ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

Σχεδιασμός και Υλοποίηση Παραμετροποιήσιμου Προσομοιωτή Μη Επανδρωμένων Εναέριων Οχημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΗΛΙΩΤΗΣ Α. ΣΤΕΦΑΝΟΣ

Επιβλέπων : Κωνσταντίνος Τζαφέστας
Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2019

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

Σχεδιασμός και Υλοποίηση Παραμετροποιήσιμου Προσομοιωτή Μη Επανδρωμένων Εναέριων Οχημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΗΛΙΩΤΗΣ Α. ΣΤΕΦΑΝΟΣ

Επιβλέπων : Κωνσταντίνος Τζαφέστας
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9η Δεκεμβρίου 2019.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Κωνσταντίνος Τζαφέστας	Χαράλαμπος Ψυλλάκης	Κωνσταντίνος Κυριακόπουλος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2019

(Υπογραφή)

.....

ΜΗΛΙΩΤΗΣ Α. ΣΤΕΦΑΝΟΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η συγκεκριμένη διπλωματική εργασία αφορά την ανάπτυξη ενός προσομοιωτή ιπτάμενων ρομπότ, όπως fixed-wing και multirotors, βασισμένο σε εργαλεία ελεύθερου λογισμικού.

Ο προσομοιωτής ονομάζεται Last_letter_2 και αποτελεί ένα νέο πακέτο του ROS σε συνεργασία με το Gazebo. Ο Last_letter_2 δημιουργήθηκε για να δώσει την δυνατότητα στους ερευνητές ιπτάμενων ρομπότ, να κατασκευάζουν εύκολα τα μοντέλα τους, να προγραμματίζουν τους αλγορίθμους ελέγχου τους και να πραγματοποιούν γρήγορα και κυρίως αξιόπιστα τις αναγκαίες προσομοιώσεις πριν προχωρήσουν στις πειραματικές πτήσεις.

Το κυριότερο και μεγαλύτερο μέρος του προσομοιωτή είναι γραμμένο στο ROS. Στο Gazebo οπτικοποιούνται τα αποτελέσματα της προσομοίωσης και λύνεται η κινηματική του μοντέλου χάρη στην μηχανή φυσικής του Gazebo. Αυτό καθιστά τον Last_letter_2 ένα σχεδόν ολοκληρωμένο και αυτόνομο ROS πακέτο που μπορεί να συνεργαστεί και με άλλους ρομποτικούς προσομοιωτές γενικής χρήσης πέραν του Gazebo.

Σε αυτήν την εργασία εξηγείται όλη η λογική του προσομοιωτή, η αρχιτεκτονική του, τα default μοντέλα και controllers που διαθέτει, ενώ δίνονται όλες οι απαραίτητες οδηγίες για την χρήση και ανάπτυξή του.

Λέξεις Κλειδιά: <<UAV, flight simulator, ROS, Gazebo, aerial, fixed-wing, quadcopter, hexacopter, multirotor, gazebo plugin, controller>>

Abstract

This thesis concerns the development of a simulator for flight models, such as fixed-wing and multirotors, based on open source tools.

The simulator is called Last letter 2 and is a ROS package combined with Gazebo. The Last letter 2 was created in order to help flight robot researchers build their models, program the control algorithms and preform quick and reliable useful simulations before the experimental tests.

The main part of simulator is written in ROS. The Gazebo helps in visualizing the results of simulation and in solving the model kinematics thanks to its physics machine. That makes Last letter 2 an almost complete and autonomous system that can work with many other robotic simulations, not only Gazebo.

In this thesis is explained the idea of Last letter 2, the architecture, the default models and controllers it has. In addition, all the helpful instructions for the use and development are offered too.

Key words: <<UAV, flight simulator, ROS, Gazebo, aerial, fixed-wing, quadcopter, hexacopter, multirotor, gazebo plugin, controller>>

Πίνακας περιεχομένων

Α' Μέρος

Παρουσίαση Εφαρμογής

1

Εισαγωγή

1.1 Ρομποτικοί Προσομοιωτές

Στην έρευνα των ρομπότ, οι προσομοιωτές αποτελούν απαραίτητα εργαλεία. Οι ερευνητές πριν δοκιμάσουν τους νέους αλγορίθμους στα υπαρκτά ρομπότ, τους ελέγχουν σε εικονικά ρομπότ μέσω των προσομοιωτών. Αυτό αφενός εξοικονομεί χρόνο και κόστος, αφετέρου προσφέρει ασφάλεια καθώς πιθανά λάθη στον πραγματικό κόσμο μπορούν να εκθέσουν σε κίνδυνο τον άνθρωπο.

Στις μέρες μας, λόγω του μεγάλου φάσματος χρήσης των ρομπότ έχουν αναπτυχθεί προσομοιωτές τόσο γενικής χρήσης όσο και ειδικής, που εστιάζουν δηλαδή σε κάποιο μόνο είδος ρομπότ, προσπαθώντας να καλύψουν κάθε πιθανή ανάγκη. Όσον αφορά τους γενικής χρήσης προσομοιωτές ρομπότ, δίνουν την δυνατότητα στους χρήστες να κατασκευάσουν τα δικά τους μοντέλα ρομπότ, να δημιουργήσουν τα δικά τους περιβάλλοντα και να πραγματοποιήσουν την προσομοίωση. Οι δυνάμεις που αναπτύσσονται και οι κινήσεις των ρομπότ υπολογίζονται από τις πανίσχυρες μηχανές φυσικής που διαθέτουν όπως οι ODE, Bullet, Simbody, DART, Open Dynamics engine και Vortex Studio engine. Οι προσομοιωτές αυτοί παρέχουν παράλληλα και ένα πλήθος εργαλείων προσπαθώντας να κάνουν την εμπειρία του χρήστη ευκολότερη και πιο ποιοτική. Οι επεκτάσεις αυτές αφορούν την αλληλεπίδραση του χρήστη με το γραφικό περιβάλλον, τον τρόπο περιγραφής του ρομπότ, βοηθητικές βιβλιοθήκες για την συγγραφή των controller και άλλα. Οι προσομοιωτές γενικής χρήσης ρομπότ χρησιμοποιούνται κυρίως για προσομοιώσεις manipulator και mobile ρομπότ. Ωστόσο, συναντά κανείς και ορισμένα παραδείγματα εφαρμογών με ιπτάμενα, όπως multirotors και fixed-wing. Τέσσερις από τους πιο ευρέως διαδεδομένους προσομοιωτές γενικής χρήσης είναι οι:

- Gazebo
- Morse
- Webots
- V_REP

Όσον αφορά του προσομοιωτές ειδικής χρήσης θα εστιάσουμε σε αυτούς που προσομοιώνουν ιπτάμενα ρομπότ. Η προσομοίωση πτήσης διαφέρει κάπως από την προσομοίωση ενός επίγειου ρομπότ. Τα ιπτάμενα μοντέλα οφείλουν τις κινήσεις τους κυρίως στην αλληλεπίδρασή τους με τον αέρα και τις δυνάμεις που αναπτύσσονται εξαιτίας αυτού, δυνάμεις σχεδόν αμελητέες για τα περισσότερα ρομπότ καθώς οι κινήσεις τους είναι περιορισμένες στον χώρο και οι ταχύτητές του μικρές. Όσο πιο καλός ο προσομοιωτής τόσο πιο κοντά σε πραγματική πτήση θα είναι η προσομοίωση. Ένας προσομοιωτής πτήσης επιτρέπει τον ασφαλή έλεγχο ενός πειραματικού κώδικα και ρυθμίσεων κάποιου μοντέλου, ενώ βοηθάει στην εξοικονόμηση χρόνου και χρημάτων καθώς μια πτώση στον προσομοιωτή στοιχίζει ... λιγότερο!

Ορισμένοι από τους πιο ευρέως διαδεδομένους προσομοιωτές[1] πτήσης είναι οι εξής:

- hector_quadrotor
- X Plane -11
- FlightGear
- RealFlight
- CRRCSim
- jMAVSim
- AirSim

1.2 Αντικείμενο διπλωματικής

Η συγκεκριμένη διπλωματική εργασία αφορά την κατασκευή ενός νέου προσομοιωτή πτήσης, του Last Letter 2. Ο Last Letter 2 είναι ένας ερευνητικός προσομοιωτής Μη Επανδρωμένων Αεροσκαφών Μικρής Κλίμακας (fixed-wing & multirotors), βασισμένος σε εργαλεία ελεύθερου λογισμικού, για προσομοίωση νέων αλγορίθμων ελέγχου πάνω σε οποιοδήποτε υπαρκτό ή πειραματικό μοντέλο ιπτάμενου ρομπότ.

Ο Last letter 2 δίνει την δυνατότητα στον χρήστη:

- να κατασκευάσει κάθε γεωμετρία μοντέλου αποτελούμενη από κομμάτια μάζας, αεροδυναμικές επιφάνειες (πτερύγια) και κινητήρες ώθησης
- να προγραμματίσει τον αλγόριθμο ελέγχου (controller) που θα ελέγχει το μοντέλο
- να ελέγχει την προσομοίωση στέλνοντας σήματα μέσω joystick εν ώρα πτήσης
- να παρακολουθεί σε πραγματικό χρόνο την πτήση του μοντέλου
- να έχει πρόσβαση στα δεδομένα της πτήσης, καθώς και να μπορεί να τα αποθηκεύσει για μελλοντική επεξεργασία
- να ρυθμίζει τα κέρδη του ελεγκτή σε πραγματικό χρόνο πτήσης

[1] Λεπτομέρειες στο Παράρτημα Α.

2

Δήλωση Τεχνικού Προβλήματος

Αν και υπάρχουν και άλλες εφαρμογές που αφορούν την προσομοίωση πτήσεων μοντέλων, ήταν ανάγκη να δημιουργηθεί ο Last letter 2.

Πολλοί από τους προσομοιωτές πτήσης είναι εμπορικοί, δηλαδή όχι ελεύθερου λογισμικού. Κάποιοι εστιάζουν μόνο σε εμπορικούς flight controllers και όχι σε ερευνητικούς. Άλλοι προσφέρουν περιορισμένο αριθμό μοντέλων, ενώ δεν δίνουν την επιλογή κατασκευής μοντέλου από τον χρήστη. Άλλοι εστιάζουν σε πολύ συγκεκριμένες τεχνικές control, όπως machine learning, visual control, εμποδίζοντας την ανάπτυξη διαφορετικής φύσεως αλγορίθμων ελέγχου.

Από την άλλη, η ανάπτυξη ορισμένων plugins σχετικών με την προσομοίωση ιπτάμενων ρομπότ σε έναν από τους προσομοιωτές γενικής χρήσης του προηγούμενου κεφαλαίου, θα αποτελούσε κάτι ειδικό και θα περιόριζε τους χρήστες στα πλαίσια του συγκεκριμένου μόνο προσομοιωτή.

Η ανάγκη δημιουργίας του Last letter 2 προήλθε από την ιδέα ύπαρξης ενός συστήματος προσομοίωσης σχετικά αυτόνομου, εύκολα κατανοητού και τροποποιήσιμου σύμφωνα με τις όποιες ερευνητικές ανάγκες, που θα μπορεί να συνεργαστεί με περισσότερους από έναν προσομοιωτές γενικής χρήσης.

3

Σχεδίαση Συστήματος – Τεχνική προσέγγιση

3.1 Γιατί ROS - Gazebo

Σε αυτό το κεφάλαιο περιγράφεται ο τρόπος λειτουργίας του προσομοιωτή, αφού πρώτα εξηγηθεί η λογική πάνω στην οποία βασίζεται και ο ρόλος κάθε εργαλείου που χρησιμοποιείται. Ο προσομοιωτής βασίζεται πάνω σε δύο συστήματα ελεύθερου λογισμικού, το ROS και το Gazebo.

Ο Gazebo[2] είναι ένας ευρέως διαδεδομένος και ισχυρός προσομοιωτής ρομπότ ελεύθερου λογισμικού. Περιέχει τεράστια γκάμα επιλογών μοντέλων και αισθητήρων προσφέροντας αξιόπιστες προσομοιώσεις σε πολύπλοκα περιβάλλοντα τόσο εσωτερικού όσο και εξωτερικού χώρου. Οι δυνατές μηχανές φυσικής που χρησιμοποιεί για την επίλυση της κινηματικής των μοντέλων είναι άλλο ένα πλεονέκτημά του, όπως επίσης και η δυνατότητα που προσφέρει στον χρήστη για πλήρη έλεγχο τόσο των μοντέλων όσο και του κόσμου της προσομοίωσης. Τέλος συνεργάζεται με πολλά άλλα εργαλεία όπως είναι και το ROS. Παρά όμως αυτά τα πλεονεκτήματα, υστερεί στην προσομοίωση ιπτάμενων ρομπότ, καθώς δεν προσομοιώνει τον αέρα, ούτε και την κίνηση των μοντέλων λόγω της αλληλεπίδρασή τους με αυτόν.

Το ROS[3] (Robot Operating System), είναι λειτουργικό σύστημα για ρομπότ. Χρησιμοποιείται ευρέως για τον προγραμματισμό και έλεγχο ρομπότ, κυρίως mobile και manipulators. Οι δυνατότητες που προσφέρει το καθιστούν από τα πρώτα σε προτίμηση λειτουργικά για έλεγχο ρομπότ. Ένα σύστημα σε ROS, είναι χωρισμένο σε πολλά ανεξάρτητα nodes, κάνοντας τον κώδικα εύχρηστο και ευανάγνωστο. Κάθε node έχει τον δικό του σκοπό, ενώ όλα μπορούν να επικοινωνούν μεταξύ τους μέσω μηνυμάτων που στέλνουν, είτε σύγχρονα (services) είτε ασύγχρονα (topics). Συνεργάζεται εύκολα με άλλα λογισμικά για ρομπότ, όπως το Gazebo, και προσφέρει επιλογές στην γλώσσα προγραμματισμού των nodes (Python, C++ και Lisp μέχρι στιγμής). Παράλληλα, πίσω από το ROS υπάρχει μια τεράστια κοινότητα που

[2] Βλ. Παράρτημα

[3] Βλ. Παράρτημα

το αναβαθμίζει ή χρησιμοποιεί κάνοντας την χρήση του, ιδιαίτερα στην αρχή, αρκετά εύκολη. Με λίγα λόγια το ROS είναι ένα κατάλληλο λειτουργικό για ανάπτυξη μεγάλων και περίπλοκων συστημάτων που αφορούν ρομπότ.

Έτσι επιλέχθηκε το ROS ως η βάση του προσομοιωτή ώστε να οργανωθεί σωστά και ευδιάκριτα, μιας και αποτελεί ένα πανίσχυρο εργαλείο στον προγραμματισμό των ρομπότ. Αυτό κάνει ακόμα ευκολότερη την χρήση του σε ήδη γνώστες του ROS, ενώ επιτρέπει την πραγματοποίηση αλλαγών σε οποιοδήποτε σημείο του κώδικα. Σκοπός είναι οι χρήστες έναν κατανοήσουν την λειτουργία του να μπορούν να τον τροποποιήσουν σύμφωνα με τις ανάγκες τους. Το Gazebo επιλέχθηκε για τους εξής λόγους. Αποτελεί έναν από τους πιο διαδεδομένους προσομοιωτές ρομπότ και είναι γνωστό σε μεγάλη ομάδα χρηστών, διαθέτει πανίσχυρες μηχανές φυσικής με αποτέλεσμα να μην χρειάζεται να υπολογιστεί η κινηματική του μοντέλου στο ROS και υπάρχει αρκετό documentation για την συνεργασία ROS-Gazebo διαθέσιμο, κάνοντας την ανάπτυξη αλλά και την μελλοντική τροποποίηση της εφαρμογής ευκολότερη.

3.2 Εξήγηση λογικής

Όλη η δυναμική του μοντέλου υπολογίζεται στο ROS. Εκεί τρέχει και ο αλγόριθμος του αλγορίθμου ελέγχου που παράγει τα σήματα εισόδου στο μοντέλο. Το Gazebo είναι υπεύθυνο να ασκεί τις υπολογισμένες δυνάμεις και ροπές στο μοντέλο, να υπολογίζει την επόμενη κατάσταση του μοντέλου λύνοντας τις εξισώσεις κινηματικής με την μηχανή φυσικής που διαθέτει και να ενημερώνει το ROS με την νέα κατάσταση του μοντέλου. Τα δύο συστήματα συνεργάζονται άριστα και συγχρονισμένα σε κάθε κύκλο προσομοίωσης.

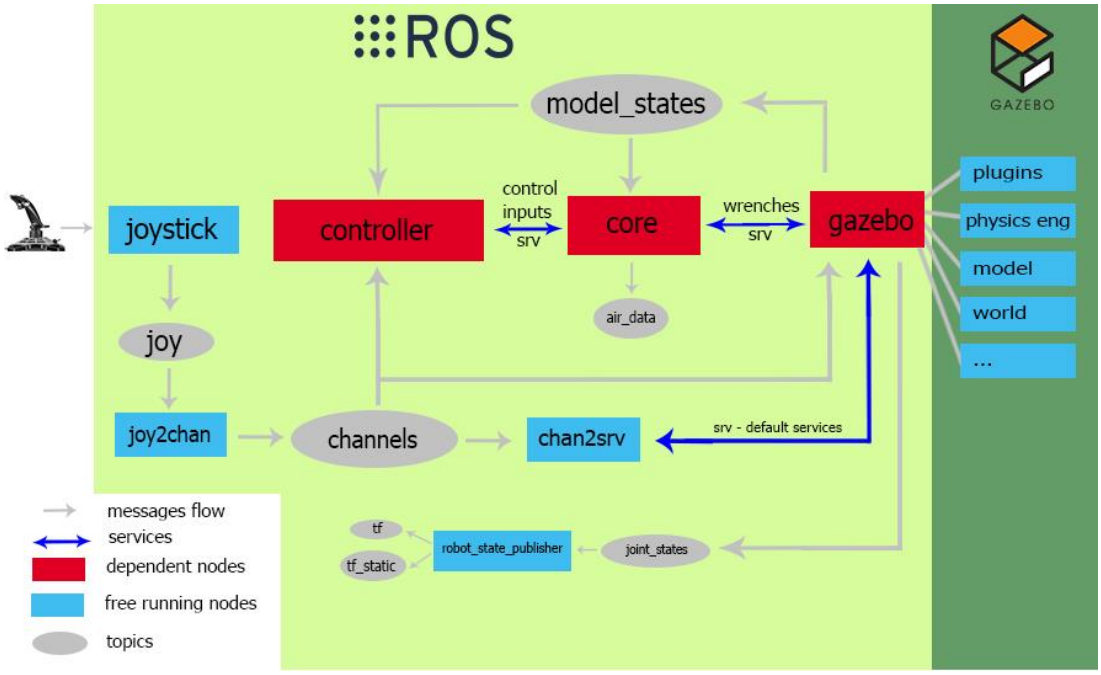
3.3 Τι κάνει ξεχωριστό τον Last letter 2

Ο Last letter 2 αποτελεί κάτι ξεχωριστό και δίνει στους χρήστες αρκετή ελευθερία επιλογών. Η αντιμετώπιση του μοντέλου ως συνδυασμός πτερυγίων, κινητήρων και άλλων μερών, επιτρέπουν στον χρήστη να κατασκευάσει fixed-wing αεροπλάνα, multirotor μοντέλα και ο,τι άλλο μπορεί να σχεδιαστεί από αυτά τα μέρη. Ο χρήστης επιλέγει με μεγάλη ακρίβεια την θέση κάθε μέλους και τις ιδιότητές του. Με πολύ εύκολο τρόπο μπορεί να προγραμματίσει και τον αλγόριθμο ελέγχου του μοντέλου, αφού γράφεται και τρέχει σε συγκεκριμένο μέρος του συστήματος. Παράλληλα ο Last Letter 2 είναι ένας απόλυτα συγχρονισμένος και αξιόπιστος προσομοιωτής, χάρη στην τεχνική Step lock[4] που εφαρμόστηκε. Η σειρά με την οποία τρέχουν οι εντολές είναι η ίδια σε κάθε βήμα προσομοίωσης, παρά τυχόν καθυστερήσεις. Το Gazebo τρέχει ανεξάρτητα από το ROS. Για συχνότητες όπου οι υπολογισμοί στο ROS χρειάζονται περισσότερο χρόνο από ένα βήμα προσομοίωσης, ο προσομοιωτής δεν συνεχίζει προσπερνώντας βήματα αλλά περιμένει. Αυτό βέβαια επιβραδύνει την προσομοίωση αλλά δεν χάνει σε αξιοπιστία. Ο κώδικας του προσομοιωτή είναι δομημένος με τρόπο ώστε ο χρήστης, αποκτώντας λίγη εμπειρία να μπορεί να επέμβει, τροποποιώντας οποιοδήποτε σημείο ώστε να πετύχει επιπλέον δυνατότητες ανάλογα με τις ανάγκες του. Παράλληλα, προσφέρει στον χρήστη και πολλές από τις επεκτάσεις των πανίσχυρων εργαλείων που βασίζεται, όπως το Rviz και το rqt του ROS για οπτικοποίηση των σημάτων από τους αισθητήρες και προβολή των δεδομένων, dynamic_reconfigure για την ρύθμιση παραμέτρων κατά την διάρκεια της προσομοίωσης (χρήσιμο για control tuning) ή την καταγραφή την προσομοίωσης σε αρχεία log μέσω του Gazebo. Τέλος, έχοντας αναπτυχθεί στο ROS (στο Gazebo έχουν προστεθεί οι ελάχιστες γραμμές κώδικα αναγκαίες για την επικοινωνίας τους), αποτελεί ένα ολοκληρωμένο και σχεδόν αυτόνομο πακέτο

[4] Βλ. Παράρτημα

προσομοίωσης ιπτάμενων ρομπότ με σκοπό να μπορεί να συνεργαστεί με οποιονδήποτε άλλον προσομοιωτή γενικής χρήσης σαν το Gazebo, επιτρέποντας σε έμπειρους χρήστες να επιλέξουν σύμφωνα με τις ανάγκες τους.

3.4 Αρχιτεκτονική



Εικόνα 1. Αρχιτεκτονικό διάγραμμα nodes – topics

Ο προσομοιωτής τρέχει ανάμεσα σε δύο χώρους, του ROS και του Gazebo. Σε κάθε βήμα της προσομοίωσης αυτοί οι δύο χώροι επεξεργάζονται, παράγουν και ανταλλάσουν δεδομένα. Για την αξιοπιστία της προσομοίωσης ο ένας χώρος περιμένει δεδομένα από τον άλλο χώρο, όπου είναι αναγκαίο.

Στην μεριά του ROS υπολογίζεται όλη η δυναμική του μοντέλου. Οι δυνάμεις αυτές εξάγονται βασισμένες σε θεωρητικούς τύπους υπολογισμού των αεροδυναμικών δυνάμεων και ροπών του μοντέλου, λαμβάνοντας υπ' όψιν ταχύτητες, προσανατολισμό και τη θέση του μοντέλου στο χώρο, δεδομένα που έρχονται από την μεριά του Gazebo, καθώς επίσης και σήματα ελέγχου προερχόμενα από το node του controller, το οποίο τρέχει επίσης στην μεριά του ROS. Οι υπολογισμένες αυτές δυνάμεις στέλνονται με την σειρά τους στο Gazebo, όπου ασκούνται στα σωστά μέρη του μοντέλου, πλησιάζοντας με αυτόν το τρόπο όσο γίνεται πιο κοντά σε μια ρεαλιστική προσομοίωση, και έπειτα ανανεώνεται κατά ένα βήμα η προσομοίωση. Αυτό προϋποθέτει τον υπολογισμό της κινηματικής του μοντέλου για την επόμενη χρονική στιγμή, πράγμα που επιτελεί η μηχανή φυσικής του Gazebo. Τέλος, το Gazebo, στέλνει τα νέα δεδομένα που εξάγονται από την μηχανή φυσικής και περιγράφουν την κατάσταση του μοντέλου πίσω στο ROS, ώστε να χρησιμοποιηθούν για τον υπολογισμό του επόμενου βήματος και να επαναληφθεί η διαδικασία.

Η παραπάνω διαδικασία είναι συγχρονισμένη, δηλαδή όσο τρέχουν οι βασικοί υπολογισμοί από την μεριά του ROS, στο Gazebo υπάρχει αναμονή και το αντίστροφο. Αυτό συμβαίνει για να υπάρχει αξιοπιστία και πλήρης έλεγχος των βημάτων της προσομοίωσης, καθώς τα δύο μέρη τρέχουν τελείως ανεξάρτητα. Απλούστερα δηλαδή, αφού το ROS και το Gazebo δεν ακολουθούν κοινό ρολόι, ο μόνος τρόπος να συγχρονιστούν είναι να περιμένει το ένα το άλλο.

3.4.1 Επεξήγηση της Αρχιτεκτονικής

Στο ROS τρέχουν επτά ROS nodes , τρία εξαρτημένα από μεταβλητές του συστήματος και άλλα τέσσερα αυτόνομα. Στο Gazebo, ως αυτόνομο σύστημα τρέχουν διάφορες διεργασίες. Η επικοινωνία των δύο συστημάτων επιτυγχάνεται μέσω του *gazebo_n*[5]. Αυτό αποτελεί την διεπαφή μεταξύ τους, και είναι υπεύθυνο για την ανταλλαγή των μηνυμάτων τους. Κάνει ορατή την ύπαρξη του ενός στο άλλο σύστημα. Τα εξαρτημένα nodes είναι τα *controller_n*, *core_n* και *gazebo_n*. Κάθε στιγμή τρέχει μόνο ένα από τα τρία, ενώ τα υπόλοιπα περιμένουν, και αυτό για να επιτυγχάνεται αξιοπιστία και ακρίβεια στα βήματα της προσομοίωσης. Αυτά τα τρία nodes αποτελούν και τον πυρήνα του προσομοιωτή[6].

Επιπλέον, τρέχουν παράλληλα και ασύγχρονα δύο ακόμη nodes. Το *joy2chan_n* και το *chan2srv_n*. Το πρώτο είναι υπεύθυνο να δημιουργεί τον πίνακα καναλιών με τα σήματα που στέλνει ο χρήστης μέσω του joystick. Το δεύτερο εξυπηρετεί στην άμεση κλήση κάποιων λειτουργιών του Gazebo.

Τέλος, ο προσομοιωτής κάνει χρήση και δύο έτοιμων nodes, από πακέτα του ROS. Αυτά είναι το *joystick_n* και το *robot_state_publisher_n*. Το *joystick_n* συνδέει κάθε joystick με το ROS, περιγράφοντας τους άξονες και τα κουμπιά του joystick σε μορφή κατανοητή για το ROS, ενώ το *robot_state_publisher_n* εξάγει σε κάθε βήμα, βασισμένο στις γωνίες των αρθρώσεων του μοντέλου, το δέντρο μετασχηματισμών μεταξύ των πλαισίων συντεταγμένων του μοντέλου.

3.4.2 Nodes

Αναλυτική εξήγηση κάθε node του προσομοιωτή.

joy2chan_n

Εικόνα 2. joy2chan node

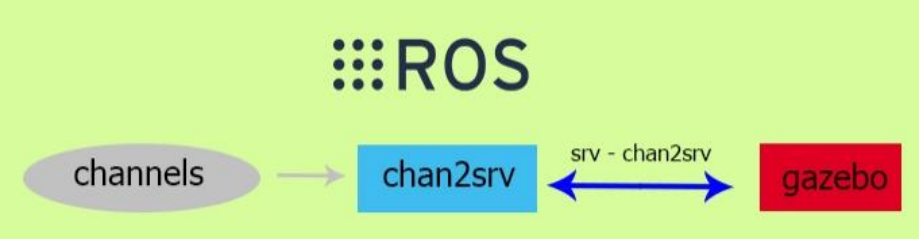
Διαβάζει από το topic */joy*[7] τις οδηγίες που στέλνει ο χρήστης μέσω του joystick και δημιουργεί το πίνακα καναλιών για να διαβαστεί από τα επόμενα nodes. Αυτό το ROS node είναι αυτόνομο, δηλαδή καλείται ασύγχρονα κάθε φορά που υπάρχουν νέα δεδομένα από το joystick, και δεν επηρεάζει την λειτουργία του υπόλοιπου συστήματος.

[5] Όλα τα node συμβολίζονται με την μορφή: *nodeName_n*

[6] Βλ. step lock στο Παράρτημα

[7] Όλα τα topic συμβολίζονται με την μορφή: */topicName*

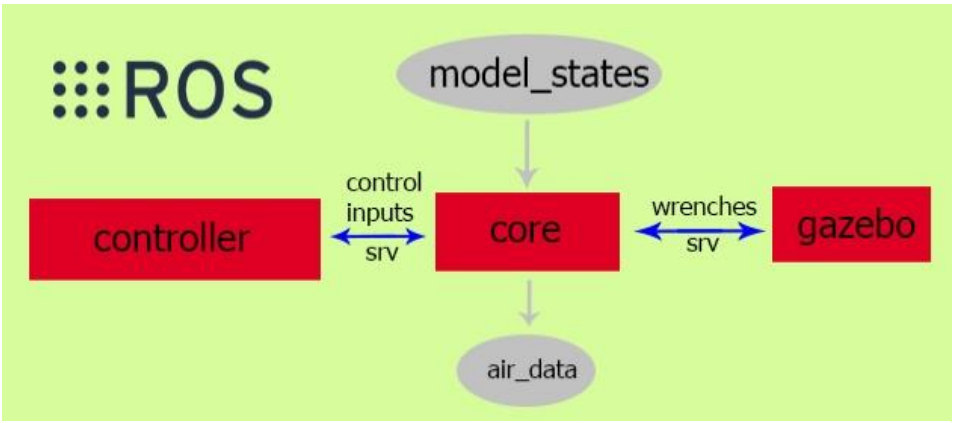
chan2srv_n



Εικόνα 3. chan2srv node

Διαβάζει από το topic */channels* τα διαθέσιμα κανάλια για ενδεχόμενες επιπλέον λειτουργίες μέσω των υπόλοιπων καναλιών, πέραν των σημάτων ελέγχου, που αφορούν τον κόσμο του Gazebo, όπως η εισαγωγή και η διαγραφή αντικειμένων στο κόσμο της προσομοίωσης. Οι επιπλέον αυτές λειτουργίες, εξυπηρετούνται μέσω έτοιμων services του Gazebo. Η εξυπηρέτηση των λειτουργιών αυτών γίνεται ασύγχρονα καθώς δεν προκαλείται πρόβλημα στην σωστή και αξιόπιστη λειτουργία της προσομοίωσης, εάν εξυπηρετηθούν σε άλλον κύκλο από αυτόν που καλούνται.

core_n



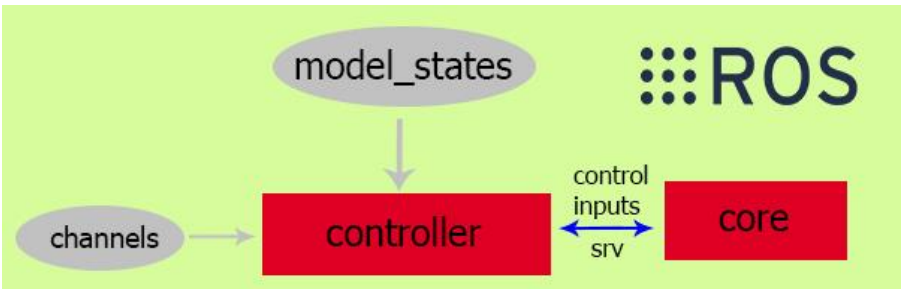
Εικόνα 4. core node

Η καρδία του προσομοιωτή. Διαβάζει από το topic */model_states* δεδομένα σχετικά με την κατάσταση του μοντέλου, ενώ επικοινωνεί τόσο με το *controller_n* όσο και με το *gazebo_n* μέσω services, για σύγχρονη και ταχύτατη επικοινωνία. Από το topic */model_states*, παίρνει δεδομένα σχετικά με την κατάσταση του μοντέλου και σε συνδυασμό με τα σήματα ελέγχου που ζητά από τον controller, υπολογίζει τις ανάλογες δυνάμεις και ροπές που αναπτύσσονται πάνω σε κάθε πτερύγιο και κινητήρα, όπως lift, drag, thrust, torque. Όταν τελειώσουν οι υπολογισμοί αυτοί, τα αποτελέσματα στέλνονται στο Gazebo (μέσω service call[8]), και το *core_n*, περιμένει να διαβάσει τα νέα δεδομένα από το topic */model_states*, τα οποία θα είναι έτοιμα μόλις το Gazebo ολοκληρώσει το επόμενο βήμα προσομοίωσης. Αξίζει να σημειωθεί πως καθώς γίνονται οι υπολογισμοί των αεροδυναμικών δυνάμεων, το Gazebo παραμένει κολλημένο, περιμένοντας το ROS να τελειώσει και να του στείλει τα δεδομένα.

Σε αυτό το node, υπολογίζονται επίσης και τιμές μεταβλητών σχετικών με το περιβάλλον στο οποίο γίνεται η προσομοίωση, όπως άνεμος, θερμοκρασία, πίεση, πυκνότητα ανέμου, εξίσου σημαντικές για τους παραπάνω υπολογισμούς. Τα δεδομένα αυτά δημοσιεύονται παράλληλα στο topic */air_data* για πιθανή μελλοντική χρήση.

