήσης, ρυθμίζονται οι αλλάγες που προκύπτουν από τις εντολές που δίνει ο χειριστής. Μία ή περισσότερες από αυτές τις εντολές μπορούν γίνουν υπεύθυνες για την εκτέλεση μιας προκαθορισμένης διαδρομής. Η αποστολή αυτών γίνεται καταγράφοντας τα δεδομένα σε ένα string με όνομα sendBuff και με τη χρήση socket. Το sendBuff περιέχει τις

γωνίες Euler, τις συντεταγμένες στο τρισδιάστατο χώρο και ένα flag επίτρεψης αποδοχής εντολών.

Η προκαθορισμένη διαδρομή μπορεί να γίνει βάσει των αλλαγών στις καρτεσιανές συντεταγμένες στο χώρο. Αρχικά, πραγματώνεται μια προγραμματισμένη διαδρομή, δίνοντας εντολή στο quadcopter να μετατοπιστεί σε μια τελική θέση, σύμφωνα με το καρτεσιανό σύστημα συντεταγμένων, ανεξαρτήτως του τρόπου μετάβασής του σε αυτή. Αυτό υλοποιείται ορίζοντας ως αρχική θέση (X = 0 , Y = 0), τη θέση απογείωσής του ή τη θέση της κάμερας βάσει των landmarks εαν αυτά ανιχνευτούν. Για να μεταβεί το quadcopter στη τελική θέση που είναι και ο προορισμός του χρησιμοποιεί τον optical flow για να μπορέσει να υπολογίσει την απόσταση που διανύει. Σε αυτή την περίπτωση δηλαδή η κάμερα, και κατ' επέκταση η λειτουργία του optical flow, είναι η ανάδραση του συστήματος, όπου μέσω αυτής το quadcopter καταφέρνει να μεταβεί στο τελικό προορισμό του με επιτυχία. Θέτοντας πολλά σημεία 'στόχους' μπορεί το quadcopter να διαγράψει οποιαδήποτε πορεία θέλει ο χρήστης, δίνοντας απλά τις συντεταγμένες του κάθε σημείου στο καρτεσιανό επίπεδο.

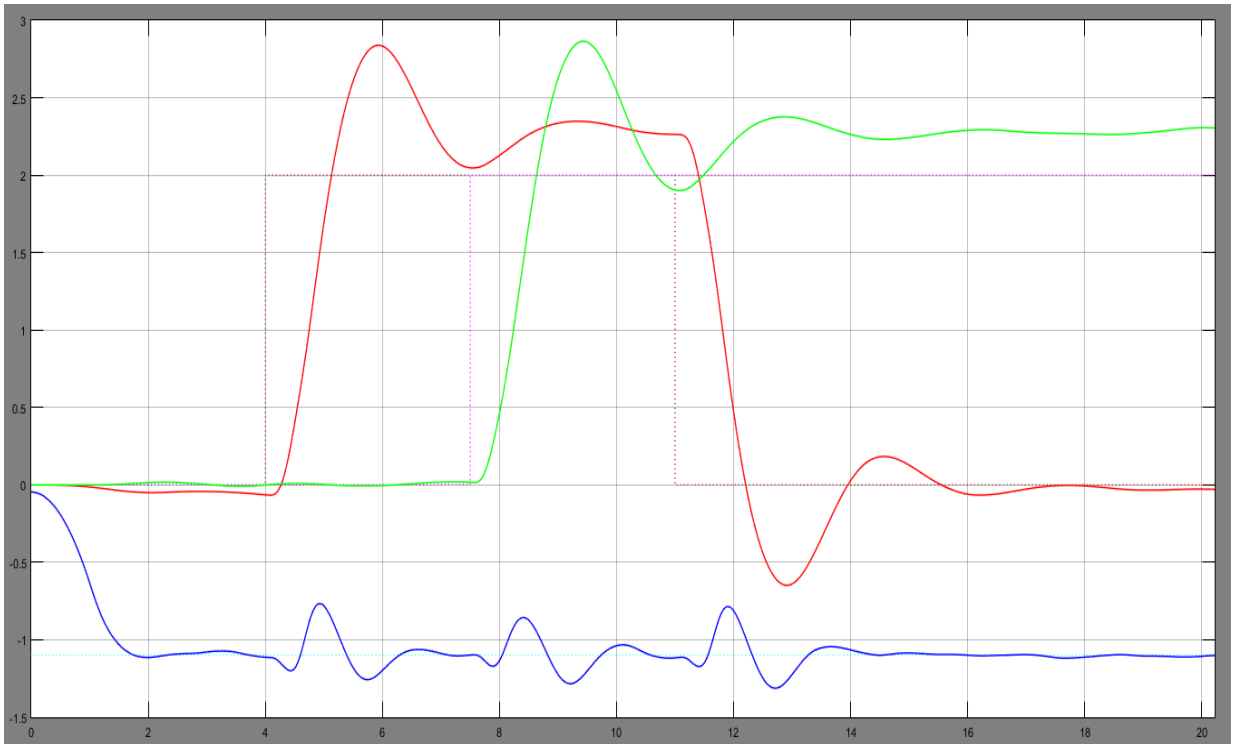
Για τη φόρτωση αυτών των εντολών στο quadcopter, μετά την αποστολή των τιμών των συντεταγμένων στον κώδικα rsedu_control.c, μπαίνουν στην είσοδο του Drone_Compensator_U_pos_refin του Drone_Compensator ([0]:x, [1]:y, [2]:z).

Στο συγκεκριμένο πείραμα δίνονται τρείς στόχοι στο quadcopter με το πάτημα του πλήκτρου 'z' και με συντεταγμένες:

- Σ1 (X = 2 , Y = 0) τη χρονική στιγμή που θα πατηθεί το πλήκτρο 'z'.
- Σ2 (X = 2 , Y = 2) έπειτα από t = 3.5 sec
- Σ3 (X = 0 , Y = 2) έπειτα από t = 3.5 sec

Γίνεται η προσπάθεια δηλαδή το quadcopter να διαγράψει πορεία σχήματος ' Π '.

Στην αρχή το συγκεκριμένο πείραμα εκτελείται σε περιβάλλον εξομοίωσης για την αντιμετώπιση τυχόν ανεπιθύμητων συμπεριφορών από το quadcopter. Γίνεται χρήση βηματικών εισόδων τις χρονικές στιγμές στις οποίες θέλει ο χρήστης το όχημα να μεταβεί στους στόχους που του έχουν τεθεί.

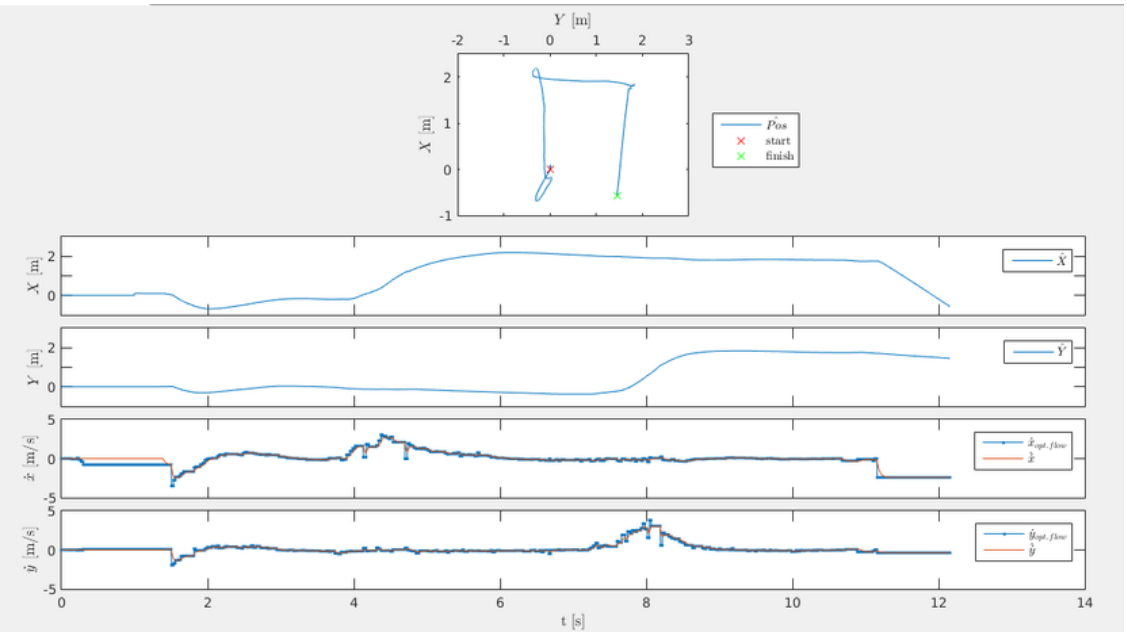


Εικόνα 6.3.1 Αποτέλεσμα εξομοίωσης της θέσης του quadcopter έπειτα από βηματκές εισόδους

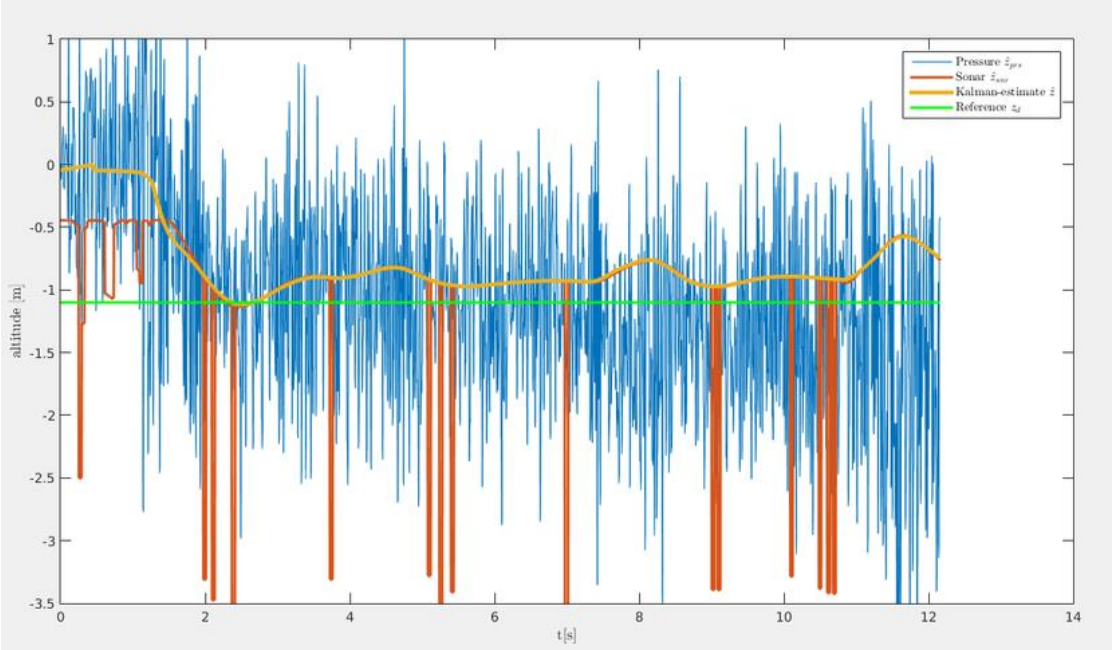
Είναι λογικό να υπάρχει καθυστέρηση στην επίτευξη της επιθυμητής τιμής κυρίως λόγω αδράνειας της μάζας του οχήματος τόσο στην αρχή της κίνησής του όσο και στο τέλος της. Όπως φαίνεται στη παραπάνω γραφική παράσταση οι διακεκομένες γραμμές είναι οι βηματικές είσοδοι καθώς και οι επιθυμητές τιμές του συστήματος. Η κόκκινη καμπύλη είναι η μετατόπισή του στον άξονα X, η πράσινη καμπύλη είναι η μετατόπισή του στον άξονα Y ενώ η μπλέ καμπύλη ειναι η μετατόπισή του στον άξονα Z.

Παρατηρείται ότι κάθε φορά που εισερχεται στο σύστημα μια είσοδος υπάρχει μια μικρή ταλάντωση στον άξονα Z, γιατί για να μεταβεί σε αυτές τις θέσεις το όχημα θα πρέπει να προβεί σε κινήσεις που επηρεάζουν τις γωνίες Euler, άρα κατά συνέπεια και το ύψος.

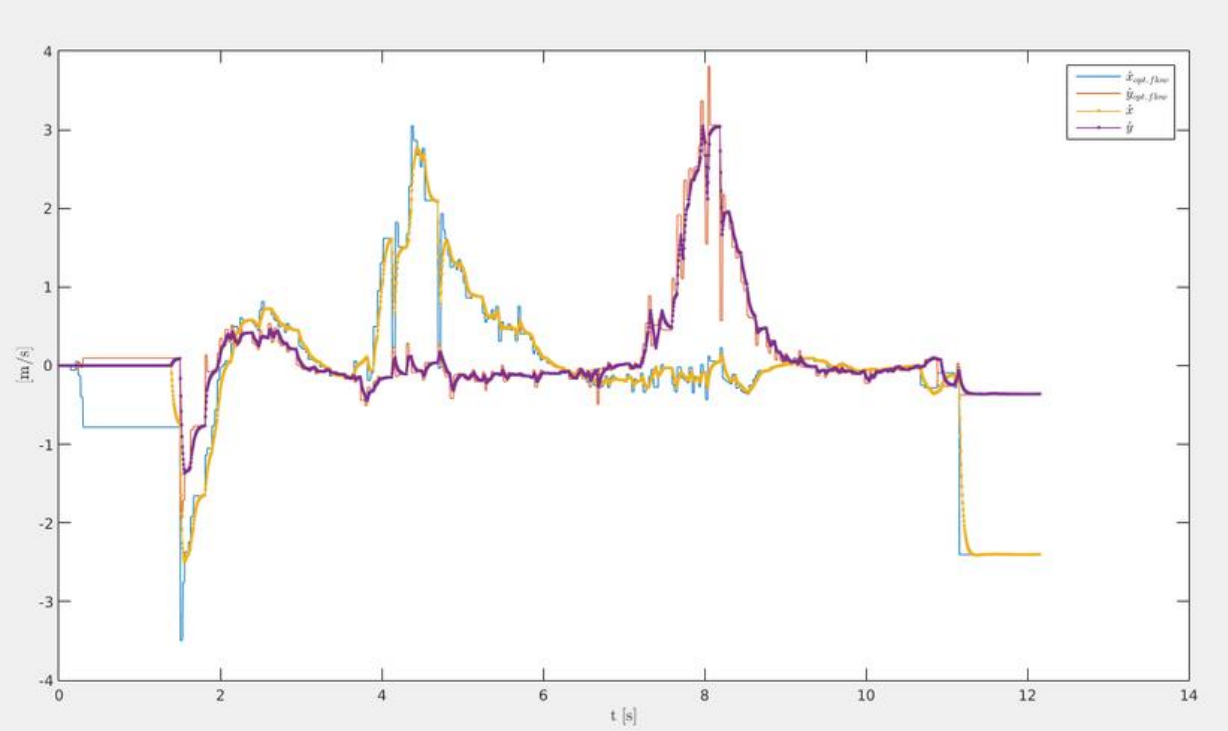
Μετά το πέρας της πτήσης γίνεται η λήψη των δεδομένων, όπως φαίνεται στις παρακάτω εικόνες.