[8] Βλ. Παράρτημα

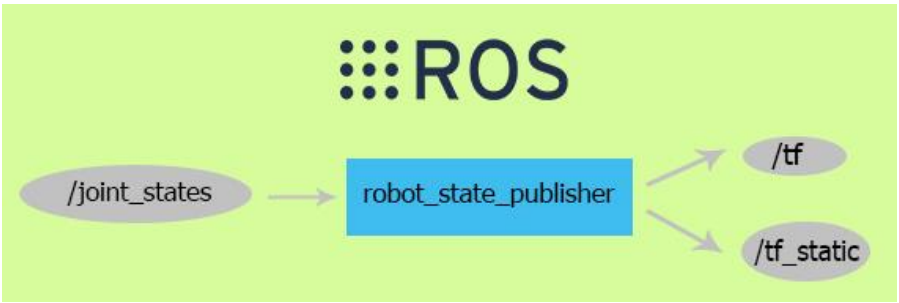
controller_n



Εικόνα 5. controller node

Αυτό είναι το node, στο οποίο ο χρήστης προγραμματίζει τον controller του μοντέλου. Διαβάζει από το topic */channels* τις εισόδους που δίνει ο χρήστης και από το */model_states* δεδομένα σχετικά με την κατάσταση του μοντέλου, δηλαδή τη θέση, τον προσανατολισμό, τη γραμμική και γωνιακή ταχύτητα κάθε μέλους του μοντέλου. Έπειτα, όταν το *core_n* του ζητήσει τα νέα control inputs, τα υπολογίζει με βάση τον τύπο του controller που του έχει οριστεί, και τα επιστρέφει μέσω service πίσω στο *core_n*. Πριν ξεκινήσει όμως να υπολογίζει τα νέα control inputs, βεβαιώνεται πως έχει στην διάθεσή τους τα πιο πρόσφατα model_states. Σε αντίθετη περίπτωση περιμένει μέχρι να τα αποκτήσει και μετά συνεχίζει. Αξίζει να σημειωθεί πως το node αυτό, φορτώνει τα νέα control inputs στις πρώτες θέσεις του πίνακα καναλιών.

robot_state_publisher_n



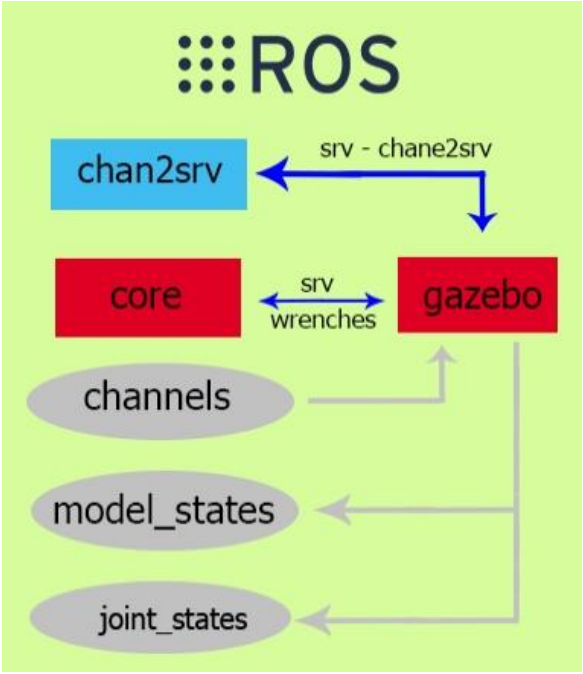
Εικόνα 6. robot_state_publisher node

Ένα έτοιμο πακέτο του ROS που επιτρέπει την δημοσίευση της κατάστασης του ρομπότ στην βιβλιοθήκη tf. Διαβάζει από το topic */joint_states* τις καταστάσεις των μεταβλητών αρθρώσεων του ρομπότ και υπολογίζει τις θέσεις των διάφορων μερών του ρομπότ. Το δέντρο της βιβλιοθήκης tf που δημιουργείται δεν παίζει κάποιο ρόλο στην προσομοίωση. Χρησιμοποιείται όμως από το Rviz[9], όπου για να αναπαρασταθεί το μοντέλο πρέπει να είναι γνωστές οι σχέσεις μεταξύ των link.

[9] Βλ. Κεφάλαιο 6.1

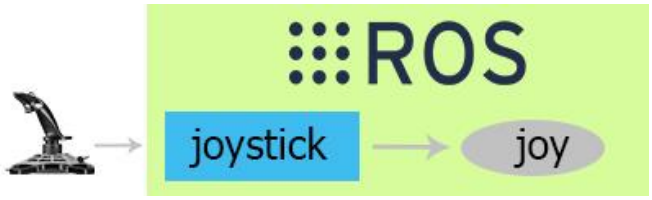
gazebo_n

Η επικοινωνία του ROS με το Gazebo επιτυγχάνεται μέσω του gazebo_n[10], που δημιουργείται από το πακέτο gazebo_ros, ένα χρήσιμο πακέτο για την συνεργασία του ROS με το Gazebo. Με αυτόν το τρόπο επιτρέπεται στον χρήστη να επικοινωνεί με τα gazebo plugin και να ελέγχει το περιβάλλον προσομοίωσης και το μοντέλο. Στο Gazebo το κύριο plugin για έλεγχο του μοντέλου και της μηχανής φυσικής είναι το model_plugin. Το plugin αυτό κρατάει παγωμένη την μηχανή φυσικής του Gazebo για όσο χρόνο υπολογίζεται η δυναμική στο *core_n*. Έπειτα λαμβάνει και εφαρμόζει τις δυνάμεις, επιτρέπει στην μηχανή φυσικής να τρέξει το επόμενο βήμα προσομοίωσης και ανανεώνει τις τιμές των topic */model_states* και */joint_states*. Η δημοσίευση των νέων model_states καλεί στο *core_n* την διαδικασία υπολογισμού της δυναμικής του επόμενου βήματος. Έτσι ξεκινάει πάλι το ROS και το Gazebo περιμένει. Το *gazebo_n* επικοινωνεί μέσω service και με το *chan2srv_n*, από το οποίο λαμβάνει τα αιτήματα στα default Gazebo services. Τέλος διαβάζει και τον πίνακα καναλιών από το topic */channels*.



Εικόνα 7. gazebo node

joystick_n



Εικόνα 8. joystick node

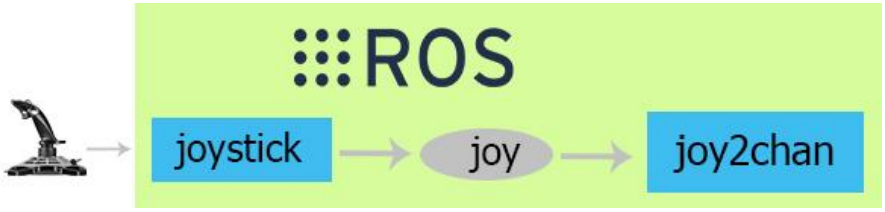
Ένα node του έτοιμου ROS πακέτου Joy, που συνδέει ένα γενικό Linux joystick με το ROS. Τα παραγόμενα αποτελέσματα τα γράφει στο /*joy* topic.

[10] Περισσότερες λεπτομέρειες για το *gazebo_n* υπάρχουν στο site του Gazebo: http://gazebosim.org/tutorials/?tut=ros_comm.

3.4.3 Topics

Παρακάτω αναλύεται το περιεχόμενο[11] των topics που χρησιμοποιούνται στον προσομοιωτή.

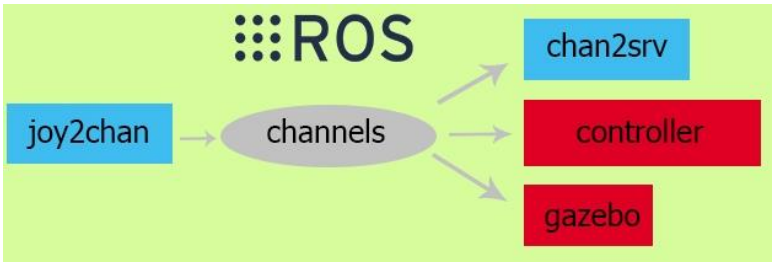
/joy



Εικόνα 9. joy topic

Σε αυτό το topic δημοσιεύεται η πιο πρόσφατη κατάσταση για κάθε έναν από τους άξονες και τα κουμπιά του joystick. Σε αυτό γράφει το *joystick_n* και διαβάζει το *joy2chan_n*. Τα δεδομένα του είναι εκφρασμένα σε μορφή sensor_msgs/Joy.

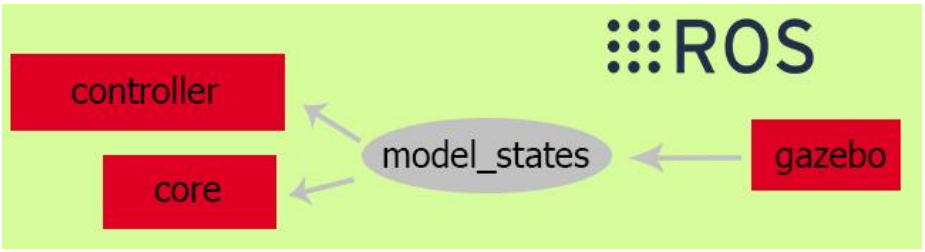
/channels



Εικόνα 10. channels topic

Εδώ γράφει το *joy2chan_n* και διαβάζουν τα *controller_n, chan2srv_n* και *gazebo_n*. Περιέχει τον πίνακα καναλιών με τα σήματα που στέλνει ο χρήστης μέσω του joystick. Τα δεδομένα του είναι εκφρασμένα σε μορφή last_letter_2_msgs/joystick_input.

/model_states



Εικόνα 11. model_states topic

Περιέχει την πιο πρόσφατη κατάσταση του μοντέλου, τιμές δηλαδή που σχετίζονται με την θέση, προσανατολισμό, γραμμική ταχύτητα, γωνιακή ταχύτητα. Αυτό το topic ανανεώνεται μόνο από το model_plugin του Gazebo, στο τέλος κάθε κύκλου προσομοίωσης, καθώς μόνο εκεί υπάρχει διαθέσιμη αυτή πληροφορία. Από το topic αυτό διαβάζουν τα *controller_n* και *core_n*. Αξίζει να σημειωθεί πως για το *core_n*, η ανανέωση των τιμών σε αυτό το topic

[11] Λεπτομέρειες στο Κεφάλαιο 7.3

αποτελεί το σήμα έναυσής του. Η ανανέωση γίνεται αφού και μόνο το Gazebo ολοκληρώσει τον κύκλο του, οπότε αποτελεί και σημείο ένδειξης πως το Gazebo είναι σε κατάσταση αναμονής και το *core_n* έχει την ελευθερία πια να κάνει τον δικό του κύκλο. Τα δεδομένα του είναι εκφρασμένα σε μορφή last_letter_2_msgs/model_states.

/tf-/tf_static

Εικόνα 12. tf topics

Τα tf topics, διατηρούν τη σχέση μεταξύ των πλαισίων συντεταγμένων σε δομή δέντρου. Το */tf_static* περιέχει τις σχέσεις μεταξύ των πλαισίων που δεν αλλάζουν με την πάροδο του χρόνου, ενώ το */tf* των μεταβλητών. Για τον λόγο αυτό, το */tf_static* φορτώνεται μόνο στην αρχή της προσομοίωσης και διατηρεί τα δεδομένα του, ενώ το */tf* ανανεώνεται σε κάθε κύκλο από το ένα και μοναδικό node που γράφει σε αυτά, το *robot_state_publisher_n*. Τα δεδομένα αυτά χρησιμοποιούνται από το Rviz για την οπτικοποίηση του μοντέλου και είναι εκφρασμένα σε μορφή tf2_msgs/TFMessage.

/air_data

Εικόνα 13. air_data topic

Εδώ γράφονται οι τιμές που σχετίζονται με το περιβάλλον προσομοίωσης από το *core_n*, στο οποίο και υπολογίζονται. Οι τιμές αυτές αφορούν την ταχύτητα του ανέμου στις τρεις διαστάσεις, την πυκνότητα και την πίεση του αέρα και την θερμοκρασία του περιβάλλοντος. Τα δεδομένα αυτού του topic είναι εκφρασμένα σε μορφή last_letter_2_msgs/air_data.

4

Διεπαφή Χρήστη

Ένα από τα κύρια χαρακτηριστικά του Last letter 2, στο οποίο δόθηκε μεγάλη σημασία κατά την κατασκευή του, είναι η εύκολη χρήση του από μελλοντικούς ερευνητές. Αφού έγινε κατανοητή η βασική ιδέα και λειτουργία του προσομοιωτή, είναι χρήσιμο κανείς να γνωρίζει πώς μπορεί να επέμβει στο σύστημα, να αλλάξει παραμέτρους, να γράψει νέους αλγορίθμους, να φτιάξει νέα μοντέλα, νέους αισθητήρες και ό,τι άλλο θέλει. Ο προσομοιωτής παρέχει την δυνατότητα στους χρήστες να τροποποιούν και να προσθέτουν/αφαιρούν κομμάτια κώδικα, σύμφωνα με τις ανάγκες τους εύκολα και γρήγορα.

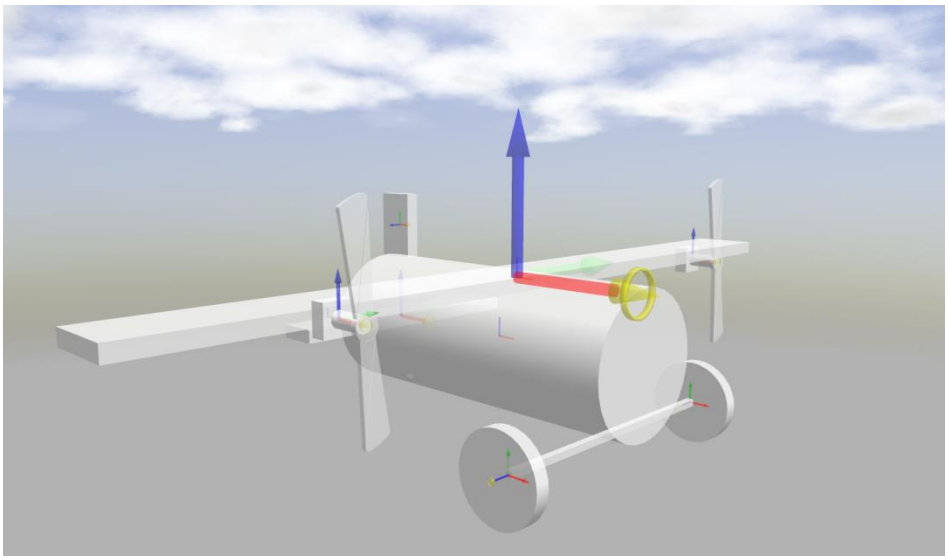
Οι τρόποι με τους οποίους ο χρήστης μπορεί να επέμβει στο σύστημα είναι τέσσερις και αφορούν:

1. την περιγραφή μοντέλου
2. την δήλωση παραμέτρων που αφορούν το μοντέλο, το περιβάλλον, την μηχανή φυσικής και την συσκευή επικοινωνίας του χρήστη με τον προσομοιωτή (joystick)
3. την συγγραφή αλγόριθμου ελέγχου (controller)
4. την προσθήκη λειτουργιών στα υπόλοιπα κανάλια ελέγχου

Πριν την ανάλυση την παραπάνω, είναι χρήσιμο να γίνει μια αναφορά στα συστήματα συντεταγμένων του προσομοιωτή.

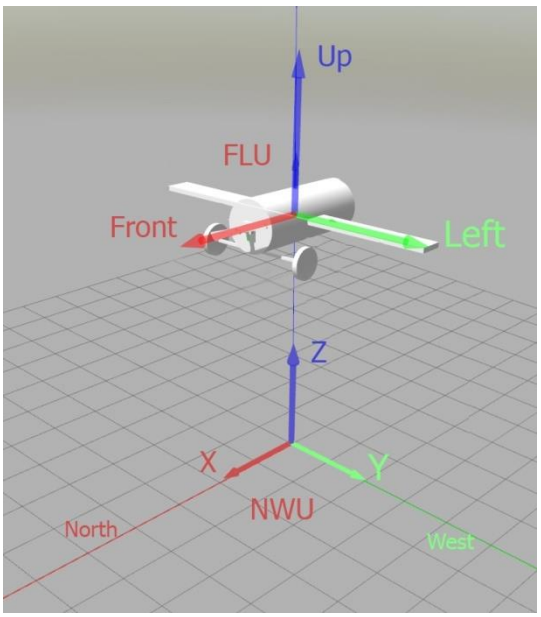
4.1 Συστήματα συντεταγμένων

Στον προσομοιωτή υπάρχουν αρκετά συστήματα συντεταγμένων. Κάθε link του μοντέλου έχει το δικό του σύστημα συντεταγμένων. Στη συνέχεια, ό,τι αναφέρεται για το βασικό link του μοντέλου, το body_FLU, ισχύει και για όλα τα υπόλοιπα.



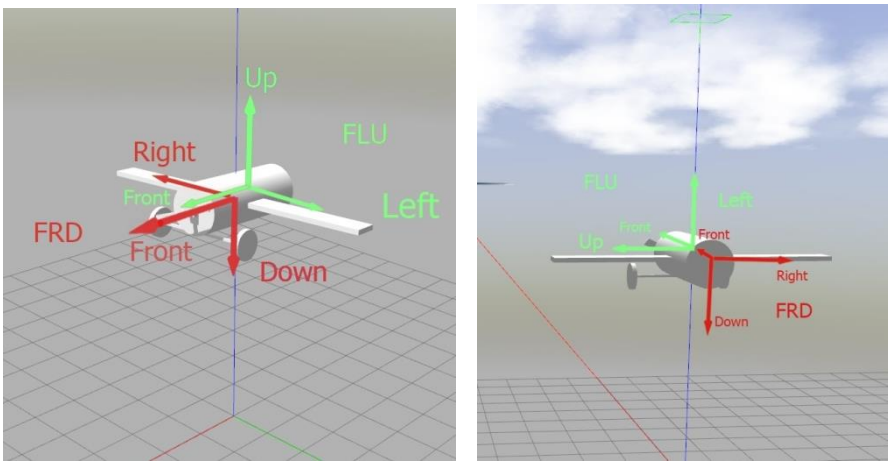
Εικόνα 14. Link ενός μοντέλου

Στο Gazebo, όλα τα συστήματα συντεταγμένων είναι της μορφής FLU (Front Left Up), δηλαδή οι θετικοί άξονες κάθε τοπικού καρτεσιανού συστήματος τριών αξόνων εκτείνονται από το σημείο τομής των αξόνων μπροστά, αριστερά και πάνω, όπως φαίνεται και στο σχήμα. Η κατάσταση των link του μοντέλου είναι εκφρασμένη ως προς αυτά. Το ίδιο ισχύει και με τις δυνάμεις και ροπές. Προκειμένου να εφαρμοστούν χρειάζεται να είναι εκφρασμένες σε FLU μορφή του αντίστοιχου συστήματος συντεταγμένων.



Εικόνα 15. Το αρχικό σύστημα συντεταγμένων του κόσμου της προσομοίωσης (inertial_NWU) και το σύστημα συντεταγμένων του αρχικού link του μοντέλου (body_FLU) όπως τα διαχειρίζεται το Gazebo.

Από την άλλη πλευρά, οι μαθηματικοί τύποι υπολογισμού της δυναμικής απαιτούν τα δεδομένα εκφρασμένα σε μορφή FRD (Front Right Down) ενώ οι εξαγόμενες δυνάμεις και ροπές είναι εκφρασμένες στην ίδια μορφή.



Εικόνα 16. FLU vs FRD frames

Έτσι στους υπολογισμούς παρατηρείται συχνά η μεταφορά διανυσμάτων από ένα σύστημα της μορφής FLU στο ίδιο σύστημα της μορφής FRD και αντίστροφα. Η στροφή αυτή των συστημάτων συντεταγμένων επιτυγχάνεται μέσω της συνάρτησης **FLUtoFRD**[12]**.** Ο ακριβώς αντίστροφος μετασχηματισμός επιτυγχάνεται μέσω της **FRDtoFLU**.

4.2 Περιγραφή μοντέλου

Τα μοντέλα που προσομοιώνονται στον προσομοιωτή, περιγράφονται σε μορφή URDF[13] (Unified Robot Description Format), μια μορφή κατανοητή και αναγνωρίσιμη από το ROS.

Στα URDFs, τα μοντέλα περιγράφονται σαν ένα συνδυασμό από links ενωμένων μέσω joints. Έτσι κάθε μοντέλο απεικονίζεται σαν μια μορφή δέντρου, με αρχικό κόμβο ένα βασικό link (π.χ. body_FLU), συνεχίζοντας με τα υπόλοιπα μέρη του μοντέλου να ακολουθούν σε μορφή φύλλων, ενωμένα με joints.

Οδηγίες περιγραφής μοντέλου σε URDF

Όλα τα μοντέλα του προσομοιωτή βρίσκονται στον φάκελο ***<catkin_workspace>/src/last_letter_2/last_letter_2/config/param/models***. Κάθε μοντέλο έχει τον δικό του υποφάκελο, με όνομα ίδιο με του αεροσκάφους. Εκεί υπάρχει ένα υποφάκελος με το όνομα **/urdf**, όπου βρίσκεται και το αρχείο περιγραφής του μοντέλου, με όνομα **model.urdf.xacro**. Σε αυτό το αρχείο περιγράφεται όλη η δομή ενός URDF αρχείου μοντέλου, ξεκινώντας από τα μέρη που το αποτελούν μέχρι στις σχετικές με αυτό βιβλιοθήκες, π.χ. model plugin.

[12] Βλ. Κεφάλαιο 7.4

[13] Βλ. Παράρτημα

Ένα μοντέλο αποτελείται από 4 είδη link:

- το βασικό link, δηλαδή ο κορμός του μοντέλου
- link που αποτελούν τα πτερύγια του μοντέλου
- link που αποτελούν τους κινητήρες του, και
- συμπληρωματικά link

Στο τέλος προστίθενται και οι απαραίτητες βιβλιοθήκες που θα τρέξουν στο Gazebo.

4.2.1 Βασικό link

Το βασικό link είναι το πρώτο link του μοντέλου. Από αυτό ξεκινάνε να συνδέονται όλα τα υπόλοιπα link που απαρτίζουν το μοντέλο. Το όνομα αυτού του link είναι αυστηρώς **body_FLU**. Ο προσομοιωτής μπορεί να βλέπει και να χειρίζεται αυτό το link μόνο μέσω αυτού του ονόματος.

4.2.2 Πτερύγια link

Τα πτερύγια παίρνουν το όνομα **airfoil&**, όπου **&** ο αριθμός του πτερύγιου ξεκινώντας από το νούμερο 1. Προτεινόμενος τύπος joint (link για joint types) για την ένωση των πτερυγίων με το body_FLU ή όποιο άλλο link είναι ο revolute, με μηδενικές γωνίες στροφής. Τεχνικά πιο σωστό είναι να χρησιμοποιηθεί fixed joint, καθώς η σύνδεσή των πτερυγίων με τον κορμό είναι σταθερή, αλλά το Gazebo λόγω απλοποίησης διαγράφει τα fixed joints και ενώνει τα link πατέρα παιδιού, διατηρώντας το όνομα του πατέρα. Αυτό φυσικά κάνει την πρόσβαση στο link του παιδιού (πτερυγίου) αδύνατη, πράγμα απαγορευτικό για την προσομοίωση. Ο προσομοιωτής μπορεί να διαχειριστεί μέχρι δέκα link πτερυγίων. Αν και τα link πτερυγίων χρησιμοποιούνται κυρίως σε fixed-wing μοντέλα, μπορούν κάλλιστα να χρησιμοποιηθούν και σε multirotor μοντέλα. (πχ κατασκευή υβριδικών αεροσκαφών).

4.2.3 Κινητήρες link

Όσον αφορά τους κινητήρες, δεν απεικονίζονται από ένα link ο καθένας αλλά από 3, δηλαδή μία βάση όπου ασκείται η δύναμη της ώθησης του κινητήρα, ένας άξονας και μια προπέλα. Αυτά τα link πρέπει να έχουν επίσης συγκεκριμένα ονόματα. Η βάση παίρνει το όνομα **motor&**, ο άξονας το όνομα **axle&,** και η προπέλα το όνομα **propeller&,** όπου & ο αριθμός του κινητήρα ξεκινώντας από το 1. Το Gazebo θα ψάξει αυτά τα ονόματα για να ασκήσει την δύναμη ώθησης και την ροπή κάθε κινητήρα στην βάση του, και μια ροπή στρέψης του άξονα που συγκρατεί την προπέλα, δίνοντας έτσι κίνηση και στην προπέλα. Τα link κινητήρων, χρησιμοποιούνται τόσο στα fix-wing, όσο και στα multirotors.

4.2.4 Συμπληρωματικά link

Ένα μοντέλο μπορεί να αποτελείται και από link τα οποία δεν είναι μέρος της αεροδυναμικής ούτε του συστήματος προώθησης του μοντέλου, αλλά είναι χρήσιμα για την προσομοίωση. Τέτοια links μπορεί να είναι οι τροχοί του μοντέλου, βάσεις για αισθητήρες, προσθετικές μάζες για καλύτερη προσέγγιση του ρεαλιστικού μοντέλου κτλ. Τα ονόματα αυτών των link δεν παίζουν κάποιο ρόλο οπότε δίνονται τυχαία.

Πέρα από τις παραπάνω οδηγίες, τα υπόλοιπα ρυθμίζονται ανάλογα τις ανάγκες του χρήστη, δηλαδή το μέγεθος, το σχήμα, η αδράνεια και όποιο άλλο χαρακτηριστικό. Τα links συνδέονται μεταξύ τους μέσω joints, τα ονόματα και το είδος των οποίων επιλέγεται επίσης

ανάλογα με τις ανάγκες. Προτείνεται να χρησιμοποιούνται ονόματα που κάνουν εύκολη την ανάγνωση του URDF αρχείου.

4.2.5 Gazebo Plugins

Τέλος, πέρα από τα δομικά στοιχεία ενός μοντέλου, ένα URDF αρχείο περιλαμβάνει και κάποιες πληροφορίες που αφορούν το Gazebo. Το Gazebo διαβάζει το URDF του μοντέλο και το μετατρέπει σε μορφή SDF, για να μπορεί να το διαχειριστεί. Επίσης από το URDF διαβάζει και τις βιβλιοθήκες των gazebo plugins που θα τρέξουν για το μοντέλο, όπως model και sensor plugins. Αξίζει να σημειωθεί πως κάθε αισθητήρας που προστίθεται σε ένα μοντέλο, πρέπει να αντιστοιχίζεται με κάποιο link.

Παρακάτω εξηγείται ο τρόπος με τον οποίο προστέθηκαν κάποιοι βασικοί αισθητήρες στο plane_1wing_sensors μοντέλο του Last_letter_2. Σκοπός είναι ο χρήστης αφενός να καταλάβει την χρήση και ρύθμισή τους, αφετέρου να αποκτήσει μια βασική ιδέα ώστε να μπορέσει εύκολα να ασχοληθεί και με άλλους τύπους αισθητήρων.

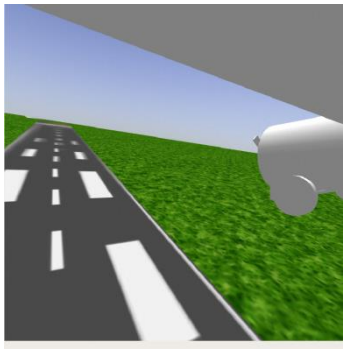
Για κάθε αισθητήρα έχει φορτωθεί ένα έτοιμο Sensor plugin. Τα Sensor Plugins δηλώνονται και περιγράφονται στο URDF αρχείο μέσα στο tag <gazebo>, όπως ακριβώς και τα Model Plugins. Μόνο που στην περίπτωση των Sensors πρέπει να δοθεί και ένα link αναφοράς. Τα δεδομένα των Sensor Plugin είναι διαθέσιμα μέσω αντίστοιχων topic.

Οι βασικοί έτοιμοι αισθητήρες[14] που χρησιμοποιούνται ως παράδειγμα στον Last_letter_2 είναι οι:

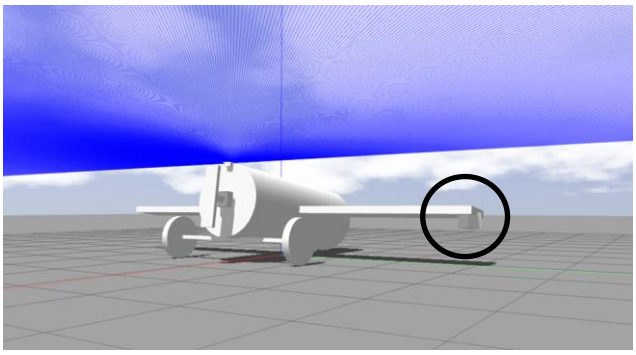
- Camera
- GPU Laser
- IMU

4.2.5.1 Camera Sensors

Ο αισθητήρας αυτός είναι μια απλή RGB κάμερα. Πριν την δήλωσή του πρέπει να δημιουργηθεί ένα link πάνω στο οποίο θα αντιστοιχηθεί η κάμερα. Έτσι λοιπόν κατασκευάζεται ένα απλό link με σχήμα κουτιού το οποίο συνδέεται με το πτερύγιο και δίνεται σε αυτό το όνομα camera_link. Το link αυτό της κάμερας τοποθετείται στην αριστερή πλευρά του πτερυγίου από την κάτω πλευρά και μπορεί να περιστρέφεται.



Εικόνα 17. Η εικόνα της κάμερας



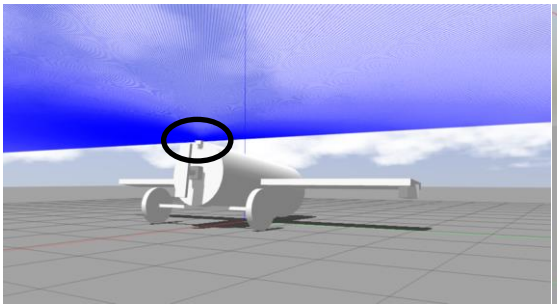
Εικόνα 18. Η θέση της κάμερας

[14] Αναλυτικότερη περιγραφή εδώ: http://gazebosim.org/tutorials?tut=ros_gzplugins

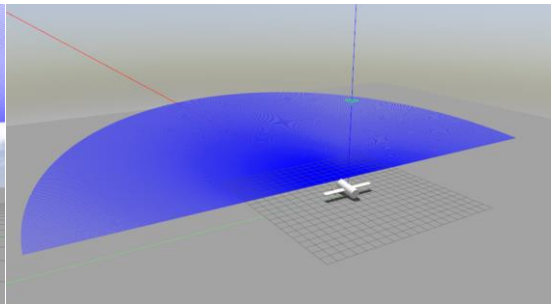
4.2.5.2 GPU Laser Sensor

Ο αισθητήρας αυτός είναι ένας αισθητήρας εύρους απόστασης lazer[15].

Η ιδέα είναι παρόμοια με τον αισθητήρα της κάμερας. Προστίθεται στο URDF του μοντέλου ένα νέο link, στο οποίο θα συνδεθεί ο αισθητήρας. Το link αυτό έχει σχήμα τετραγώνου και τοποθετείται πάνω από το μπροστινό μέρος του αεροπλάνου ώστε ο αισθητήρας να καταγράφει την περιοχή από αριστερά μέχρι δεξιά του αεροπλάνου, γωνία εύρους 180 μοιρών.



Εικόνα 20. Η θέση του laser scanner



Εικόνα 19. Το εύρος γωνίας βολής του laser

4.2.5.3 IMU

Αισθητήρας IMU που αναγνωρίζει την θέση και τις κινήσεις του μοντέλου. Για αυτόν τον αισθητήρα δεν δημιουργείται νέο link. Χρησιμοποιείται το βασικό link body_FLU, μιας και ο αισθητήρας τοποθετείται συνήθως στο κέντρο του μοντέλου.