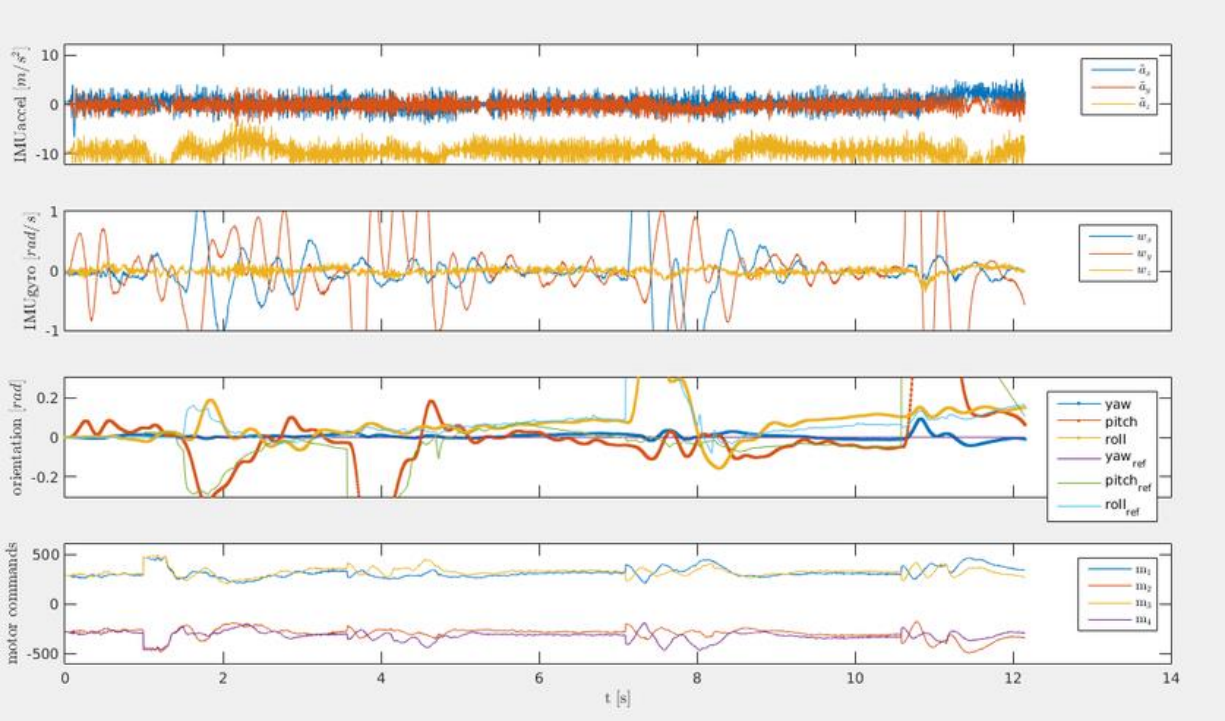
Εικόνα 6.3.2 Μετατόπιση και ταχύτητα στον άξονα X,Y



Εικόνα 6.3.3 Ύψος, μετρήσεις αισθητηρίων πίεσης και sonar και εκτίμηση Kalman



Εικόνα 6.3.4 Ταχύτητες στον X, Y άξονα και εκτίμηση ταχύτητας από τον optical flow



Εικόνα 6.3.5 Αισθητήρια IMU, προσανατολισμός και εντολές κινητήρων

Με το πάτημα του πλήκτρου ‘ f ’ μπορεί ο χρήστης κατά την διάρκεια της πτήσης να εισάγει τις συντεταγμένες X, Y εντός των ορίων (-5, +5) . Έτσι δίνεται στο χρήστη η πλήρης ελευθερία να εισάγει τη διαδρομή που εκείνος επιθυμεί.

6.4. Εκτέλεση Ακολουθίας Κινήσεων

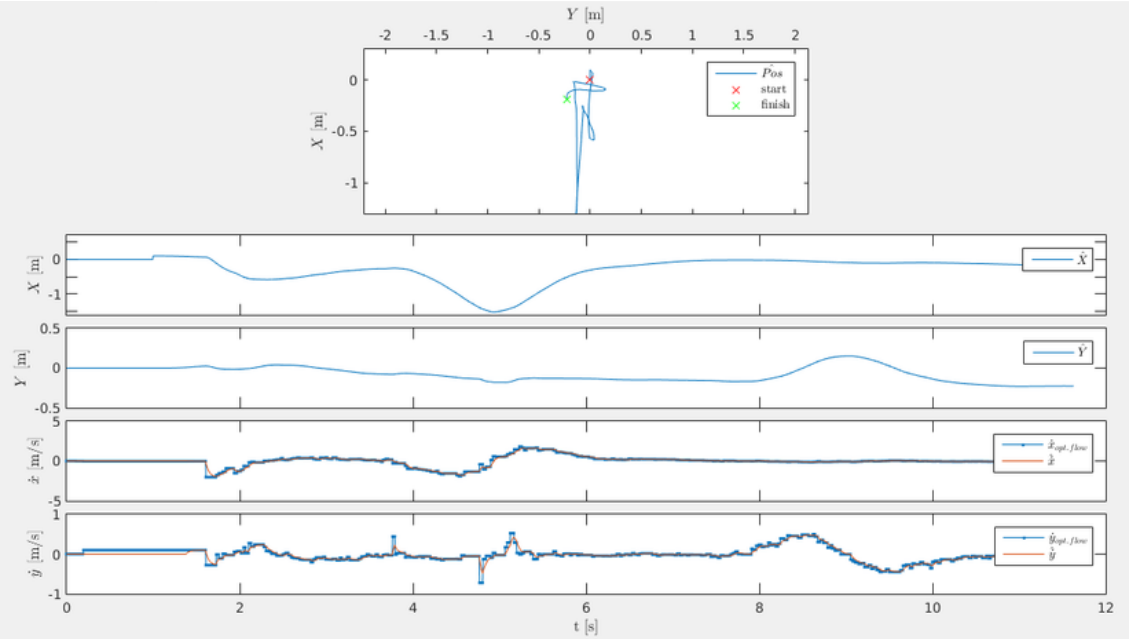
Σε αυτό το πείραμα εκτελείται μια ακολουθία κινήσεων, και συγκεκριμένα μια ακολουθία μεταβολών των γωνιών Euler σε διαφορετικές χρονικές στιγμές. Η ενέργεια αυτή εκτελείται επίσης με την αποστολή μόνο ενός σήματος από το πληκτρολόγιο από τον χρήστη μέσω του DroneKeyboardPilot.sh.

Εντός του ReferenceValueServer.c, δημιουργείται η ακολουθία όπου με το πάτημα του πλήκτρου ‘x’ γίνεται η εκτέλεση της διαδικασίας:

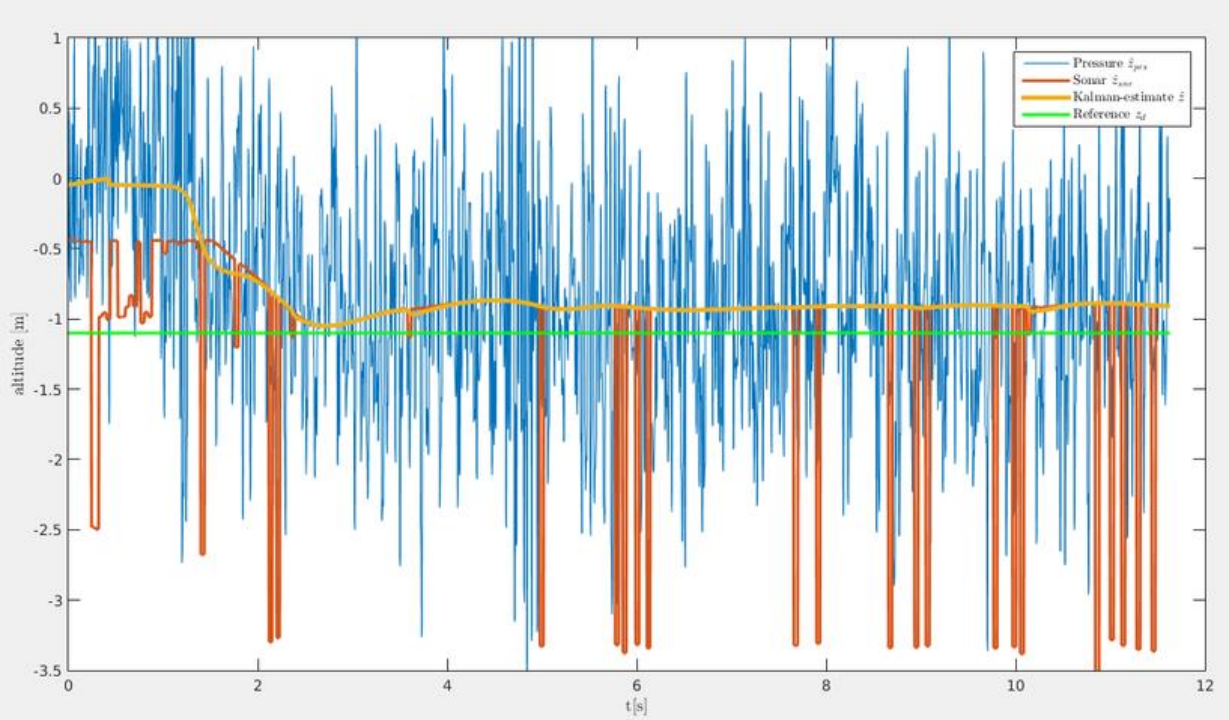
- Pitch = 0.2 για t = 1 sec
- Hover για t = 3 sec
- Roll = 0.1 για t = 1 sec
- Hover για t = 3 sec

Οι εντολές αυτές αποστέλλονται μέσω του socket στο κώδικα rsedu_control.c. Οι τιμές τους μπαίνουν στην είσοδο του Drone_Compensator_U_orient_refin που υλοποιήθηκε από το Simulink ([0]:yaw, [1]:pitch, [2]:roll).

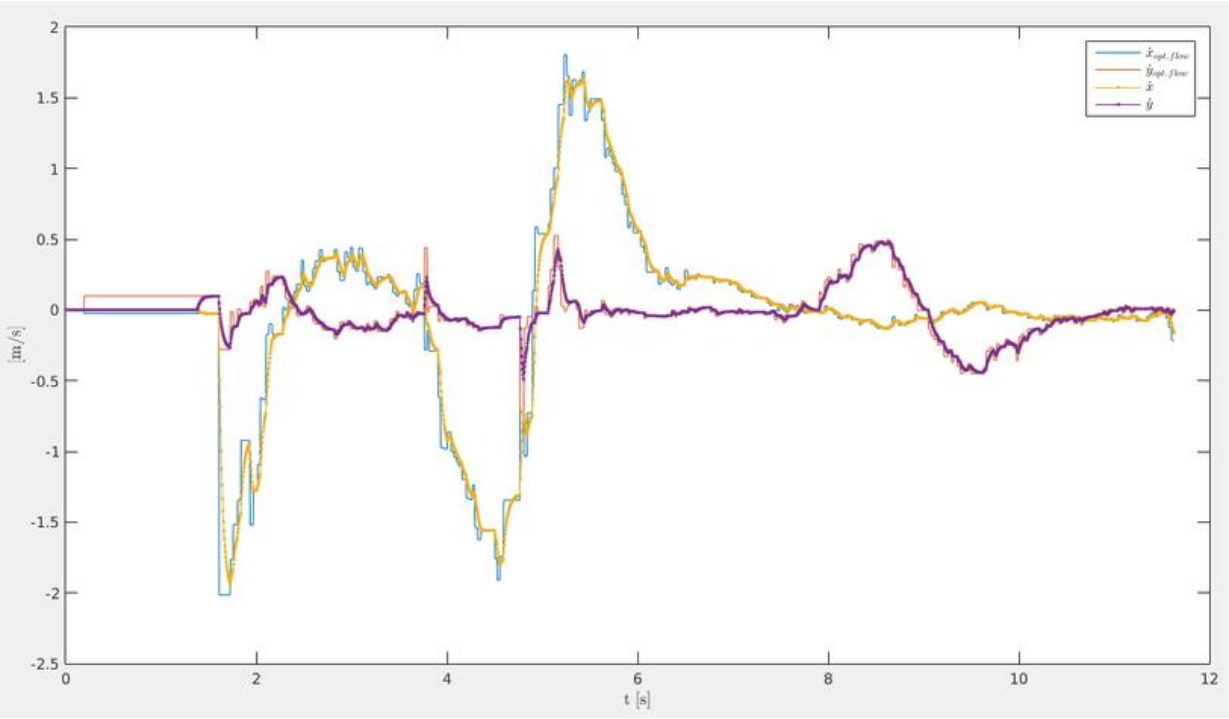
Τα αποτελέσματα μετά το πέρας της πτήσης είναι:



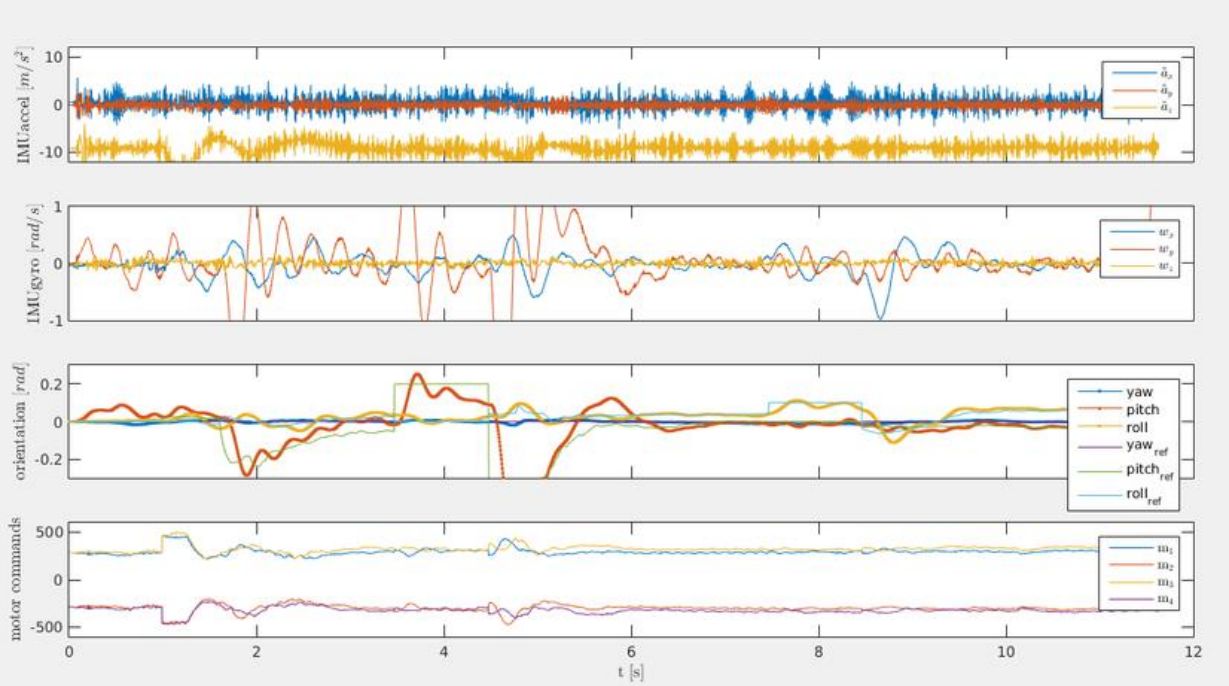
Εικόνα 6.4.1 Θέσεις στον X, Y άξονα και ταχύτητες



Εικόνα 6.4.2 Ύψος, μετρήσεις αισθητηρίων πίεσης και sonar και εκτίμηση Kalman



Εικόνα 6.4.3 Ταχύτητες στον X, Y άξονα και εκτίμηση ταχύτητας από τον optical flow

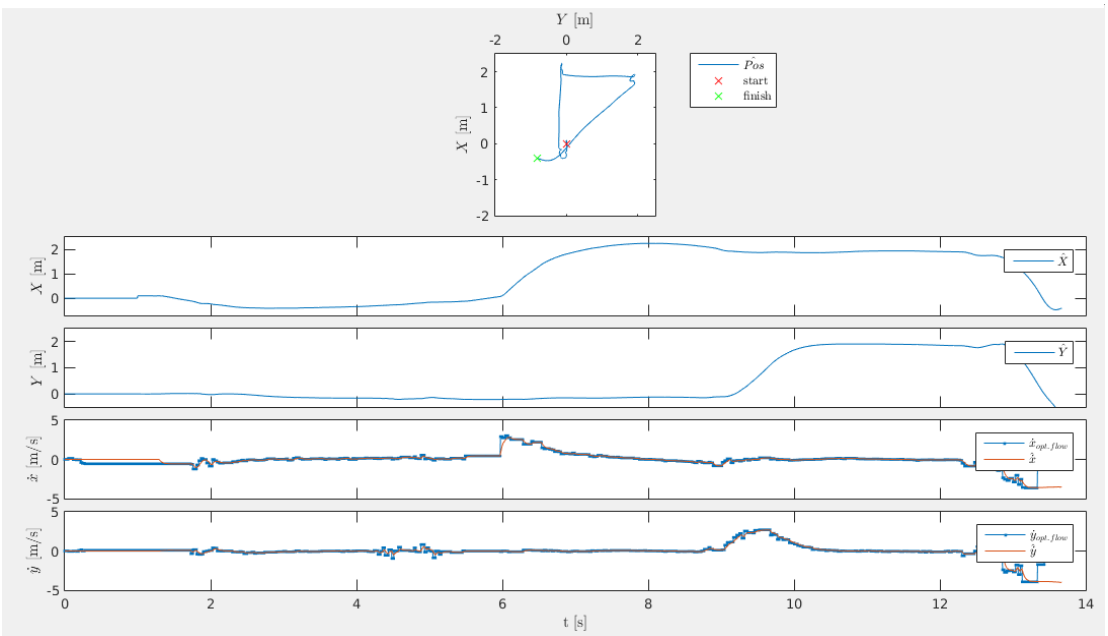


Εικόνα 6.4.4 Αισθητήρια IMU, προσανατολισμός και εντολές κινητήρων

6.5. Επιστροφή στην Αρχική Θέση

Με την ίδια λογική, σε οποιαδήποτε θέση βρίσκεται το quadcopter, μπορεί να επιστρέψει στην αρχική του θέση θέτοντας εντός του ReferenceValueServer.c στο πλήκτρο ‘q’ τιμές αναφοράς:

- X = 0
- Y = 0
- Yaw = 0



Εικόνα 6.5.1 Παράδειγμα επιστροφής στην αρχική θέση μετά απο κίνηση σχήματος ‘Γ’

Κεφάλαιο 7: Επίλογος

7.1. Προβλήματα που Συναντήθηκαν

Κατά την διάρκεια της έρευνας της παρούσας πτυχιακής, αντιμετωπίσαμε πολλά προβλήματα όσο αναφορά το υλικό που χρησιμοποιήθηκε αλλά και το λογισμικό.

Αρχικά, ένα από τα σημαντικότερα προβλήματα ήταν η προσπάθεια υλοποίησης του συνολικού project εντός ενός custom λογισμικού Linux Ubuntu 16.30 που διατείθεται ελεύθερα. Αυτό δημιουργήθηκε στο MIT με σκοπό την αποφυγή εγκατάστασης όλων των απαραίτητων βιβλιοθηκών που απαιτούνται για την λειτουργία της προκείμενης εργασίας, καθώς παρέχει και όλα τα απαραίτητα αρχεία, κώδικες και αρχιτεκτονική φακέλων που χρειάζεται. Η λειτουργία αυτού του λογισμικού γίνεται εντός του VMware. Το VMware είναι ένα πρόγραμμα που ‘εικονοποιεί’ ένα λογισμικό, στην περιπτωσή μας το Ubuntu 16.30, δηλαδή δεν απαιτείται η εγκατάσταση του εκάστοτε λογισμικού εντός του υπολογιστή για να ‘τρέξει’. Η δυσκολία που αντιμετωπίσαμε ήταν ότι, παρότι είχαμε πρόσβαση στο συγκεκριμένο λογισμικό μέσω VMware, δεν ήταν δυνατή η εγκατάσταση του Matlab λόγω ανεπαρκή χώρου στο δίσκο. Το αποτέλεσμα ήταν να αφήσουμε την συγκεκριμένη διαδικασία και να εγκαταστήσουμε από την αρχή τα Ubuntu 14.04 και να κάνουμε εμείς όλες τις απαραίτητες εγκαταστάσεις βιβλιοθηκών και τις ρυθμίσεις που χρειάζονταν. Αν και χάσαμε αρκετό χρόνο από αυτό, επωφεληθήκαμε, καθώς κατανοήσαμε σε μεγαλύτερο βαθμό την λειτουργία των διαδικασιών που γίνονται στο backround.

Επίσης, πολλές φορες κατά την σύνδεση του υπολογιστή με το quadcopter μέσω της διεργασίας DroneConnect.sh, παρατηρήσαμε οτι ενώ δεχόμασταν μήνυμα από το terminal πως η σύνδεση υπάρχει, δεν είχαμε επικοινωνία με quadcopter. Αυτό οφειλόταν στην σύνδεση του υπολογιστή με το Internet μέσω Wifi. Σε συγκεκριμένα δίκτυα Wifi, η IP του quadcopter μπερδεύοταν με του router, με αποτέλεσμα να νομίζει ο υπολογιστής οτι συνδέθηκε. Η λύση αυτού ήταν απλή αποσύνδεση από το Internet.

Εντός του περιβάλλοντος εξομοίωσης, πολλές φορές χρειάστηκε η αλλαγή κάποιων ονομάτων εισόδων και εξόδων των δομικών διαγραμμάτων ή η δημιουργία νέων. Η αλλαγή αυτή απαιτούσε και ρύθμισή τους εντός του κώδικα C, καθώς η σχέση εξομοιωσης στο Simulink και του κώδικα C είναι αλληλοεξαρτώμενη. Έτσι, σε κάθε αλλαγή τους, χρειαζόταν αλλαγή της δήλωσής τους εντός του κώδικα ert_main.c και rsedu_control.c.

Στο κεφάλαιο 6, αναλύθηκε η χρήση των landmarks και πώς το quadcopter μπορεί να τα ανιχνεύσει και να ελεγχθεί βάσει των θέσεων τους. Η διάταξη τους αλλά και το χρώμα τους είναι πολύ συγκεκριμένα και η αλλαγή τους μπορεί να οδηγήσει σε ανεπιθύμητες συμπεριφορές, εκτός και αν αλλάξουμε τις παραμέτρους τους, εντός του κώδικα main_offboard_image.c για τα χρώματα και του κώδικα rsedu_vis_helpers.c για τις τιμές των συντεταγμένων των θέσεων των χρωμάτων. Κατά τη πτήση, στη προσπάθεια ανίχνευσης των landmarks, παρατηρήσαμε ότι το landmark με το χρώμα πράσινο δεν ανιχνευόταν ιδιαίτερα σε σχέση με τα υπόλοιπα. Ο λόγος ήταν η φωτεινότητα του, καθώς τα χρώματα μετατρέπονταν στη μορφή HSV ή HSL (οπού L = luminosity, δηλαδή φωτεινότητα). Η έντονη φωτεινότητα αυτού του χρώματος το έκανε να μην διαφέρει πολύ από το λευκό του εδάφους. Για το λόγο αυτό, η λειτουργία των landmarks πολλές φορές δεν έχει τα επιθυμητά αποτελέσματα.

Ο αλγόριθμος optical flow στην εξομοίωση (Drone_Sensorsystem/camera) δεν ακολουθεί την λογική του πραγματικού optical flow που υλοποιείται εντός του quadcopter. Απέχει πολύ από την πραγματικότητα, και ο λόγος είναι ότι προσπαθούμε να κάνουμε απλώς μια εξομοίωση της λειτουργίας του και όχι ακριβής αναπαράσταση όλων των αλγορίθμων που υλοποιούνται. Ένας ακόμα λόγος είναι ότι δεν έχουμε καθόλου πρόσβαση στον optical flow. Ότι γίνεται στον αλγόριθμο συμβαίνει εντός του quadcopter και μπορούμε μόνο να πάρουμε δεδομένα όπως τις ταχύτητες vx, vy, vz. Στο Παράρτημα 1, που δίνεται ένα παράδειγμα του optical flow είναι για να εξηγηθεί απλά η λειτουργία του και δεν αντιπροσωπεύει τον αλγόριθμο που υλοποιείται εδω.

Σχετικά με την πτήση και την συλλογή δεδομένων-εικόνων online, εξηγήθηκε σε προηγούμενα κεφάλαια ότι η επικοινωνία αυτή υλοποιείται με τη χρήση fifo pipes. Ουσιαστικά, μεταφέρονται οι εικόνες σε ένα fifo pipe με το όνομα /tmp/picture/, αποστέλλονται στον

υπολογιστή και μεταφέρονται σε ένα καινούργιο fifo pipe με το όνομα /tmp/rollingspiderpicture/. Πολλές φορές, ακολουθώντας την διαδικασία πτήσης και λήψης εικόνων, λαμβάναμε μόνο το πρώτο frame της κάμερας του quadcopter και ενώ η επικοινωνία συνεχιζόταν να υπάρχει, δεν λαμβάναμε άλλες εικόνες. Ένας πιθανός λόγος που συμβαίνει αυτό είναι διαφορά συχνοτήτων των δύο συστημάτων, του quadcopter και του υπολογιστή. Δεν μπορούσαν να συγχρονιστούν τα δύο συστήματα παρά μόνο στη αρχή της επικοινωνίας, και λόγω της λογικής των fifo pipe (First In – First Out), οι εικόνες 'στοιβάζονταν' η μία πάνω στην άλλη μέσα στο pipe.

Όσο αφορά το υλικό του quadcopter, αντιμετωπίσαμε ιδιαίτερη δυσκολία με την διάρκεια πτήσης που μας παρείχε η μπαταρία. Άν και στο datasheet του Rolling Spider ανέφερε 7-8 λεπτά πτήσης, στην πραγματικότητα είχαμε μέγιστο χρόνο πτήσης 4 λεπτά. Αυτό οφείλεται στον μεγάλο όγκο υπολογισμών από τον επεξεργαστή εντός του quadcopter. Η λύση αυτού δεν ήταν εύκολη καθώς δύσκολα μπορούμε να μειώσουμε τον φόρτο εργασίας του επεξεργαστή, και δεν υπάρχει η οικονομική δυνατότητα αγοράς περισσότερων μπαταριών λόγω του αυξημένου κόστους. Αυτό που κάναμε εμείς ήταν να μειώσουμε το ποσοστό της μπαταρίας που απομένει για να σταματήσει η πτήση από 50% σε 30%. Μεγαλύτερη μείωση θα δημιουργούσε προβλήματα στην πτήση και ανεπιθύμητη συμπεριφορά. Η αλλαγή αυτή έγινε εντός του κώδικα rsedu_control.c στις δηλώσεις των μεταβλητών:

static float MIN_BATTTAKEOFF = 30.0
static float MIN_BATT = 30.0.

Κατά τη διάρκεια της πτήσης, και σε περιπτώσεις απλού hover, παρατηρήθηκε βίαιη αύξηση ύψους χωρίς να βρεθεί κάποιο αντικείμενο μπροστά στο αισθητήριο του sonar. Αυτό οφείλεται στις μεγάλες παρεμβολές των υπερηχητικών κυμάτων που εκπέμπει το sonar, καθώς δεν θεωρείται πολύ αξιόπιστο αισθητήριο. Μια λύση αυτού του προβλήματος είναι η δημιουργία υποσυστήματος εναλλαγής από το sonar σε αισθητήριο πίεσης. Όταν το sonar αρχίζει και διαβάζει μεγάλες αυξομειώσεις ύψους, άρα θεωρείται αναξιόπιστο, θα γίνεται αλλαγή λειτουργίας σε αισθητήριο πίεσης.

7.2. Μελλοντική Έρευνα

Το σύστημα ελέγχου που αναλύθηκε σε αυτή την πτυχιακή μπορεί να υλοποιηθεί και με LQR (Linear Quadratic Regulator), μια διαφορετική μέθοδος ελεγκτή σε σχέση με τον κλασσικό έλεγχο PID που υλοποιήσαμε εμείς.

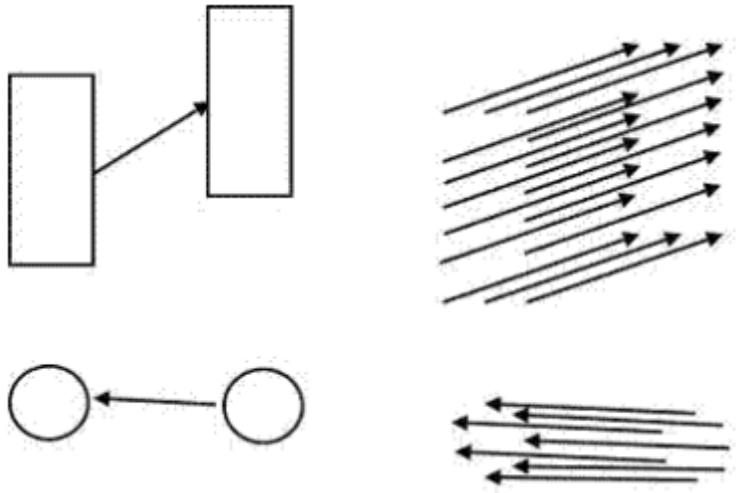
Σε περίπτωση που αντιμετωπιζόταν το πρόβλημα αναγνώρισης του πράσινου χρώματος θα ήταν εφικτή η σταθερή αιώρηση του quadcopter πάνω από κάθε landmark. Θα μπορούσε ο χρήστης πατώντας ένα πλήκτρο από το πληκτρολόγιο με κώδικα που έχουμε ενσωματώσει στο ReferenceValueServer.c να επιλέξει το χρώμα του landmark που επιθυμεί για να κάνει hover το quadcopter πάνω από αυτό.

Σε πιο προχωρημένο επίπεδο, αν μας το επέτρεπε ο προυπολογισμός, θα μπορούσαμε να εφαρμόσουμε αλγόριθμους, όπως το Extended Kalman Filter (EKF), την οικογένεια αλγορίθμων PREMONN (PREdictive Modular Neural Networks), γενετικούς αλγορίθμους κ.λ.π, για την υλοποίηση του S.L.A.M. (Simultaneous Localization and Mapping). Ο λόγος που δεν μπορέσαμε να προχωρήσουμε παραπάνω σε αυτό το κομμάτι είναι χαμηλή επεξεργαστική ισχύς που μας δίνει ο επεξεργαστής, η έλλειψη μνήμης για την αποθήκευση τόσου μεγάλου όγκου δεδομένων και ο πολύ μικρός χρόνος πτήσης λόγω μπαταρίας. Με το τρόπο αυτό το quadcopter θα ήταν εξ΄ολοκλήρου αυτόνομο αφού θα μπορούσε να κατασκευάζει τον χάρτη του περιβάλλοντος που βρίσκεται εντελώς μόνο του. Ο χρήστης, το μόνο που θα χρειαζόταν να κάνει, είναι να δώσει τον τελικό προορισμό. Εδώ να αναφερθεί ότι, με τη χρήση κάποιου αλγορίθμου όπως είναι ο A*, το quadcopter θα μπορούσε να σχεδιάσει μόνο του τις πιθανές πορείες που θα ακολουθήσει και μάλιστα την πιο σύντομη από αυτές, αποφεύγοντας τη σύγκρουση με κάθε εμπόδιο που υπάρχει στο χώρο.