Κώδικας URDF plane_1wing_sensors (συντετμημένος)

```
<?xml version="1.0"?>
<robot name="plane_1wing_sensors"
                      xmlns:xacro="http://www.ros.org/wiki/xacro">

  <link name="body_FLU">
      …
  </link>

  <link name="airfoil1">
      …
  </link>

  <joint name="body_FLU_to_airfoil1" type="revolute">
      …
  </joint>

  <link name="motor1">
```

[15] Περισσότερες πληροφορίες εδώ: https://wiki.ROS.org/hokuyo_node

```
  <joint name="body_FLU_to_motor1" type="revolute">

  <link name="axle1">
  <joint name="motor1_to_axle1" type="continuous">


  <link name="prop1">
  <joint name="axle1_to_propeller1" type="fixed">

  <link name="wheel_base">
  <joint name=" body_FLU_to_wheel_base" type="fixed">

  <link name="left_wheel">
  <joint name="wheel_base_to_left_wheel" type="continuous">

  <link name="right_wheel">
  <joint name="wheel_base_to_right_wheel" type="continuous">

  <link name="rear_wheel">
  <joint name="body_FLU_to_rear_wheel" type="continuous">
```

Link αισθητήρων

Link κάμερας (ένα απλό link σχήματος κύβου)

```
  <link name="camera_link">
    <collision>
      <geometry>
        <box size="0.1 0.1 0.1"/>
      </geometry>
    </collision>

    <visual>
      <geometry>
        <box size="0.1 0.1 0.1"/>
      </geometry>
    </visual>

    <inertial>
      <mass value="1e-5" />
      <inertia    ixx="1e-6" ixy="0" ixz="0"
                  iyy="1e-6" iyz="0" izz="1e-6" />
    </inertial>
  </link>

  <joint name="camera_joint" type="revolute">
    <axis xyz="0 0 1" />
    <origin xyz="0 ${airfoil_y/2-0.1} -${airfoil_z/2+0.05}" rpy="0 0
-0.6"/>
```

```
    <parent link="airfoil1"/>
    <child link="camera_link"/>
    <limit effort="1000.0" velocity="0.5" lower="-2" upper="2" />
  </joint>

  Link laser scanner (ένα απλό link σχήματος κύβου)
  <link name="laser_link">
    <collision>
      <geometry>
        <box size="0.1 0.1 0.1"/>
      </geometry>
    </collision>

    <visual>
      <geometry>
        <box size="0.1 0.1 0.1"/>
      </geometry>
    </visual>

    <inertial>
      <mass value="1e-5" />
      <inertia    ixx="1e-6" ixy="0" ixz="0"
                  iyy="1e-6" iyz="0" izz="1e-6" />
    </inertial>
  </link>

    <joint name="laser_joint" type="revolute">
    <axis xyz="0 1 0" />
    <origin xyz="${body_h/2-0.05} 0 ${-body_r}" rpy="0 0 0"/>
    <parent link="body_FLU"/>
    <child link="laser_link"/>
    <limit effort="1000.0" velocity="0.5" lower="-1" upper="1" />
  </joint>

Δήλωση βιβλιοθηκών Gazebo (gazebo plugins)
  <gazebo>
    Model plugin
    <plugin name="model_plugin" filename="libmodel_plugin.so"/>

 Joint state publisher (δημοσιεύει τις καταστάσεις των joint για να
 σχηματιστεί το tf δέντρο από τον robot_state_publisher)
 Λεπτομέρειες εδώ: http://wiki.ros.org/joint_state_publisher
    <plugin name="joint_state_publisher"
            filename="libgazebo_ros_joint_state_publisher.so">
      <jointName> body_FLU_to_airfoil1,
                  body_FLU_to_motor1,
                  motor1_to_axle1,
                  wheel_base_to_left_wheel,
                  wheel_base_to_right_wheel,
```

```
                  body_FLU_to_rear_wheel,
                  camera_joint,
                  laser_joint
      </jointName>
      <robotNamespace>last_letter_2</robotNamespace>
      <updateRate>0</updateRate>
    </plugin>
  </gazebo>

  Plugin κάμερας
  <gazebo reference="camera_link"> Δήλωση link αναφοράς
    <sensor type="camera" name="camera_sensor">
      Δήλωση Παραμέτρων κάμερας
      <update_rate>30.0</update_rate>
      <camera name="head">
        <horizontal_fov>1.3962634</horizontal_fov>
        <image>
          <width>800</width>
          <height>800</height>
          <format>R8G8B8</format>
        </image>
        <clip>
          <near>0.02</near>
          <far>300</far>
        </clip>
        <noise>
          <type>gaussian</type>
          <!-- Noise is sampled independently per pixel on each
            frame. That pixel's noise value is added to each of its
            color channels, which at that point lie in the range
            [0,1]. -->
          <mean>0.0</mean>
          <stddev>0.007</stddev>
        </noise>
      </camera>
Βιβλιοθήκη αισθητήρα κάμερας
      <plugin name="camera_controller"
                              filename="libgazebo_ros_camera.so">
        <alwaysOn>true</alwaysOn>
Συχνότητα αισθητήρα. 0 για συχνότητα ίσης με της προσομοίωσης
        <updateRate>0.0</updateRate>
        <cameraName>last_letter_2/sensors/camera</cameraName>
        Δήλωση topic στο οποίο δημοσιεύει την εικόνα
        <imageTopicName>image</imageTopicName>
        <cameraInfoTopicName>camera_info</cameraInfoTopicName>
        <frameName>camera_frame</frameName>
        <hackBaseline>0.07</hackBaseline>
        <distortionK1>0.0</distortionK1>
        <distortionK2>0.0</distortionK2>
```

```
      <distortionK3>0.0</distortionK3>
      <distortionT1>0.0</distortionT1>
      <distortionT2>0.0</distortionT2>
    </plugin>
  </sensor>
</gazebo>
```

Plugin laser scanner

```
<gazebo reference="laser_link"> Δήλωση link αναφοράς
  <sensor type="ray" name="laser_sensor">
    <pose>0 0 0 0 0 0</pose>
    Δήλωση Παραμέτρων αισθητήρα
    <visualize>true</visualize>
    <update_rate>40</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>720</samples>
          <resolution>1</resolution>
          <min_angle>-1.570796</min_angle>
          <max_angle>1.570796</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.1</min>
        <max>30.0</max>
        <resolution>0.1</resolution>
      </range>
      <noise>
        <type>gaussian</type>
        <!-- Noise parameters based on published spec for Hokuyo
          laser achieving "+-30mm" accuracy at range < 10m.  A
          mean of 0.0m and stddev of 0.01m will put 99.7% of
          samples within 0.03m of the true reading. -->
        <mean>0.0</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
    Βιβλιοθήκη αισθητήρα laser scanner
    <plugin name="gazebo_ros_laser_scanner_controller"
                            filename="libgazebo_ros_laser.so">
      Δήλωση topic που δημοσιεύει τα δεδομένα
      <topicName>last_letter_2/sensors/laser</topicName>
      <frameName>laser_link</frameName>
    </plugin>
  </sensor>
</gazebo>
```

```
  Αισθητήρας IMU
Για τον αισθητήρα αυτό δεν δημιουργήθηκε νέο link. Χρησιμοποιήθηκε το
body_FLU
  <gazebo reference="body_FLU">
    <gravity>true</gravity>
    <sensor name="imu_sensor" type="imu">
      <always_on>true</always_on>
      <update_rate>100</update_rate>
      <visualize>true</visualize>
      <topic>__default_topic__</topic>
      <plugin filename="libgazebo_ros_imu_sensor.so"
name="imu_plugin">
        Δήλωση topic δεδομένων IMU
        <topicName>last_letter_2/sensors/imu</topicName>
        <bodyName>body_FLU</bodyName>
        <updateRateHZ>10.0</updateRateHZ>
        <gaussianNoise>0.0</gaussianNoise>
        <xyzOffset>0 0 0</xyzOffset>
        <rpyOffset>0 0 0</rpyOffset>
        <frameName>body_FLU</frameName>
      </plugin>
      <pose>0 0 0 0 0 0</pose>
    </sensor>
  </gazebo>

</robot>
```

4.2.6 Παράδειγμα

Στο URDF του plane_3wing_2motor μοντέλου (αρχείο ***<catkin_workspace>/src/last_letter_2/last_letter_2/config/param/model/plane_3wing_2motor/urdf/model.urdf.xacro***) παρατηρείται πως υπάρχουν 3 links με τα ονόματα airfoi1, airfoil2, airfoil3, ενωμένα στο βασικό link, το body_FLU. Επίσης υπάρχουν και δύο κινητήρες τοποθετημένοι πάνω στο κεντρικό πτερύγιο. Κάθε ένας από αυτούς ξεκινάει με ένα link motor1/2, ακολουθεί ένα axle1/2 και τελειώνει με το propeller1/2. Όλα τα παραπάνω πρέπει να δηλωθούν με αυτά τα ακριβή ονόματα. Πέραν αυτών υπάρχουν και τα link wheel_base, left_wheel, right_wheel και rear_wheel, τα οποία όπως δηλώνουν και τα ονόματά τους είναι οι βάση για τους μπροστινούς τροχούς μαζί με αυτούς και ο πίσω τροχός του μοντέλου. Τα ονόματά τους είναι τυχαία. Στο τέλος δηλώνονται και τα Gazebo plugins.

Παρακάτω φαίνονται τα βασικά σημεία του URDF του μοντέλου αυτού.

```
<robot name="plane_3wing_2motor">

    Βασικό link
    <link name="body_FLU">
       <visual> …    </visual>
       <collision> … </collision>
       <inertial> … </inertial>
    </link>
```

```
    Link Πτερυγίων
    <link name="airfoil1">
          …
    </link>
    <joint name="body_FLU_to_airfoil1" type="revolute">
           . . .
       <parent link="body_FLU" />
       <child link="airfoil1" />
    </joint>

    <link name="airfoil2"> …  </link>
    <joint name="body_FLU_to_airfoil2" … > … </joint>

    <link name="airfoil3"> … </link>
    <joint name="body_FLU_to_airfoil3" … > … </joint>

    Link Κινητήρων
    <link name="motor1"> … </link>
    <joint name="airfoil1_to_motor1" … > … </joint>

    <link name="motor2"> … </link>
    <joint name=" airfoil1_to_motor2" … > … </joint>

    <link name="axle1"> … </link>
    <joint name="motor1_to_axle1" … > … </joint>

    <link name="axle2"> … </link>
    <joint name=" motor2_to_axle2" … > … </joint>

    <link name="propeller1"> … </link>
    <joint name="axle1_to_propeller1" … > … </joint>

    <link name="propeller2"> … </link>
    <joint name=" axle2_to_propeller2" … > … </joint>

    Συμπληρωματικά Link (Τροχοί)
    <link name="wheel_base"> …. </link>
    <joint name="body_FLU_to_wheel_base" … > … </joint>

    <link name="left_wheel"> ….</link>
    <joint name="wheel_base _to_left_wheel" … > … </joint>

    <link name="right_wheel"> … </link>
    <joint name="wheel_base _to_right_wheel" … > … </joint>

    <link name="rear_wheel"> … </link>
    <joint name="body_FLU_to_rear_wheel" … > … </joint>

     Δήλωση Gazebo Plugin
    <gazebo>
        <plugin name="model_plugin" filename="libmodel_plugin.so"/>
        <plugin name="joint_state_publisher"
filename="libgazebo_ros_joint_state_publisher.so">
                      …
        </plugin>
    </gazebo>
</robot>
```

4.3 Δήλωση Παραμέτρων

Ο χρήστης περιγράφει τις ιδιότητες του μοντέλου και κάποιες του προσομοιωτή γενικότερα μέσα από παραμέτρους. Υπάρχουν παράμετροι που αφορούν γενικά την προσομοίωση και παράμετροι που αφορούν ειδικά το μοντέλο.

4.3.1 Γενικές Παράμετροι

Κάθε προσομοίωση χαρακτηρίζεται από ορισμένες παραμέτρους. Ο προσομοιωτής δίνει την δυνατότητα στον χρήστη να ελέγχει αυτές τις παραμέτρους μέσω αρχείων .yaml. Όλες οι παράμετροι είναι συγκεντρωμένες στον φάκελο ***<catkin_workspace>/src/last_letter_2/ last_letter_2/config/param.*** Εκεί υπάρχουν τρία αρχεία γενικών παραμέτρων οι οποίες αφορούν:

- την μηχανή φυσικής της προσομοίωσης (**updatePhysics.yaml**)
- το φυσικό περιβάλλον μέσα στο οποίο γίνεται η προσομοίωση (**environment.yaml**)
- την συσκευή επικοινωνίας του χρήστη με τον προσομοιωτή (joystick) (**HID.yaml**)

4.3.1.1 updatePhysics.yaml - Παράμετροι Μηχανής Φυσικής

Παράμετροι σχετικοί με την μηχανής φυσικής. Αυτές οι παράμετροι έπονται του ονόματος /updatePhysics/ και είναι οι εξής τρεις:

simRate: συχνότητα προσομοίωσης
deltaT: χρονικό βήμα προσομοίωσης
paused: έναρξη προσομοίωσης σε κατάσταση παύσης ή όχι (0,1)

Αξίζει να σημειωθεί πως Ταχύτητα προσομοίωσης = simRate * deltaT
Δηλαδή για simRate * deltaT = 1, η προσομοίωση τρέχει σε πραγματικό χρόνο.

4.3.1.2 environment.yaml - Παράμετροι Φυσικού Περιβάλλοντος

Αυτές οι παράμετροι περιγράφουν μεταβλητές του περιβάλλοντος μέσα στο οποίο πραγματοποιείται η πτήση. Οι παράμετροι έπονται του ονόματος /environment/ και είναι οι εξής:

rho: Πυκνότητα του αέρα ($\frac{kg}{m^3}$)
Rd: Σταθερά αερίου σε ξηρό αέρα **(**$\frac{J}{K*mol}$)
Lo: Ρυθμός πτώσης της θερμοκρασίας με την αύξηση του υψόμετρου, από την επιφάνεια της θάλασσας ($\frac{K}{km}$)
gravity: Επιτάχυνση της βαρύτητας, στην επιφάνεια της θάλασσας ($\frac{m}{s^2}$)
groundTemp: Θερμοκρασία ατμόσφαιρας στο υψόμετρο αρχικοποίησης ($Celsius$)
groundPres: Πίεση ατμόσφαιρας στο υψόμετρο αρχικοποίησης ($mBar$)
windRef: Ταχύτητα ανέμου στο υψόμετρο αναφοράς ($\frac{m}{s}$)

windRefAlt: Υψόμετρο αναφοράς για την ταχύτητα του ανέμου (m)

windDir: Κατεύθυνση ανέμου στο επίπεδο xy - z συνιστώσα μηδενική (deg)

surf/Smooth: Εκθέτης Hellman. Ρυθμίζει πόσο γρήγορα αυξάνεται η ταχύτητα του ανέμου καθώς το υψόμετρο μεγαλώνει. (s^{-1})

Ακολουθούν παράμετροι σχετικές με τις ριπές ανέμου. Οι ριπές ανέμου διαμορφώνονται χρησιμοποιώντας Dryden συναρτήσεις μεταφοράς. Αυτές οι στοχαστικές συναρτήσεις μεταφοράς παίρνουν ως είσοδο λευκό θόρυβο μοναδιαίας ισχύος και διαμορφώνουν το φάσμα ισχύος τους ώστε να ταιριάζουν με αυτό των πραγματικών μετρήσεων. Περισσότερη τεκμηρίωση, ενδεικτικές τιμές και αναφορές μπορούν να βρεθούν στην ενότητα ``Wind Disturbances`` of `Modeling a Fixed-Wing UAV https://github.com/Georacer/uav-modeling.

Dryden/Lu: Παράμετρος που ελέγχει το μήκος κύματος της οριζόντιας ροής (m)

Dryden/Lw: Παράμετρος που ελέγχει το μήκος κύματος της κάθετης ροής (m)

Dryden/sigmau: Συνολική ενέργεια οριζόντιας ροής

Dryden/sigmaw: Συνολική ενέργεια κάθετης ροής

Dryden/use: Ενεργοποίηση / Απενεργοποίηση των ριπών ανέμου.

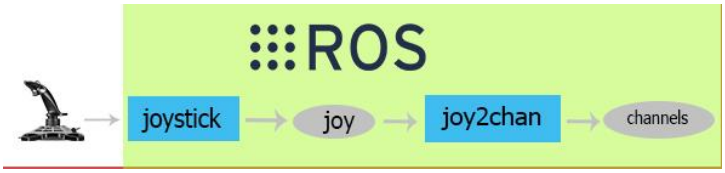
4.3.1.3 HID.yaml - Παράμετροι συσκευής αλληλεπίδρασης

Οι παράμετροι αυτού του αρχείου περιγράφουν τον τρόπο με τον οποίο τα σήματα που στέλνει ο χρήστης μέσω του joystick αντιστοιχίζονται στον πίνακα καναλιών. Περιέχει τρεις πίνακες 20 θέσεων, που έπονται του ονόματος /HID/ και των οποίων ο ρόλος εξηγείται παρακάτω.

axis: πίνακας αντιστοίχησης των αξόνων του joystick με τον πίνακα καναλιών

buttons: πίνακας αντιστοίχησης των κουμπιών του joystick με τον πίνακα καναλιών

throws: δήλωση πρόσημου κατά την στρέψη ενός άξονα με θετική φορά ή το πάτημα ενός κουμπιού



Εικόνα 21. Ροή σημάτων από το joystick στον πίνακα καναλιών

Κάθε joystick που συνδέεται στο σύστημα, οδηγείται από το *joystick_n* του πακέτο joy του ROS. Το node αυτό δημοσιεύει στο topic */joy* έναν πίνακα με την πιο πρόσφατη κατάσταση των αξόνων και έναν με των κουμπιών του joystick. Το περιεχόμενο αυτών το πινάκων πρέπει να μεταφερθεί σε έναν πίνακα 20 θέσεων, το πίνακα καναλιών.

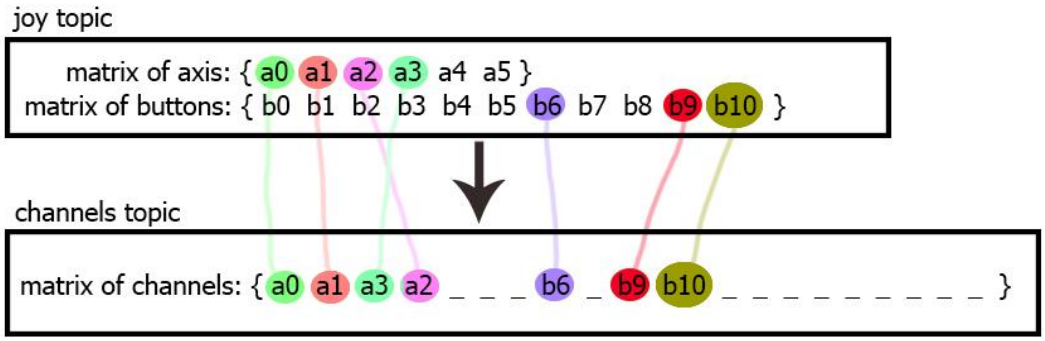
Για να επιτευχθεί αυτό, ο χρήστης σημειώνει σε μια από τις 20 θέσεις του πίνακα καναλιών, το νούμερο της θέσης του αρχικού πίνακα στην οποία βρίσκεται ο άξονας ή το κουμπί που τον ενδιαφέρει. Πιο συγκεκριμένα, στον πρώτο πίνακα axes[], αντιστοιχίζονται οι άξονες του joystick με μια από τις 20 θέσεις του πίνακα καναλιών. Στον δεύτερο πίνακα buttons[], αντιστοιχίζονται τα κουμπιά του joystick με μια από τις 20 θέσεις του πίνακα καναλιών. Τέλος, ο πίνακας throws γεμίζεται με την τιμή που δίνει το *joystick_n* για κάποιον άξονα στραμμένο προς την θετική πλευρά (κανόνας τριών δακτύλων δεξιού χεριού) ή για κάποιο

πατημένο κουμπί. Αυτό βοηθάει αργότερα στην σωστή δημιουργία των πρόσημων. Προκειμένου να μην γίνονται επικαλύψεις στον πίνακα καναλιών, ο χρήστης καλείται να αφήνει για κάθε θέση του πίνακα καναλιών την μία τιμή μεταξύ των πινάκων axes και buttons -1.0, δηλαδή κενή.

Παράδειγμα

Με τις παρακάτω παραμέτρους ο πίνακας των καναλιών γεμίζει όπως φαίνεται στην εικόνα. Στις τέσσερις πρώτες θέσεις αντιστοιχίζονται τέσσερις άξονες και κάπου στη μέση του πίνακα 3 κουμπιά. Οι υπόλοιπες θέσεις μένουν κενές.

```
#Chan  [  0    1    2   3    4   5   6    7   8   9   10  11  12  13  14  15  16  17  18  19]
axes:  [  0,   1,   3,  2,  -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
buttons: [ -1, -1, -1, -1, -1, -1, -1,  6, -1,  9, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1]
throws:  [-1.0, -1.0, -1.0, 1.0,  1.0, 1.0, 1.0, 1.0,  1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,  1.0, 1.0, 1.0]
```



4.3.2 Παράμετροι μοντέλου

Μέσα στον φάκελο ***<catkin_workspace>/src/last_letter_2/last_letter_2/config/param*** υπάρχει ο φάκελος των μοντέλων **/models**. Εκεί κάθε μοντέλο έχει τον δικό του υποφάκελο, ο οποίος περιέχει το URDF αρχείο του μοντέλου και κάποιες παραμέτρους σχετικές με το μοντέλο χωρισμένες στα αρχεία:

- init.yaml
- channel_mix.yaml
- aerodynamics.yaml
- propulsion.yaml

4.3.2.1 inti.yaml – Αρχική κατάσταση μοντέλου

Εδώ δηλώνεται η κατάσταση του μοντέλου μόλις ξεκινάει η προσομοίωση. Οι παράμετροι έπονται του ονόματος /init/.

position: Διάνυσμα θέσης μοντέλου (x, y, z)
orientation: Διάνυσμα προσανατολισμού μοντέλου (around x, y, z axis)
velLin: Διάνυσμα γραμμικής ταχύτητας μοντέλου (x, y, z)
velAng: Διάνυσμα γωνιακής ταχύτητας (around x, y, z axis)

Όλα τα παραπάνω δηλώνονται σε μορφή πινάκων τριών στοιχείων, όσες και οι διαστάσεις. Η θέση και ο προσανατολισμός είναι εκφρασμένες ως προς το σύστημα συντεταγμένων του κόσμου, ενώ οι ταχύτητες ως προς το σύστημα συντεταγμένων του μοντέλου.

4.3.2.2 channel_mix.yaml – Δήλωση χρήσης καναλιών

Εδώ αντιστοιχίζονται τα σήματα που στέλνει ο χρήστης μέσω του joystick με κάποιο κανάλι. Οι παράμετροι έπονται του ονόματος /channels/ και είναι οι εξής:

roll_in_chan: κανάλι ελέγχου γωνίας roll
pitch_in_chan: κανάλι ελέγχου γωνίας pitch
yaw_ in_chan: κανάλι ελέγχου γωνίας yaw
throttle_in_chan: κανάλι ελέγχου throttle
camera_angle_chan: κανάλι ελέγχου γωνίας κάμερας
laser_angle_chan: κανάλι ελέγχου γωνίας laser
spawn_can_chan: κανάλι εισαγωγής ενός μικρού κουτιού κάτω από το μοντέλο
despawn_cans_chan: κανάλι διαγραφής όλων τον εισαγόμενων κουτιών
reset_sim_chan: κανάλι επανεκκίνησης προσομοίωσης

4.3.2.3 aerodynamics.yaml – Παράμετροι Πτερυγίων

Εδώ δηλώνονται οι παράμετροι που περιγράφουν την αεροδυναμική των πτερυγίων του μοντέλου.
Αρχικά δηλώνεται ο αριθμός των πτερυγίων και έπειτα για κάθε πτερύγιο οι αεροδυναμικές παράμετροι.

nWings: Συνολικός αριθμός πτερυγίων μοντέλου

Οι παράμετροι ενός πτερυγίου έπονται του ονόματος /airfoil&/, όπου & ο αριθμός του πτερυγίου ξεκινώντας την αρίθμηση από το 1.

aerodynamicsType: Τύπος υπολογισμού της αεροδυναμικής του πτερυγίου.
0 για NoAerodynamics, 1 για stdLinearAero, 2 για polyAero

Υπάρχουν 3 τύποι υπολογισμού της αεροδυναμικής για κάθε πτερύγιο. Οι NoAerodynamics, stdLinearAero και polyAero. Κάθε αεροδυναμικός τύπος πτερυγίου απαιτεί την δήλωση κάποιων παραμέτρων προκειμένου να γίνουν σωστά οι υπολογισμοί της δυναμικής τους.

Ο NoAerodynamics δηλώνει απουσία αεροδυναμικής συμπεριφοράς του πτερυγίου. Χρησιμοποιείται όταν ο χρήστης θέλει να απενεργοποιήσει ή να μην χρησιμοποιήσει το πτερύγιο κατά την προσομοίωση. Αυτός ο τύπος εξοικονομεί χρόνο καθώς ο χρήστης δεν χρειάζεται να τροποποιήσει το URDF μοντέλο και δεν απαιτεί την δήλωση καμιάς παραμέτρου.

Ο stdLinearAero[16] υπολογίζει την αεροδυναμική του πτερυγίου αποκλειστικά με χρήση αεροδυναμικών συντελεστών δηλωμένων από τον χρήστη.

[16] Βλ. Παράρτημα

Ο polyAero[17] τύπος, υπολογίζει την αεροδυναμική του πτερυγίου με χρήση των αεροδυναμικών παραμέτρων του stdLinearAero τύπου αλλά χρησιμοποιεί και δύο πολυωνυμικές συναρτήσεις για να εξάγει τα βασικά μεγέθη των συντελεστές drag και lift του πτερυγίου. Συνήθως χρησιμοποιείται όταν ο χρήστης έχει στην διάθεσή του τις γραφικές παραστάσεις lift-alpha, drag-alpha εξαγόμενες για παράδειγμα από αεροδυναμικό τούνελ. Έτσι περιγράφει ως πολυώνυμα αυτές τις γραφικές και οι τιμές του lift και drag εξάγονται σε κάθε βήμα από αυτά τα πολυώνυμα.

Γενικές παράμετροι πτερυγίου

s: Επιφάνεια πτερυγίου s=b*c
b: Άνοιγμα πτερυγίου (απόσταση από άκρη σε άκρη)
c: Μήκος μέσης χορδής πτερυγίου
deltax_max: Μέγιστη εκτροπή πτερυγίου γύρω από τον xx' άξονα (rad)
deltay_max: Μέγιστη εκτροπή πτερυγίου γύρω από τον yy' άξονα (rad)
deltaz_max: Μέγιστη εκτροπή πτερυγίου γύρω από τον zz' άξονα (rad)
input_x_chan: Κανάλι ελέγχου κίνησης πτερυγίου γύρω από τον xx'
input_y_chan: Κανάλι ελέγχου κίνησης πτερυγίου γύρω από τον yy'
input_z_chan: Κανάλι ελέγχου κίνησης πτερυγίου γύρω από τον zz'

Εάν [-1] το πτερύγιο δεν κινείται γύρω από τον αντίστοιχο άξονα. Διαφορετικά, το πτερύγιο κινείται γύρω από τον αντίστοιχο άξονα ακούγοντας στο κανάλι του οποίο ο αριθμός έχει δηλωθεί.

NoAerodynamics

Δεν χρειάζονται παράμετροι για τον συγκεκριμένο τύπο πτερυγίου

StdLinearAero

Συντελεστές του Lift

oswald*: Συντελεστής απόδοσης Oswald , που σχετίζεται με την αποδοτικότητα (για lift) της γεωμετρίας του πτερυγίου
mcoeff*: Παράγοντας που σχετίζεται με το πόσο απότομη είναι η αλλαγή μεταξύ του γραμμικού και του flat plate[18] μοντέλου
alpha_stall*: Stall angle-of-attack του αεροσκάφους
c_lift_0*: Μηδενικής angle_of_attack συντελεστής lift
c_lift_a*: Συντελεστής σχέσης lift με angle-of-attack
c_lift_q: Συντελεστής lift ως προς γωνία theta (pitch)
c_lift_input_y: Elevator deflection to lift coefficient

[17] Βλ. Παράρτημα
[18] Βλ. Παράρτημα

Συντελεστές του drag

c_drag_p*:	Συντελεστής παρασιτικού drag
c_drag_q:	Συντελεστής drag ως προς γωνία theta (pitch)
c_drag_input_y:	Συντελεστής drag ως προς είσοδο y πτερυγίου

Από τις παραπάνω παραμέτρους, όσες είναι με αστερίσκο (*), χρησιμοποιούνται από τον τύπο 1 μόνο, τον stdLinearAero για την εξαγωγή του συντελεστή lift και drag. Στον τύπο 2 polyAero, αυτές δεν παίζουν κάποιο ρόλο αλλά πρέπει να δηλωθούν με μηδενική τιμή.

Υπόλοιπες παράμετροι[19]

c_y_0:	Bias side force coefficient
c_y_b:	Side force coefficient in relation to angle-of-sideslip
c_y_p:	Side force coefficient in relation to roll-rate
c_y_r:	Side force coefficient in relation to yaw-rate
c_y_input_x:	Side force coefficient in relation to input_x
c_y_input_z:	Side force coefficient in relation to input_z

c_l_0:	Bias rolling moment coefficient
c_l_p:	Rolling moment coefficient in relation to roll rate
c_l_r:	Rolling moment coefficient in relation to yaw rate
c_l_b:	Rolling moment coefficient in relation to angle-of-sideslip
c_l_input_x:	Rolling moment coefficient in relation to input_x (must be positive)
c_l_input_z:	Rolling moment coefficient in relation to input_z

c_m_0:	Bias pitching moment coefficient
c_m_a:	Pitching moment coefficient in relation to angle-of-attack
c_m_q:	Pitching moment coefficient in relation to pitch rate
c_m_input_y:	Pitching moment coefficient in relation to input_y (must be positive)

c_n_0:	Bias pitching moment coefficient
c_n_b:	Yawing moment coefficient in relation to angle-of-sideslip
c_n_p:	Yawing moment coefficient in relation to roll-rate
c_n_r:	Yawing moment coefficient in relation to yaw-rate
c_n_input_x:	Yawing moment coefficient in relation to input_x
c_n_input_z:	Yawing moment coefficient in relation to input_z (must be positive)

[19] Στην περιγραφή αυτών των παραμέτρων χρησιμοποιήθηκαν οι αγγλικές ορολογίες για πιο κατανοητή περιγραφή.

Αυτός ο τύπος υπολογισμού της αεροδυναμικής του πτερυγίου χρησιμοποιεί τις ίδιες παραμέτρους με τον προηγούμενο τύπο, με την διαφορά πως αντί για τις **bold** παραμέτρους δηλώνονται δύο πολυώνυμα, ένα για τον συντελεστή lift και ένα για τον drag.

Οι παρακάτω παράμετροι χρησιμοποιούνται για να ορίσουν πολυώνυμα[20] ή splines που περιγράφουν τους συντελεστές lift και drag σαν μια συνάρτηση μιας μεταβλητής της angle-of-attack. Χρειάζονται δύο ομάδες παραμέτρων, οι cLiftPoly και οι cDragPoly.

cLiftPoly: ένα πολυώνυμο μιας μεταβλητής περιγράφει τον συντελεστή του lift, σαν συνάρτηση του angle-of-attack (σε radians). Αυτή είναι μια ομάδα παραμέτρων και όλες οι πολυωνυμικές παράμετροι πρέπει να ορίζονται κάτω από αυτήν την ομάδα. Προτείνεται spline πολυώνυμο που καλύπτει όλο το διάστημα [-π, π].

cDragPoly: ένα πολυώνυμο μιας μεταβλητής περιγράφει τον συντελεστή του drag, σαν συνάρτηση του angle-of-attack (σε radians). Αυτή είναι μια ομάδα παραμέτρων και όλες οι πολυωνυμικές παράμετροι πρέπει να ορίζονται κάτω από αυτήν την ομάδα. Προτείνεται spline πολυώνυμο που καλύπτει όλο το διάστημα [-π, π].

4.3.2.4 propulsion.yaml

Εδώ δηλώνονται οι παράμετροι που περιγράφουν την ώθηση που παράγουν οι κινητήρες του μοντέλου.

Αρχικά δηλώνεται ο αριθμός των κινητήρων και έπειτα για κάθε κινητήρα οι παράμετροί του.

nMotors: Αριθμός κινητήρων

Οι παράμετροι ενός κινητήρα έπονται του ονόματος /motor&/, όπου & ο αριθμός του κινητήρα ξεκινώντας την αρίθμηση από το 1.

motorType: Τύπος κινητήρα
0 για NoEngine, 1 για genericEngine, 2 για electricEngine

Υπάρχουν τρία είδη κινητήρα. Ο NoEngine, ο genericEngine και ο electricEngine.

Ο NoEngine δηλώνει απουσία κινητήρα. Χρησιμοποιείται όταν ο χρήστης θέλει να απενεργοποιήσει ή να μην χρησιμοποιήσει έναν κινητήρα κατά την προσομοίωση. Αυτός ο τύπος εξοικονομεί χρόνο καθώς ο χρήστης δεν χρειάζεται να τροποποιήσει το URDF μοντέλο και δεν απαιτεί την δήλωση καμιάς παραμέτρου.

Ο genericEngine υπολογίζει τις δυνάμεις που παράγει και ασκεί ένας κινητήρας με προπέλα στο μοντέλο, μέσω κάποιων συντελεστών ώθησης. Λεπτομέρειες των μαθηματικών υπολογισμών των δυνάμεων στο appedix.

Ο electricEngine χρησιμοποιείται συγκεκριμένα για ηλεκτροκινητήρα με προπέλα. Οι μαθηματικοί τύποι υπάρχουν στο Παράρτημα.