Παράρτημα 1- Αλγορίθμοι

Optical flow

Ο optical flow είναι μεγάλο κομμάτι της μηχανικής όρασης και της επιστήμης των υπολογιστών γενικότερα. Αποτελεί έναν αλγόριθμο ο οποίος χρησιμοποιείται κυρίως για την εύρεση ταχυτήτων των αντικειμένων, των επιφανειών ή των ακμών, άρα τον υπολογισμό της σχετικής κίνησης μεταξύ παρατηρητή και περιβάλλοντος, αναλύοντας ένα video σε εικόνες (frames).



Εικόνα Π.1.1 Διανύσματα Optical Flow

Ο υπολογισμός αυτός γίνεται συγκρίνοντας τα pixels δύο διαδοχικών εικόνων τις στιγμές t_1 και t_2. Η βασική ιδέα είναι ότι κατά την κίνηση του αντικειμένου ανάμεσα σε αυτά τα δύο frames, η φωτεινότητα του και η αντανάκλαση του φωτός στο αντικείμενο δεν αλλάζουν. Αυτό μαθηματικά αναλύεται ως:

$$f(x + \Delta x, y + \Delta y, t + \Delta t) \approx f(x, y, t)$$

όπου f(x, y, t) είναι η ένταση της εικόνας στην θέση (x,y) την στιγμή t, και Δx, Δy η αλλαγή της θέσης και Δt η αλλαγή στον χρόνο.

Στη συνέχεια, γίνεται χρήση της σειράς Taylor. Αυτή η μαθηματική ακολουθία είναι η αναπαράσταση μιας συνάρτησης ως άθροισμα απείρων όρων οι οποίοι υπολογίζονται από τις τιμές των παραγώγων της σε ένα συγκεκριμένο σημείο. Αυτό μαθηματικά ορίζεται ως:

$$f(a) + \frac{f'(a)}{1!}(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \frac{f'''(a)}{3!}(x - a)^3 + ...$$

ή

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x - a)^n$$

Με τη χρήση της σειράς Taylor, η εξίσωση γίνεται:

$$f(x + \Delta x, y + \Delta y, t + \Delta t) \approx f(x, y, t) + \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y + \frac{\partial f}{\partial y}\Delta y + h.o.t.$$

προσθέτοντας δηλαδή της μερικές παραγώγους της συνάρτησης του x, y, t και τους όρους υψηλής προτεραιότητας (high order terms) οι οποίοι αγνοούνται. Συνεπάγεται:

$$\nabla I \bullet v + I_t = 0$$

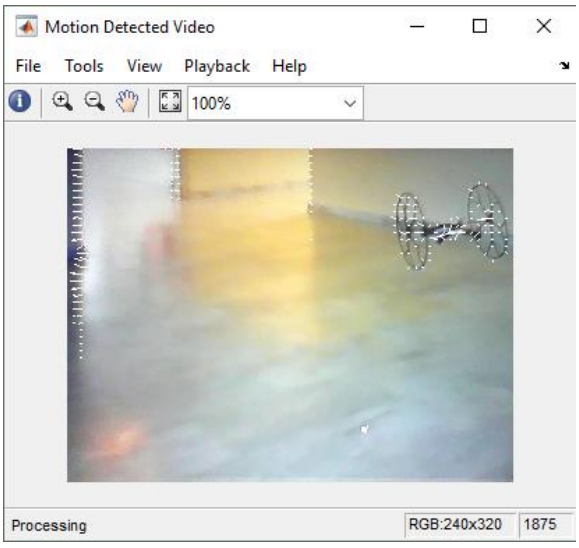
όπου $\nabla I = (I_x, I_y)$ και $v = (u, \upsilon) = (\Delta x, \Delta y)$ όπου είναι το διάνυσμα του optical flow. Το διάνυσμα $v = (u, \upsilon)$ είναι αυτό που αναζητάται.

Αν και η προηγούμενη διαδικασία είναι αρκετά χρήσιμη ως προς την εύρεση της ταχύτητας, περιορίζεται μόνο σε ένα διάνυσμα το οποίο δημιουργεί προβλημα σε καταστάσεις που σε ένα σημείο υπάρχουν και άλλα διανύσματα, όπως π.χ. σε μια γωνία. Αυτό επιλύεται με διάφορες μεθόδους. Μια από τις πιο διαδεδομένες είναι η Horn-Schunck, βασική αρχή της οποίας είναι η ομαλοποίηση των διανυσμάτων σε σχέση με την όλη εικόνα.

Εδώ πρέπει να επισημανθεί ότι ο optical flow δεν υλοποιείται ακριβώς με αυτή την διαδικασία μέσα στο Simulink της εξομοίωσης, όπως και τα υπόλοιπα αισθητήρια. Η ακριβής υλοποίηση αυτού του αλγορίθμου γίνεται εντός του quadcopter. Επειδή λοιπόν, η λειτουργία του δεν εξηγείται πουθενά ενώ η σημασία του στη συγκεκριμένη εργασία είναι πολύ μεγάλη παρακάτω δίνεται ένα παράδειγμα χρήσης αυτού του αλγόριθμου για τον εντοπισμό της κίνησης σε εικόνες που λαμβάνονται από τη κάμερα του υπολογιστή για να κατανοηθεί παραπάνω ο τρόπος λειτουργίας του.

Κώδικας Παραδείγματος Optical-Flow:

```
%%Αρχικοποίηση και ιδιότητες κάμερας
vidDevice = imaq.VideoDevice('winvideo', 2, 'YUY2_320x240', ...
                             'ROI', [1 1 320 240], ...
                             'ReturnedColorSpace', 'rgb', ...
                             'DeviceProperties.Brightness', 64, ...
                             'DeviceProperties.Sharpness', 5);
%% Εντοπισμός κατεύθυνσης και ταχύτητας του κινούμενου αντικειμένου
optical = vision.OpticalFlow( ...
    'OutputValue', 'Horizontal and vertical components in complex form');
%% Παράμετροι και μορφοποίηση κάμερας
maxWidth = imaqhwinfo(vidDevice,'MaxWidth');
maxHeight = imaqhwinfo(vidDevice,'MaxHeight');
shapes = vision.ShapeInserter;
shapes.Shape = 'Lines';
shapes.BorderColor = 'white';
r = 1:5:maxHeight;
c = 1:5:maxWidth;
[Y, X] = meshgrid(c,r);
%%Δημιουργία VideoPlayer για απεικόνηση video
hVideoIn = vision.VideoPlayer;
hVideoIn.Name  = 'Original Video';
hVideoOut = vision.VideoPlayer;
hVideoOut.Name  = 'Motion Detected Video';
%% Λήψη και επεξεργασία video
nFrames = 0;
while (nFrames<600)
    rgbData = step(vidDevice);
    % Υπολογισμός opticalflow για τη συγκεκριμένη εικόνα
    optFlow = step(optical,rgb2gray(rgbData));
    optFlow_DS = optFlow(r, c);
    H = imag(optFlow_DS)*50;
    V = real(optFlow_DS)*50;
    lines = [Y(:)'; X(:)'; Y(:)'+V(:)'; X(:)'+H(:)'];
    rgb_Out = step(shapes, rgbData,  lines');
    step(hVideoIn, rgbData);
    step(hVideoOut, rgb_Out);
    nFrames = nFrames + 1;
end
% Εμφάνιση video με τα αποτελέσματα του optical flow στο videoPlayer του Matlab
release(vidDevice);
release(hVideoIn);
release(hVideoOut);
displayEndOfDemoMessage(mfilename)
```



Εικόνα Π.1.2 Αποτέλεσμα παραδείγματος optical flow

Φίλτρο Kalman

Το φίλτρο Kalman είναι ένας αλγόριθμος ο οποίος χρησιμοποιείται για την εκτίμηση άγνωστων μεταβλητών από μια σειρά μετρήσεων όπου εμπεριέχεται θόρυβος και άλλες ανακρίβειες. Το φίλτρο πήρε το όνομά του από τον Rudolf E. Kálmán, έναν από τους βασικούς κατασκευαστές της θεωρίας του. Η εκτίμηση των άγνωστων αυτών μεταβλητών που εξάγονται από το φίλτρο περιέχουν μεγαλύτερη ακρίβεια σε σχέση με την εκτίμηση που έχει γίνει απο τη μέτρησή τους. Το φίλτρο Kalman είναι η πλέον διαδεδομένη μέθοδος υλοποίησης του φίλτρου Bayes. Χρησιμοποιείται για φιλτράρισμα, για πρόγνωση, για την εκτίμηση και την εξαγωγή συμπεράσματος για τις άγνωστες μεταβλητές κάθε χρονική στιγμή σε γραμμικά Γκαουσιανά συστήματα, γι'αυτό ονομάζεται Γκαουσιανό φίλτρο.

Η λειτουργία αυτού του αλγορίθμου χωρίζεται σε δύο στάδια . Το πρώτο στάδιο αφορά τη πρόβλεψη. Το φίλτρο παράγει εκτιμήσεις για τις υπάρχουσες τιμές των μεταβλητών κατάστασης μαζί με τις αβεβαιότητες που τις διέπουν. Το δεύτερο στάδιο αφορά τη διαδικασία της επόμενης μέτρησης, η οποία είνα αδύνατο να μη περιέχει σφάλματα και θόρυβο. Η πρόβλεψη ανανεώνεται χρησιμοποιώντας ένα σταθμισμένο μέσο όρο δίνοντας προτεραιότητα στις μετρήσεις με τη μεγαλύτερη βαρύτητα, δηλαδή σε αυτές που η τιμή τους έχουν μικρότερη διαφορά από τη μέση τιμή. Μπορεί να λειτουργήσει σε πραγματικό χρόνο, χρησιμοποιώντας μόνο τις μετρήσεις εισόδου της παρούσας χρονικής στιγμής και την υπολογισμένη μέτρηση της προηγούμενης χρονικής στιγμής μαζί με τον πίνακα αβεβαιότητας της, δηλαδή δεν απαιτούνται πρόσθετες πληροφορίες από όλες τις προηγούμενες μετρήσεις του παρελθόντος.

Το φίλτρο Kalman λειτουργεί επίσης για τη μοντελοποίηση ελέγχου. Λόγω της χρονικής καθυστέρησης μεταξύ της αποστολής εντολών στους κινητήρες και της λήψης μετρήσεων από τα αισθητήρια στην ανάδραση, η χρήση του φίλτρου Kalman βοηθά στη δημιουργία ενός ρεαλιστικού μοντέλου εκτιμήσεων για την υπάρχουσα κατάσταση του συστήματος του κάθε κινητήρα.

Algorithm Kalman_filter$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

$\bar{\mu}_t = A_t\, \mu_{t-1} + B_t\, u_t$

$\bar{\Sigma}_t = A_t\, \Sigma_{t-1}\, A_t^T + R_t$

$K_t = \bar{\Sigma}_t\, C_t^T (C_t\, \bar{\Sigma}_t\, C_t^T + Q_t)^{-1}$

$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\, \bar{\mu}_t)$

$\Sigma_t = (I - K_t\, C_t)\, \bar{\Sigma}_t$

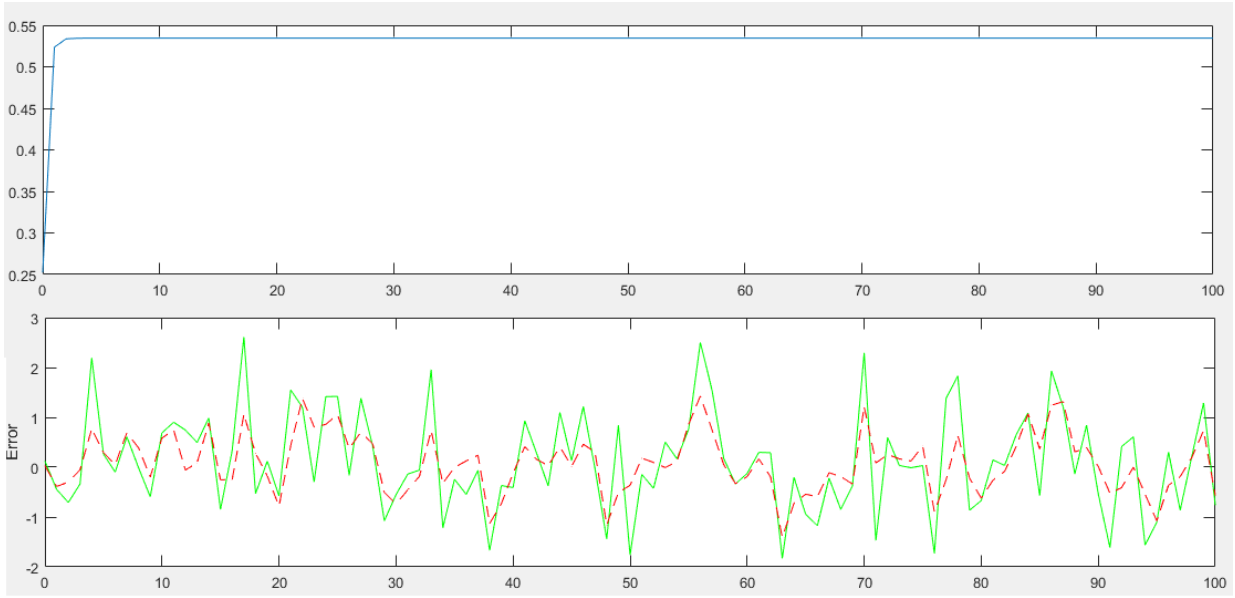
return μ_t, Σ_t

Εικόνα Π.1.3 Αλγόριθμος Kalman

Κώδικας Παραδείγματος Φίλτρου Kalman:

```
%Είσοδοι συστήματος
A = [1.1269   -0.4940    0.1129,
     1.0000         0         0,
          0    1.0000         0];

B = [-0.3832
      0.5919
      0.5191];

C = [1 0 0];

D = 0;
Plant = ss(A,[B B],C,0,-1,'inputname',{'u' 'w'},'outputname','y');

% Παράμετροι συστήματος
R = 1;
Q=2;
%Εκτέλεση φίλτρου KALMAN
[kalmf,L,~,M,Z] = kalman(Plant,Q,R);
kalmf = kalmf(1,:);

a = A;
b = [B B 0*B];
c = [C;C];
d = [0 0 0;0 0 1];
P = ss(a,b,c,d,-1,'inputname',{'u' 'w' 'v'},'outputname',{'y' 'yv'});

sys = parallel(P,kalmf,1,1,[],[]);
SimModel = feedback(sys,1,4,2,1);
SimModel = SimModel([1 3],[1 2 3]);
SimModel.inputname
SimModel.outputname

t = (0:100)';
u = sin(t/5);

rng(10,'twister');
w = sqrt(Q)*randn(length(t),1);
v = sqrt(R)*randn(length(t),1);
out = lsim(SimModel,[w,v,u]);

y = out(:,1);
ye = out(:,2);
yv = y + v;

%Σχόλια Γραφικών Παραστάσεων
clf
subplot(211), plot(t,y,'b',t,ye,'r--'),
xlabel('No. of samples'), ylabel('Output')
title('Kalman filter response')
subplot(212), plot(t,y-yv,'g',t,y-ye,'r--'),
xlabel('No. of samples'), ylabel('Error')

MeasErr = y-yv;
MeasErrCov = sum(MeasErr.*MeasErr)/length(MeasErr);
EstErr = y-ye;
EstErrCov = sum(EstErr.*EstErr)/length(EstErr);

sys = ss(A,B,C,D,-1);
y = lsim(sys,u+w);
yv = y + v;

P=B*Q*B';
x=zeros(3,1);
ye = zeros(length(t),1);
ycov = zeros(length(t),1);
errcov = zeros(length(t),1);

for i=1:length(t)

  Mn = P*C'/(C*P*C'+R);
  x = x + Mn*(yv(i)-C*x);
  P = (eye(3)-Mn*C)*P;
  ye(i) = C*x;
```

```
  errcov(i) = C*P*C';

  x = A*x + B*u(i);
  P = A*P*A' + B*Q*B';
end

%Εμφάνιση Αποτελεσμάτων
subplot(211), plot(t,y,'b',t,ye,'r--'),
xlabel('No. of samples'), ylabel('Output')
title('Response with time-varying Kalman filter')
subplot(212), plot(t,y-yv,'g',t,y-ye,'r--'),
xlabel('No. of samples'), ylabel('Error')
subplot(211)
plot(t,errcov), ylabel('Error Covar'),
MeasErr = y-yv;
MeasErrCov = sum(MeasErr.*MeasErr)/length(MeasErr);
EstErr = y-ye;
EstErrCov = sum(EstErr.*EstErr)/length(EstErr);
```