[20] Δήλωση πολυωνύμων παρακάτω στο 4.3.2.5

Γενικές παράμετροι

rotationDir: Διεύθυνση περιστροφής κινητήρα
input_chan: Κανάλι ελέγχου του κινητήρα

noEngine

Δεν χρειάζονται παράμετροι για τον συγκεκριμένο τύπο πτερυγίου

genericEngine

s_prop: Δίσκος περιοχής που καλύπτει η προπέλα
c_prop: Συντελεστής αποδοτικότητας ώθησης κινητήρα. Αυτός μπορεί να πάρει την τιμή 1.0 και να συγχωνευτεί με τον s_prop.
k_motor: Συντελεστής ώθησης που πολλαπλασιάζεται με την είσοδο ελέγχου του κινητήρα και σχετίζεται με την ώθηση του κινητήρα.
k_t_p: Συντελεστής ροπής που παράγεται από τον κινητήρα, πολλαπλασιάζεται με το τετράγωνο της γωνιακής ταχύτητας του κινητήρα.
k_omega: Συντελεστής που πολλαπλασιάζεται με την είσοδο ελέγχου του κινητήρα (με εύρος [0, 1]) για να παραχθεί η γωνιακή ταχύτητα του κινητήρα.

electricEngine

maxThrust: Μέγιστη ώθηση κινητήρα
Kv: Σταθερή ταχύτητα κινητήρα $\left(\frac{RPS}{V}\right)$
Rm: Εσωτερική αντίσταση κινητήρα $(Ohms)$
I0: Ρεύμα ενεργοποίησης κινητήρα $(Ampere)$
RadPSLimits: Το κάτω και άνω όριο του κινητήρα σε μορφή πίνακα $(\frac{rad}{s})$

Παράμετροι προπέλας

propDiam: Διάμετρος προπέλας (m)
nCoeffPoly: Πολυώνυμο μιας μεταβλητής που περιγράφει τον συντελεστής απόδοσης έλικας ως συνάρτηση του advance ratio. Αυτή είναι μια ομάδα μεταβλητών και όλες οι σχετικές με αυτήν παράμετροι πρέπει να δηλώνονται μετά το όνομα αυτό.
propPowerPoly: Πολυώνυμο μιας μεταβλητής που περιγράφει συντελεστής ισχύος έλικας ως συνάρτηση του advance ratio. Αυτή είναι μια ομάδα μεταβλητών και όλες οι σχετικές με αυτήν παράμετροι πρέπει να δηλώνονται μετά το όνομα αυτό.

Άλλοι παράμετροι

engInertia: Η αδράνεια του κινητήρα και της προπέλας μαζί $\left(\frac{Kg}{m^3}\right)$

Rs: Εσωτερική αντίσταση μπαταρίας $(Ohms)$

Cells: Αριθμός των LiPo cells της μπαταρίας (ακέραιος)

4.3.2.5 Δήλωση Πολυωνύμων – Παράμετροι πολυωνύμων

Σε διάφορες περιπτώσεις στον Last_letter_2, χρησιμοποιούνται πολυώνυμα μιας ή περισσοτέρων μεταβλητών για την εξαγωγή κάποιων παραμέτρων. Προκειμένου να προσδιοριστούν αυτά τα πολυώνυμα χρειάζονται να δηλωθούν κάποιες παράμετροι. Παρακάτω εξηγείται ο ρόλος της κάθε παραμέτρου και ο τρόπος με τον οποίο δηλώνεται.

polyType: (ακέραιος) Ο τύπος του πολυωνύμου
0 για μιας μεταβλητής (1D), 1 για πολυώνυμο δύο μεταβλητών (2D), 2 για cubic spline

Μιας μεταβλητής (1D) πολυώνυμο

Για πολυώνυμα μιας μεταβλητής είναι απαραίτητο να δηλωθούν οι παρακάτω μεταβλητές.

polyNo: Βαθμός πολυωνύμου

coeffs: Λίστα που περιέχει τους πολυωνυμικούς συντελεστές, ξεκινώντας από τον μη μηδενικό όρο. Η λίστα πρέπει να έχει polyNo+1 στοιχεία. Για παράδειγμα, για έναν $3^{ου}$ βαθμού 1D πολυώνυμο, το συγκεκριμένο πεδίο δηλώνεται κάπως έτσι:

myPoly/coeffs: [0.0, 2.5, -0.8, -2,6]

Δύο μεταβλητών (2D) πολυώνυμο

Για πολυώνυμα δύο μεταβλητών είναι απαραίτητο να δηλωθούν οι παρακάτω μεταβλητές:

polyNo: (ακέραιος) Η λίστα με του βαθμούς του πολυωνύμου n1 και n1, έναν για κάθε μεταβλητή, v1 και v2. Ο βαθμός της v1 πρέπει να είναι μικρότερος από τον βαθμό της v2, αλλιώς οι μεταβλητές εισόδου θα πρέπει να αλλάξουν σειρά στον κώδικα. Παράδειγμα:

myPoly/polyNo: [1, 5]

coeffs: Λίστα που περιέχει τους πολυωνυμικούς συντελεστές, ξεκινώντας από τον μη μηδενικό όρο. Οι συντελεστές δηλώνονται ξεκινώντας από αυτόν που αντιστοιχίζεται στον όρο $v1^0 * v2^1$, έπειτα στον όρο $v1^1 * v2^1$ κ.λ.π. μέχρι τον όρο $v1^{n1} * v2^0$. Η λίστα πρέπει να περιέχει $\frac{2*n2+2*n1+n1-n1^2+2}{2}$ στοιχεία. Για παράδειγμα, για n1=1 και n2=5, το συγκριμένο πεδίο δηλώνεται κάπως έτσι:

myPoly/coeffs: [7.554e+4, -1076.0, 5.768, -0.01416, 1.66e-5, -7.553e-9,
1.676e+5, -1487.0, 4.831, -0.006781, 3.493e-6]

Για cubic spline είναι απαραίτητο να δηλωθούν οι παρακάτω μεταβλητές:

breaksNo: (ακέραιος) Αριθμός των σημείων μετάβασης που περιέχονται στην spline

breaks: Οι τιμές σημείων μετάβασης. Το μήκος αυτής της λίστας πρέπει να είναι breaksNo+1 με τον πρώτο στοιχείο να ενεργεί ως το κατώτατο όριο του αναμενόμενου εύρους.

coeffs: Λίστα που περιέχει τους cubic spline συντελεστές για κάθε τμήμα spline. Κάθε γραμμή έχει τέσσερα στοιχεία, ξεκινώντας από τον μη μηδενικό όρο μέχρι τον 3^{o} όρο. Ο πίνακας έχει 4*breaksNo στοιχεία. Ωστόσο, δεδομένου ότι τα .yaml αρχεία δέχονται μόνο εισόδους σε μορφή λίστας, όλες οι γραμμές του πίνακα ενώνονται σε μια γραμμή. Για παράδειγμα, για μια spline με 3 σημεία μετάβασης, το συγκεκριμένο πεδίο δηλώνεται κάπως έτσι:

```
myPoly/coeffs: [0.1, 0.2,  0.3,  0.4,
                0.15, 0.25, 0.24, 0.45,
                0.28, 0.28, 0.38, 0.48]
```

4.4 Συγγραφή Αλγορίθμου Ελέγχου

Ο συγκεκριμένος προσομοιωτής δίνει στον χρήστη την δυνατότητα να επιλέξει και να προγραμματίσει τον αλγόριθμο του controller που θα τρέχει στον μοντέλο του. Ο κώδικας του controller τρέχει σε ξεχωριστό[21] node και βρίσκεται σε ένα αρχείο μόνο. Αυτό, κάνει αφενός πιο ευδιάκριτη την εικόνα, πιο εύκολη την πρόσβαση στο συγκεκριμένο node και γρήγορη την προσθήκη ή τροποποίησή ενός νέου αλγορίθμου controller.

Κάθε νέος controller προγραμματίζεται σύμφωνα με την δομή της κλάσης Controller, όπως στα ήδη υπάρχοντα αρχεία των default controller του προσομοιωτή στον φάκελο ***<catkin_workspace>/src/last_letter_2/last_letter_2/src/controllers*.** Ο χρήστης καλείται να ακολουθήσει αυτήν την δομή και πάνω εκεί να προγραμματίσει τον αλγόριθμό του. Μια κλάση τύπου Controller περιέχει κάποιες μεθόδους. Η **chan2signal()** κρατάει ενήμερο το *controller_n* με την πιο πρόσφατη κατάσταση του πίνακα καναλιών παρακολουθώντας το topic */channels*. Η **storeStates()** φορτώνει σε κάθε κύκλο την πρόσφατη κατάσταση του μοντέλου παρακολουθώντας το topic */model_states*. Η **controlLaw()** περιέχει τον αλγόριθμο ελέγχου του μοντέλου. Τέλος, η **returnControlInputs()** υπολογίζει και στέλνει στο *core_n* τα σήματα ελέγχου. Ο χρήστης ασχολείται και τροποποιεί κυρίως την **controlLaw**() μέθοδο σύμφωνα με τις ανάγκες του. Έχοντας στην διάθεσή τα σήματα εισόδου του χρήστη και την κατάσταση του μοντέλου, καλείται να συνθέσει οποιονδήποτε αλγόριθμο ελέγχου επιθυμεί.

4.5 Προσθήκη επιμέρους λειτουργιών

Ο χρήστης μέσω του joystick έχει την δυνατότητα να στείλει πολλά επιπλέον σήματα στον προσομοιωτή, πέραν των βασικών τεσσάρων. Τα σήματα αυτά μπορούν να αξιοποιηθούν για την επιτέλεση κάποιων επιπλέον λειτουργιών, για παράδειγμα την κλήση ενός default service του Gazebo, όπως επανεκκίνηση της προσομοίωσης ή και άλλων λειτουργιών στην μεριά του ROS όπως η επιλογή ή απομόνωση κομμάτι κώδικα.

[21] Αναλυτική περιγραφή του controller node στο Κεφάλαιο 7.1.2

Καθώς ο κώδικας του προσομοιωτή είναι χωρισμένος σε πολλά node και τρέχει σε δύο προγράμματα, η εξυπηρέτηση αυτών των λειτουργιών από ένα μόνο σημείο είναι αδύνατη. Έτσι έχουν οριστεί τρία σημεία στα οποία ο χρήστης έχει την δυνατότητα να αξιοποιήσει κάποιο από τα κανάλια σημάτων. Αυτά είναι:

- *controller_n*
- *chan2srv_n*
- *gazebo_n* (model plugin)

controller_n

Στον *controller_n* υπάρχει η μέθοδος **channelFunctions()**, στην οποίο ο χρήστης έχει την δυνατότητα να αντιστοιχίσει διάφορες λειτουργίες στα κανάλια εισόδου του χρήστη, γράφοντας τον δικό του κώδικα.

chan2srv_n

Το node αυτό δημιουργήθηκε εξ ολοκλήρου για αυτόν τον σκοπό. Αφορά κυρίως την κλήση έτοιμων default services του Gazebo, από την μεριά του ROS. Στην παρούσα μορφή του, εισάγει μοντέλα μικρών κύβων κάτω από το μοντέλο μέσω ενός κουμπιού και διαγράφει όλα τα ήδη υπάρχοντα μοντέλα κύβων μέσω άλλου κουμπιού.

***gazebo_n* (model plugin)**

Στο κώδικα του model plugin του μοντέλου, υπάρχει η μέθοδος **manageChan()**, για κλήση λειτουργιών μέσω σημάτων καναλιών. Εκεί ο χρήστης μπορεί να προσθέσει τον δικό του κώδικα που θα ακούει στην τιμή ενός καναλιού. Αυτήν την στιγμή, μέσω του 5ου, και 6ου καναλιού, ελέγχεται η στρέψη των δύο link αισθητήρων του μοντέλου plane_1wing_sensors, δηλαδή της κάμερας και του laser scanner.

5

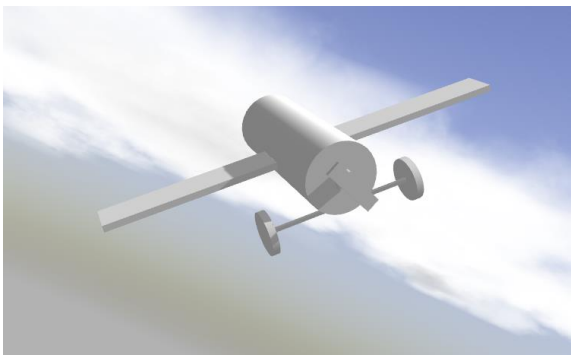
Βασικά Μοντέλα

Ο προσομοιωτής περιέχει 5 έτοιμα μοντέλα, τρία αεροπλάνα, ένα quadcopter και ένα hexacopter.

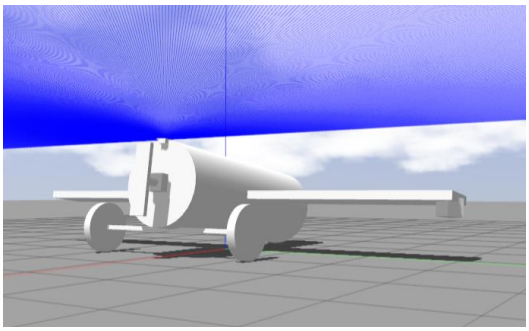
Τα έτοιμα μοντέλα, αποτελούν μια αφετηρία για τους μελλοντικούς χρήστες ώστε αφενός να μπορούν να ξεκινήσουν μια προσομοίωση και να δοκιμάσουν έναν controller σύντομα, αποκτώντας έτσι εμπειρία για την λειτουργία και λογική του προσομοιωτή, αφετέρου για να κατανοήσουν καλύτερα την δομή και την περιγραφή ενός μοντέλου πριν επιχειρήσουν να χτίσουν το δικό τους ερευνητικό ρομπότ.

Τα έτοιμα μοντέλα του προσομοιωτή είναι τα εξής:

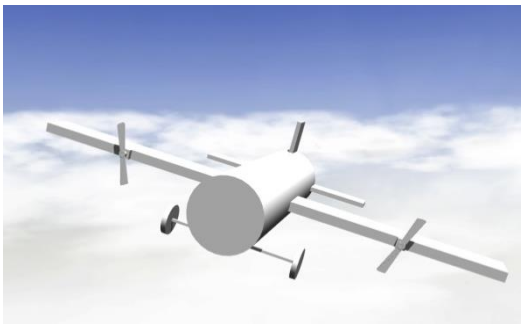
1-wing plane



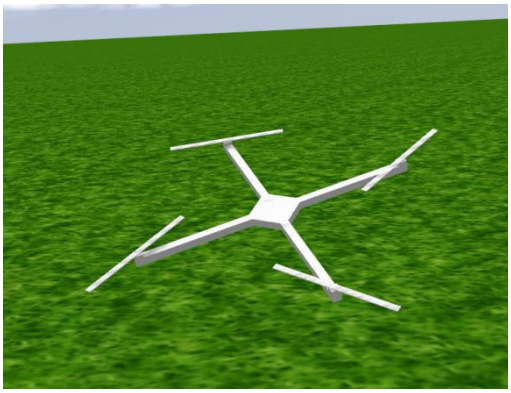
1-wing plane with sensors



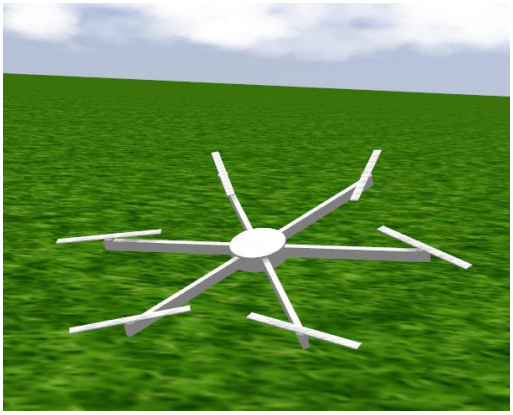
3-wing plane



quadcopter

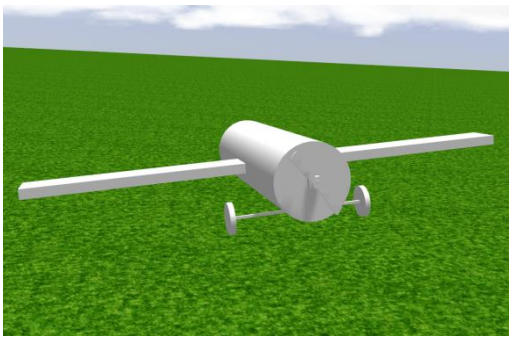


hexacopter



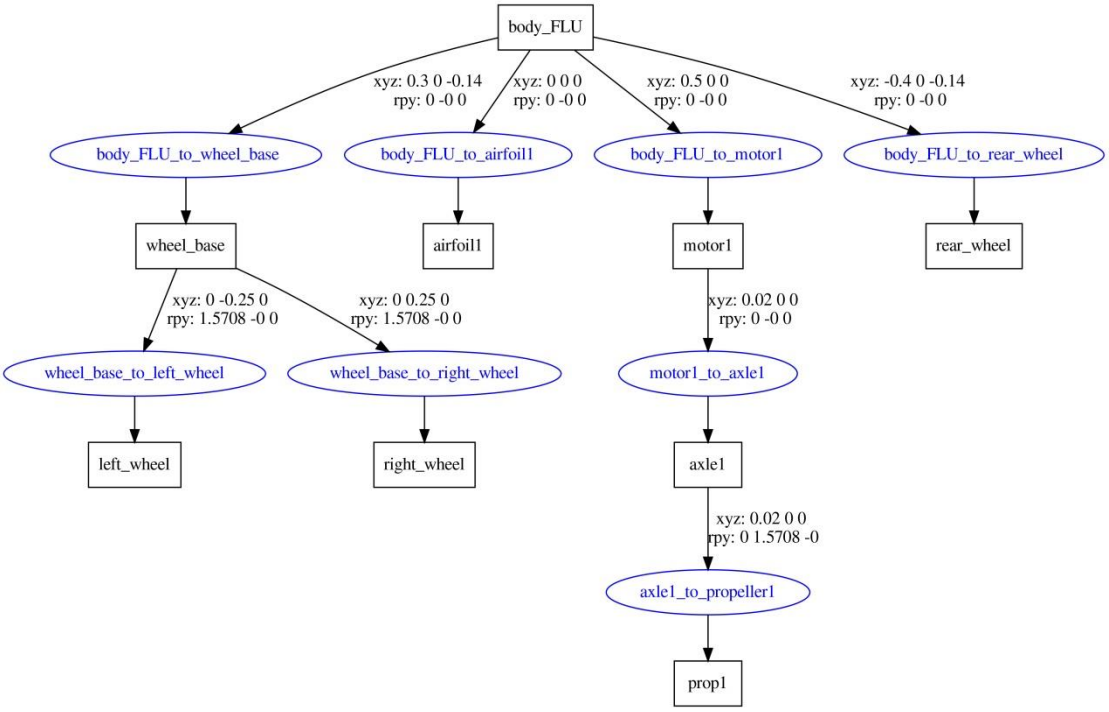
5.1 ***1-wing plane*** *(plane_1wing)*

Αριθμός πτερυγίων: 1
Τύπος πτερυγίων: stdLinearAero
Αριθμός κινητήρων: 1
Τύπος κινητήρων: genericEngine



Εικόνα 22. plane_1wing model

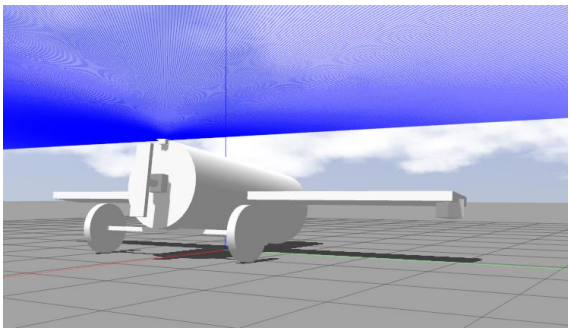
Αυτό είναι το πιο απλό μοντέλο fix-wing μοντέλου. Αποτελείται από τον κορμό του αεροπλάνου, ένα πτερύγιο που περνάει από το κέντρο μάζας του, τον κινητήρα του και κάποια links που σχηματίζουν τους τροχούς, χρήσιμα για την απογείωση και προσγείωση. Η αεροδυναμική του πτερυγίου είναι τύπου stdLinearAero και ο κινητήρας τύπου genericEngine. Το μοντέλο αυτό χρησιμοποιείται όταν ο χρήστης έχει γνωστά τα μαθηματικά και την αεροδυναμική για όλο το αεροπλάνο, δηλαδή το αντιμετωπίζει σαν σύνολο και όχι σαν συνδυασμό αεροδυναμικών επιφανειών. Έτσι όλες οι δυνάμεις και ροπές που οφείλονται στην αεροδυναμική του μοντέλου ασκούνται στο κέντρο μάζας του. Ο κινητήρας αποτελεί ξεχωριστό φυσικά τμήμα καθώς δεν είναι κομμάτι της αεροδυναμικής αλλά του συστήματος προώθησης του αεροπλάνου.



Εικόνα 23. URDF graph with Links & Joints – plane_1wing

5.2 ***1-wing plane with sensors*** *(plane_1wing_sensors)*

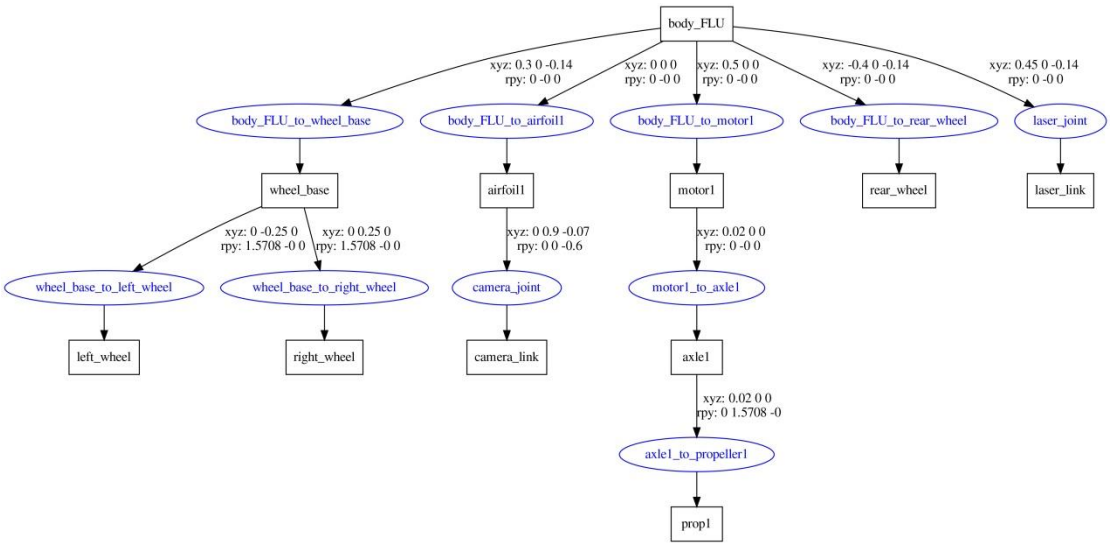
Αριθμός πτερυγίων: 1
Τύπος πτερυγίων: polyAero
Αριθμός κινητήρων: 1
Τύπος κινητήρων: genericEngine



Εικόνα 24. plane_1wing_sensors model

Το μοντέλο αυτό είναι σχεδόν ίδιο με το προηγούμενο. Οι διαφορές είναι ότι η αεροδυναμική των συντελεστών του lift και drag γίνεται μέσω πολυωνύμων που έχει ορίσει ο χρήστης και πως σε αυτό το μοντέλο έχουν προστεθεί τρεις default αισθητήρες του Gazebo, μια κάμερα, ένα laser scanner και ένας αισθητήρας IMU.

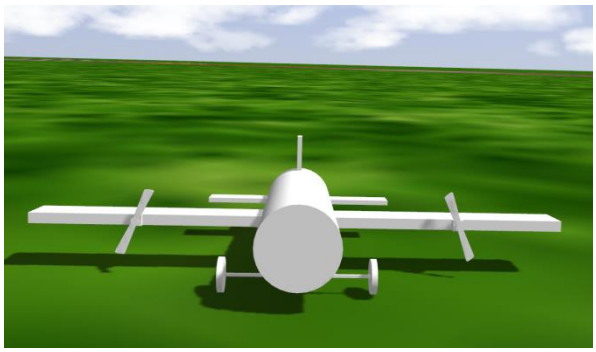
Για κάθε αισθητήρα έχει φορτωθεί ένα έτοιμο Sensor plugin. Τα Sensor Plugins δηλώνονται και περιγράφονται στο URDF αρχείο μέσα στο tag <gazebo>, όπως ακριβώς και τα Model Plugins. Μόνο που στην περίπτωση των Sensors πρέπει να δοθεί και ένα link αναφοράς. Τα δεδομένα των Sensor Plugin είναι διαθέσιμα μέσω αντίστοιχων topic.



Εικόνα 25. URDF graph with Links & Joints – plane_1wing_sensors

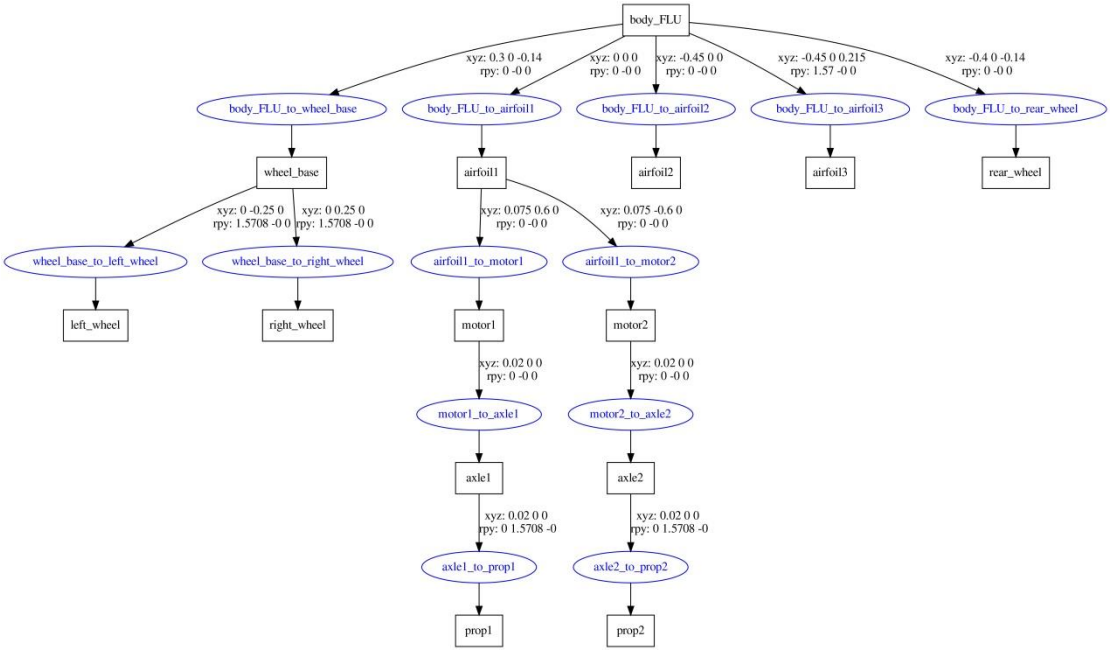
5.3 ***3-wing plane*** *(plane_3wing_2motor)*

Αριθμός πτερυγίων:	3
Τύπος πτερυγίων:	stdLinearAero
Αριθμός κινητήρων:	2
Τύπος κινητήρων:	electricEngine



Εικόνα 26. plane_3wing_2motor model

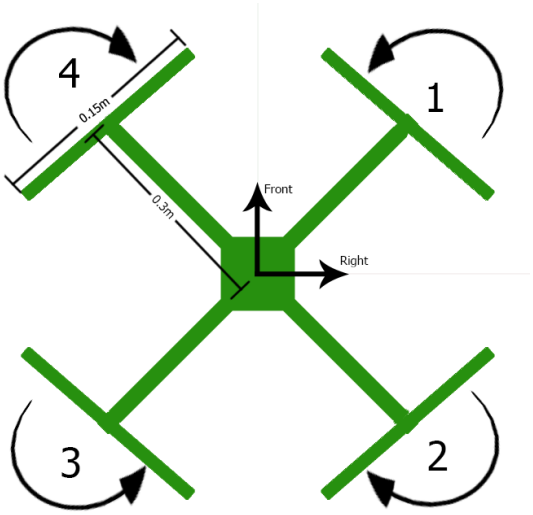
Αυτό είναι το δεύτερο fixed-wing μοντέλο του προσομοιωτή. Αποτελείται από τον κορμό του αεροπλάνου, 3 πτερύγια, τον κινητήρα του αεροπλάνου και κάποια links που σχηματίζουν τις ρόδες του, χρήσιμες για την απογείωση και προσγείωση. Τα 3 πτερύγια αντιστοιχίζονται στα πτερύγια ενός πραγματικού αεροπλάνου. Το πρώτο, το μεγαλύτερο που βρίσκεται τοποθετημένο οριζόντια στο κέντρο περίπου του κορμού του μοντέλου, παίζει τον ρόλο των φτερών ενός αεροπλάνου (wings), και είναι υπεύθυνο να παράγει το βασικό lift του αεροπλάνου καθώς επίσης και την στροφή roll (ailerons). Τα άλλα δύο είναι τοποθετημένα στην ουρά του μοντέλου, το ένα οριζόντια και το άλλο κάθετα και παίζουν τον ρόλο του οριζόντιου και κάθετου σταθεροποιητή, υπεύθυνων για τις κινήσεις pitch (elevators) και yaw (rudder). Οι κινητήρες όπως και στο προηγούμενο μοντέλο αποτελούν ξεχωριστό τμήμα ως σύστημα προώθησης. Η αεροδυναμική των πτερυγίων είναι τύπου stdLinearAero και οι κινητήρες τύπου electricEngine. Το μοντέλο αυτό χρησιμοποιείται όταν ο χρήστης έχει φτιάξει ένα αεροπλάνο ως ένα συνδυασμό από πτερύγια και κινητήρες και δεν γνωρίζει την συνολική αεροδυναμική συμπεριφορά του μοντέλου αλλά έχει πληροφορίες για την αεροδυναμική κάθε επιφάνειας και τις ωθήσεις των κινητήρων αυτών. Σε αυτήν την περίπτωση δίνονται τιμές στις παραμέτρους των πτερυγίων που αφορούν το lift, drag και m (torque). Οι υπόλοιπες μηδενίζονται.



Εικόνα 27. URDF graph with Links & Joints – plane_1wing_sensors

5.4 *Quadcopter*

Εικόνα 29. Quadcopter model



Εικόνα 28. Quadcopter design

Αριθμός πτερυγίων: 0
Τύπος πτερυγίων: -
Αριθμός κινητήρων: 4
Τύπος κινητήρων: genericEngine

Αυτό το μοντέλο ανήκει στην κατηγορία των multirotors και είναι ένα quadcopter. Το μοντέλο αυτό αποτελείται από ένα βασικό κορμό στον οποίο είναι ενωμένοι 4 άξονες. Κάθε άξονας στην άκρη του βαστάει έναν κινητήρα. Τα multirotors δεν έχουν αεροδυναμικές επιφάνειες παρά μόνο κινητήρες ώθησης. Οι κινητήρες είναι τύποι genericEngine.

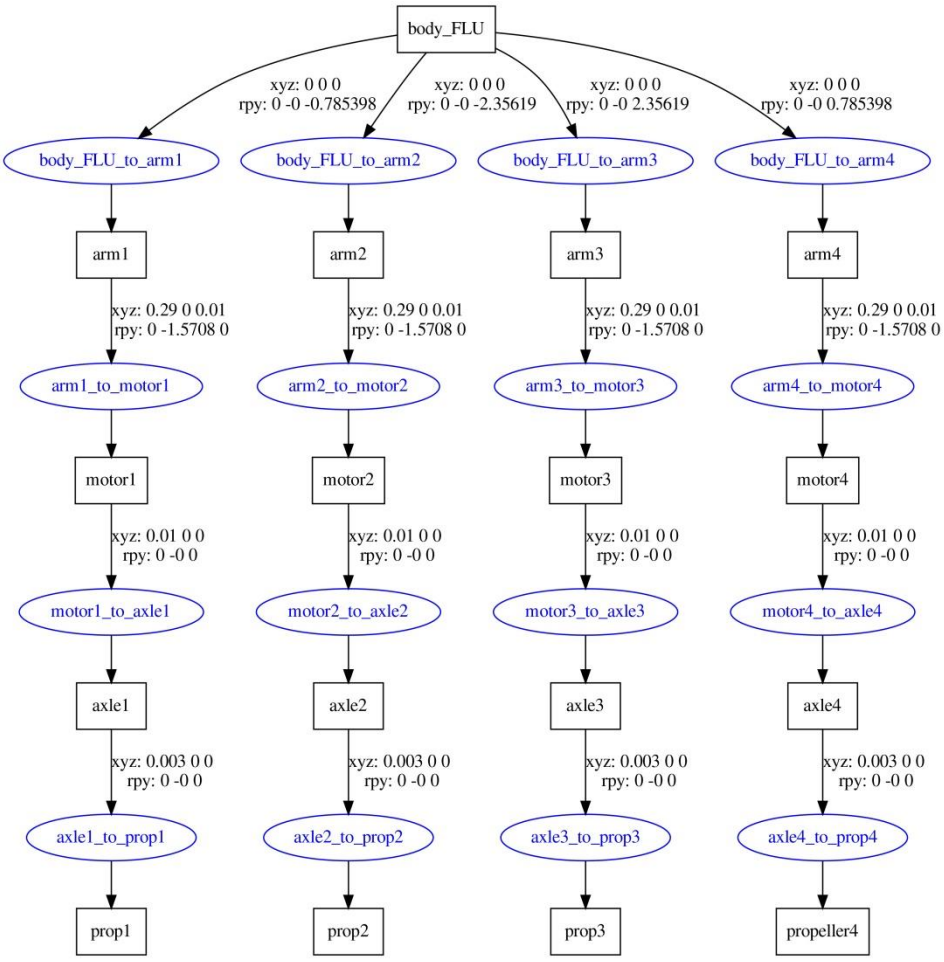
Τα multirotor χαρακτηρίζονται και από έναν πίνακα που μεταφράζει τις εισόδους των κινητήρων σε σήματα ελέγχου κινήσεων, δηλαδή throttle, roll, pitch, yaw. Για το quadcopter ο πίνακας είναι ο εξής.

$$[u] = \begin{bmatrix} u\_throttle \\ u\_roll \\ u\_pitch \\ u\_yaw \end{bmatrix} = \frac{1}{4} * \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix} * \begin{bmatrix} M1 \\ M2 \\ M3 \\ M4 \end{bmatrix}$$

Ο αντίστροφος αυτού, μετατρέπει τα σήματα κινήσεων σε σήματα εισόδου κινητήρων.