Εικόνα Π.1.4 Απόκριση κώδικα Kalman Filter

Παράρτημα 2 – Κώδικες

Κώδικας quadrotor_dynamics.m:

```
function [sys,x0,str,ts] = quadrotor_dynamics(t,x,u,flag,quad)

    %   Παράμετροι
    %   u              Τιμές Εισόδων          1x4
    %   crash         (1 or 0)                1x1
    %   init           Αρχικές Συνθήκες       1x12
    %   u = [N S E W]  Εντολές Κινητήρων      1x4

    %   Μεταβλητές Κατάστασης Και Ταχύτητες Κινητήρων
    %   z      Θέση                          3x1   (X,Y,Z)
    %   v      Ταχύτητα                      3x1   (dX,dY,dZ)
    %   n      Προσανατολισμός               3x1   (Y,P,R)
    %   o      Γωνιακή Ταχύτητα              3x1   (wx,wy,wz)
    %   w      Γωνιακές Ταχύτητες Κινητήρων 4x1

    % Πίνακας Μεταβλητών Κατάστασης
    % x = [z1 zzn1 n2 n3 z1 z2 z3 o1 o2 o3 w1 w2 w3 w4]

    %Αρχικές Καταστάσεις
    z0 = [0 0 -0.046];          %   z0      Αρχικές Τιμές Θέσης              1x3
    n0 = [0 0 0];               %   n0      Αρχικές Τιμές Γωνιών Euler       1x3
    v0 = [0 0 0];               %   v0      Αρχικές Τιμές Ταχυτήτων          1x3
    o0 = [0 0 0];               %   o0      Αρχικές Τιμές Γωνιακών ταχυτήτων 1x3
    init = [z0 n0 v0 o0];

    switch flag
        case 0
            [sys,x0,str,ts]=mdlInitializeSizes(init,quad);   % Αρχικοποίηση
        case 1
            sys = mdlDerivatives(t,x,u,quad);                % Υπολογισμός Παραγώγων
        case 3
            sys = mdlOutputs(t,x,quad);                      % Υπολογισμός Εξόδων
        case { 2, 4, 9 }
            sys = [];
        otherwise
            error(['Unhandled flag = ',num2str(flag)]);
    end
end

% Επιστροφή Μεγεθών των Πινάκων, Αρχικών Καταστάσεων και Δειγματοληψία της
function [sys,x0,str,ts] = mdlInitializeSizes(init,quad)

    sizes = simsizes;
    sizes.NumContStates  = 12;
    sizes.NumDiscStates  = 0;
    sizes.NumOutputs     = 12;
    sizes.NumInputs      = 4;
    sizes.DirFeedthrough = 0;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);

    % Αρχικοποίηση των αρχικών καταστάσεων
    x0 = init;

    % Όπου str ένας κενός πίνακας
    str = [];

    % Δειγματοληψία
    ts = [0 0];

    if quad.verbose

disp(sprintf('t\t\tz1\t\tz2\t\tz3\t\tn1\t\tn2\t\tn3\t\tv1\t\tv2\t\tv3\t\to1\t\to2\t\to3\t\tw1\t\tw2\t\tw
3\t\tw4\t\tu1\t\tu2\t\tu3\t\tu4'))
    end
end
```

```
% Υπολογισμός μεταβλητών κατάστασης της επόμενης δειγματοληψίας

function sys = mdlDerivatives(t,x,u,quad)
    global a1s b1s

    %Σταθερές
    %Δείκτες Κατεύθυνσης
    N = 1;                      %   N       'North'                             1x1
    E = 2;                      %   S       'South'                             1x1
    S = 3;                      %   E       'East'                              1x1
    W = 4;                      %   W       'West'                              1x1

    d=sqrt(2)/2*quad.d;

    D(:,1) = [ d;-d;quad.h];    %   Di Σφάλμα από την επιθυμητή θέση του κινητήρα 1x3
    D(:,2) = [ d; d;quad.h];
    D(:,3) = [-d; d;quad.h];
    D(:,4) = [-d;-d;quad.h];

    %Τιμές πλαισίου αναφοράς
    e1 = [1;0;0];
    e2 = [0;1;0];
    e3 = [0;0;1];

    %Συλλογή καταστάσεων από το U
    w = u(1:4);

    %Συλλογή καταστάσεων από το X
    z = x(1:3);   % Θέση         {W}
    n = x(4:6);   % Θ,Φ,Ψ γωνίες {W}
    v = x(7:9);   % Ταχύτητα     {W}
    o = x(10:12); % Γωνιακή Ταχύτητα {B} (wx,wy,wz) (FR)

    %Προεπεξεργασία στρέψης και πινάκων WRONSKIAN
    phi = n(1);    % yaw
    the = n(2);    % pitch
    psi = n(3);    % roll

    if (abs(the)>pi/2) error('|pitch| greater than pi/2!'); end;
    if (abs(psi)>pi/2) error('|roll| greater than pi/2!'); end;
    if (z(3)>0) error('z greater than 0 (below ground)!'); end;


    % Πίνακας μεταφοράς από το πλαίσιο αναφοράς στο σταθερό πλαίσιο
R_Body2World = [cos(the)*cos(phi) sin(psi)*sin(the)*cos(phi)-cos(psi)*sin(phi)
cos(psi)*sin(the)*cos(phi)+sin(psi)*sin(phi);
cos(the)*sin(phi) sin(psi)*sin(the)*sin(phi)+cos(psi)*cos(phi) cos(psi)*sin(the)*sin(phi)-
sin(psi)*cos(phi);
     -sin(the)         sin(psi)*cos(the)                              cos(psi)*cos(the)];

    % Πίνακας στρέψης
    iW = [0        sin(psi)          cos(psi);
          0        cos(psi)*cos(the) -sin(psi)*cos(the);
          cos(the) sin(psi)*sin(the) cos(psi)*sin(the)] / cos(the);


    %Μοντέλο Κινητήρα
    for i=[N E S W]  % Σχετική κίνηση για τον κάθε κινητήρα

        Vr = cross(o,D(:,i)) + inv(R_Body2World)*v;
        mu = sqrt(sum(Vr(1:2).^2)) / (abs(w(i))*quad.r);
        lc = Vr(3) / (abs(w(i))*quad.r);
        li = mu;
        alphas = atan2(lc,mu);
        j = atan2(Vr(2),Vr(1));
        J = [cos(j) -sin(j);
            sin(j) cos(j)];

        %Flapping
        beta = [((8/3*quad.theta0 + 2*quad.theta1) - 2*(lc))/(1/mu-mu/2);0;];
        beta = J'*beta;
        a1s(i) = beta(1) - 16/quad.gamma/abs(w(i)) * o(2);
        b1s(i) = beta(2) - 16/quad.gamma/abs(w(i)) * o(1);
```

```
        %Δυνάμεις και Ροπές
T(:,i) = quad.Ct*quad.rho*quad.A*quad.r^2*w(i)^2 * [-cos(b1s(i))*sin(a1s(i));   sin(b1s(i));-
cos(a1s(i))*cos(b1s(i))];

        %Ώθηση κινητήρων, γραμμικοποίηση εκτιμήσεων γωνιών
        Q(:,i) = -quad.Cq*quad.rho*quad.A*quad.r^3*w(i)*abs(w(i)) * e3;     %Ροπή αδράνειας

        tau(:,i) = cross(D(:,i),T(:,i));  %Ροπή εξαιτίας της ώθησης των κινητήρων

    end

    %Περιγραφή δυναμικού μοντέλου
    dz = v;
    dn = iW*o;

    dv = (quad.g*e3 + R_Body2World*(1/quad.M)*sum(T,2));
    do = inv(quad.J)*(-cross(o,quad.J*o) + sum(tau,2) + sum(Q,2)); %Άθροισμα ροπών
    if isnan(do)
      error('system rotating NaN!')
      do = zeros(size(do));
    end
    sys = [dz;dn;dv;do];                          %Διανύσματα παραγώγων μεταβλητών κατάστασης
end

% Υπολογισμός διανύσματος εξόδου για αυτή τη βηματική

function sys = mdlOutputs(t,x, quad)
    %Αποφυγή σύγκρουσης
    if x(3)>0
        x(3) = 0;
        if x(6) > 0
            x(6) = 0;
        end
    end

    if quad.verbose
        disp(sprintf('%0.3f\t',t,x))
    end

    % Υπολογισμός διανύσματος εξόδου ως συνάρτηση του διανύσματος κατάστασης
    %   z      Θέση                           3x1   (x,y,z)
    %   v      Ταχύτητα                       3x1   (xd,yd,zd)
    %   n      Προσανατολισμός                3x1   (Y,P,R)
    %   o      Γωνική ταχύτητα                3x1   (Yd,Pd,Rd)
    n = x(4:6);    % Φ,Θ,Ψ γωνίες
    phi = n(1);    % yaw
    the = n(2);    % pitch
    psi = n(3);    % roll

    % Πίνακας μεταφοράς από το πλαίσιο αναφοράς στο σταθερό πλαίσιο(Wronskian)
R_Body2World = [cos(the)*cos(phi) sin(psi)*sin(the)*cos(phi)-cos(psi)*sin(phi)
cos(psi)*sin(the)*cos(phi)+sin(psi)*sin(phi);
cos(the)*sin(phi) sin(psi)*sin(the)*sin(phi)+cos(psi)*cos(phi) cos(psi)*sin(the)*sin(phi)-
sin(psi)*cos(phi);
-sin(the)         sin(psi)*cos(the)                            cos(psi)*cos(the)];

    iW = [0        sin(psi)          cos(psi);              % Αντίστροφος Wronskian
          0        cos(psi)*cos(the) -sin(psi)*cos(the);
          cos(the) sin(psi)*sin(the) cos(psi)*sin(the)] / cos(the);

    % Επιστροφή της ταχύτητας στο πλαίσιο αναφοράς
    sys = [ x(1:6);                          % μεταφορά θέσεων και γωνιών Euler
            inv(R_Body2World)*x(7:9);        % μεταφορά ταχύτητας στο πλαίσιο αναφοράς
            x(10:12)];                       % pqr βάσει πλαισίου αναφοράς

end
```

Κώδικας rsedu_control.c:

```
#include "rsedu_control.h"
#include <stdbool.h>
```

Βιβλιοθήκες κώδικα

```
int FEAT_TIME            = 0;
int FEAT_OF_ACTIVE       = 1;
int FEAT_POSVIS_RUN      = 1;
int FEAT_POSVIS_USE      = 0;
int FEAT_IMSAVE          = 0;
int FEAT_NOLOOK          = 0;
int FEAT_NOSAFETY        = 0;

int onCycles             = 4000;
int calibCycles          = 400;
int takeoffCycles        = 200;
```

paramsEDU.dat και καθορισμός κύκλων μηχανής άνα στάδιο

```
// Εδώ γίνεται η εισαγωγή του κώδικα C που υλοποιήθηκε από το Simulink. Η εισαγωγή
// αυτή γίνεται μέσω του κώδικα PackEmbeddedCode.c. Λόγω του μεγάλου όγκου του, η
// παρουσίασή του εδώ παραλείφθηκε.

// Αρχή main προγράμματος


void RSEDU_control(HAL_acquisition_t* hal_sensors_data, HAL_command_t* hal_sensors_cmd)
{

    static int run_flag = 1;
    static int counter = 0;
    static int counter_noOF = 0;

    static float MAX_ACCELL      = 6.0;
    static float MAX_DELTADXY    = 6.0;
    static float MAX_RANGE               = 10.0;
    static float MIN_BATTTAKEOFF         = 30.0;
    static float MIN_BATT                = 30.0;
    static int MAX_noOF          = 200;

    double powerGain = 0;
    static double powerGain_paramsFile = 0.1;

    static double sensorCal[7];
    static double battLevelAvg;
    static float of_data[5];


    float vis_data[4];


    static bool pressureSensorOk = false;
    static float ofQuality;


    static char serverIP[16];
    strncpy(serverIP, "192.168.1.2", 16);
    static int sockfd = 0;
    static char recvBuff[100];
    static struct sockaddr_in serv_addr;

    int pitch_ref_buff, roll_ref_buff, yaw_ref_buff, alt_ref_buff
    int x_ref_buff, y_ref_buff;

    static int of_fifo, vis_fifo;

    HAL_acquisition_t* in = hal_sensors_data;
    HAL_command_t    * out = hal_sensors_cmd;

    long long start;
    static FILE *ptfile;
    ptimer_start(FEAT_TIME, counter, &(start));
```

Αρχικοποίηση και δήλωση μεταβλητών πτήσης και επικοινωνίας

Δήλωση Ptiming

```
    if(run_flag == 0)
    {
        if(counter > calibCycles)
        {
            printf("Saving logged data after %i cycles... \n", counter);
            rt_StopDataLogging(MATFILE, Drone_Compensator_M->rtwLogInfo);
        }

        if(FEAT_OF_ACTIVE)
        {
            close(of_fifo);
        }

        if(FEAT_TIME)
        {
            fclose(ptfile);
        };


        printf("Saving logged data... DONE \n");
        printf("Good night! \n");
        out->motors_speed[0] = 0;
        out->motors_speed[1] = 0;
        out->motors_speed[2] = 0;
        out->motors_speed[3] = 0;
        out->command = BLDC_CMD_STOP;
        usleep(100);
        exit(0);
    }


    counter++;
```