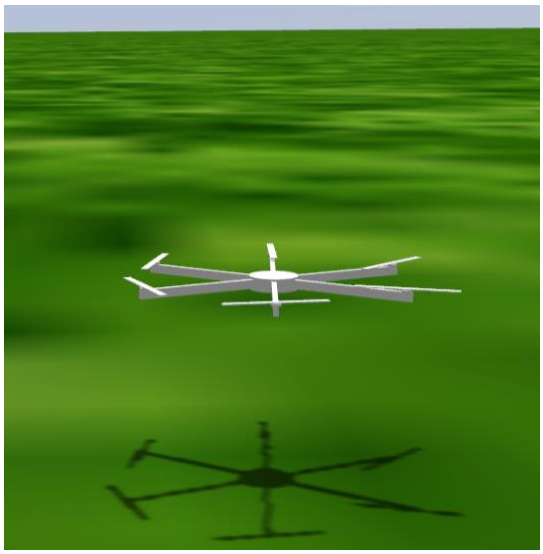
$$[M] = \begin{bmatrix} M1 \\ M2 \\ M3 \\ M4 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} u\_throttle \\ u\_roll \\ u\_pitch \\ u\_yaw \end{bmatrix}$$

Όπου [u] ο πίνακας με τα σήματα ελέγχου του drone και [M] ο πίνακας με τα σήματα ελέγχου των κινητήρων. Ο controller του quadcopter αφού υπολογίσει τα σήματα ελέγχου κινήσεων, τα μεταφράζει μέσω του αντίστροφου πίνακα σε σήματα κινητήρων και μετά τα στέλνει στο *core_n*, φορτώνοντάς τα στον πίνακα καναλιών.

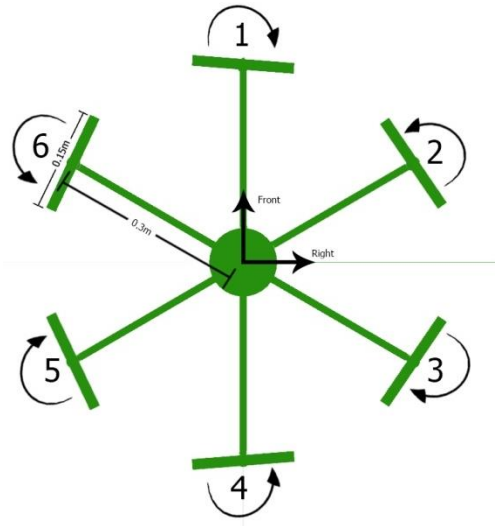


Εικόνα 30. URDF graph with Links & Joints – quadcopter

5.5 *Hexacopter*



Εικόνα 32. hexacopter model



Εικόνα 31. Hexacopter design

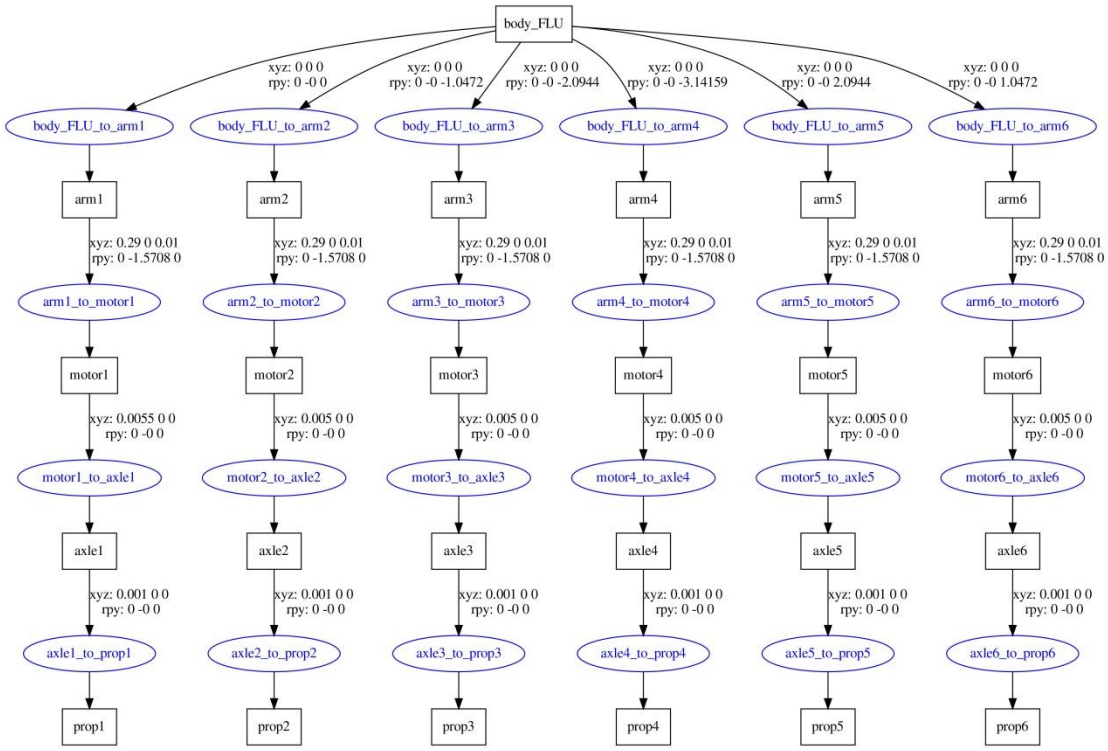
Αριθμός πτερυγίων: 0
Τύπος πτερυγίων: -
Αριθμός κινητήρων: 6
Τύπος κινητήρων: genericEngine

Άλλο ένα multirotor μοντέλο, το hexacopter, αποτελούμενο από έναν κορμό, 6 βραχίονες και 6 κινητήρες. Οι κινητήρες είναι τύπου genericEngine.

Οι πίνακες του hexacopter είναι οι εξής:

$$[u] = \begin{bmatrix} u\_throttle \\ u\_roll \\ u\_pitch \\ u\_yaw \end{bmatrix} = \frac{1}{6} * \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & -1.5 & -1.5 & 0 & 1.5 & 1.5 \\ 2 & 1 & -1 & -2 & -1 & 1 \\ -2 & 0.5 & -0.5 & 2 & -0.5 & 0.5 \end{bmatrix} * \begin{bmatrix} M1 \\ M2 \\ M3 \\ M4 \\ M5 \\ M6 \end{bmatrix}$$

$$[M] = \begin{bmatrix} M1 \\ M2 \\ M3 \\ M4 \\ M5 \\ M6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0.5 & -1 \\ 1 & -1 & 1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & 0 & -0.5 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} u\_throttle \\ u\_roll \\ u\_pitch \\ u\_yaw \end{bmatrix}$$



Εικόνα 33. URDF graph with Links & Joints – hexacopter

6

Συμπληρωματικές λειτουργίες

Ο Last_letter_2, ως εφαρμογή ανεπτυγμένη πάνω στο ROS, συνδυάζεται με τις επιπλέον δυνατότητα και χαρακτηριστικά που προφέρει το ROS. Σε αυτό το κεφάλαιο παρουσιάζονται κάποιες χρήσιμες για τον προσομοιωτή επεκτάσεις του ROS.

6.1 Rviz

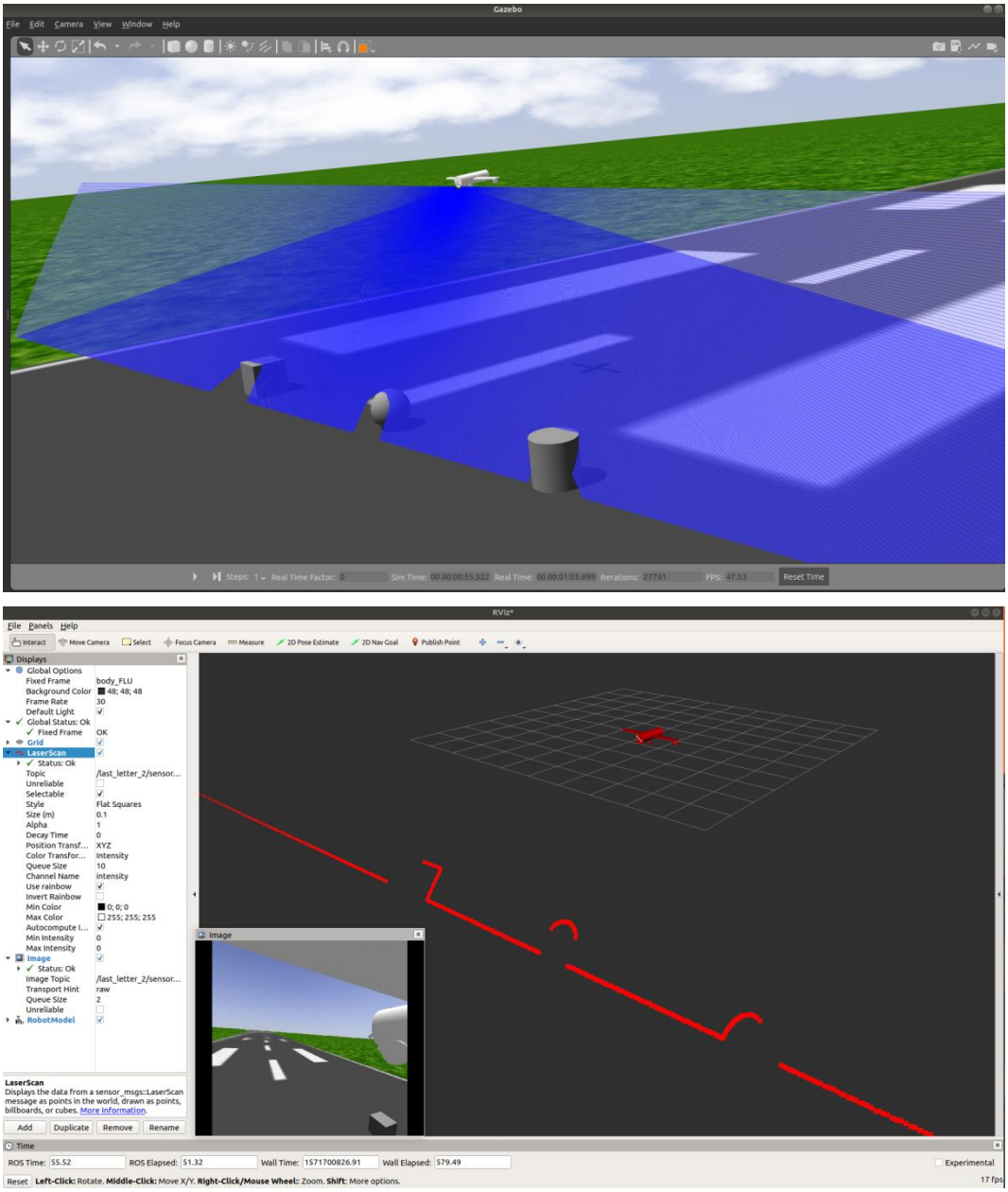
Το RVI είναι ένα 3D περιβάλλον για οπτικοποίηση των παραμέτρων του ROS. Με αυτό ο χρήστης μπορεί να οπτικοποιήσει το μοντέλο, το δέντρο tf, τα δεδομένα από τους αισθητήρες που είναι φορτωμένοι στο μοντέλο και άλλα. Μία χρήσιμη αξιοποίηση του Rviz είναι η οπτικοποίηση των δεδομένων που παράγουν οι αισθητήρες εν ώρα προσομοίωσης.

Το plane_1wing_sensors μοντέλο, διαθέτει έναν αισθητήρα lazer scanner και μια κάμερα. Ο χρήστης επιλέγοντας στην δεξιά στήλη τον τύπο δεδομένων που θέλει να οπτικοποιήσει μπορεί εύκολα να παρατηρεί τις τιμές αυτών των αισθητήρων.

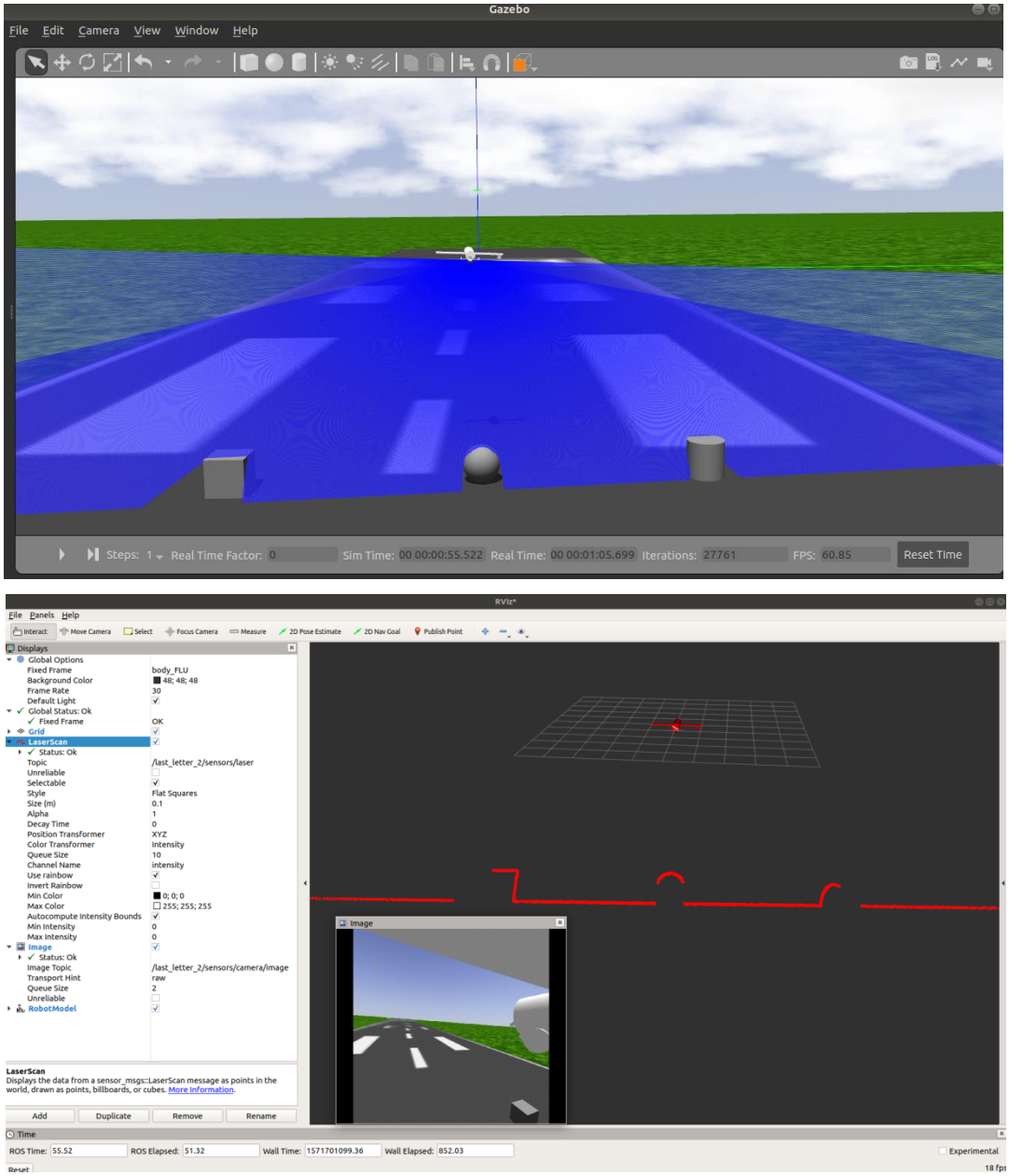
Για την οπτικοποίηση της δέσμης του laser scanner, φορτώνεται ο τύπος LaserScan από την λίστα τύπων αισθητήρα και επιλέγεται το κατάλληλο topic στο οποίο δημοσιεύονται τα δεδομένα του αισθητήρα.

Για την οπτικοποίηση της εικόνας της κάμερα, φορτώνεται ο τύπος Image από την λίστα τύπων αισθητήρα και επιλέγεται το κατάλληλο topic στο οποίο δημοσιεύεται η εικόνα της κάμερας.

Παρακάτω προβάλλεται ένα στιγμιότυπο της προσομοίωσης, όπως φαίνεται από το περιβάλλον προσομοίωσης (Gazebo) και όπως από το περιβάλλον οπτικοποίησης του ROS (Rviz). Το στιγμιότυπο δείχνει το αεροπλάνο με τον αισθητήρα να περνάει πάνω από κάποια αντικείμενα. Τα σημεία στα οποία η δέσμη βρίσκει κάποιο εμπόδιο, είτε αντικείμενο είτε το έδαφος είναι χρωματισμένα με κόκκινο χρώμα.



Εικόνα 34. Στιγμιότυπο προσομοίωσης όπως φαίνεται από το περιβάλλον του Gazebo (πάνω) και όπως από το περιβάλλον του Rviz (κάτω)



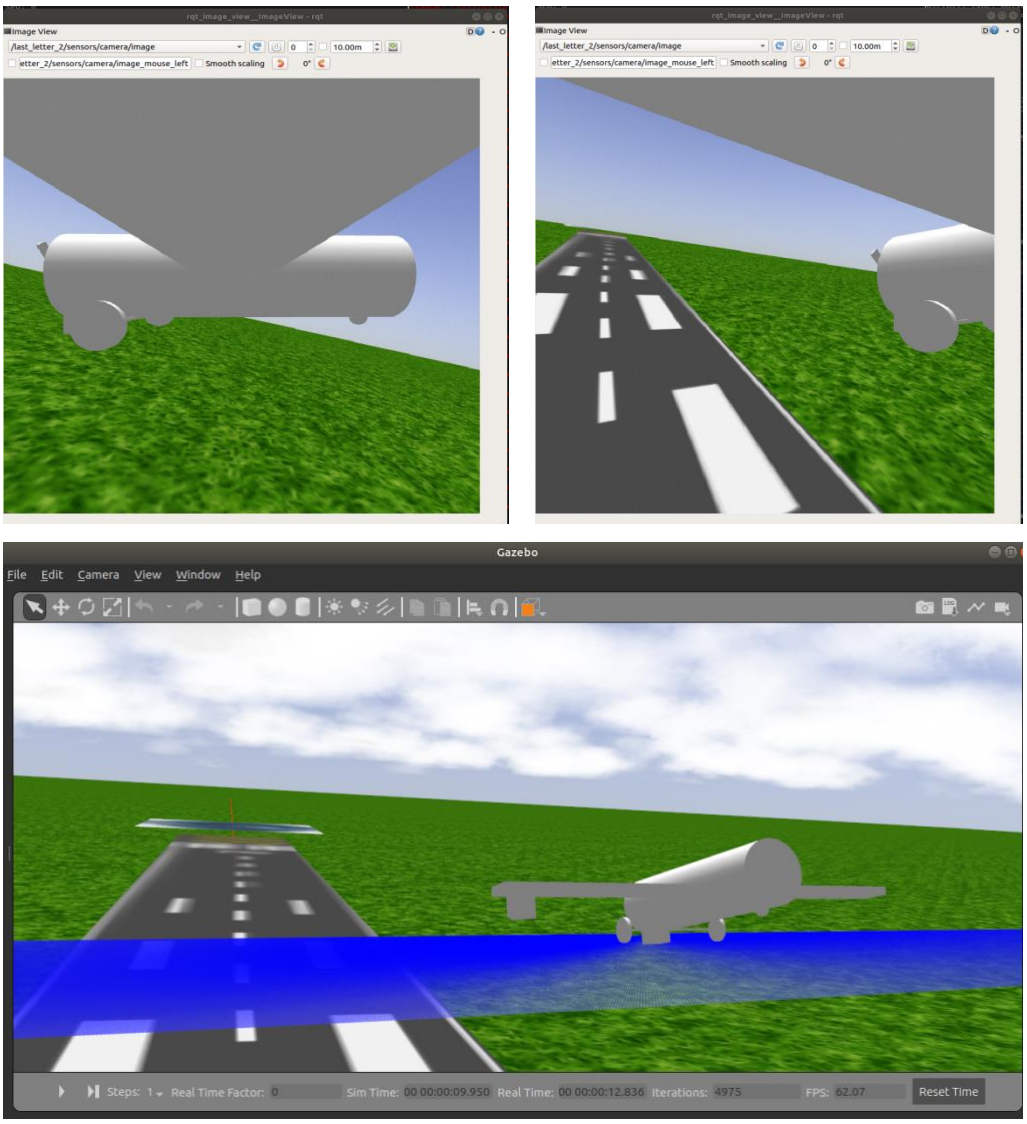
Εικόνα 35. Το ίδιο στιγμιότυπο από άλλη οπτική γωνία

6.2 Rqt

Το rqt είναι ένα software framework του ROS που προσφέρει διάφορα εργαλεία GUI. Ο χρήστης μπορεί να καλεί και να χρησιμοποιεί αυτά τα εργαλεία μέσω του terminal.

6.2.1 Rqt_image_view

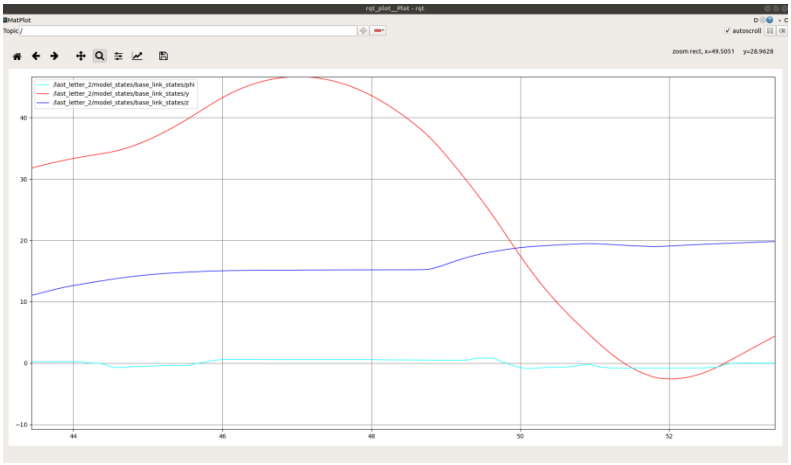
Εργαλείο οπτικοποίησης της εικόνας της κάμερας. Δίνει ακριβώς την ίδια εικόνα με αυτήν του Rviz. Όμως είναι ένας πιο γρήγορος και απλός τρόπος. Καλείται μέσω της εντολής **rqt_image_view**.



Εικόνα 36. Εικόνες της κάμερας μέσω του rqt_image_view (πάνω). Η θέση του μοντέλου (κάτω)

6.2.2 Rqt_plot

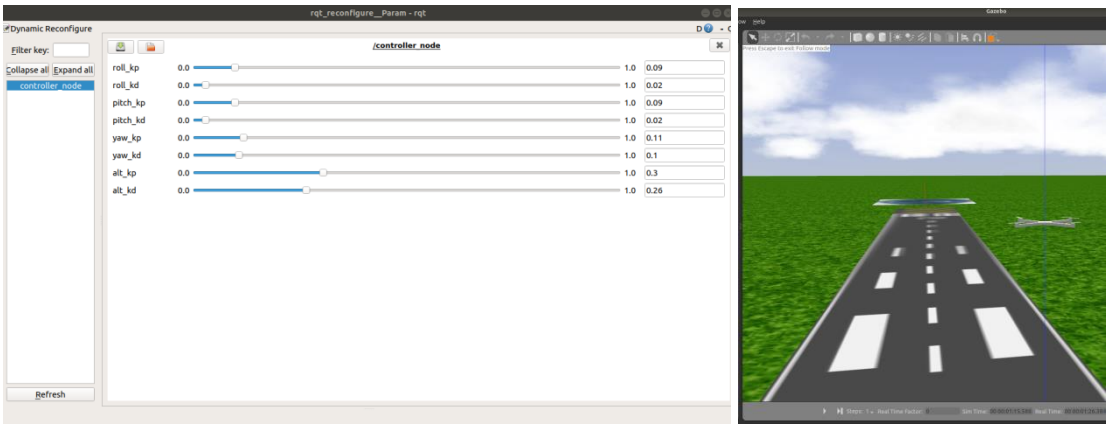
Εργαλείο οπτικοποίησης δεδομένων που δημοσιεύονται σε topics συναρτήσει του χρόνου σε δισδιάστατη μορφή κυματομορφής. Καλείται μέσω της εντολής **rqt_plot** και τα δεδομένα φορτώνονται σημειώνοντας στο πάνω μέρος το πλήρες όνομα του topic.



Εικόνα 37. Στιγμιότυπο από το rqt_plot. Οι γραμμές δείχνουν τις τιμές θέση στον άξονα z (μπλε), ταχύτητα στον άξονα y (κόκκινη) και γωνία φι (γαλάζια) ενός quadcopter σε συνάρτηση του χρόνου.

6.2.3 Rqt_dynamic_reconfigure[22] – Live tuning

Εργαλείο ελέγχου τιμής παραμέτρων σε πραγματικό χρόνο. Αυτό το εργαλείο δίνει την δυνατότητα για tuning των controller σε πραγματικό χρόνο. Από τους τρείς default controller του προσομοιωτή, ο controller του quadcopter περιλαμβάνει κάποιες γραμμές κώδικα για αυτόν ακριβώς τον σκοπό. Αξίζει να σημειωθεί πως ο χρήστης μπορεί να αποθηκεύει τις παραμέτρους του σε μορφή αρχείων .yaml πατώντας στο πάνω αριστερό μέρος της εφαρμογής το εικονίδιο αποθήκευσης.



Εικόνα 38. Πίνακας rqt_dynamic_reconfigure για ρύθμιση των μεταβλητών σε πραγματικό χρόνο (αριστερά). Στιγμιότυπο από το Gazebo (δεξιά)

[22] Λεπτομέρειες για το πώς χρησιμοποιείται βρίσκονται εδώ:
http://wiki.ros.org/dynamic_reconfigure/Tutorials

Β' Μέρος

Κατασκευή Εφαρμογής

7

Δομή Προσομοιωτή –Επεξήγηση κώδικα

Ο προσομοιωτής αποτελείται από τέσσερα πακέτα.

- last_letter_2
- last_letter_2_gazebo_plugins
- last_letter_2_libs
- last_letter_2_msgs

7.1 Πακέτο last_letter_2

Σε αυτό το πακέτο βρίσκεται το μεγαλύτερο και πιο κύριο μέρος του κώδικα και αφορά τα node *core_n, controller_n, joy2chan_n* και *chan2srv_n*. Σε αυτά τρέχουν οι υπολογισμοί της δυναμικής του μοντέλου, ο αλγόριθμος ελέγχου του μοντέλου και κάποιες επιπλέον βοηθητικές συναρτήσεις. Ο κώδικας του *core_n*, είναι χωρισμένος σε επιμέρους αρχεία. Αυτό βοηθάει τόσο στην κατανόησή του όσο και στην διαχείριση και διόρθωσή του. Στην συνέχεια εξηγούνται όλα τα επιμέρους αρχεία αυτού του πακέτου.

7.1.1 Core node

Όπως έχει αναφερθεί, η δυναμική του συστήματος υπολογίζεται στο *core_n*. Εξαιτίας της πολυπλοκότητας αυτού του node και των πολλών κλάσεων, υποκλάσεων και μεθόδων που περιέχει, ο κώδικας έχει μοιραστεί σε επιμέρους μικρότερα αρχεία. Τα αρχεία αυτά είναι τα εξής:

- **core.cpp**
- **model.cpp**
- **environment.cpp**
- **dynamics.cpp**
- **factory.cpp**

- **aerodynamics.cpp**
- **noAerodynamics.cpp**
- **stdLinearAero.cpp**
- **polyAero.cpp**
- **propulsion.cpp**
- **noEngine.cpp**
- **genericEngine.cpp**
- **electricEngine.cpp**

Ακολουθεί λεπτομερή ανάλυση για κάθε ένα από τα παραπάνω αρχεία του προσομοιωτή.

7.1.1.1 core.cpp

Εδώ δημιουργείται ένα αντικείμενο τύπου Model, που όπως δηλώνει και το όνομα είναι μια κλάση που επιτελεί όλες τις ενέργειες και υπολογισμούς που σχετίζονται με το μοντέλο της προσομοίωσης, και ξεκινάει να τρέχει ένας spinner όπου είναι υπεύθυνος να εξυπηρετεί τις callback του αντικειμένου model. Ως spinner επιλέχθηκε τόσο σε αυτό το node όσο και σε κάποια άλλα ο ασύγχρονος spinner (AsyncSpinner) που δίνει ως επιλογή το ROS έναντι του κλασικού απλού spinner που χρησιμοποιείται τις περισσότερες φορές, διότι παρατηρήθηκε έπειτα από δοκιμές, πως ο AsyncSpinner εξυπηρετεί πιο γρήγορα τα αιτήματα των callbacks βελτιώνοντας έτσι την συνολική απόδοση της εφαρμογής.

7.1.1.2 model.cpp

Το αρχείο αυτό περιέχει όλη την δομή της κλάσης Model. Στην συνέχεια αναλύονται οι μέθοδοι της κλάσης αυτής.

Model(): environment(this), dynamics(this) - constructor

Εδώ αρχικά δημιουργούνται δύο υποκλάσεις, οι environment και dynamics, στις οποίες στέλνεται ένας δείκτης στη κλάσης Model, ώστε να μπορούν να έχουν πρόσβαση στις μεταβλητές της. Στην πρώτη υπολογίζονται τιμές σχετικές με το περιβάλλον της προσομοίωσης ενώ στην δεύτερη υπολογίζεται η δυναμική του μοντέλου. Έπειτα φορτώνονται από τον Parameter server κάποιες παράμετροι του μοντέλου. Ακολουθεί δήλωση του subscriber στο topic */model_states* και των services με το *gazebo_n* και το *controller_n*. Τέλος στέλνεται ένα service call στο gazebo για να παγώσει η προσομοίωση, περιμένοντας τον χρήστη να δώσει το σήμα έναρξής της και αρχικοποιούνται κάποιες μεταβλητές.

gazeboStatesClb(const last_letter_2_msgs::model_states::ConstPtr& msg)

Αυτή η μέθοδος είναι callback στο topic */model_states*, στο οποίο γράφει μόνο το Gazebo στο τέλος κάθε κύκλου προσομοίωσης. Με άλλα λόγια, καλείται κάθε φορά που γεμίζει το topic με ένα νέο μήνυμα. Ο ρόλος της είναι να ανανεώνει τις μεταβλητές της κλάσης που αφορούν την κατάσταση του μοντέλου, βάση του νέου μηνύματος που στάλθηκε από το gazebo. Τέλος, η callback αυτή πυροδοτεί την έναρξη όλων των υπόλοιπων υπολογισμών της δυναμικής μέσω της συνάρτησης modelStep().

modelStep()

Η μέθοδος αυτή καλεί με τις σειρά όσες μεθόδους παίζουν ρόλο στον κύκλο των υπολογισμών της δυναμικής. Ο ρόλος της είναι να κάνει πιο ευανάγνωστο τον κώδικα.

getControlInputs()

Η συνάρτηση αυτή στέλνει αίτημα στο *controller_n* για τα control inputs μέσω service call. Ο controller αφού υπολογίσει τα νέα σήματα εισόδου του μοντέλου, τα φορτώνει στον πίνακα καναλιών και επιστρέφει τον πίνακα. Μόλις ανακτηθεί ο ανανεωμένος πίνακας καναλιών, αποθηκεύονται οι είσοδοι των πτερυγίων και των κινητήρων σε σχετικές μεταβλητές της κλάσης Model.

getAirdata()

Εδώ καλείται η μέθοδος calculateAirdata της κλάσης μέλους environment, ώστε να υπολογιστούν οι τιμές των περιβαλλοντικών συνθηκών γύρω από το μοντέλο, δεδομένου της θέσης του, οι οποίες είναι η θερμοκρασία, η πυκνότητα του αέρα, η ατμοσφαιρική πίεση και τα διανύσματα του αέρα. Έπειτα, τα διανύσματα του αέρα που είναι εκφρασμένα ως προς το σταθερό σύστημα συντεταγμένων του κόσμου inertial_NWU στρέφονται ώστε να εκφραστούν στο σύστημα συντεταγμένων του μοντέλου body_FLU.

calcDynamics()

Εδώ καλούνται οι δύο μέθοδοι της κλάσης μέλους dynamics, οι calcAero και clacProp. Όπως δηλώνουν και τα ονόματα, οι μέθοδοι αυτοί υπολογίζουν την δυναμική των πτερυγίων και των κινητήρων του μοντέλου αντίστοιχα.

applyWrenches()

Τέλος, αφού έχουν προηγηθεί όλες οι παραπάνω μέθοδοι, οι δυνάμεις και ροπές του μοντέλου για το νέο βήμα προσομοίωσης είναι έτοιμες, οπότε στέλνονται μέσω αυτής της μεθόδου στο Gazebo μέσω του *gazebo_n* για να εφαρμοστούν στα σωστά σημεία του μοντέλου (links). Η αποστολή τους γίνεται μέσω service. Πριν σταλούν προηγείται ο μετασχηματισμός τους από το FRD σύστημα όπου είναι υπολογισμένες στο FLU σύστημα συντεταγμένων, το οποίο διαχειρίζεται το Gazebo.