Έξοδος εάν run_flag=0

// Βήμα 1:
// Αρχικοποίηση (Σύνδεση Server, Ρυθμίσεις Χρήστη)

```
    if(counter == 1)
    {
        printf("\nBattery output voltage: %5.2f V - %0d percents\n", in->HAL_vbat_SI.vbat_V, (int)in->HAL_vbat_SI.vbat_percentage);
        printf("used: %d, users: %d, gyrotemp %f, acctemp %f, presstmp %f \n", (int)in->used, (int)in->count_user, in->HAL_gyro_SI.temperature, in->HAL_acc_SI.temperature, in->HAL_pressure_SI.temperature);

        FILE *paramFile;
        paramFile = fopen("/data/edu/params/paramsEDU.dat", "r");
        if(paramFile == NULL)
        {
            printf("ParamsEDU.dat parameter file not found, using default! \n");
        }
        else
        {
            char tmpbuff[50];
            char tmpstr1[16];
            char tmpstr2[16];
            while(!feof(paramFile))
            {
                if(fgets(tmpbuff, 50, paramFile))
                {
                    sscanf(tmpbuff, "%15s : %15[^;];", tmpstr1, tmpstr2);

                    if(!strcmp(tmpstr1, "FEAT_OF_ACTIVE"))
                    {
                        FEAT_OF_ACTIVE = atoi(tmpstr2);
                    }
                    else if(!strcmp(tmpstr1, "FEAT_POSVIS_RUN"))
                    {
                        FEAT_POSVIS_RUN = atoi(tmpstr2);
                    }
                    else if(!strcmp(tmpstr1, "POWERGAIN"))
                    {
```

Έλεγχος αρχείου και ανάγνωση του paramsEDU.dat

```
                    powerGain_paramsFile = atoi(tmpstr2) / 100.0;
                }
                else if(!strcmp(tmpstr1, "FEAT_POSVIS_USE"))
                {
                    FEAT_POSVIS_USE = atoi(tmpstr2);
                }
                else if(!strcmp(tmpstr1, "FEAT_NOLOOK"))
                {
                    FEAT_NOLOOK = atoi(tmpstr2);
                }
                else if(!strcmp(tmpstr1, "FEAT_IMSAVE"))
                {
                    FEAT_IMSAVE = atoi(tmpstr2);
                }
                else if(!strcmp(tmpstr1, "FEAT_TIME"))
                {
                    FEAT_TIME = atoi(tmpstr2);
                }
                else if(!strcmp(tmpstr1, "FEAT_NOSAFETY"))
                {
                    FEAT_NOSAFETY = atoi(tmpstr2);
                }
                else if(!strcmp(tmpstr1, "IP"))
                {
                    memcpy(serverIP, tmpstr2, sizeof(tmpstr2));
                };

            }
        }
        fclose(paramFile);
    }

    printf("\nSettings:\n -----\n FEAT_TIME: %d \n FEAT_OF_ACTIVE: %d \n FEAT_POSVIS_RUN: %d \n
FEAT_POSVIS_USE: %d \n FEAT_NOLOOK: %d \n FEAT_IMSAVE: %d \n FEAT_NOSAFETY: %d \n -----\n", FEAT_TIME,
FEAT_OF_ACTIVE, FEAT_POSVIS_RUN, FEAT_POSVIS_USE, FEAT_NOLOOK, FEAT_IMSAVE, FEAT_NOSAFETY);

    ptimer_init(FEAT_TIME, __func__, &(ptfile), &(run_flag));

    printf("Waiting for connection to reference value server... \n");
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n ERROR : Could not create socket \n");
        exit(0);
    }

    memset(&serv_addr, '0', sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(12345);

    if(inet_pton(AF_INET, serverIP, &serv_addr.sin_addr) <= 0)
    {
        printf("ERROR inet_pton error occured \n");
        exit(0);
    }

    if(connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("ERROR Connection to reference value server failed \n");
        exit(0);
    }else
    {
        printf("Connected to reference value server! \n");
    }

}
```

Έλεγχος αρχείου και ανάγνωση του paramsEDU.dat

Εκκίνηση Ptiming

Εκκίνηση επικοινωνίας με Reference Value Server

```
else if(counter == 2)
{
    printf("Waiting for optical flow connection...\n");
    usleep(100);
    of_fifo = open("/tmp/of_fifo", O_RDONLY);
    fcntl(of_fifo, F_SETFL, fcntl(of_fifo, F_GETFL) | O_NONBLOCK);

    if(FEAT_OF_ACTIVE)
    {
        read(of_fifo, (float*)(&of_data), sizeof(of_data));
        if(of_fifo < 0)
        {
            printf("WARNING of might not be running, %d!\n\n", of_fifo);
        }
        else
        {
            printf("Got optical flow connection, %d!\n", of_fifo);
        }

    }
    else
    {
        printf("Optical Flow deactivated! \n");
    }

    printf("Waiting for image processing connection...\n");
    usleep(100);
    vis_fifo = open("/tmp/vis_fifo", O_RDONLY);
    fcntl(vis_fifo, F_SETFL, fcntl(vis_fifo, F_GETFL) | O_NONBLOCK);

    if(FEAT_POSVIS_RUN)
    {
        read(vis_fifo, (float*)(&vis_data), sizeof(vis_data));
        if(vis_fifo < 0)
        {
            printf("WARNING vis might not be running, %d!\n", vis_fifo);
        }
        else
        {
            printf("Got image processing connection, %d !\n", vis_fifo);
        }
    }
    else
    {
        printf("POSVIS computations deactivated! \n");
    };

    sensorCal[0] = in->HAL_acc_SI.x;
    sensorCal[1] = in->HAL_acc_SI.y;
    sensorCal[2] = in->HAL_acc_SI.z;
    sensorCal[3] = in->HAL_gyro_SI.x;
    sensorCal[4] = in->HAL_gyro_SI.y;
    sensorCal[5] = in->HAL_gyro_SI.z;
    sensorCal[6] = in->HAL_pressure_SI.pressure;

    battLevelAvg = (double)((int)in->HAL_vbat_SI.vbat_percentage);

    out->command = BLDC_CMD_START;
}
```

Δημιουργία επικοινωνίας με Optical Flow

Δημιουργία επικοινωνίας με Image Processing

Calibration αισθητηρίων και ενεργοποίηση κινητήρων

// Βήμα 2:
// Καταγραφή Δεδομένων Calibration

```
    else if(counter < calibCycles)
    {
        int counterCalib = counter - 2;
if (counterCalib<=0) printf("error in counter %d for Calibration!\n",counterCalib);
        sensorCal[0] = sensorCal[0] * (counterCalib - 1) / counterCalib + in->
HAL_acc_SI.x / counterCalib;
        sensorCal[1] = sensorCal[1] * (counterCalib - 1) / counterCalib + in->
HAL_acc_SI.y / counterCalib;
        sensorCal[2] = sensorCal[2] * (counterCalib - 1) / counterCalib + in->
HAL_acc_SI.z / counterCalib;
        sensorCal[3] = sensorCal[3] * (counterCalib - 1) / counterCalib + in->
HAL_gyro_SI.x / counterCalib;
        sensorCal[4] = sensorCal[4] * (counterCalib - 1) / counterCalib + in->
HAL_gyro_SI.y / counterCalib;
        sensorCal[5] = sensorCal[5] * (counterCalib - 1) / counterCalib + in->
HAL_gyro_SI.z / counterCalib;
        sensorCal[6] = sensorCal[6] * (counterCalib - 1) / counterCalib + in->
HAL_pressure_SI.pressure / counterCalib;

        if(in->HAL_pressure_SI.pressure != 0) pressureSensorOk = true;

        battLevelAvg = battLevelAvg * (counterCalib - 1) / counterCalib +
(double)((int)in->HAL_vbat_SI.vbat_percentage) / counterCalib;

        if(FEAT_OF_ACTIVE)  read(of_fifo, (float*)(&of_data), sizeof(of_data));
        if(FEAT_POSVIS_RUN) read(vis_fifo, (float*)(&vis_data), sizeof(vis_data));


        out->motors_speed[0] = 0;
        out->motors_speed[1] = 0;
        out->motors_speed[2] = 0;
        out->motors_speed[3] = 0;
        out->command = BLDC_CMD_STOP;
        return;
    }
```

Καταγραφή δεδομένων Calibration

Ενεργοποίηση κινητήρων με U=0

// Βήμα 3:
// Αρχικοποίηση Δυναμικού Μοντέλου για έλεγχο

```
    else if(counter == calibCycles)
    {
        printf("Batterylevel: %f\n", battLevelAvg);
        printf("Sensorcal: %f :: %f :: %f :: %f :: %f :: %f :: %f \n", sensorCal[0],
sensorCal[1], sensorCal[2], sensorCal[3], sensorCal[4], sensorCal[5], sensorCal[6]);

        if((!FEAT_NOSAFETY) && fabs(9.81 + sensorCal[2]) > 0.7)
        {
            run_flag = 0;
            printf("ERROR: Please take off from a level surface! \n");
            out->motors_speed[0] = 0;
            out->motors_speed[1] = 0;
            out->motors_speed[2] = 0;
            out->motors_speed[3] = 0;
            out->command = BLDC_CMD_STOP;
            return;
        }
        if(!pressureSensorOk)
        {
            run_flag = 0;
            printf("ERROR: Pressure sensor appears damaged! \n");
            out->motors_speed[0] = 0;
            out->motors_speed[1] = 0;
            out->motors_speed[2] = 0;
            out->motors_speed[3] = 0;
            out->command = BLDC_CMD_STOP;
            return;
        }
```

Έλεγχος ισορροπημένης απογείωσης

Έλεγχος λειτουργίας αισθητηρίου πίεσης

// Simulink Drone Compensator
// Δήλωση Εισόδων και Εξόδων

```
    Drone_Compensator_M->ModelData.defaultParam = &Drone_Compensator_P;
    Drone_Compensator_M->ModelData.blockIO = &Drone_Compensator_B;
    Drone_Compensator_M->ModelData.dwork = &Drone_Compensator_DW;

    Drone_Compensator_initialize(Drone_Compensator_M,
      &Drone_Compensator_U_controlModePosVSOrient_flagin,
      Drone_Compensator_U_pos_refin, &Drone_Compensator_U_takeoff_flag,
      Drone_Compensator_U_orient_refin, &Drone_Compensator_U_ddx,
      &Drone_Compensator_U_ddy, &Drone_Compensator_U_ddz, &Drone_Compensator_U_p,
      &Drone_Compensator_U_q, &Drone_Compensator_U_r,
      &Drone_Compensator_U_altitude_sonar, &Drone_Compensator_U_prs,
      Drone_Compensator_U_opticalFlow_datin,
      Drone_Compensator_U_sensordataCalib_datin, Drone_Compensator_U_posVIS_datin,
      &Drone_Compensator_U_usePosVIS_flagin,
      Drone_Compensator_U_batteryStatus_datin, Drone_Compensator_Y_motors_refout,
      &Drone_Compensator_Y_X, &Drone_Compensator_Y_Y, &Drone_Compensator_Y_Z,
      &Drone_Compensator_Y_yaw, &Drone_Compensator_Y_pitch,
      &Drone_Compensator_Y_roll, &Drone_Compensator_Y_dx, &Drone_Compensator_Y_dy,
      &Drone_Compensator_Y_dz, &Drone_Compensator_Y_pb, &Drone_Compensator_Y_qb,
      &Drone_Compensator_Y_rb, &Drone_Compensator_Y_controlModePosVSOrient_flagout,
      Drone_Compensator_Y_pose_refout, &Drone_Compensator_Y_ddxb,
      &Drone_Compensator_Y_ddyb, &Drone_Compensator_Y_ddzb,
      &Drone_Compensator_Y_pa, &Drone_Compensator_Y_qa, &Drone_Compensator_Y_ra,
      &Drone_Compensator_Y_altitude_sonarb, &Drone_Compensator_Y_prsb,
      Drone_Compensator_Y_opticalFlow_datout,
      Drone_Compensator_Y_sensordataCalib_datout,
      Drone_Compensator_Y_posVIS_datout, &Drone_Compensator_Y_usePosVIS_flagout,
      Drone_Compensator_Y_batteryStatus_datout,
      &Drone_Compensator_Y_takeoff_flagout);

  Drone_Compensator_U_posVIS_datin[0] = NO_VIS_X;
  Drone_Compensator_U_posVIS_datin[1] = 0.0;
  Drone_Compensator_U_posVIS_datin[2] = 0.0;
  Drone_Compensator_U_posVIS_datin[3] = 0.0;

  if((FEAT_POSVIS_RUN) && (vis_fifo < 0))
  {
      vis_fifo = open("/tmp/vis_fifo", O_RDONLY);
      fcntl(vis_fifo, F_SETFL, fcntl(vis_fifo, F_GETFL) | O_NONBLOCK);
      if(vis_fifo < 0) printf("WARNING image processing not connected!\n");
  }

  if((FEAT_OF_ACTIVE) && (of_fifo < 0))
  {
      of_fifo = open("/tmp/of_fifo", O_RDONLY);
      fcntl(of_fifo, F_SETFL, fcntl(of_fifo, F_GETFL) | O_NONBLOCK);
      if(of_fifo < 0) printf("ERROR optical flow not running!\n");
      run_flag = 0;
  }

  Drone_Compensator_U_sensordataCalib_datin[0] = sensorCal[0];
  Drone_Compensator_U_sensordataCalib_datin[1] = sensorCal[1];
  Drone_Compensator_U_sensordataCalib_datin[2] = sensorCal[2];
  Drone_Compensator_U_sensordataCalib_datin[3] = sensorCal[3];
  Drone_Compensator_U_sensordataCalib_datin[4] = sensorCal[4];
  Drone_Compensator_U_sensordataCalib_datin[5] = sensorCal[5];
  Drone_Compensator_U_sensordataCalib_datin[6] = sensorCal[6];

  out->command = BLDC_CMD_RUN;

}
```

Έλεγχος of_fifo και vis_fifo

Εισαγωγή Αισθητηρίων στο Simulink

// Βήμα 4:
// Πτήση

```
    else if(counter <= onCycles)
    {

        if(counter < calibCycles + takeoffCycles)
        {
            powerGain = powerGain_paramsFile;
            Drone_Compensator_U_takeoff_flag = 1;
            Drone_Compensator_U_pos_refin[2] = -1.1;
```

Παράμετροι ισορροπίας και ύψους πρίν την απογείωση

```
            if((Drone_Compensator_U_batteryStatus_datin[1] < MIN_BATTTAKEOFF) &&
(Drone_Compensator_U_batteryStatus_datin[1] > 0.1))
            {
                run_flag = 0;
                printf("Flight aborted due to low voltage (%f %%): shutting down
motors now, charge battery!\n", Drone_Compensator_U_batteryStatus_datin[1]);
                out->motors_speed[0] = 0;
                out->motors_speed[1] = 0;
                out->motors_speed[2] = 0;
                out->motors_speed[3] = 0;
                out->command = BLDC_CMD_STOP;
                return;
            }
        }
```

Έλεγχος μπαταρίας για απογείωση

```
        else if(counter == calibCycles + takeoffCycles)
        {
            Drone_Compensator_U_takeoff_flag = 0;
            Drone_Compensator_U_pos_refin[2] = -1.1;
        }
```