7.1.1.3 environment.cpp

Εδώ βρίσκεται η κλάση Environment, στην οποία υπολογίζονται η θερμοκρασία, τα διανύσματα του ανέμου, η πυκνότητα του αέρα και η ατμοσφαιρική πίεση. Οι τιμές αυτές είναι σημαντικές για τον υπολογισμό της δυναμικής του μοντέλου. Στον Constructor της κλάσης φορτώνονται σχετικές με το περιβάλλον παράμετροι, ενώ στις μεθόδους της κλάσης υπολογίζονται τα παραπάνω μεγέθη. Ο άνεμος υπολογίζεται στο inertial_FLU σύστημα συντεταγμένων, το σταθερό δηλαδή σύστημα του κόσμου. Για να χρησιμοποιηθεί στους υπολογισμούς πρέπει πρώτα να εκφραστεί ως προς το κατάλληλο σύστημα συντεταγμένων, όπως ενός πτερυγίου ή ενός κινητήρα.

7.1.1.4 dynamics.cpp

Η κλάση αυτή είναι υπεύθυνη να καλεί τις σωστές συναρτήσεις υπολογισμού των δυνάμεων για κάθε πτερύγιο και κινητήρα του μοντέλου ανάλογα με τον τύπο του.

Dynamics() - constructor

Εδώ δημιουργείται μια κλάση υπολογισμού δυναμικής για κάθε πτερύγιο και κινητήρα του μοντέλου και προστίθεται σε μια λίστα για ευκολότερη πρόσβαση. Η μία λίστα κλάσεων ονομάζεται listOfAerodynamics και περιέχει τις κλάσεις υπολογισμού της αεροδυναμικής του κάθε πτερυγίου και η άλλη ονομάζεται listOfPropulsion και περιέχει τις κλάσεις υπολογισμού των δυνάμεων κάθε κινητήρα του μοντέλου. Οι κλάσεις αυτές που είναι μέλη λίστας, κατασκευάζονται από την factory, ανάλογα με τον τύπο του πτερυγίου ή κινητήρα.

calcAero()

Εδώ καλείται η συνάρτηση calculationCycle, για κάθε κλάση της λίστας listOfAerodynamics, πυροδοτώντας έναν κύκλο υπολογισμού των αεροδυναμικών δυνάμεων ενός πτερυγίου.

calcProp()

Εδώ καλείται η συνάρτηση calculationCycle, για κάθε κλάσης της λίστας listOfPropulsion, πυροδοτώντας έναν κύκλο υπολογισμού των δυνάμεων ενός κινητήρα.

7.1.1.5 factory.cpp

Σε αυτήν την κλάση κατασκευάζονται άλλες κλάσεις ανάλογα με τις ανάγκες της προσομοίωσης. Οι νέες κλάσεις αφορούν τον υπολογισμό της δυναμικής των πτερυγίων, των κινητήρων και την δημιουργία πολυωνύμων. Οι μέθοδοί της καλούνται κατά την έναρξη της προσομοίωσης, ετοιμάζοντας έτσι ένα πρόσφορο έδαφος για τους μελλοντικούς υπολογισμούς. Για κάθε πτερύγιο και κινητήρα υπάρχουν διαφορετικοί τρόποι να υπολογιστεί η δυναμική τους. Το ίδιο ισχύει και για την δημιουργία πολυωνύμων. Οι τρόποι αυτοί εξαρτώνται από τον χρήστη και δηλώνονται στα αντίστοιχα .yaml αρχεία των πτερυγίων και των κινητήρων, όπως εξηγούνται σε επόμενο κεφάλαιο.

buildAerodynamics(Model *parent, int id)

Στην μέθοδο αυτήν κατασκευάζονται οι κλάσεις υπολογισμού της αεροδυναμικής κάθε πτερυγίου, βάση του τύπου υπολογισμού αεροδυναμικής του πτερυγίου που έχει δηλώσει ο χρήστης στο aerodynamics.yaml αρχείο.

Η μέθοδος επιστρέφει έναν δείκτη σε κλάση τύπου Aerodynamics, η οποία στη συνέχεια προστίθεται στην λίστα listOfAerodynamics μέσω της κλάσης dynamics όπως αναφέρθηκε παραπάνω. Η κλάση παίρνει ως παραμέτρους έναν δείκτη στην βασική κλάση του μοντέλου model, ώστε να έχει πρόσβαση στα μέλη της, και ένα αριθμό id, αναγνωριστικό στοιχείο του πτερυγίου για το οποίο είναι υπεύθυνη.

Για κάθε πτερύγιο προσφέρονται 3 τρόποι υπολογισμού της δυναμικής του μέσω των κλάσεων noAerodynamics, StdLinearAero και polyAero.

Η πρώτη, η noAerodynamics χρησιμοποιείται όταν δεν παράγονται αεροδυναμικές δυνάμεις γύρω από το πτερύγιο καθώς κινείται μέσα στον αέρα. Η περίπτωση αυτή βοηθάει τον χρήστη να απενεργοποιεί ή να καταργεί πτερύγια χωρίς να τα αφαιρεί, εξοικονομώντας χρόνο.

Η δεύτερη, η StdLinearAero, χρησιμοποιείται όταν ο χρήστης έχει στην διάθεσή του τους αεροδυναμικούς συντελεστές του πτερυγίου, συντελεστές που χρησιμοποιούνται στους μαθηματικούς τύπους για να υπολογίσουν δυνάμεις όπως lift, drag, torque δεδομένων μεγεθών όπως angle of attack και airspeed του μοντέλου.

Τέλος η τρίτη κλάση, η polyAero, είναι σχεδόν ίδια με την προηγούμενη. Η μόνη διαφορά είναι πως οι συντελεστές των lift και drag δεν υπολογίζονται μέσω παραμέτρων αλλά εξάγονται μέσω πολυωνυμικών συναρτήσεων που έχει ορίσει ο χρήστης. Επιλέγεται όταν ο

χρήστης δεν έχει στην διάθεσή του τους συντελεστές αεροδυναμικής αλλά πειραματικές τιμές μεταξύ lift-α, drag-α. Η κλάση αυτή δημιουργεί πρώτα τα πολυώνυμα αυτών των σχέσεων, και έπειτα τα χρησιμοποιεί για τον υπολογισμό των παραπάνω συντελεστών.

buildPropulsion(Model *parent, int id)

Στην μέθοδο αυτήν κατασκευάζονται οι κλάσεις υπολογισμού της ώθησης κάθε κινητήρα, βάση του τύπου υπολογισμού ώθησης του κινητήρα που έχει δηλώσει ο χρήστης στο propulsion.yaml αρχείο.

Η μέθοδος επιστρέφει έναν δείκτη σε κλάση τύπου Propulsion , η οποία προστίθεται στην λίστα listOfPropulsion μέσω της κλάσης dynamics όπως αναφέρθηκε παραπάνω. Η κλάση παίρνει ως παραμέτρους έναν δείκτη στην βασική κλάση του μοντέλου model, ώστε να έχει πρόσβαση στα μέλη της, και ένα αριθμό id, αναγνωριστικό στοιχείο του κινητήρα για τον οποίο είναι υπεύθυνη.

Για κάθε κινητήρα προσφέρονται 3 τρόποι υπολογισμού της δυναμικής του μέσω των κλάσεων noEngine, genericEngine και electricEngine.

Η πρώτη, η noEngine, επιλέγεται όταν ο κινητήρας δεν παράγει καθόλου ώθηση. Όπως και στην περίπτωση της noAerodynamics, η κλάση αυτή δίνει την δυνατότητα στον χρήστη να απενεργοποιεί κινητήρες χωρίς να τους αφαιρεί, γλυτώνοντας χρόνο και κόπο.

Η δεύτερη, η genericEngine υπολογίζει τις δυνάμεις ενός γενικού τύπου κινητήρα με προπέλα βασιζόμενη σε σχετικές παραμέτρους.

Τέλος, η τρίτη κλάση υπολογίζει τις δυνάμεις ειδικά για ηλεκτροκινητήρες βασιζόμενη σε σχετικές παραμέτρους.

buildPolynomial(char *baseParam)

Η μέθοδος αυτή δημιουργεί πολυώνυμα 1ου βαθμού, 2ου βαθμού και cubic splines. Τα πολυώνυμα θα αξιοποιηθούν στον υπολογισμό μελλοντικών συντελεστών των πτερυγίων και των κινητήρων. Η μέθοδος παίρνει ως παράμετρο έναν δείκτη τύπου char, με βάση τον οποίο επιλέγει ο χρήστης με ποιόν από τους τρεις τρόπους θέλει να κατασκευάσει τις παραπάνω γραφικές παραστάσεις, δηλαδή με πολυώνυμα 1ου βαθμού, πολυώνυμα 2ου βαθμού ή cubic splines.

7.1.1.6 aerodynamics.cpp

Στο αρχείο αυτό περιγράφεται η κλάση Aerodynamics, μία για κάθε πτερυγίου του μοντέλου. Εδώ υπολογίζονται όλες οι αεροδυναμικές δυνάμεις και ροπές του πτερυγίου που δημιουργούνται από την κίνησή του μέσα στον αέρα.

Aerodynamics(Model *parent, int id) -constructor

Εδώ αποθηκεύονται ο δείκτης στην κλάση model, η οποία είναι πατέρας της συγκεκριμένης κλάσης Aerodynamics, και το αναγνωριστικό id του πτερυγίου.

calculationCycle()

Η μέθοδος αυτή είναι ο συντονιστής της όλης διαδικασίας αφού καλεί με την σειρά τις απαραίτητες ενέργειες που πρέπει να γίνουν σε έναν κύκλο υπολογισμού των αεροδυναμικών δυνάμενων.

getStates()

Φορτώνεται η κατάσταση του πτερυγίου (θέση, προσανατολισμός, γραμμική ταχύτητα, γωνιακή ταχύτητα), μέσω του δείκτη στην κλάση-πατέρα model.

getInputSignals()

Φορτώνονται τα σήματα εισόδου που αφορούν το πτερύγιο με το συγκεκριμένο id, μέσω του δείκτη στην κλάση-πατέρα model.

rotateWind()

Μεταφέρονται τα διανύσματα του ανέμου από το σταθερό σύστημα συντεταγμένων του κόσμου στο σύστημα συντεταγμένων του link του πτερυγίου, απαραίτητο για τους υπολογισμούς.

calculateTriplet()

Υπολογίζονται τα τρία βασικά μεγέθη airspeed, alpha, beta του πτερυγίου, λόγω της υψηλής συχνότητάς τους στους βασικούς υπολογισμούς της δυναμικής.

calcWrench()

Καλούνται οι συναρτήσεις calcForces() και calcTorques() που υπολογίζουν τις αεροδυναμικές δυνάμεις και ροπές του πτερυγίου αντίστοιχα. Οι συναρτήσεις αυτές ανήκουν σε μια από τις κλάσεις noAerodynamics, StdLinearAero, polyAero ανάλογα με τον δηλωμένο τύπο αεροδυναμικής του πτερυγίου.

Οι κλάσεις noAerodynamics, stdLinearAero και polyAero είναι υποκλάσεις της βασικής κλάσης Aerodynamics. Οι κλάσεις αυτές περιέχουν δύο μόνο μεθόδους, τις calcForces και calcTorques, τις οποίες η βασική κλάση Aerodynamics τις έχει δηλωμένες ως virtual. Οπότε ανάλογα με το ποια έχει επιλέξει ο χρήστης, καλούνται οι calcForces και calcTorques της αντίστοιχης υποκλάσης.

7.1.1.7 noAerodynamics.cpp

Υπολογίζει μηδενικές δυνάμεις και ροπές για το πτερύγιο. Χρησιμοποιείται σε περιπτώσεις όπου ο χρήστης θέλει να απενεργοποιήσει τις ιδιότητες ενός πτερυγίου χωρίς να το αφαιρέσει, παρεμβαίνοντας και τροποποιώντας κομμάτια του αρχείου URDF.

7.1.1.8 stdLinearAero.cpp

Υπολογίζει τις δυνάμεις και τις ροπές του πτερυγίου βασιζόμενη σε αεροδυναμικές παραμέτρους. Οι εξισώσεις βρίσκονται στην προηγούμενη ενότητα.

7.1.1.9 polyAero.cpp

Η κλάση polyAero είναι υποκλάση της κλάσης stdLinearAero. Περιέχει τις μεθόδους liftCoeff και dragCoeff τις οποίες η stdLinearAero τις έχει δηλωμένες ως virtual. Οι μέθοδοι αυτές υπολογίζουν την τιμή του lift και drag δεδομένου του alpha μέσω πολυωνύμων και χρησιμοποιούνται όταν ο χρήστες έχει στην διάθεσή του τις πειραματικές τιμές των lift-alpha και drag-alpha. Αρχικά, στον constructor της κλάσης κατασκευάζονται τα πολυώνυμα της

σχέσης lift-alpha και drag-alpha με την βοήθεια της κλάσης factory και έπειτα σε κάθε κύκλο, μέσω των μεθόδων που προαναφέρθηκαν υπολογίζεται η σωστή τιμή του lift και drag δεδομένου του alpha.

7.1.1.10 propulsion.cpp

Στο αρχείο αυτό περιγράφεται η κλάση propulsion, μία για κάθε κινητήρα του μοντέλου. Εδώ υπολογίζονται η ώθηση και η ροπή που παράγει ο κινητήρας και ασκεί στον link του μοντέλου όπου είναι συνδεδεμένος.

Propulsion(Model *parent, int id) -constructor

Εδώ αποθηκεύονται ο δείκτης στην κλάση model, η οποία είναι πατέρας της συγκεκριμένης κλάσης Propulsion, και το αναγνωριστικό id του κινητήρα.

getStates()

Φορτώνεται η κατάσταση του κινητήρα (θέση, προσανατολισμός, γραμμική ταχύτητα, γωνιακή ταχύτητα) μέσω του δείκτη στην κλάση-πατέρα model.

getInputSignals()

Φορτώνεται το σήμα ελέγχου του κινητήρα με το συγκεκριμένο id, μέσω του δείκτη στην κλάση-πατέρα model.

rotateWind()

Μεταφέρονται τα διανύσματα του ανέμου από το σταθερό σύστημα συντεταγμένων του κόσμου στο σύστημα συντεταγμένων του link του κινητήρα, απαραίτητο για τους υπολογισμούς.

calcAirspeed()

Υπολογίζονται το βασικό μέγεθος airspeed του κινητήρα, λόγω της υψηλής συχνότητάς του στους βασικούς υπολογισμούς της δυναμικής.

calcWrench()

Καλούνται οι συναρτήσεις calcThrust(), calcTorque() και calcOmega() που υπολογίζουν την παραγόμενη ώθηση του κινητήρα, την ροπή που ασκεί στο υπόλοιπο σώμα και την γωνιακή ταχύτητα της προπέλας. Οι συναρτήσεις αυτές ανήκουν σε μια από τις κλάσεις noEngine, genericEngine ή electricEngine ανάλογα με τον δηλωμένο τύπο του κινητήρα.

Οι κλάσεις noEngine, genericEngine και electricEngine είναι υποκλάσεις της βασικής κλάσης Propulsion. Οι κλάσεις αυτές περιέχουν τρείς μεθόδους, τις calcThrust, calcTorque και calcOmega οι οποίες υπολογίζουν την ώθηση που παράγεται από τον κινητήρα, την ροπή και την γωνιακή ταχύτητα της προπέλας του και είναι δηλωμένες ως virtual. Οπότε ανάλογα με το ποια έχει επιλέξει ο χρήστης, καλούνται οι calcThrust, calcTorque και calcOmega της αντίστοιχης υποκλάσης.

7.1.1.11 noEngine.cpp

Υπολογίζει μηδενικές τιμές για ώθηση, ροπή και γωνιακή ταχύτητα προπέλας. Χρησιμοποιείται, όπως και στην περίπτωση της noAerodynamics, σε περιπτώσεις που δεν χρησιμοποιείται ο κινητήρας ή είναι ανάγκη να προσομοιωθεί το μοντέλο με απουσία του κινητήρα.

7.1.1.12 genericEngine.cpp

Υπολογίζονται η ώθηση, ροπή και γωνιακή ταχύτητα της προπέλας ενός οποιουδήποτε κινητήρα με προπέλα. Οι εξισώσεις υπάρχουν στην προηγούμενη ενότητα.

7.1.1.13 electricEngine.cpp

Υπολογίζονται η ώθηση, ροπή και γωνιακή ταχύτητα της προπέλας ενός ηλεκτροκινητήρα με προπέλα. Οι εξισώσεις υπάρχουν στην προηγούμενη ενότητα.

7.1.2 Controller Node

7.1.2.1 quadcopter_contr_node.cpp

Ο αλγόριθμος ελέγχου το μοντέλου τρέχει στο *controller_n*, το οποίο βρίσκεται σε ένα από τα αρχεία του φακέλου **/controllers**. Ο προσομοιωτής περιέχει 3 αρχεία default controller για τα μοντέλα που διαθέτει. Παρακάτω θα ασχοληθούμε με τον controller του quadcopter ως παράδειγμα. Όλα τα αρχεία ακολουθούν ακριβώς την ίδια δομή, η οποία προτείνεται και για την συγγραφή νέων μελλοντικών αρχείων controller.

Σε κάθε κύκλο συλλέγονται όσα δεδομένα χρησιμοποιούνται από τον controller για τον υπολογισμό των control inputs. Τα δεδομένα αυτά είναι ο πίνακας καναλιών των σημάτων που στέλνει ο χρήστης μέσω του joystick (*/channels*) και η κατάσταση κάθε βασικού link του μοντέλου (*/model_states*), δηλαδή η θέση, ο προσανατολισμός, η γραμμική και η γωνιακή ταχύτητα. Αφού γίνει η συλλογή των πιο πρόσφατων παραπάνω απαραίτητων δεδομένων, ο controller υπολογίζει όταν του ζητηθεί τα νέα control inputs και τα επιστρέφει στο υπόλοιπο σύστημα ανανεώνοντας κάποιες θέσεις του πίνακα καναλιών με τις νέες τιμές. Η ζήτηση των control inputs και η αποστολή τους γίνεται μέσω service.

Controller() - constructor

Δήλωση των subscriber και του service που θα χρησιμοποιηθούν και φόρτωση όλων των χρήσιμων παραμέτρων από τον Parameter server. Κλήση της **initControllerVariables()** για την αρχικοποίηση των μεταβλητών που χρησιμοποιούνται.

initControlVariables()

Αρχικοποίηση των μεταβλητών που χρησιμοποιούνται από τις υπόλοιπες μεθόδους.

Για την περίπτωση των multirotor, δηλώνεται και ο πίνακας με τον οποίο μετασχηματίζονται οι είσοδοι των κινητήρων σε σήματα ελέγχου του multirotor, δηλαδή σήμα throttle, roll, pitch, yaw. Από αυτόν εξάγεται ο αντίστροφός του, με βάση τον οποίο ο controller μετατρέπει τα σήματα ελέγχου που παράχθηκαν από τον αλγόριθμο ελέγχου σε σήματα εισόδου για τους κινητήρες.

Κάτι τέτοιο δεν ισχύει για περιπτώσεις αεροπλάνων, καθώς όλοι οι κινητήρες παίρνουν συνήθως το ίδιο σήμα εισόδου, που είναι το σήμα throttle.

chan2signal(last_letter_2_msgs::joystick_input msg)

Callback που ακούει στο topic */channels*, στο οποίο δημοσιεύονται τα κανάλια σημάτων του χρήστη. Αρχικά αποθηκεύεται ο πίνακας καναλιών με τα σήματα που στέλνει ο χρήστης. Έπειτα εξάγονται από αυτόν τα 4 βασικά σήματα κατεύθυνσης roll_input, pitch_input, yaw_input, thrust_input. Στο τέλος, καλείται η συνάρτηση **channelFunctions()**, υπεύθυνη για την διαχείριση των υπόλοιπων καναλιών. Στους default controller η συνάρτηση αυτήν δεν χρησιμοποιείται. Προστέθηκε για τυχόν μελλοντικές ανάγκες.

channelFunctions()

Μέθοδος διαχείρισης των υπόλοιπων καναλιών.

storeStates(const last_letter_2_msgs::model_states msg)

Callback που ακούει στο topic */model_states*, στο οποίο δημοσιεύεται η κατάσταση των σημαντικών link του μοντέλου και ενημερώνεται από το Gazebo. Η μέθοδος αυτή αποθηκεύει την κατάσταση των link αυτών, για μελλοντική χρήση.

returnControlInputs(last_letter_2_msgs::get_control_inputs_srv::Request &req,
last_letter_2_msgs::get_control_inputs_srv::Response &res)

Αυτή η μέθοδος είναι ένας εξυπηρετητής Service (Service Server). Το αίτημα καλείται από την κλάση Model, την βασική δηλαδή κλάση του μοντέλου που αποτελεί και τον συντονιστή. Η κλάση αυτή ζητάει από το *controller_n* τα καινούρια control inputs του νέου βήματος το οποίο βρίσκεται σε εξέλιξη. Μόλις έρθει το αίτημα, η μέθοδος διασφαλίζει ότι έχουν κληθεί οι callbacks που αποθηκεύουν τα πιο πρόσφατα δεδομένα πριν προχωρήσει στους υπολογισμούς. Σε περίπτωση που δεν έχει συμβεί αυτό, καλεί ένα **spinOnce()**, ώστε να τρέξουν οι callback και να ενημερωθούν τα απαραίτητα δεδομένα. Έπειτα υπολογίζονται τα νέα control inputs του μοντέλου και επιστρέφονται. Τα control inputs, είτε αφήνονται όπως ακριβώς τα έστειλε ο χρήστης (περίπτωση plane_contr_node, χρήσιμο για χειροκίνητο έλεγχο αεροπλάνου μέσω του joystick) είτε χρησιμοποιούνται για τον υπολογισμό νέων σημάτων ελέγχου μέσω του αλγορίθμου ελέγχου. Ο αλγόριθμος ελέγχου βρίσκεται στην μέθοδο controlLaw().

Όταν είναι έτοιμα τα νέα σήματα ελέγχου, τοποθετούνται σε κάποιες θέσεις του πίνακα καναλιών. Έπειτα επιστρέφεται όλος ο πίνακας στο *core_n*.

Για την περίπτωση multirotor, από τα νέα σήματα ελέγχου εξάγονται τα σήματα εισόδου κάθε κινητήρα με την βοήθεια του αντιστρόφου του πίνακα του multirotor, τα οποία και φορτώνονται στον πίνακα καναλιών.

controlLaw()

Για τα default αεροπλάνα του προσομοιωτή δεν χρησιμοποιείται κάποιος αλγόριθμος ελέγχου. Τα σήματα που στέλνει ο χρήστης μεταφέρονται απευθείας στο μοντέλο.

Για τα default multirotor του προσομοιωτή έχει δημιουργηθεί ένας PD, ο οποίος ελέγχει το roll, το pitch, την κατεύθυνση του yaw και το ύψος του multirotor.

7.1.3 Joy2chan node

7.1.3.1 joy2chan_node.cpp

Node υπεύθυνο για το γέμισμα του πίνακα καναλιών με τα σήματα που στέλνει ο χρήστης μέσω του joystick. Διαβάσει τις τιμές από το topic */joy*.

main()

Στην main συνάρτηση του node, ορίζεται ένας subscriber στο topic */joy* με callback την **buildChannels(),** ορίζεται ένας publisher στο topic */channels* και φορτώνονται χρήσιμοι παράμετροι από τον Parameter Server. Έπειτα ορίζεται ένας spinner για την εξυπηρέτηση των μηνυμάτων που φτάνουν στο topic */joy* από το joystick_n.

buildChannels(sensor_msgs::Joy joyMsg)

Callback που ακούει στο topic */joy*, στο οποίο δημοσιεύεται η κατάσταση των αξόνων και κουμπιών του joystick. Τα σήματα που έρχονται από το joystick είναι χωρισμένα σε δύο πίνακες, ένας με τους άξονες του joystick και ένας με τα κουμπιά του. Η μέθοδος αυτή τοποθετεί τις τιμές των δύο πινάκων σε κοινό πίνακα, τον πίνακα καναλιών. Οι θέσεις του πίνακα καναλιών φορτώνονται από τις θέσεις των δύο πινάκων, όπως έχει δηλώσει ο χρήστης στο **HID.yaml** αρχείο.

7.1.4 Chan2srv node

7.1.4.1 chan2srv_node.cpp

Το node αυτό είναι υπεύθυνο για την κλήση δυο έτοιμων (default) services του Gazebo. Αυτά είναι το spawn_model και delete_model, και επιτρέπουν στον χρήστη να εισάγει και να διαγράφει ένα κουτάκι κάτω από το μοντέλο στον κόσμο της προσομοίωσης εν ώρα πτήσης. Το αίτημα στέλνεται από την μεριά του ROS οπότε επιταχύνεται και διευκολύνεται κάπως η επικοινωνία με το Gazebo, για δευτερεύον θέματα. Η εισαγωγή αντικειμένων μπορεί να τροποποιηθεί σύμφωνα με τις ανάγκες του χρήστη. Επίσης μπορούν να χρησιμοποιηθούν και άλλες κλήσεις στα υπόλοιπα default service του Gazebo.

7.2 Πακέτο last_letter_2_gazebo_plugins

Αυτό το πακέτο περιέχει τα κομμάτια κώδικα των gazebo plugins[23] που χρησιμοποιούνται, δηλαδή του model και world plugin. Με βάση αυτά τα plugins, ο χρήστης μπορεί και επικοινωνεί με τον κόσμο της προσομοίωσης και το μοντέλο.

Στον προσομοιωτή έχουν αναπτυχθεί δύο plugins. Ένα model plugin, που φέρνει τον χρήστη σε άμεση επαφή με το μοντέλο που προσομοιώνεται και ένα world plugin, που φέρνει τον χρήστη σε άμεση επαφή με τον κόσμο της προσομοίωσης. Πέρα αυτών όμως, γίνεται χρήση και τριών έτοιμων sensor plugin, στην περίπτωση του μοντέλου plane_1wing_sensors, ένα για κάθε αισθητήρα που κουβαλάει.

Αξίζει να σημειωθεί, πως μπορούν να χρησιμοποιηθούν και άλλα plugins ή να τροποποιηθούν τα υπάρχοντα ανάλογα με τις ανάγκες των μελλοντικών χρηστών.

Στη συνέχεια εξηγείται η λειτουργία του κάθε plugin, όσον αφορά την προσομοίωση.

7.2.1 Model Plugin

7.2.1.1 model_plugin.cpp

Το model plugin επιτρέπει στον χρήστη να ελέγχει το μοντέλο, αλλά και την προσομοίωση έμμεσα.

Πιο συγκεκριμένα το model plugin επιτελεί τις εξής λειτουργίες. Ασκεί σε κάθε κύκλο τις δυνάμεις και τις ροπές πάνω στα αντίστοιχα link του μοντέλου, πριν η μηχανή φυσικής ξεκινήσει τους υπολογισμούς του επόμενου βήματος προσομοίωσης. Ανανεώνει το δέντρο tf του μοντέλου, που κρατάει όλες τις σχέσεις μετασχηματισμών μεταξύ των διαφόρων συστημάτων συντεταγμένων του. Τέλος, γράφει στο topic */model_states* τις νέες τιμές που αφορούν την κατάσταση του μοντέλου και υπολογίζονται από την μηχανή φυσικής του Gazebo. Παράλληλα, είναι υπεύθυνο και για τον συγχρονισμό[24] του με το ROS.

Load (physics::ModelPtr _model, sdf::ElementPtr _sdf)

Η Load είναι μια συνάρτηση που καλείται με το που φτιάχνεται ένα plugin και δεν πρέπει να μπλοκάρεται. Αυτό σημαίνει πως στην Load πρέπει να στηθούν όλες οι γέφυρες επικοινωνίας του plugin με το υπόλοιπο σύστημα. Ως παράμετροι της Load δίνονται ένας pointer στο μοντέλο για το οποίο καλείται, και ένας pointer στο αρχείο sdf του μοντέλου. Αρχικά αποθηκεύεται ο pointer του μοντέλου και γεννιέται ένα βοηθητικό thread. Δημιουργείται η callback συνάρτηση **BeforeUpdate()**, η οποία ακούει σε σήμα που στέλνεται πριν η μηχανή φυσικής ξεκινήσει τους υπολογισμούς του νέου βήματος προσομοίωσης. Δημιουργείται η callback συνάρτηση **OnUpdate()**, η οποία ακούει σε σήμα που στέλνεται στο τέλος ενός βήματος του κόσμου προσομοίωσης. Δηλώνεται η **applyWrenchesOnModel()** ως εξυπηρετητής του service */apply_model_wrenches_srv*. Δημιουργείται ο subscriber που ακούει στο topic /*channes* και ο publisher στο topic /*model_states*. Έπειτα φορτώνονται κάποιες παράμετροι του μοντέλου από τον Parameter Server και καλείται η **modelStateInit()**.

[23] Βλ. Παράρτημα

[24] Βλ. Κεφάλαιο 8

modelStateInit()

Εδώ ορίζονται οι αρχική κατάσταση του μοντέλου όπως έχει δηλωθεί από τον χρήστη στο αρχείο init.yaml, δηλαδή η αρχική θέση και προσανατολισμός, η αρχική γραμμική και γωνιακή ταχύτητα. Η θέση και ο προσανατολισμός είναι εκφρασμένα ως προς το σύστημα συντεταγμένων του κόσμου. Οι ταχύτητες όμως είναι εκφρασμένες ως προς το σύστημα συντεταμένων του μοντέλου. Οπότε τα διανύσματα ταχυτήτων πρέπει να στραφούν ως προς το σταθερό σύστημα του κόσμου πριν εφαρμοστούν. Αυτό επιτυγχάνεται μέσω της βιβλιοθήκης KDL.

QueueThread()

Αυτό είναι ένα βοηθητικό thread, για την διαχείριση των callback του model plugin. Κάθε τόσο ελέγχει εάν υπάρχουν νέα αιτήματα και καλεί την αντίστοιχη callback. Έπειτα από δοκιμές παρατηρήθηκε πώς η πιο αποδοτική συχνότητα για έλεγχο νέων μηνυμάτων είναι 2.2 φορές μεγαλύτερη από την συχνότητα της προσομοίωσης. Για άλλες συχνότητες η προσομοίωση επιβράδυνε.

applyWrenchOnModel

(last_letter_2_msgs::apply_model_wrenches_srv::Request &req,
last_letter_2_msgs::apply_model_wrenches_srv::Response &res)

Αυτή η μέθοδος είναι ο εξυπηρετητής του service call του *core_n*, με το οποίο το ROS στέλνει τις υπολογισμένες δυνάμεις και ροπές στο Gazebo. Εδώ αρχικά ασκούνται όλες οι δυνάμεις και ροπές όλων των πτερυγίων και κινητήρων του μοντέλου. Οι δυνάμεις αυτές ασκούνται στο πλαίσιο συντεταγμένων κάθε link. Πρέπει να δοθεί προσοχή στο τρόπου που στήνεται το μοντέλο και στο πώς ενώνονται τα links, καθώς το Gazebo ορίζει σαν πλαίσιο συντεταγμένων ενός link, το πλαίσιο συντεταγμένων του joint στο οποίο είναι παιδί του. Στην συνέχεια ρυθμίζονται οι γωνίες των joints camera_joint και laser_joint (ισχύει μόνο για το μοντέλο plane_1wing_sensors). Τέλος η μεταβλητή wrenches_applied δηλώνεται true. Αυτό επιτρέπει στο Gazebo να ξεκολλήσει από την **BeforeUpdate()** και να συνεχίσει την λειτουργία του.

manageChan (const last_letter_2_msgs::joystick_input::ConstPtr &channels)

Η μέθοδος αυτή ακούει στο topic */channels* και ο ρόλος της είναι να αντιστοιχίζονται λειτουργίες σε κανάλια σημάτων. Αυτήν την στιγμή εξυπηρετούνται δύο μόνο σήματα μέσω του *camera_angle_chan* και *laser_angle_chan* καναλιού που ακούν σε δύο άξονες του Joystick. Και οι δύο λειτουργίες αλλάζουν τις τιμές των γωνιών του camera_joint και του laser_joint, στρέφοντας με αυτό το τρόπο τα link των αντίστοιχων αισθητήρων.

BeforeUpdate()

Callback που ακούει σε σήμα που στέλνεται πριν την έναρξη της μηχανής φυσικής του βήματος. Εδώ κολλάει το Gazebo περιμένοντας την μεταβλητή wrenches_applied να πάρει την τιμή true. Αυτό βοηθάει στον συγχρονισμό[25] ROS-Gazebo. Το Gazebo αρχικά τρέχει 50 κύκλους χωρίς να σταματήσει. Αυτό είναι αναγκαίο προκειμένου να στηθούν όλοι οι δίαυλοι επικοινωνίας του με το ROS.

OnUpdate()

Callback που ακούει σε σήμα που στέλνεται στο τέλος κάθε ανανέωσης του κόσμου, δηλαδή στο τέλος κάθε βήματος προσομοίωσης. Εδώ δημοσιεύονται οι σχέσεις μεταξύ των πλαισίων συντεταγμένων στο topic */tf*, χρήσιμες στην χρήση του Rviz. Στην πλειοψηφία των

[25] Βλ. Κεφάλαιο 8

περιπτώσεων, οι χρήσιμοι μετασχηματισμοί, δηλαδή αυτοί που αφορούν τα βασικά links δεν αλλάζουν. Όμως για να καλυφθούν και κάποιες λίγες περιπτώσεις εξαίρεσης επιλέχθηκε να ανανεώνονται σε κάθε βήμα. Επίσης δημοσιεύονται στο topic */model_states*, η κατάσταση του μοντέλου μετά το νέο βήμα προσομοίωσης. Η δημοσίευση σε αυτό το topic, όπως έχει αναφερθεί και σε άλλα κεφάλαια, αποτελεί το σήμα για να ξεκινήσει το ROS τους υπολογισμούς του επόμενου βήματος.

7.2.2 World Plugin

7.2.2.1 world_plugin.cpp

Η χρήση του world plugin είναι πιο απλή. Το world plugin καλείται στην αρχή της προσομοίωσης και το μόνο που κάνει είναι να δημοσιεύσει στο topic */tf_static*, το δέντρο μετασχηματισμών μεταξύ των σταθερών μόνο σχέσεων των συστημάτων συντεταγμένων, δηλαδή μετασχηματισμούς που θα μείνουν απαράλλαχτοι μέχρι το τέλος της προσομοίωσης. Οι πληροφορίες παραμένουν αποθηκευμένες μέσα στο topic αυτό, και δεν χάνονται όσες φορές και να ζητηθούν από κάποιον subscriber. Έτσι το ROS μπορεί και διαβάζει σε κάθε κύκλο τις τιμές αυτού του topic, που δημοσιεύθηκαν μια φορά μόνο, στην αρχή της προσομοίωσης.

7.3 Πακέτο last_letter_2_msgs

Ο προσομοιωτής τρέχει σε πολλά ξεχωριστά nodes. Αυτά επικοινωνούν είτε ασύγχρονα μέσω των topics είτε σύγχρονα μέσω των services. Τα χρήσιμα αυτά μηνύματα που στέλνονται μεταξύ των nodes βρίσκονται σε αυτό το πακέτο.

Για την ασύγχρονη επικοινωνία των nodes χρησιμοποιούνται τα topics. Τα πακέτα μηνυμάτων που στέλνονται μέσω των topics είναι τα εξής:

- air_data
- channels
- link_states
- model_states
- aero_wrenches
- prop_wrenches

Για την σύγχρονη επικοινωνία των nodes χρησιμοποιούνται τα services. Τα services αποτελούνται από ένα request και ένα response. Οπότε για κάθε service call κινούνται δύο μηνύματα. ένα προς τον server και ένα προς τον client. Τα πακέτα services του προσομοιωτή είναι τα εξής:

- apply_model_wrenches_srv
- get_control_inputs_srv

7.3.1 air_data.msg

Το μήνυμα αυτό περιέχει τα δεδομένα του περιβάλλοντος

wind_x: x συνιστώσα διανύσματος ανέμου (εκφρασμένη στο παγκόσμιο σύστημα συντεταγμένων σε $\frac{m}{s}$)

wind_y: y συνιστώσα διανύσματος ανέμου

wind_z: z συνιστώσα διανύσματος ανέμου

density: πυκνότητα αέρα ($\frac{kg}{m^3}$)

pressure: ατμοσφαιρική πίεση ($mBar$)

temperature: θερμοκρασία ($Kelvin$)

7.3.2 channels.msg

Το μήνυμα αυτό περιέχει τα σήματα των αξόνων και των κουμπιών του joystick συγκεντρωμένα σε έναν πίνακα 20 θέσεων

value[20]: πίνακας σημάτων joystick 20 θέσεων

7.3.3 link_states.msg

Το μήνυμα αυτό περιέχει τις 12 τιμές που περιγράφουν την κατάσταση ενός link.

x: x συνιστώσα θέσης
y: y συνιστώσα θέσης
z: z συνιστώσα θέσης
phi: γωνία phi
theta: γωνία theta
psi: γωνία psi
u: γραμμική ταχύτητα στον άξονα x
v: γραμμική ταχύτητα στον άξονα y
w: γραμμική ταχύτητα στον άξονα z
p: γωνιακή ταχύτητα στον άξονα x
q: γωνιακή ταχύτητα στον άξονα y
r: γωνιακή ταχύτητα στον άξονα z

7.3.4 model_states.msg

Το μήνυμα αυτό περιέχει την κατάσταση κάθε link του μοντέλου, δηλαδή του βασικού link, των πτερυγίων και των κινητήρων. Τα δεδομένα είναι τοποθετημένα σε πίνακες στοιχείων της μορφής link_states msgs.

base_link_states: Κατάσταση βασικού link
airfoil_states[10]: Πίνακας κατάστασης πτερυγίων
motor_states[10]: Πίνακας κατάστασης κινητήρων

Η διάσταση των πινάκων είναι 10 στοιχεία. Αυτό συνεπάγεται πως ο προσομοιωτής μπορεί να διαχειριστεί μέχρι 10 πτερύγια και 10 κινητήρες σε ένα μοντέλο.

7.3.5 aero_wrenches.msg

Το μήνυμα αυτό περιέχει τις δυνάμεις και ροπές που ασκούνται σε ένα πτερύγιο, δηλαδή

drag: δύναμη πάνω στον άξονα x του πτερυγίου
fy: δύναμη πάνω στον άξονα y του πτερυγίου
lift: δύναμη πάνω στον άξονα z του πτερυγίου
l: ροπή γύρω από τον άξονα x του πτερυγίου
m: ροπή γύρω από τον άξονα y του πτερυγίου
n: ροπή γύρω από τον άξονα z του πτερυγίου

7.3.6 prop_wrenches.msg

Το μήνυμα αυτό περιέχει την δύναμη και ροπή που παράγει ο κινητήρας και την γωνιακή ταχύτητα του άξονα ενός κινητήρα:

thrust: δύναμη ώθησης κινητήρα
torque: ροπή κινητήρα
omega: γωνιακή ταχύτητα άξονα κινητήρα

7.3.7 apply_model_wrenches_srv.srv

Με αυτό το service στέλνονται οι υπολογισμένες δυνάμεις και ροπές στο Gazebo, προκειμένου να εφαρμοστούν στο μοντέλο

Request (to Gazebo-model_plugin)
airfoil_forces[3]: Πίνακας των τριών συνιστωσών δύναμης για κάθε πτερύγιο
airfoil_torques[3]: Πίνακας των τριών συνιστωσών ροπής για κάθε πτερύγιο
motor_thrust[6]: Πίνακας των δυνάμεων ώθησης για κάθε κινητήρα
motor_torque[6]: Πίνακας των ροπής ώθησης για κάθε κινητήρα
motor_omega [6]: Πίνακας της γωνιακής ταχύτητας του άξονα κάθε κινητήρα

Response
success: Λογική μεταβλητή επιτυχίας ή όχι του αιτήματος

7.3.8 get_control_inputs_srv.srv

Με αυτό το service call το *core_n* ζητάει από τον controller τον νέο πίνακα καναλιών όπως έχει δημιουργηθεί μετά την εφαρμογή του νόμου ελέγχου.

Request (to controller_n)
Κενό μήνυμα

Response
Channels[20]: Ο ανανεωμένος με τα σήματα ελέγχου των κινητήρων πίνακας καναλιών

7.4 Πακέτο last_letter_2_libs

Το τελευταίο πακέτο του προσομοιωτή περιέχει κάποιες χρήσιμες βοηθητικές συναρτήσεις. Όλες βρίσκονται σε ένα μόνο αρχείο, το **math_lib.cpp** και είναι οι εξής:

- Κλάση Polynomial1D, υπεύθυνη για τον υπολογισμό τιμής πολυωνύμου μιας μεταβλητής
- Κλάση Polynomial2D, υπεύθυνη για τον υπολογισμό τιμής πολυωνύμου δύο μεταβλητών
- Κλάση Spline, υπεύθυνη για τον υπολογισμό τιμής μιας spline
- Συνάρτηση FLUtoFRD, με την οποία στρέφονται τα δεδομένα από το FLU σύστημα συντεταγμένων που διαχειρίζεται το Gazebo στο FRD σύστημα συντεταγμένων που χρησιμοποιείται στους υπολογισμούς.
- Συνάρτηση FRDtoFLU, με την οποία στρέφονται τα δεδομένα από το FRD σύστημα συντεταγμένων στο FLU.

8

Συγχρονισμός ROS με Gazebo

Πρωτίστης σημασίας του προσομοιωτή είναι να υλοποιεί όλα τα βήματα της προσομοίωσης αξιόπιστα. Αυτό σημαίνει πως σε κάθε κύκλο πρέπει να υπάρχουν πάντα διαθέσιμα τα πιο πρόσφατα δεδομένα και να ακολουθούνται ακριβώς τα ίδια βήματα για τους υπολογισμούς, χωρίς τυχόν παραλήψεις υπολογισμών ή χρήση δεδομένων προηγούμενων βημάτων λόγω καθυστερήσεων. Με λίγα λόγια χρειάζεται άριστος συγχρονισμός στο σύστημα.

Ο συγκεκριμένος προσομοιωτής αποτελείται από node που τρέχουν ασύγχρονα, αφού ορισμένα ελέγχονται από το ROS και άλλα από το Gazebo. Η μόνη επικοινωνία μεταξύ των node αυτών είναι τα μηνύματα που στέλνονται μέσω topics και services. Είναι όμως αναγκαίο αυτά τα node να τρέχουν με μια σειρά, καθώς και να περιμένουν κάποιες φορές άλλα node να τελειώσουν ώστε να γίνονται αξιόπιστα όλα τα βήματα ενός κύκλου προσομοίωσης.

Το Gazebo εξ ορισμού αποτελεί ένα αυτόνομο σύστημα και τρέχει τους υπολογισμούς του τελείως ανεξάρτητα από το ROS. Το Gazebo ακολουθεί ένα ρολόι και κάθε φορά που έχει περάσει ο χρόνος ενός βήματος προσομοίωσης καλεί την μηχανή φυσικής για να τρέξει το επόμενο. Πριν κληθεί όμως η μηχανή φυσικής πρέπει το ROS να έχει προλάβει να στείλει τα νέα δεδομένα. Είναι πολύ πιθανό στο ROS να συμβούν καθυστερήσεις. Αυτό μπορεί να οφείλεται στο πολύ μικρό χρονικό βήμα προσομοίωσης (στις μεγάλες συχνότητες), σε διάφορες καθυστερήσεις μεταξύ της ανταλλαγής μηνυμάτων των nodes, και φυσικά σε τυχόν χαμηλές επιδόσεις του συστήματος που τρέχει η προσομοίωση. Με την τεχνική του Step Lock που χρησιμοποιείται, το Gazebo ακούει φυσικά στο ρολόι αλλά περιμένει και το ROS να στείλει τα δεδομένα του προτού καλέσει την μηχανή φυσικής.

Η λογική με την οποία επιτυγχάνεται συγχρονισμός είναι η εξής. Τόσο το ROS όσο και το Gazebo, στέλνουν το ένα στο άλλο και λαμβάνουν ένα σήμα έναρξης. Το Gazebo στέλνει αυτό το σήμα έναρξης στο ROS, δημοσιεύοντας στο topic */model_states*. Το ROS, περιμένει να γεμίσει το topic */model_states* με τα νέα δεδομένα ώστε να ξεκινήσει τον δικό του κύκλο. Το topic αυτό ανανεώνεται μόνο από το Gazebo, συγκεκριμένα από το model_plugin του Gazebo. Οπότε ενημερώνοντας αυτό το topic, το Gazebo επιτρέπει στο ROS να ξεκινήσει τα βήματά του, ενώ περιμένει ξανά την δική του σειρά. Μόλις το ROS τελειώσει με τους υπολογισμούς, στέλνει πίσω στο Gazebo μέσω service τα αποτελέσματά του περιμένοντας

ξανά να γεμίσει το topic */model_states*. Αυτό το service call αποτελεί και το σύνθημα για το Gazebo να συνεχίσει την λειτουργία του. Έτσι λοιπόν πετυχαίνεται ο συγχρονισμός μεταξύ του ROS και Gazebo.

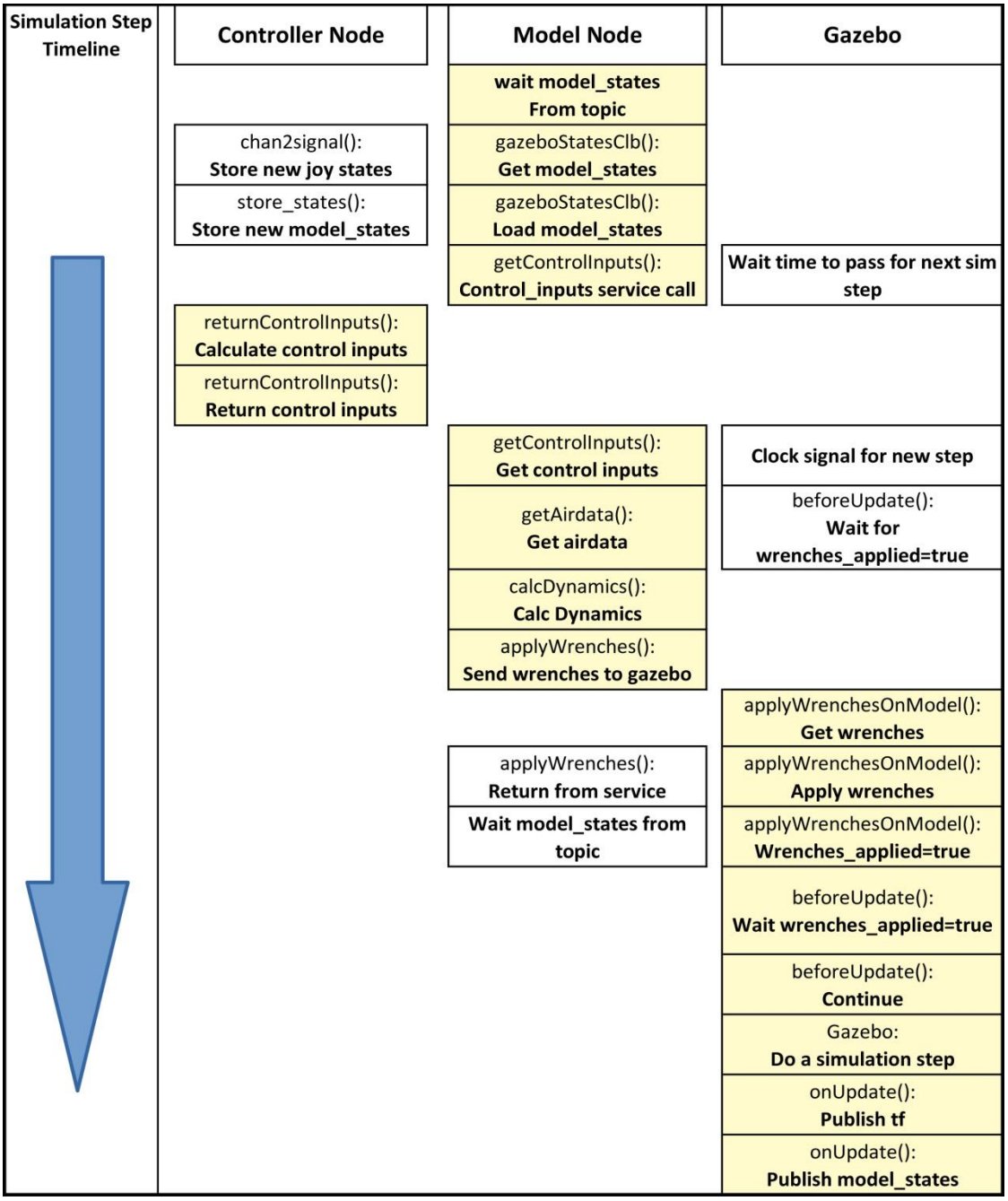
Στην μεριά του ROS, εξαιτίας του γεγονότος ότι από την στιγμή που λάβει την νέα κατάσταση του μοντέλου μέσω του topic */model_states* όλα τα βήματα γίνονται διαδοχικά, δεν χρειάζεται κάποια άλλη αλλαγή ή προσαρμογή στον κώδικά. Από την άλλη μεριά, στο Gazebo δεν συμβαίνει το ίδιο. Στο gazebo τρέχουν πολλά κομμάτια κώδικα ασύγχρονα και όχι υπό τον έλεγχο του χρήστη. Για να τηρηθεί η σειρά προτεραιότητας μεταξύ των βημάτων γίνεται χρήση μιας condition_variable στον κώδικα του model_plugin, της wrenches_applied.

```
Bool applyWrenchOnModel(....)
{
    std::lock_guard<std::mutex> lk(m);
    ...
    //apply  wrenches to each airfoil and motor
    //Handle sensors
    ...
    //unlock gazebo step
    wrenches_applied = true;
    cv.notify_one();
    ....
}

Void BeforeUpdate()
{
    std::unique_lock<std::mutex> lk(m);
    //wait until wrenches are ready
    cv.wait(lk, [] { return wrenches_applied == true; });
    wrenches_applied=false;
 }

 void OnUpdate()
 {
    .....
    //Read model states of body, airfoils and motors
    //Publish model_states
    this->states_pub.publish(model_states);
 }
```

Στο model_plugin του Gazebo, υπάρχουν 2 callbacks συνδεδεμένες με γεγονότα του Gazebo, οι **BeforeUpdate()** και **OnUpdate()**. Η πρώτη ακούει σε ένα σήμα που στέλνεται πριν η μηχανή φυσικής ξεκινήσει να υπολογίζει την ένα κατάσταση του μοντέλου και η δεύτερη σε ένα σήμα που στέλνεται στο τέλος ενός βήματος προσομοίωσης. Παράλληλα υπάρχει και ένας service server, που εξυπηρετείται στην μέθοδο **applyWrenchesOnModel()** και καλείται όταν το ROS κάνει το αντίστοιχο service call. Άρα, ενώ αυτές οι 3 callbacks πρέπει να τρέχουν με συγκεκριμένη σειρά σε κάθε κύκλο, τα σήματα που τις ενεργοποιούν δεν στέλνονται πάντα με την σωστή σειρά. Στο Gazebo, ο χρήστης δηλώνει την συχνότητα προσομοίωσης που επιθυμεί. Αυτό με την σειρά του ακολουθεί ένα ρολόι και κάθε φορά που περνάει ο χρόνος που αντιστοιχεί σε έναν κύκλο προσομοίωσης στέλνει ένα σήμα που καλεί την **BeforeUpdate()**. Υπάρχει όμως το ROS να μην έχει προλάβει να στείλει τις δυνάμεις για εφαρμογή οπότε το Gazebo πρέπει να περιμένει πρώτα τις δυνάμεις αυτές. Αυτό

επιτυγχάνεται μέσω της condition_variable wrenches_applied. Η **BeforeUpdate()** περιμένει αυτήν την μεταβλητή να πάρει την τιμή true για να συνεχίσει. Η τιμή της μεταβλητής αυτής αλλάζει σε true μόνο στο τέλος της **applyWrenchesOnModel()**, όταν δηλαδή έχουν έρθει και εφαρμοστεί οι δυνάμεις. Τότε η **BeforeUpdate()** συνεχίζει αφού δώσει πάλι την τιμή false στην condition_variable wrenches_applied ώστε να είναι έτοιμη για τον επόμενο κύκλο. Μόλις η **BeforeUpdate()** τελειώσει γίνονται οι υπολογισμοί του νέου βήματος από την μηχανή φυσικής και καλείται η **OnUpdate()**. Στο τέλος αυτής ανανεώνεται το topic */model_states*, και το ROS ξεκινάει να κάνει τους υπολογισμούς του. Παράλληλα το Gazebo περιμένει να περάσει ο κατάλληλος χρόνος και κολλάει στην **BeforeUpdate()** όπως και πριν. Αυτό συνεχίζει να συμβαίνει σε κάθε κύκλο.

Παρακάτω φαίνεται ένα χρονικά διάγραμμα των παραπάνω βημάτων:



Εικόνα 39. Χρονοδιάγραμμα με την σειρά που τρέχουν οι μέθοδοι των κλάσεων. Στον πίνακα παρουσιάζονται ποιοτικά τα βήματα ενός κύκλου προσομοίωσης που συμβαίνουν στα 3 βασικά nodes. Τα υποσκιασμένα κουτάκια είναι αυτά που πρέπει να τρέχουν με την σειρά. Τα υπόλοιπα τρέχουν παράλληλα.

Γ' Μέρος

Παράρτημα

Παράρτημα Α

Ρομποτικοί Προσομοιωτές

Στην έρευνα των ρομπότ υπάρχουν διαθέσιμα αρκετά ελεύθερα λογισμικά προσομοίωσης. Αυτά αποτελούν κύρια εργαλεία για τους ερευνητές ώστε να ελέγξουν τα αποτελέσματα της έρευνάς τους πριν προχωρήσουν στο πείραμα. Οι προσομοιωτές ρομπότ είναι έναν ασφαλές περιβάλλον για τον έλεγχο νέων μοντέλων και controller. Συνδυάζουν γραφικά, μηχανή φυσικής, μοντέλα, controllers, βοηθητικές βιβλιοθήκες και plugins ώστε ο χρήστης να μπορεί να προσομοιώσει οποιονδήποτε τύπο μοντέλου σε ότι περιβάλλον επιθυμεί. Επειδή οι ανάγκες και οι περιπτώσεις είναι αρκετές υπάρχουν προσομοιωτές γενικής χρήσης αλλά και κάποιοι πιο ειδικοί, για συγκεκριμένες μόνο εφαρμογές όπως οι προσομοιωτές πτήσης που αφορούν ιπτάμενα ρομπότ.

Α.1 Γενικοί προσομοιωτές ρομπότ

Ορισμένοι από τους πιο ευρέως διαδεδομένους και χρησιμοποιούμενους προσομοιωτές Ρομπότ Γενικής Χρήσης είναι οι εξής:

- Gazebo
- Morse
- Webots
- V_REP

A.1.1 Gazebo

Ο Gazebo είναι ένας 3D δυναμικός προσομοιωτής με την ικανότητα να προσομοιώνει με ακρίβεια και αποδοτικότητα ρομπότ, σε πολύπλοκα περιβάλλοντα τόσο εσωτερικού όσο και εξωτερικού χώρου. Όπως οι μηχανές παιχνιδιών, ο Gazebo προσφέρει προσομοίωση της φυσικής σε πολύ υψηλό βαθμό πιστότητας, μια σειρά αισθητήρων και διεπαφών τόσο για τους χρήστες όσο για άλλα προγράμματα. Μια τυπική χρήση του Gazebo δίνει την δυνατότητα για δοκιμές ρομποτικών αλγορίθμων, σχεδιασμό ρομπότ, regression testing. Προγραμματίζεται σε C++, περιλαμβάνει πολλές μηχανές φυσικής (ODE, Bullet, Simbody, DART), πλήθος ρομποτικών μοντέλων και περιβαλλόντων, ένα ευρύ φάσμα αισθητήρων, και είναι συμβατό προγραμματιστικά και γραφικά, με άλλα εργαλεία όπως το ROS. Πλήθος εφαρμογών βασίζονται σε συνεργασία ROS-Gazebo. Κυρίως χρησιμοποιείται για manipulators και mobile robots. Η χρήση του στα ιπτάμενα είναι περιορισμένη. Καθώς ο Gazebo είναι ένας αρκετά πλούσιος σε χαρακτηριστικά προσομοιωτής, καθιστά δύσκολη την δημιουργία μεγάλης κλίμακας πολύπλοκων γραφικών περιβαλλόντων που πλησιάζουν τον

πραγματικό κόσμο και υστερεί σε τεχνικές rendering που κάνουν άλλες πλατφόρμες όπως η Unreal engine ή η Unity. Τέλος, είναι ένα ώριμο λογισμικό με αρκετά μεγάλη κοινότητα που το χρησιμοποιεί και το αναβαθμίζει και αρκετά παραδείγματα εφαρμογών.

A.1.2 Morse

Ο Morse είναι ένας γενικός προσομοιωτής για ακαδημαϊκά ρομπότ. Εστιάζει στην τρισδιάστατη προσομοίωση μικρών ή μεγάλων περιβαλλόντων, εσωτερικού ή εξωτερικού χώρου, για ένα ή δεκάδες αυτόνομα ρομπότ. Τα περιβάλλοντα προσομοίωσης περιγράφονται σε απλά Python scripts. Ο Morse δίνει στον χρήστη ένα σετ από βασικούς αισθητήρες (κάμερα, laser scanner, GPS, οδομετρία,…), actuators (speed controllers, high-level waypoints controllers, generic joint controllers) και ρομπότ (quadrotors, ATRV, Pioneer3DX, generic 4 wheel vehicle, PR2). Νέα μοντέλα μπορούν να προστεθούν αρκετά εύκολα. Αυτήν την στιγμή υποστηρίζει 6 ανοικτού κώδικα middlewares (ROS,YARP, Pocolibs, MOOS, HLA and Mavlink). Όσον αφορά τα ιπτάμενα ρομπότ, υστερεί στο ότι η μηχανή φυσικής δεν λαμβάνει υπόψιν την βαρύτητα, ενώ δεν προσομοιώνονται ρεύματα αέρα και ανέμου. Αν και πιο πρόσφατη εφαρμογή από αυτή του Gazebo, με μικρότερη κοινότητα λογισμικό, ενδείκνυται για περιπτώσεις vision control ή όπου η ακρίβεια στα γραφικά είναι υψίστης σημασίας για την προσομοίωση, αφού χρησιμοποιεί το ελεύθερο λογισμικό Blender, ένα πανίσχυρο λογισμικό για 3D γραφικά. Όμως δεν παύει να μην προσφέρει GUI και να χρησιμοποιείται μέσω γραμμής εντολών.

A.1.3 Webots

Ο Webots είναι ένας ελεύθερου λογισμικού προσομοιωτής ρομπότ που παρέχει ένα ολοκληρωμένο περιβάλλον ανάπτυξης για την κατασκευή του μοντέλου, τον προγραμματισμό και την προσομοίωσης. Χρησιμοποιείται ευρέως σε πάρα πολλά ιδρύματα παγκοσμίως τόσο στην έρευνα όσο και στην εκπαίδευση. Προσομοιώνει πολύπλοκα ρομποτικά συστήματα, αυτόματα κινούμενα ρομπότ εξυπηρετώντας την έρευνα αλλά και την εκμάθηση χειρισμού κάποιων ρομπότ όπως τα ιπτάμενα και τα τροχοφόρα. Παράλληλα συνοδεύεται με έτοιμες βιβλιοθήκες ρομπότ, αισθητήρων, actuators και αντικειμένων ενώ προγραμματίζεται σε αρκετές γλώσσες όπως C++, Python, Java, Matlab, ROS.

A.1.4 V_REP

Ο προσομοιωτής V_REP, βασίζεται σε μια αρχιτεκτονική κατανεμημένου ελέγχου. Κάθε μοντέλο/αντικείμενο μπορεί να ελέγχεται ξεχωριστά μέσω ενσωματωμένου script, ROS node ή απομακρυσμένου API πελάτη. Αυτό καθιστά τον V_REP αρκετά ευπροσάρμοστο και κατάλληλο για εφαρμογές με πολλαπλά ρομπότ. Οι controllers μπορούν να γραφούν σε C/C++, Python, Java, Lua, Matlab ή Octave. Ο V_REP χρησιμοποιείται για γρήγορη ανάπτυξη αλγορίθμων, προσομοιώσεις αυτοματισμών εργοστασίου, γρήγορη δημιουργία πρωτοτύπων, εκπαίδευση σχετική με ρομπότ, απομακρυσμένη παρακολούθηση, διπλό έλεγχο ασφαλείας και πολλά ακόμη.

A.2 Προσομοιωτές πτήσεις

Πέρα από τους Προσομοιωτές Γενικής Χρήσης, έχουν αναπτυχθεί και ορισμένες εφαρμογές που αφορούν συγκεκριμένα προσομοιώσεις ιπτάμενων μοντέλων, όπως αεροπλάνων και ελικοφόρων μοντέλων. Ορισμένοι από τους πιο ευρέως διαδεδομένους προσομοιωτές πτήσης είναι οι εξής:

- hector_quadrotor
- X Plane -11
- FlightGear
- RealFlight
- CRRCSim
- jMAVSim
- AirSim

A.2.1 Hector_quadrotor

Ένα πακέτο του ROS που συνεργάζεται με το Gazebo και αφορά την σχεδίαση, έλεγχο και προσομοίωση quadrotor UAV systems. Ο hector_quadrotor αν και δίνει την δυνατότητα στον χρήστη να επιλέξει ή να φτιάξει δικά του μοντέλα και controller, περιορίζεται στην προσομοίωση μόνο quadrotors.

A.2.2 Xplane – 11

Ο Xplane-11 είναι ένας εμπορικός προσομοιωτής πτήσης, (διανέμεται και demo version δωρεάν) εξοπλισμένος με πανίσχυρα εργαλεία και αφορά την προσομοίωση εμπορικών, πολεμικών και άλλου είδους αεροπλάνων σε βασικά παγκόσμια τοπία που καλύπτουν την μεγαλύτερη επιφάνεια της Γης. Δίνει επίσης την δυνατότητα τροποποίησης και αλλαγής των παραμέτρων που αφορούν τόσο τα μοντέλα όσο και το περιβάλλον. Η χρήση του είναι πιο επαγγελματική παρά ερευνητική.

A.2.3 FligthGear

Ένας ανοικτού κώδικα προσομοιωτής πτήσης. Υποστηρίζει όλες τις δημοφιλείς πλατφόρμες (Windows, Mac, Linux) και έχει αναπτυχθεί από επιδέξιους εθελοντές από όλο τον κόσμο. Ο στόχος του είναι να δημιουργήσει ένα περιβάλλον προσομοίωσης με χρήση στην έρευνα και σε ακαδημαϊκά περιβάλλοντα, στην εκπαίδευση πιλότων, ως ένα επαγγελματικό εργαλείο μηχανικού για όσους ενδιαφέρονται να υλοποιήσουν τις δικές τους ιδέες με ιπτάμενα, όντας παράλληλα αστείος, ρεαλιστικός και προκλητικός desktop προσομοιωτής πτήσης.

A.2.4 RealFlight

Ένας εμπορικός προσομοιωτής πτήσης, με ικανότητες σχεδίασης και ελέγχου ρεαλιστικών και προσαρμοσμένων μοντέλων. Το ότι είναι όμως συμβατός με περιβάλλον Windows μόνο, περιορίζει την συνεργασία του με πανίσχυρα εργαλεία όπως το ROS, που βασίζονται σε περιβάλλον Unix.

A.2.5 CRRCSim

Ένας λιγότερο χρησιμοποιούμενος προσομοιωτής για ελικόπτερα και αεροπλάνα. Αφορά κυρίως την εκμάθηση χειρισμού μοντέλων αεροσκάφους.

A.2.6 JMAVSim

Ο Java Micro Air Vehicle Simulator (jMAVSim) είναι ένας απλός και ελαφρύς προσομοιωτής. Υποστηρίζει MAVLink πρωτόκολλο, χρησιμοποιεί την βιβλιοθήκη Java3D για οπτικοποίηση, και αφορά μόνο ελικοφόρα ρομπότ (quad/multirotors) που τρέχουν τον αυτόματο πιλότο PX4. Εύκολος να στηθεί και να ελέγξει εάν το όχημα μπορεί να απογειωθεί, πετάξει, προσγειωθεί, ενώ προσομοιώνει και διάφορες περιπτώσεις αποτυχίας (πχ αποτυχία GPS).

A.2.7 AirSim

Προσομοιωτής που χρησιμοποιείται κυρίως για δημοφιλείς αυτόματους πιλότους (π.χ. PX4). Ο AirSim στοχεύει στην ανάπτυξη και στον έλεγχο αλγορίθμων εφαρμογές αυτόνομων οχημάτων, όπως deep learning, computer vision, και re-inforcement learning αλγορίθμους. Η φυσική μηχανικής του βασίζεται στην Unreal Engine 4 (UE4) και μπορεί να υποστηρίξει στα 470 Hz προσομοίωση Hardware-in-the-Loop (HIL) σε πραγματικό χρόνο. Υποστηρίζει UAVs, Iris και PX4 quadrotor. Ο AirSim έχει έναν απλό flight controller ως προεπιλογή. Όμως μπορεί να χρησιμοποιηθεί και χωρίς flight controller.