Flag απογείωσης και ορισμός ύψους

// Πτήση και υλοποίηση εντολών από το ReferenceValueServer.c

```
        else if(counter < onCycles)
        {
            powerGain = powerGain_paramsFile;

            fcntl(sockfd, F_SETFL, O_NONBLOCK);
            memset(recvBuff, '\0', sizeof(recvBuff));
            recv(sockfd, recvBuff, sizeof(recvBuff), O_NONBLOCK);

            if((recvBuff[0]) != '\0')
            {
                sscanf(recvBuff, "%i %i %i %i %i %i %i", &run_flag, &pitch_ref_buff, &roll_ref_buff,
&yaw_ref_buff, &x_ref_buff, &y_ref_buff, &alt_ref_buff);
                Drone_Compensator_U_orient_refin[0] = (double)((yaw_ref_buff - 10000) / 1000.0);
                Drone_Compensator_U_orient_refin[1] = (double)((pitch_ref_buff - 10000) / 1000.0);
                Drone_Compensator_U_orient_refin[2] = (double)((roll_ref_buff - 10000) / 1000.0);
                 Drone_Compensator_U_pos_refin[0] = (double)(x_ref_buff / 100.0);
                 Drone_Compensator_U_pos_refin[1] = (double)(y_ref_buff / 100.0);
                if(((double)(alt_ref_buff / 100.0)) >= -4.0)
                {
                    Drone_Compensator_U_pos_refin[2]    = (double)(alt_ref_buff / 100.0);
                }
            }
```

```
            if((Drone_Compensator_U_orient_refin[1] == 0.0) &&
(Drone_Compensator_U_orient_refin[2] == 0.0))
                Drone_Compensator_U_controlModePosVSOrient_flagin = 1;
            else
                Drone_Compensator_U_controlModePosVSOrient_flagin = 0;
```

Flag ισορροπίας

```
            Drone_Compensator_U_usePosVIS_flagin = FEAT_POSVIS_USE;
```

Εκτίμηση θέσης από vision

```
            if(run_flag == 0)
            {
                printf("Flight abort request: shutting down motors now\n");
                out->motors_speed[0] = 0;
                out->motors_speed[1] = 0;
                out->motors_speed[2] = 0;
                out->motors_speed[3] = 0;
                out->command = BLDC_CMD_STOP;
                return;
            }

        }
        else
        {
            powerGain = 0;
        };
```

Διακοπή πτήσης

// Παράμετροι διακοπής πτήσης

```
        bool crash_detected;
        if( (!FEAT_NOSAFETY) && (counter>(calibCycles + takeoffCycles)))
          crash_detected = (fabs(in->HAL_acc_SI.x) > MAX_ACCELL) || (fabs(in->HAL_acc_SI.y) >
MAX_ACCELL) || (in->HAL_acc_SI.z > 0);
        else
          crash_detected = (fabs(in->HAL_acc_SI.x) > MAX_ACCELL * 3) || (fabs(in->HAL_acc_SI.y) >
MAX_ACCELL * 3);


        bool out_of_range = (fabs(Drone_Compensator_Y_X) > MAX_RANGE) || (fabs(Drone_Compensator_Y_Y) >
MAX_RANGE);

        bool battery_low = (Drone_Compensator_U_batteryStatus_datin[1] < MIN_BATT) &&
(Drone_Compensator_U_batteryStatus_datin[1] > 1.0);

        if(crash_detected) printf("Flight crash detected (accelerometer): shutting down motors now\n");
        if(out_of_range)   printf("Drone out of range: shutting down motors now\n");
        if(battery_low)    printf("Flight aborted due to low voltage (%f %%): shutting down motors now,
charge battery!\n", Drone_Compensator_U_batteryStatus_datin[1]);

        if(crash_detected || battery_low || out_of_range) {
          run_flag = 0;
          out->motors_speed[0] = 0;
          out->motors_speed[1] = 0;
          out->motors_speed[2] = 0;
          out->motors_speed[3] = 0;
          out->command = BLDC_CMD_STOP;
          return;
        }

        if((FEAT_OF_ACTIVE) && (of_fifo > 0))
        {
            if((read(of_fifo, (float*)(&of_data), sizeof(of_data)) > 0) &&
((of_data[0] != 0.0) || (of_data[1] != 0.0)))
            {
                ofQuality = of_data[3];
                if(ofQuality > 0)
                {
                    counter_noOF = 0;
                    Drone_Compensator_U_opticalFlow_datin[0] = (double)of_data[0];
                    Drone_Compensator_U_opticalFlow_datin[1] = (double)of_data[1];
                    Drone_Compensator_U_opticalFlow_datin[2] = (double)of_data[2];
                }
            }
            else
            {
                if(counter > (calibCycles + takeoffCycles))
                {
                    counter_noOF += 1;

                    if(counter_noOF >= MAX_noOF)
                    {
                        run_flag = 0;
```

Είσοδος αποτελεσμάτων of στο μοντέλο

```
                        printf("Problem with optical flow, there has been no flow
for %d cycles in cycle %d: shutting down motors now\n", counter_noOF, counter);
                        out->motors_speed[0] = 0;
                        out->motors_speed[1] = 0;
                        out->motors_speed[2] = 0;
                        out->motors_speed[3] = 0;
                        out->command = BLDC_CMD_STOP;
                        return;
                    }
                }
```

Είσοδος αποτελεσμάτων of στο μοντέλο

```
            };

        }
```

// Διακοπή πτήσης εάν ο optical flow διαφέρει πολύ από τις ταχύτητες

```
        if(ofQuality > 0 &&
            (counter > (calibCycles + takeoffCycles))
            &&
            (
                ((fabs(of_data[0]) > 0.01) && (fabs(20 * of_data[0] - Drone_Compensator_Y_dx) >
MAX_DELTADXY))
                || ((fabs(of_data[1]) > 0.01) && (fabs(20 * of_data[1] - Drone_Compensator_Y_dy) >
MAX_DELTADXY))
            )

        )
        {
            run_flag = 0;
            printf("Flight crash about to happen, mismatch optical flow (%f, %f) and state estimate (%f,
%f): shutting down motors now\n",
              20 * of_data[0],
              20 * of_data[1],
              Drone_Compensator_Y_dx,
              Drone_Compensator_Y_dy);
            out->motors_speed[0] = 0;
            out->motors_speed[1] = 0;
            out->motors_speed[2] = 0;
            out->motors_speed[3] = 0;
            out->command = BLDC_CMD_STOP;
            return;
        }

        if((FEAT_POSVIS_RUN) && (vis_fifo > 0))
        {

            if((read(vis_fifo, (float*)(&vis_data), sizeof(vis_data)) > 0)
&& ((vis_data[0] != 0.0) || (vis_data[1]) || (vis_data[3])))
            {
                Drone_Compensator_U_posVIS_datin[0] = (double)vis_data[0];
                Drone_Compensator_U_posVIS_datin[1] = (double)vis_data[1];
                Drone_Compensator_U_posVIS_datin[2] = (double)vis_data[2];
                Drone_Compensator_U_posVIS_datin[3] = (double)vis_data[3];
            }
            else
            {
                Drone_Compensator_U_posVIS_datin[0] = NO_VIS_X;
                Drone_Compensator_U_posVIS_datin[1] = 0.0;
                Drone_Compensator_U_posVIS_datin[2] = 0.0;
                Drone_Compensator_U_posVIS_datin[3] = 0.0;
            }

        }
```

Είσοδος αποτελεσμάτων vis στο μοντέλο

// Είσοδος σημάτων αισθητηρίων

```
        Drone_Compensator_U_ddx  = in->HAL_acc_SI.x;
        Drone_Compensator_U_ddy  = in->HAL_acc_SI.y;
        Drone_Compensator_U_ddz  = in->HAL_acc_SI.z;
        Drone_Compensator_U_p    = in->HAL_gyro_SI.x;
        Drone_Compensator_U_q    = in->HAL_gyro_SI.y;
        Drone_Compensator_U_r            = in->HAL_gyro_SI.z;
        Drone_Compensator_U_altitude_sonar       = in->HAL_ultrasound_SI.altitude;
        Drone_Compensator_U_prs  = in->HAL_pressure_SI.pressure;
        Drone_Compensator_U_batteryStatus_datin[0] = in->HAL_vbat_SI.vbat_V;
        Drone_Compensator_U_batteryStatus_datin[1] = (double)((int)in->HAL_vbat_SI.vbat_percentage);

        const char* error = rtmGetErrorStatus(Drone_Compensator_M);
        if(!error)
        {
            rt_OneStep(Drone_Compensator_M);
        }
        else
        {
            run_flag = 0;
            printf("ERROR: Error from Simulink, counter=%i !\n\t%s", counter, error);
            out->motors_speed[0] = 0;
            out->motors_speed[1] = 0;
            out->motors_speed[2] = 0;
            out->motors_speed[3] = 0;
            out->command = BLDC_CMD_STOP;
            return;
        }

        if(counter < onCycles)
        {
            out->command = BLDC_CMD_RUN;
        }
        Else
        {
            out->command = BLDC_CMD_STOP;
            return;
        };
```

Υπολογισμός εντολών ελέγχου

Διακοπή πτήσης εάν υπερβεί το onCycles (20sec)

// Ενημέρωση εντολών κινητήρων από τις εντολές ελέγχου

```
        out->motors_speed[0] = (int)(powerGain * (fabs(Drone_Compensator_Y_motors_refout[0])));
        out->motors_speed[1] = (int)(powerGain * (fabs(Drone_Compensator_Y_motors_refout[1])));
        out->motors_speed[2] = (int)(powerGain * (fabs(Drone_Compensator_Y_motors_refout[2])));
        out->motors_speed[3] = (int)(powerGain * (fabs(Drone_Compensator_Y_motors_refout[3])));
        usleep(100);
    }
```

// Βήμα 5:
// Καταγραφή δεδομένων και τέλος πτήσης

```
    else
    {
        printf("Saving logged data at end of flight... \n");
        rt_StopDataLogging(MATFILE, Drone_Compensator_M->rtwLogInfo);
        if(FEAT_OF_ACTIVE)
        {
            close(of_fifo);
        }
        printf("Saving logged data... DONE \n");

        if(FEAT_TIME)
        {
            fclose(ptfile);
        }
        exit(0);
    }
    usleep(200);
    ptimer_stopstore(FEAT_TIME, counter, start, ptfile);

}   // Τέλος main προγράμματος
```

Κώδικας rsedu_vis.c:

```
#include "rsedu_vis.h"                                  } Βιβλιοθήκες κώδικα
#include <stdbool.h>


void RSEDU_image_processing(void * buffer)      // Αρχή main προγράμματος
{
    static int counter = 0;

    static float vis_data[4];
    static int vis_fifo;

    int i, j;
    int row, col;
    float yuv[3];
    int nx = 80, ny = 120;
    float feature_pps[3][4];
    static bool matchLookupLoaded = false;
    static unsigned char matchLookup[128][128][128];

    float camerapos[3];                                 } Δήλωση και αρχικοποίηση
    float camerayaw;                                    παραμέτρων

    static int filtersize = 5;
    int pxls_ftr_min       = 5;

    int lndmrk_nr = 5, lndmrk_best = 0;
    static lndmrk_t lndmrks[5];

    if(counter == 1)
    {
        usleep(20000);
    }

    int fifo;
    pixel2_t *image = buffer;


    long long start;                                    } Δήλωση Ptiming
    static FILE *ptfile;
    ptimer_start(FEAT_TIME, counter, &(start));

    counter++;

    if(counter == 1)
    {
        ptimer_init(FEAT_TIME, __func__, &(ptfile), NULL);      } Εκκίνηση Ptiming

        printf("rsedu_vis(): Init fifo-communication...\n");

        if(access("/tmp/vis_fifo", F_OK) != -1)
        {
            printf("rsedu_vis(): SUCCESS POSVIS FIFO exists! \n");

            vis_fifo = open("/tmp/vis_fifo", O_WRONLY);
            if(vis_fifo)
            {
                vis_data[0] = -99.0;
                write(vis_fifo, (float*)(&vis_data), sizeof(vis_data));
                close(vis_fifo);
                printf("rsedu_vis(): SUCCESS opening POSVIS-fifo!\n");      } Δημιουργία vis_fifo
            }
            else
            {
                printf("rsedu_vis(): ERROR opening POSVIS-fifo!\n");
            }
        }
        else
        {
            printf("rsedu_vis(): ERROR opening POSVIS-fifo!\n");
        }
```

```
        if(FEAT_NOLOOK == 0)
        {
            FILE* data;
            if((data = fopen("/data/edu/params/lookuptable.dat", "rb")) == NULL)
            {
                printf("rsedu_vis(): ERROR opening lookupfile \n");
            }
            else
            {
                fread(matchLookup, sizeof(matchLookup), 1, data);
                matchLookupLoaded = true;
                fclose(data);
            }
        }
    }
```

Άνοιγμα lookuptable

```
    if(matchLookupLoaded && (FEAT_POSVIS_RUN) && ((counter % 15) == 0) && (NULL != image))
    {
        for(i = 0; i < lndmrk_nr; i++)
        {
            lndmrks[i].n  = 0;
            lndmrks[i].weights  = 0;
            lndmrks[i].px = 0;
            lndmrks[i].py = 0;
        }

        int margin = (int)(filtersize - 1) / 4;
        int weight = 1;

        for(j = 0; j < ny - 2 * margin; j++)
        {
            for(i = 0; i < nx - 2 * margin; i++)
            {
                row = margin + j;
                col = margin + i;
                yuv[0] = (float)image[nx * row + col].y1;
                yuv[1] = (float)image[nx * row + col].u;
                yuv[2] = (float)image[nx * row + col].v;

                if(FEAT_NOLOOK)
                {
                    printf("rsedu_vis(): ERROR Unfortunately, color conversion, etc. too slow onboard.
Please set FEAT_NOLOOK = 0. \n");
                }
```

Αρχικοποίηση ιδιοτήτων landmarks

Κατάταξη ιδιοτήτων του κάθε pixel σε πίνακα YUV

// Χρήση lookuptable για συσχέτιση pixel

```
                else
                {
                    lndmrk_best = (int)matchLookup[(int)(yuv[0] / 2)][(int)(yuv[1] / 2)][(int)(yuv[2] /
2)];
                }

                if(lndmrk_best > 0)
                {
                    if((lndmrks[lndmrk_best].weights > 15) && (fabs((col + 1) -
lndmrks[lndmrk_best].px) < 20) && (fabs((row + 1) - lndmrks[lndmrk_best].py) < 20))
                    {
                        weight = 8;
                    }
                    else
                    {
                        weight = 1;
                    }

                    lndmrks[lndmrk_best].px = lndmrks[lndmrk_best].px * lndmrks[lndmrk_best].weights /
(lndmrks[lndmrk_best].weights + weight) + (((double)(col + 1)) * 2) * weight /
(lndmrks[lndmrk_best].weights + weight);
                    lndmrks[lndmrk_best].py = lndmrks[lndmrk_best].py * lndmrks[lndmrk_best].weights /
(lndmrks[lndmrk_best].weights + weight) + ((double)(row + 1)) * weight / (lndmrks[lndmrk_best].weights +
weight);
```

Αύξηση βαρύτητας pixel ανάλογα με την θέση του

```
                    lndmrks[lndmrk_best].n += 1;
                    lndmrks[lndmrk_best].weights += weight;
                }

            }
        }

        printf("image_proc(): Result of matching: \n");
        for (i=0;i<lndmrk_nr;i++)
        {
        printf("image_proc(): Lndmrk %i: nr pxls matched: %i, pixelcoordinates: (%f,%f,%f,%f)
\n",i,lndmrks[i].n,lndmrks[i].px,lndmrks[i].py);
        }


        int features_valid = 0;
        int k;

        for(k = 1; k < lndmrk_nr; k++)
        {
            if(lndmrks[k].n >= pxls_ftr_min)
            {
                features_valid += 1;
                feature_pps[0][features_valid - 1] = lndmrks[k].px;
                feature_pps[1][features_valid - 1] = lndmrks[k].py;
                feature_pps[2][features_valid - 1] = 1.0;
            }

        }

        if(features_valid == 4)
        {
```

Καταχώρηση θέσεων για το κάθε landmark

// Η συνάρτηση reconstructCameraPose υλοποιείται στον κώδικα rsedu_vis_helpers.c.
// Σκοπός της είναι η εύρεση συντεταγμένων στον χώρο βάσει των pixels.

```
            reconstructCameraPose(camerapos, &camerayaw, feature_pps, ldnmrk_pinv, intrMatrx_inv);
            printf("rsedu_vis(): SUCCESS reconstructed camera pose: (%f, %f, %f, %f)\n", camerapos[0],
camerapos[1], camerapos[2], camerayaw * 180 / 3.1415);

            vis_data[0] = camerapos[0];
            vis_data[1] = camerapos[1];
            vis_data[2] = camerapos[2];
            vis_data[3] = camerayaw;

            vis_fifo = open("/tmp/vis_fifo", O_WRONLY);
            if(vis_fifo)
            {
                write(vis_fifo, (float*)(&vis_data), sizeof(vis_data));
                close(vis_fifo);
            }
        }
        else
        {
            printf("rsedu_vis(): WARNING not enough distinct markers (colored balls) found! \n")  ;
        }

    }
```

Καταχώρηση καινούργιας θέσης κάμερας στον πίνακα vis_data

Καταχώρηση αποτελεσμάτων στο vis_fifo

```
    if((FEAT_IMSAVE == 2) && ((counter % 60) == 0) && (NULL != image))
    {
        printf("image_proc(): Write image to fifo...\n");
        mkfifo("/tmp/picture", 0777);
        fifo = open("/tmp/picture", O_WRONLY);
        if(fifo)
        {
            write(fifo, buffer, 320 * 120);
            close(fifo);
            usleep(5000);
        }

    }
```

Online εμφάνιση εικόνων

```
    if((FEAT_IMSAVE == 1) && ((counter % 6) == 0) && (NULL != image))
    {
        FILE* data;
        char filename[15];

        sprintf(filename, "/tmp/edu/imgs/img%i.bin", counter);

        mkdir("/tmp/edu", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
        mkdir("/tmp/edu/imgs", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
        if((data = fopen(filename, "wb")) == NULL)
        {
            printf("rsedu_vis(): ERROR opening img file\n");
        }

        fwrite(image, sizeof(pixel2_t) * 80 * 120, 1, data);
        fclose(data);
        usleep(5000);

    }
```

Καταγραφή εικόνων στον φάκελο

```
    usleep(4000);

    ptimer_stopstore(FEAT_TIME, counter, start, ptfile);
```

Τέλος Ptiming

```
}
```

// Τέλος main προγράμματος

Κώδικας ReferenceValueServer.c:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/fcntl.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>
#include <sys/time.h>

#include <unistd.h>
#include <termios.h>
```

Βιβλιοθήκες Κώδικα

```
static double roll_ref = 0;
static double pitch_ref = 0;
static double x_ref = 0;
static double y_ref = 0;
static double alt_ref = -1.1;
static double yaw_ref = 0;
static double pattern = 0;
double xwrite = 0;
double ywrite = 0;
```

Δήλωση και Αρχικοποίηση Μεταβλητών

```
char getch() {
        char buf = 0;
        struct termios old = {0};
        if (tcgetattr(0, &old) < 0)
                perror("tcsetattr()");
        old.c_lflag &= ~ICANON;
        old.c_lflag &= ~ECHO;
        old.c_cc[VMIN] = 1;
        old.c_cc[VTIME] = 0;
        if (tcsetattr(0, TCSANOW, &old) < 0)
                perror("tcsetattr ICANON");
        if (read(0, &buf, 1) < 0)
                perror ("read()");
        old.c_lflag |= ICANON;
        old.c_lflag |= ECHO;
        if (tcsetattr(0, TCSADRAIN, &old) < 0)
                perror ("tcsetattr ~ICANON");
        return (buf);
}
```

Συνάρτηση ανάγνωσης πλήκτρων από το πληκτρολόγιο

```
int main(int argc, char *argv[])
{
```

//Αρχή main προγράμματος

```
int listenfd = 0, connfd = 0;
struct sockaddr_in serv_addr;

char sendBuff[1025];
char recvBuff[100];
char keybch;
static int yes = 1;

static double SATU_angle   = 0.5;
static double SATU_alt_min = -0.3;
static double SATU_alt_max = -2.7;
int runcmd = 1;

double roll_ref_RS, pitch_ref_RS, x_ref_RS, y_ref_RS;


listenfd = socket(AF_INET, SOCK_STREAM, 0);
```

Δήλωση Μεταβλητών και Παραμέτρων Επικοινωνίας

```
setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes));

memset(&serv_addr, '0', sizeof(serv_addr));
memset(sendBuff, '0', sizeof(sendBuff));

serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(12345);
```

Δήλωση Μεταβλητών και Παραμέτρων Επικοινωνίας

```
if  (bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr))<0)
        {
        printf("Socket error (bind), closing all open ReferenceValueServers!\n");
        system("pkill -f DroneKeyboardPilot*");
        system("pkill -f ReferenceValueServer*");
        exit(1);
        }

printf("Waiting for connection to drone...\n");
if  (listen(listenfd, 10)<0)
        {
        printf("Socket error (listen), closing all open ReferenceValueServers!\n");
        system("pkill -f DroneKeyboardPilot*");
        system("pkill -f ReferenceValueServer*");
        exit(1);
        }

connfd = accept(listenfd, (struct sockaddr*)NULL, NULL);
setsockopt(connfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes));
```

Έξοδος σε περίπτωση αποτυχίας σύνδεσης

// Εμφάνιση Υπομνήματος για την λειτουργία του κάθε πλήκτρου

```
printf("\n d,a : roll\n w,s : pitch\n j,l : yaw\n i,k : alt 0.2\n y,h : alt 0.6\n f : coordinates
input\n z : pattern x-y\n x : pattern Euler\n q : reset position\n r : reset, e : exit\n");


while(runcmd==1)
{
  keybch = getch();
  switch (keybch)
  {
          case 'z': yaw_ref += 0; x_ref += 2; y_ref += 2; pattern = 1; break;
          case 'x': pitch_ref += 0.2; roll_ref += 0.1; yaw_ref += 1;
                    pattern = 2; break;
          case 'f': pitch_ref = 0; roll_ref = 0; yaw_ref = 0 ; x_ref = 0;
                    y_ref = 0; pattern = 3; break;
          case 'd': roll_ref += 0.05; break;
          case 'a': roll_ref -= 0.05; break;
          case 'w': pitch_ref -= 0.05; break;
          case 's': pitch_ref += 0.05; break;
          case 'i': alt_ref -= 0.2; break;
          case 'y': alt_ref -= 0.6; break;
          case 'k': alt_ref += 0.2; break;
          case 'h': alt_ref += 0.6; break;
          case 'j': yaw_ref -= 0.2; break;
          case 'l': yaw_ref += 0.2; break;
          case 'r': pitch_ref = 0.0; roll_ref = 0.0; break;
          case 'q': x_ref = 0; y_ref = 0; yaw_ref = 0; break;
          case 'e': runcmd = 0; break;
  }
```

Ανάγνωση πλήκτρων από τον χρήστη

```
        if (pattern == 1)
        {
          yaw_ref = 0;
          x_ref_RS = x_ref;
          sprintf(sendBuff,"%i %i %i %i %i %i %i",runcmd,(int)(pitch_ref_RS*1000 +
10000),(int)(roll_ref_RS*1000 + 10000), (int)(yaw_ref*1000 + 10000), (int)(x_ref_RS*100.0),
(int)(y_ref_RS*100.0), (int)(alt_ref*100.0));
          send(connfd, sendBuff, strlen(sendBuff),0);
          printf("x = %f meters \n", x_ref_RS);
          usleep(3500000);
```

‘ Π ’ με Χ-Υ

```
          y_ref_RS = y_ref;
          sprintf(sendBuff,"%i %i %i %i %i %i %i",runcmd,(int)(pitch_ref_RS*1000 +
10000),(int)(roll_ref_RS*1000 + 10000), (int)(yaw_ref*1000 + 10000), (int)(x_ref_RS*100.0),
(int)(y_ref_RS*100.0), (int)(alt_ref*100.0));
          send(connfd, sendBuff, strlen(sendBuff),0);
          printf("y = %f meters \n", y_ref_RS);
          usleep(3500000);

          x_ref_RS = -x_ref;
          sprintf(sendBuff,"%i %i %i %i %i %i %i",runcmd,(int)(pitch_ref_RS*1000 +
10000),(int)(roll_ref_RS*1000 + 10000), (int)(yaw_ref*1000 + 10000), (int)(x_ref_RS*100.0),
(int)(y_ref_RS*100.0), (int)(alt_ref*100.0));
          send(connfd, sendBuff, strlen(sendBuff),0);
          printf("x = %f meters \n", x_ref_RS);
          usleep(2000000);

          pattern = 0;
        }else if (pattern == 2)
        {
          roll_ref_RS = 0;
          pitch_ref_RS = pitch_ref;
          yaw_ref = 0;
          sprintf(sendBuff,"%i %i %i %i %i %i %i",runcmd,(int)(pitch_ref_RS*1000 +
10000),(int)(roll_ref_RS*1000 + 10000), (int)(yaw_ref*1000 + 10000), (int)(x_ref_RS*100.0),
(int)(y_ref_RS*100.0), (int)(alt_ref*100.0));
          send(connfd, sendBuff, strlen(sendBuff),0);
          usleep(1000000);

          pitch_ref_RS = 0;
          roll_ref_RS = 0;
          sprintf(sendBuff,"%i %i %i %i %i %i %i",runcmd,(int)(pitch_ref_RS*1000 +
10000),(int)(roll_ref_RS*1000 + 10000), (int)(yaw_ref*1000 + 10000), (int)(x_ref_RS*100.0),
(int)(y_ref_RS*100.0), (int)(alt_ref*100.0));
          send(connfd, sendBuff, strlen(sendBuff),0);
          usleep(3000000);

          roll_ref_RS = roll_ref;
          sprintf(sendBuff,"%i %i %i %i %i %i %i",runcmd,(int)(pitch_ref_RS*1000 +
10000),(int)(roll_ref_RS*1000 + 10000), (int)(yaw_ref*1000 + 10000), (int)(x_ref_RS*100.0),
(int)(y_ref_RS*100.0), (int)(alt_ref*100.0));
          send(connfd, sendBuff, strlen(sendBuff),0);
          usleep(1000000);

          roll_ref_RS = 0;
          sprintf(sendBuff,"%i %i %i %i %i %i %i",runcmd,(int)(pitch_ref_RS*1000 +
10000),(int)(roll_ref_RS*1000 + 10000), (int)(yaw_ref*1000 + 10000), (int)(x_ref_RS*100.0),
(int)(y_ref_RS*100.0), (int)(alt_ref*100.0));
          send(connfd, sendBuff, strlen(sendBuff),0);
          usleep(3000000);

          pattern = 0;
        }else if (pattern == 3)
        {
          printf("X = ? (-5 +5)\n");
          do
          {
          scanf("%lf", &xwrite);
          } while ( xwrite > 5 && xwrite < -5);
          do
          {
          printf("Y = ? (-5 +5)\n");
          scanf("%lf", &ywrite);
          } while ( ywrite > 5 && ywrite < -5);
          x_ref_RS = xwrite;
          y_ref_RS = ywrite;
          sprintf(sendBuff,"%i %i %i %i %i %i %i",runcmd,(int)(pitch_ref_RS*1000 +
10000),(int)(roll_ref_RS*1000 + 10000), (int)(yaw_ref*1000 + 10000), (int)(x_ref_RS*100.0),
(int)(y_ref_RS*100.0), (int)(alt_ref*100.0));
          send(connfd, sendBuff, strlen(sendBuff),0);
          usleep(3000000);

          pattern = 0;
```

‘ Π ’ με X-Y

‘ Γ ’ με Euler roll/pitch

Online επιλογή X - Y από το χρήστη

```
        }else
        {
          roll_ref_RS = roll_ref;
          pitch_ref_RS = pitch_ref;

          if (roll_ref_RS > SATU_angle) roll_ref_RS = SATU_angle;
          if (roll_ref_RS < -SATU_angle) roll_ref_RS = -SATU_angle;
          if (pitch_ref_RS > SATU_angle) pitch_ref_RS = SATU_angle;
          if (pitch_ref_RS < -SATU_angle) pitch_ref_RS = -SATU_angle;

          if (alt_ref > SATU_alt_min) alt_ref = SATU_alt_min;
          if (alt_ref < SATU_alt_max) alt_ref = SATU_alt_max;


          sprintf(sendBuff,"%i %i %i %i %i %i",runcmd,(int)(pitch_ref_RS*1000 +
10000),(int)(roll_ref_RS*1000 + 10000), (int)(yaw_ref*1000 + 10000), (int)(x_ref_RS*100.0),
(int)(y_ref_RS*100.0), (int)(alt_ref*100.0));
          send(connfd, sendBuff, strlen(sendBuff),0);
          usleep(10000);
        }
};
```

Ορισμός ορίων και εκτέλεση υπόλοιπων εντολών

```
if (shutdown(listenfd,2) || shutdown(connfd,2) || close(listenfd) || close(connfd)  )
                printf("ERROR: Shutdown Socket\n");
else
        printf("Shutdown Socket successful \n");

return 0;

}                                                    //Τέλος main προγράμματος
```

Έξοδος Διεργασίας

Βιβλιογραφία

- **Βιβλία - Έρευνες**

Σημειώσεις “Βιομηχανικοί Ελεγκτές”: Κωνσταντίνος Αλαφοδήμος, Μαρτιος 2015

“Συστήματα Αυτομάτου Ελέγχου Ι”: Δημήτριος Καλλιγερόπουλος, Σουλτάνα Βασιλειάδου, 2005

“Συστήματα Αυτομάτου Ελέγχου ΙΙ”: Δημήτριος Καλλιγερόπουλος, Σουλτάνα Βασιλειάδου, 2005

“Probabilistic Robotics”: Sebastian Thrun - Stanford University, Wolfram Budgard - University of Freiburg, Dieter Fox - University of Washington, 1999-2000

“Optical Flow Estimation” - David J. Fleet, Yair Weiss

“Robotics, Vision and Control Fundamentals - Fundamentals Algorithms in MATLAB”: Peter Corke

“Modelling and Control of Quadcopter” - Teppo Luukkonen Aalto University 2011

“Stabilization and Control of Unmanned Quadcopter” - Tomáš Jiinec Luleå University of Technology, Department of Computer Science, Electrical and Space Engineering

“Σχεδιασμος, Αναπτυξη και Υλοποιηση Τετρακοπτερου βασισμενο σε Raspberry Pi με Ελεγχο μεσω Smartphone” - Αποστολίδης Αριστείδης, Θεσσαλονίκη, Φεβρουάριος 2015

- **Links**

16.30-16.31 Feedback Control Systems: Sertac Karaman - http://peris.mit.edu/1630

http://fast.scripts.mit.edu

https://github.com/Parrot-Developers/RollingSpiderEdu

https://en.wikipedia.org/wiki/Quadcopter

https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle

https://en.wikipedia.org/wiki/Optical_flow

http://www.theuav.com

http://quadcopterarena.com

http://www.explainthatstuff.com/accelerometers.html

http://global.parrot.com/au/products/rolling-spider

http://expect.sourceforge.net

http://www.linuxjournal.com/article/2156

https://linuxprograms.wordpress.com/2008/02/14/fifo-named-pipes-mkfifo-mknod

http://www.mathworks.com/help/imaq/examples/live-motion-detection-using-optical-flow.html?requestedDomain=www.mathworks.com

https://www.mathworks.com/help/control/ug/kalman-filtering.html

https://www.ubuntu.com