Παράρτημα Β

Τεχνολογίες (ROS, Gazebo)

Οι δύο κύριες τεχνολογίες πάνω στις οποίες βασίζεται η λειτουργία του προσομοιωτή είναι το ROS και το Gazebo. Στον παρόν κεφάλαιο, εξηγούνται τα βασικά χαρακτηριστικά του κάθε ενός προγράμματος, ώστε ο χρήστης να αποκτήσει κάποιες στοιχειώδης γνώσεις σχετικές με τα εργαλεία αυτά, πριν εμβαθύνει στην λειτουργία του υπόλοιπου προσομοιωτή.

B.1 ROS

Βασικά χαρακτηριστικά του ROS:

- ROS nodes
- ROS topics
- ROS services
- ROS parameter server
- ROS URDF, xacro

B.1.1 Nodes

Στο ROS, ένα node είναι μια διαδικασία που εκτελεί υπολογισμούς. Τα node συνδυάζονται όλα μαζί δημιουργώντας έναν γράφο και επικοινωνούν μεταξύ τους χρησιμοποιώντας topics, services και τον Parameter Server. Αυτά τα nodes προορίζονται να λειτουργούν σε μικρή κλίμακα καθώς ένα ρομποτικό σύστημα ελέγχου συνήθως περιλαμβάνει πολλά από αυτά. Η χρήση των ROS nodes προσφέρει αρκετά πλεονεκτήματα στο συνολικό σύστημα. Αυξάνεται η ανοχή στα σφάλματα καθώς τυχόν αποτυχίες συμβαίνουν σε μεμονωμένα nodes. Μειώνεται η πολυπλοκότητα του κώδικα.

B.1.2 Topics

Τα topics είναι το μέσο με το οποίο τα nodes ανταλλάσουν μηνύματα. Τα topics έχουν ανώνυμη σημασιολογία δημοσίευση/ανάγνωσης, οπότε αποσυνδέεται η παραγωγή της πληροφορίας από την κατανάλωση. Με άλλα λόγια, τα node δεν γνωρίζουν με ποιόν επικοινωνούνε. Όσα ενδιαφέρονται για την πληροφορία ενός topic, εγγράφονται σε αυτό. Τα nodes που παράγουν πληροφορία, την δημοσιεύουν στο topic. Έτσι μπορεί να υπάρχουν

πολλαπλά nodes που γράφουν και διαβάζουν από ένα topic. Χρησιμοποιούνται για ασύγχρονη, μονόδρομη επικοινωνία.

B.1.3 Services

Τα services χρησιμοποιούνται από nodes που χρειάζεται να εκτελέσουν απομακρυσμένα αιτήματα, π.χ. να λάβουν μια απάντηση σε ένα αίτημα. Είναι κατάλληλα για RPC αιτήματα/αποκρίσεις που συχνά απαιτούνται σε ένα κατανεμημένο σύστημα. Το αίτημα/απάντηση ορίζεται από ένα ζεύγος μηνυμάτων: ένα για το αίτημα και ένα για την απάντηση. Ένα ROS node παρέχει ένα service μέσω ενός ονόματος και ο "πελάτης" καλεί το service στέλνοντας την αίτηση και περιμένοντας την απάντηση. Τα services δηλώνονται μέσω srv αρχείων, που μεταφράζονται σε πηγαίο κώδικα μέσω μιας ROS βιβλιοθήκης. Ένα node πελάτης μπορεί να διατηρήσει μόνιμη σύνδεση με ένα service, πράγμα που επιτρέπει μεγαλύτερη απόδοση με κόστος την μικρότερη ευρωστία σε αλλαγές του παρόχου υπηρεσιών. Τα services χρησιμοποιούνται για σύγχρονη επικοινωνία.

B.1.4 Parameter Server

Ο Parameter Server είναι ένα κοινόχρηστο, πολυμεταβητό λεξικό προσβάσιμο μέσω δικτύου API και χρησιμοποιείται για να διατηρεί μικρό μέρος μεταβλητών. Τα node χρησιμοποιούν τον Server για να αποθηκεύσουν και να ανακτήσουν παραμέτρους κατά την προσομοίωση. Καθώς δεν έχει σχεδιαστεί για υψηλή απόδοση, είναι καλύτερο να χρησιμοποιείται για στατικά, μη δυαδικά δεδομένα όπως παραμέτρους διαμόρφωσης. Έχει σκοπό να είναι ορατός σε όλο το σύστημα και υλοποιείται χρησιμοποιώντας XMLRPC.

B.1.5 URDF

Το Universal Robotic Description Format (URDF) είναι μια XML μορφοποίηση αρχείου που χρησιμοποιείται από το ROS για να περιγράψει τα στοιχεία των ρομπότ. Ενώ τα URDFs είναι μια χρήσιμη μορφοποίηση στο ROS, στερούνται κάποιων χαρακτηριστικών και δεν έχουν αναβαθμιστεί ώστε να καλύπτουν κάποιες ανάγκες των σύγχρονων ρομπότ. Τα URDF μπορούν να προσδιορίσουν τις κινηματικές και δυναμικές ιδιότητες ενός μόνο ρομπότ μεμονωμένα. Δεν μπορούν να προσδιορίσουν την θέση του ρομπότ μέσα στον κόσμο της προσομοίωσης. Παράλληλα, δεν μπορεί να ορίσει πράγματα πέρα από ρομπότ, όπως φώτα, 3D χάρτες κ.τ.λ.

B.1.6 Xacro

Συνηθίζεται, για ευκολία του χρήστη, στα URDF αρχεία να γίνεται χρήση μακροεντολών της xacro, μιας XML γλώσσας, που μετατρέπει xacro εντολές σε μεγαλύτερες XML φράσεις. Πέρα από εξοικονόμηση χρόνου στην δημιουργία ενός URDF αχρείου, η χρήση της Xacro κάνει το αρχείο ευανάγνωστο και εύκολο να κατανοηθεί και διορθωθεί. Φυσικά δεν είναι δεσμευτική η χρήση της αλλά προαιρετική.

B.2 Gazebo

Βασικά χαρακτηριστικά του Gazebo:

- gazebo sdf
- gazebo plugins

B.2.1 SDF

Για να αντιμετωπιστούν οι αδυναμίες των URDFs, δημιουργήθηκε μια νέα μορφή που ονομάζεται Simulation Description Format (SDF). Το SDF σχεδιάστηκε για τον προσομοιωτή Gazebo, έχοντας στόχο επιστημονικές εφαρμογές ρομπότ. Το SDF είναι μια πλήρης περιγραφή για οτιδήποτε υπάρχει στον κόσμο της προσομοίωσης και στο ρομπότ. Είναι μια κλιμακωτή, σταθερή ανθεκτική και επεκτάσιμη μορφή ικανή να περιγράψει κάθε είδος ρομπότ, στατικά και δυναμικά αντικείμενα, φώτα ακόμα και τη φυσική. Η μορφοποίηση SDF περιγράφεται χρησιμοποιώντας XML. Τόσο το URDF και το SDF είναι πλήρως τεκμηριωμένες μορφοποιήσεις και αφήνεται στην κρίση του χρήστη να επιλέξει την κατάλληλη ανάλογα με τις ανάγκες της εφαρμογής. Προκειμένου να χρησιμοποιηθούν URDF αρχεία στο Gazebo, πρέπει να προστεθούν κάποιες επιπλέον επεκτάσεις ώστε να μπορέσει να μετατρέψει τα URDF σε SDF αρχεία αυτόματα.

B.2.2 Gazebo plugins

Τα Gazebo plugins είναι κομμάτια κώδικα που μετατρέπονται σε κοινόχρηστες βιβλιοθήκες και εισάγονται στον προσομοιωτή. Τα plugin έχουν απευθείας πρόσβαση σε κάθε λειτουργικότητα του Gazebo μέσω C++ κλάσεων. Με αυτά οι χρήστες μπορούν και ελέγχουν σχεδόν κάθε πτυχή του Gazebo. Τα plugins είναι αυτοτελείς ρουτίνες που εύκολα κοινοποιούνται, προστίθενται και αφαιρούνται από ένα τρέχον σύστημα.

Αυτήν την στιγμή υπάρχουν 6 τύποι plugin που αφορούν τον κόσμο της προσομοίωσης, το μοντέλο, τους αισθητήρες, το σύστημα, τα οπτικά και το GUI. Τα Gazebo plugins προσφέρουν στα URDF μοντέλα μεγαλύτερη λειτουργικότητα. Περισσότερες πληροφορίες σχετικές με τα gazebo plugin εδώ http://gazebosim.org/tutorials?tut=ROS_gzplugins.

Παράρτημα Γ

Μοντέλο Δυναμικής

Στο συγκεκριμένο κεφάλαιο παρουσιάζονται οι μαθηματικοί τύποι που χρησιμοποιήθηκαν για την μοντελοποίηση των δυνάμεων. Όλοι οι μαθηματικοί τύποι προέρχονται από το σύγγραμμα **Small Unmanned Aircraft: Theory and Practise** του **Beard** και **McLain**, **κεφάλαιο 4, Forces & Moments** [1] και από το **Modeling a Fixed-Wing UAV A Collection of Equations** [2]**.** Για αναλυτική εξήγηση της θεωρίας ανατρέξτε εκεί.

Το σύνολο των δυνάμεων και ροπών που ασκούνται στο μοντέλο προέρχονται από τις αεροδυναμικές επιφάνειες (πτερύγια), του κινητήρες, και την βαρύτητα. Καθώς το Gazebo έχει την δυνατότητα και προσομοιώνει την δύναμη του βάρους, θα δοθεί σημασία στις άλλες δύο μόνο κατηγορίες, δηλαδή τις αεροδυναμικές δυνάμεις και τις δυνάμεις ώθησης των κινητήρων.

Οι μαθηματικοί τύποι παρουσιάζονται με δύο μορφές. Μία όπως στα παραπάνω συγγράμματα και μία όπως στον κώδικα, ώστε να μπορεί ο αναγνώστης να κάνει την σύνδεση μεταξύ των δύο.

Γ.1 Αεροδυναμικές δυνάμεις και ροπές

Οι παρακάτω τύποι ισχύουν και για τους δύο τύπους αεροδυναμικής.

$$lift = \frac{1}{2}\rho V_a^2 S \begin{pmatrix} [-C_D(\alpha)\sin\alpha - C_L(\alpha)\cos\alpha] \ + \\ \left[-C_{D_q} sin\,\alpha - C_{L_q}\cos\alpha\right]\frac{c}{2V_a}q + \\ \left[-C_{D_{\delta e}}\sin\alpha - C_{L_{\delta e}}\cos\alpha\right]\delta_e \end{pmatrix}$$

$$drag = \frac{1}{2}\rho V_a^2 S \begin{pmatrix} [-C_D(\alpha)\cos\alpha + C_L(\alpha)\sin\alpha] \ + \\ \left[-C_{D_q} cos\,\alpha + C_{L_q}\sin\alpha\right]\frac{c}{2V_a}q + \\ \left[-C_{D_{\delta e}}\cos\alpha + C_{L_{\delta e}}\sin\alpha\right]\delta_e \end{pmatrix}$$

$$f_y = \frac{1}{2}\rho V_a^2 S\left(C_{Y_0} + C_{Y_\beta}\beta + C_{Y_p}\frac{b}{2V_a}p + C_{Y_r}\frac{b}{2V_a}r + C_{Y_a}\delta_a + C_{Y_r}\delta_r\right)$$

$$l = \frac{1}{2}\rho V_a^2 Sb\left(C_{l_0} + C_{l_\beta}\beta + C_{l_p}\frac{b}{2V_a}p + C_{l_r}\frac{b}{2V_a}r + C_{l_{\delta_\alpha}}\delta_a + C_{l_{\delta_r}}\delta_r\right)$$

$$m = \frac{1}{2}\rho V_a^2 Sc\left(C_{m_0} + C_{m_a}a + C_{m_q}\frac{c}{2V_a}q + C_{m_e}\delta_e\right)$$

$$n = \frac{1}{2}\rho V_a^2 Sb\left(C_{n_0} + C_{n_\beta}\beta + C_{n_p}\frac{b}{2V_a}p + C_{n_r}\frac{b}{2V_a}r + C_{n_{\delta_\alpha}}\delta_a + C_{n_{\delta_r}}\delta_r\right)$$

Οι τύποι αυτοί μεταφράζονται σε κώδικα ως εξής:

```
lift=    qbar*((-c_drag_alpha*sa-c_lift_alpha*ca)+
               (-c_drag_q*sa-c_lift_q*ca)*0.5/airspeed*c*q+
               (-c_drag_deltae*sa-c_lift_deltae*ca)*input_y )

drag=    qbar*((-c_drag_alpha*ca+c_lift_alpha*sa)+
               (-c_drag_q*ca+c_lift_q*sa)*0.5/airspeed*c*q+
               (-c_drag_deltae*ca+c_lift_deltae*sa)*input_y )

fy=      qbar*(c_y_0+
               c_y_b*beta+
               c_y_p*b/2/airspeed*p+
               c_y_r*b/2/airspeed*r+
               c_y_deltaa*input_x+
               c_y_deltar*input_z      )


l=       qbar*(b*(   c_l_0+
                     c_l_b*beta+
                     c_l_p*b/2/airspeed*p+
                     c_l_r*b/2/airspeed*r+
                     c_l_deltaa*input_x+
                     c_l_deltar*input_z      ))

m=       qbar*(c*(   c_m_0+
                     c_m_a*alpha+
                     c_m_q*c/2/airspeed*q+
                     c_m_deltae*input_y      ))

n=       qbar*(b*(   c_n_0+
                     c_n_b*beta+
                     c_n_p*b/2/airspeed*p+
                     c_n_r*b/2/airspeed*r+
                     c_n_deltaa*input_x+
                     c_n_deltar*input_z      ))
qbar=    1/2*rho*airspeed^2*s
```

Γ.1.1 stdLinearAero

Για τον stdLinearAero τύπο πτερυγίου, ο συντελεστής του lift c_lift_alpha $[C_L(\alpha)]$ μοντελοποιείται ως το ζυγισμένο άθροισμα δύο μερών, μας γραμμικής συνάρτησης ως προς γωνία angle_of_attack και του συντελεστή ανύψωσης μιας επίπεδης πλάκας. Στην περιοχή εντός της γωνίας stall, επικρατεί το πρώτο μέρος. Σε αντίθετη περίπτωση, το φτερό μοντελοποιείται ως επίπεδη πλάκα.

$$C_L(\alpha) = \big(1 - \sigma(\alpha)\big)\big(C_{L_0} + C_{L_\alpha} a\big) + \sigma(\alpha)[2sign(\alpha)\sin^2 a \cos a]$$

$$\sigma(\alpha) = \frac{1 + e^{-M(a-a_0)} + e^{M(a+a_0)}}{(1 + e^{-M(a-a_0)})(1 + e^{M(a+a_0)})}$$

```
sigmoid =      (1+exp(-M*(alpha-alpha0))+exp(M*(alpha+alpha0))) /
               (1+exp(-M*(alpha-alpha0))) /
               (1+exp(M*(alpha+alpha0)))

linear =       (1.0-sigmoid) * (c_lift_0 + c_lift_a0*alpha)
flatPlate =    sigmoid*(2*sign(alpha)*(sin(alpha))^2*cos(alpha))
c_lift_alpha = linear+flatPlate
```

Ο συντελεστής του drag c_drag_a ορίζεται ως το ζυγισμένο άθροισμα παρασιτικού και επαγόμενου drag.

$$C_D(\alpha) = C_{D_p} + \frac{\big(C_{L_0} + C_{L_\alpha}\alpha\big)^2}{\pi e AR}$$

$$AR = \frac{b^2}{S}$$

```
c_drag_alpha =       c_drag_p + c_lift_0 +
                    (c_lift_a0*alpha)^2 / (M_PI*oswald*AR)

          AR =       b^2 / s
```

Γ.1.2 polyAero

Για την περίπτωση polyAero, οι μεταβλητές και $C_L(\alpha)$ και $C_D(\alpha)$ υπολογίζονται μέσω πολυωνύμων.

Γ.2 Δύναμη, ροπή ώθησης και γωνιακή ταχύτητα κινητήρα

Γ.2.1 genericEngine

Σύμφωνα με το σύγγραμμα του Beard [1], οι μαθηματικοί τύποι ενός κινητήρα με προπέλα είναι οι εξής:

$$thrust = \frac{1}{2}\rho S_{prop} C_{prop}((k_{motor}\delta_t)^2 - V_a^2)$$

$$torque = -k_{T_p}(k_\Omega \delta_t)^2$$

$$omega = k_\Omega \delta_t$$

και σε κώδικα:

```
thrust = 1/2 * rho * s_prop * c_prop *
                          (motor_input * k_motor^2 - airspeed^2)

torque = -rotationDir * k_t_p * (k_omega * motor_input)^2

omega  = k_omega * motor_input
```

Γ.2.2 electricEngine

Σύμφωνα με το σύγγραμμα Modeling a Fixed-Wing UAV A Collection of Equations [2] και το μαθηματικό μοντέλο της προπέλας του ηλεκτρονικού συγγράμματος Thermodynamics and Propulsion [3] για τον τύπο του ηλεκτροκινητήρα χρησιμοποιήθηκαν οι εξής τύποι:

$$E_i = \frac{\omega}{2\pi K_v}$$

$$I_m = \frac{Cells * 4 * \delta_t - Ei}{R_s * \delta_t + R_m}$$

$$P_{mot} = E_i(I_m - I_0)$$

$$J = \frac{V_a}{2\omega\pi D}$$

$$P_{prop} = C_p(J) * \rho * \left(\frac{\omega}{2\pi}\right)^3 D^5$$

$$thrust = P_{prop}\frac{C_t(J)}{V_a} + 0.9e^{-\frac{V_a}{4}}\left(\frac{\pi * \rho * D^2 * P_{mot}^2}{2}\right)^{\frac{1}{3}}$$

$$torque = \frac{P_{prop}}{\omega}$$

$$\delta_T = \frac{P_{mot} - P_{prop}}{\omega}$$

$$\dot{\omega} = \frac{\delta_T}{I_{eng}}$$

$$\omega = \dot{\omega}\delta_t$$

$P_{mot} = ισχύς\ κινητήρα$
$P_{prop} = ισχύς\ προπέλας$
$C_p = συντελεστής\ ισχύος$
$C_t = συντελεστής\ ώθησης$
$V_a = σχετική\ ταχύτητα\ ανέμου$
$J = advance\ ratio$
$D = διάμετρος\ προπέλας$
$r = πυκνότητα\ αέρα$
$\delta_t = σήμα\ ελέγχου\ κινητήρα$
$I_{eng} = ροπή\ αδράνειας\ κινητήρα\ και\ προπέλας$

και ο κώδικας:

```
Ei = omega / 2 / M_PI / Kv;
Im = (Cells * 4.0 * motor_input - Ei) / (Rs * motor_input + Rm)
engPower = Ei * (Im - I0)
advRatio = normalWind / (std::fabs(omega) / 2.0 / M_PI) / propDiam
propPower = propPowerPoly->evaluate(advRatio) * rho * pow(std::fabs(omega) / 2.0 / M_PI, 3) * pow(propDiam, 5)
npCoeff = npPoly->evaluate(advRatio)
prop_wrenches.thrust = propPower * std::fabs(npCoeff / (normalWind + 1.0e-10))
fadeFactor = (exp(-normalWind * 3 / 12))
staticThrust = 0.9 * fadeFactor * pow(M_PI / 2.0 * propDiam * propDiam * rho * engPower * engPower, 1.0 / 3)
prop_wrenches.thrust = prop_wrenches.thrust + staticThrust

prop_wrenches.torque = propPower / omega
deltaT = (engPower - propPower) / std::fabs(omega)
omegaDot = 1 / engInertia * deltaT
omega += rotationDir * omegaDot * motor_input
omega = rotationDir * std::max(std::min(std::fabs(omega), omegaMax), omegaMin)
prop_wrenches.omega = omega
```

Η συνολική ώθηση αποτελείται από δύο όρους. Ο πρώτος περιγράφει την ώθηση του κινητήρα όταν το μοντέλο κινείται ($V_a > 0$). Ο δεύτερος όρος υπολογίζει την ώθηση για ταχύτητες κοντά στο μηδέν ($V_a \simeq 0$), δηλαδή όταν το μοντέλο είναι σταματημένο.

Παράρτημα Δ

URDF παραδείγματα - εξηγήσεις

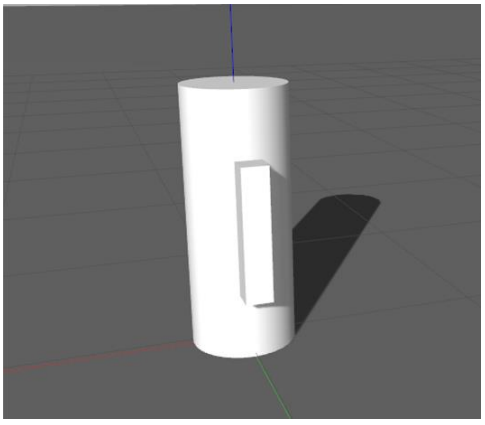
Στο ROS τα μοντέλα περιγράφονται σε αρχεία μορφής URDF (Unified Robot Description Format), XML μορφοποίησης. Σε αυτά τα αρχεία περιγράφονται όλα τα μέρη του μοντέλου (links), και ο τρόπος με τον οποίο συνδέονται (joints).

Σε αυτό το κεφάλαιο αρχικά εξηγείται η δομή ενός πολύ απλού μοντέλου. Στην συνέχεια γίνεται εκτενέστερη ανάλυση του κώδικα του URDF αρχείου του plane_3wing_2motor μοντέλου. Τέλος εξηγούνται οι γραμμές κώδικα στο αρχείο URDF του plane_1wing_sensors, που αφορούν τους έτοιμους αισθητήρες

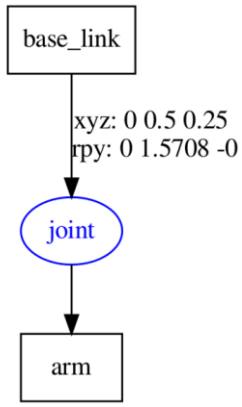
Δ.1 Ένα απλό μοντέλο

Παρακάτω αναλύεται ένα πολύ απλό μοντέλο για την περιγραφή και ανάλυση των βασικών στοιχείων ενός URDF αρχείου, προκειμένου ο χρήστης να αποκτήσει μια πρώτη ιδέα.

Το παρακάτω μοντέλο είναι ένας κύλινδρος με μία άρθρωση στην μια του πλευρά όπου συνδέεται ένα ορθογώνιο. Πλησιάζει σε λογική την άρθρωση ενός χεριού, όπου ο κύλινδρος είναι το σώμα του ρομπότ και το ορθογώνιο ένας πολύ απλός βραχίονας.



Εικόνα 40. Εικόνα απλού μοντέλου



Εικόνα 40. URDF γράφος μοντέλου

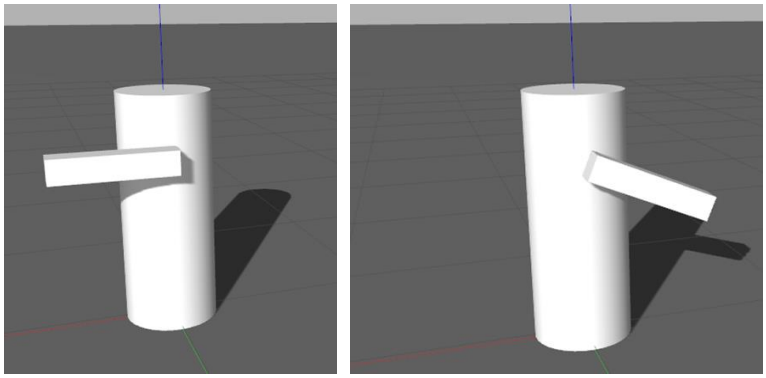
Ο κώδικας

```
<?xml version="1.0" encoding="utf-8"?>
<robot name="simple_model">

    Βασικό link
    <link name="base_link">
        <visual>
        <origin rpy="0 0 0" xyz="0 0 0"/>
          <geometry>
            <cylinder length="1" radius="1"/>
          </geometry>
        </visual>

        <collision>
        <origin rpy="0 0 0" xyz="0 0 0"/>
          <geometry>
            <cylinder length="1" radius="1"/>
          </geometry>
        </collision>

        <inertial>
        <origin rpy="0 0 0" xyz="0 0 0"/>
         <mass value="5"/>
         <inertia    ixx="1.66666666667" ixy="0" ixz="0" iyz="0"
                     iyy="1.66666666667"
                     izz="2.5"/>
        </inertial>
    </link>

    Link βραχίονα
    <link name="arm">
        <visual>
        <origin rpy="0 0 0" xyz="0.25 0.05 0"/>
          <geometry>
            <box size="0.5 0.1 0.1"/>
          </geometry>
        </visual>

        <collision>
        <origin rpy="0 0 0" xyz="0.25 0.05 0"/>
          <geometry>
            <box size="0.5 0.1 0.1"/>
          </geometry>
        </collision>

        <inertial>
        <origin rpy="0 0 0" xyz="0.25 0.05 0"/>
         <mass value="0.5"/>
```

```
            <inertia    ixx="0.00083333333" ixy="0" ixz="0" iyz="0"
                        iyy="0.01083333333"
                        izz="0.01083333333"/>
          </inertial>
      </link>

      Joint μεταξύ βασικού link και βραχίονα
      <joint name="joint" type="revolute">
        <parent link="base_link"/>
        <child link="arm"/>
        <origin rpy="0 1.57 0" xyz="0 0.5 0.25"/>
        <axis xyz="0 1 0"/>
        <limit effort="10.0" velocity="0.5" lower="-1.57"
                upper="1.57"/>
      </joint>
  </robot>
```

Το παραπάνω μοντέλο αποτελείται από δύο links ενωμένα μέσω ενός joint. Τα links είναι δύο απλά σχήματα (το ένα κύλινδρος και το άλλο ορθογώνιο). Ένα link, στο URDF αρχείο, περιγράφεται με 3 χαρακτηριστικά.

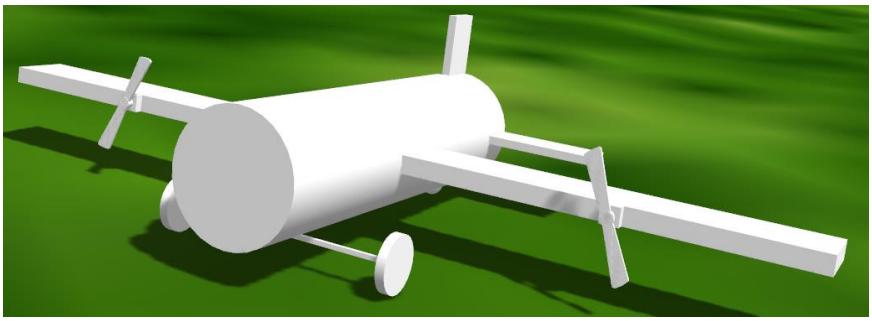
Visuals



Εικόνα 41. Δύο τυχαίες θέσεις του βραχίονα

Το πρώτο είναι το visual, όπου περιγράφονται το σχήμα, οι διαστάσεις, η θέση και ο προσανατολισμός του link όπως θα φαίνεται στην προσομοίωση, δηλαδή ότι θα βλέπει ο χρήστης. Για visual link, μπορεί να επιλεχθεί κάποιο από τα βασικά σχήματα (τετράγωνο, κύλινδρος, σφαίρα) ή να επιλεχθεί κάποιο mesh, δηλαδή μία τρισδιάστατη δομή ενός αντικειμένου, αποθηκευμένο σε Collada αρχείο. Τα αρχεία Collada (COLLAborative Design Activity) είναι μια μορφή αρχείου για διαδραστικές τρισδιάστατες εφαρμογές. Τα COLLADA είναι αρχεία XML, συνήθως με την επέκταση .dae (digital asset exchange).

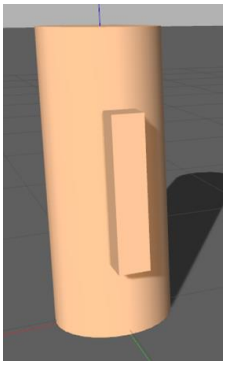
Στον Last_letter_2 γίνεται χρήση Collada αρχείου για το visual της προπέλας μοντέλα fixed-wing.



Εικόνα 42. Στο model_3wing_2motor το visual της κάθε προπέλας περιγράφεται από αρχείο Collada

Collision

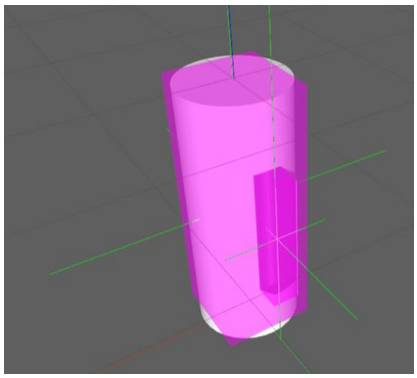
Το δεύτερο είναι το collision, όπου περιγράφονται το σχήμα, οι διαστάσεις, η θέση και ο προσανατολισμός του link όπως τα βλέπει η προσομοίωση, δηλαδή ορίζονται τα όρια του μοντέλου που θα υπολογίζει η μηχανή φυσικής στην αλληλεπίδρασή του με το περιβάλλον και τα υπόλοιπα αντικείμενα του χώρου προσομοίωσης. Συνήθως το collision είναι πιο απλό σχήμα από ότι το visual.



Εικόνα 43. Το collision του μοντέλου. Επειδή τα σχήματα είναι απλά, τυχαίνει να είναι ίδιο με το visual.

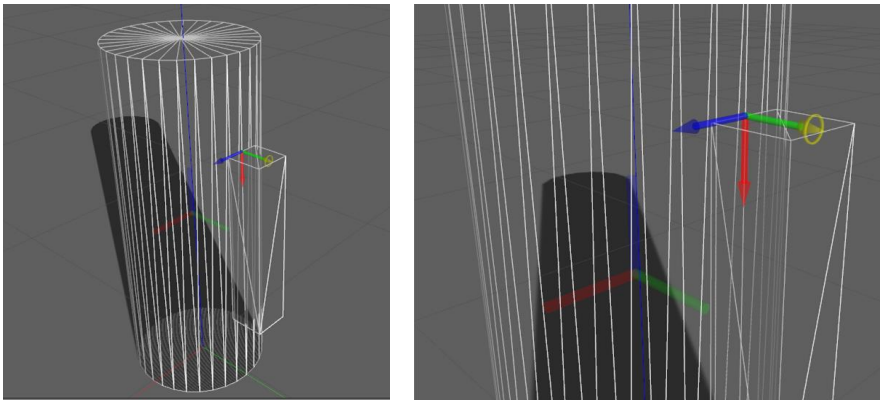
Inertial

Το τελευταίο είναι το inertial, δηλαδή η αδράνεια του link, στοιχείο απαραίτητο για την μηχανή φυσικής ώστε να γίνει μία ρεαλιστική προσομοίωση. Εδώ δηλώνεται η μάζα και ο 3x3 πίνακας αδράνειας του link, με τις τιμές να έχουν άμεση σχέση με τις διαστάσεις και την μάζα που έχουν δηλωθεί νωρίτερα. Αξίζει να σημειωθεί, πώς τυχόν λανθασμένες τιμές του πίνακα αδράνειας είναι πιθανόν να οδηγήσουν σε απρόβλεπτες κινήσεις του μοντέλου στην προσομοίωση.



Εικόνα 44. Η αδράνεια του μοντέλου όπως την οπτικοποιεί το Gazebo

Όσον αφορά το joint, είναι σημαντικό να δηλωθεί ο τύπος του (π.χ. revolute, continuous, fixed etc), η θέση, ο προσανατολισμός του, ο άξονας περιστροφής του καθώς και όποια άλλη πληροφορία πηγάζει από τον τύπο του joint.



Εικόνα 45. Στην εικόνα φαίνεται το πλαίσιο συντεταγμένων του joint. Το δαχτυλίδι δηλώνει τον άξονα περιστροφής του link που συνδέεται μέσω του συγκεκριμένου joint.

Προσοχή! Κάθε σύστημα συντεταγμένων που προσθέτουμε στο μοντέλο μας, ορίζεται με βάση το προηγούμενο. Για παράδειγμα, το σύστημα συντεταγμένων ενός joint περιγράφεται σε σχέση με αυτό του link που εφαρμόζεται, ενώ το επόμενο link που συνδέεται μέσω αυτού του joint περιγράφεται ως προς το σύστημα συντεταγμένων του joint. Με λίγα λόγια, κάθε παιδί υιοθετεί το σύστημα συντεταγμένων του πατέρα.

Δ.2 URDF του plane_3wing_2motor

Στο κεφάλαιο 4, στο τέλος του υποκεφαλαίου παρουσιάστηκαν κάποια βασικά σημεία του κώδικα URDF περιγραφής του plane_3wing_motor. Παρακάτω δίνεται ολόκληρο το αρχείο URDF με επεξηγηματικά σχόλια.

```
<?xml version="1.0"?>
<robot name="plane_3wing_2motor"
Δήλωση xacro. Απαραίτητο για την μετατροπή των μακροεντολών σε κείμενο URDF
xmlns:xacro="http://www.ros.org/wiki/xacro">

Δήλωση xacro μεταβλητής για την συνολική μάζα του μοντέλου.
Όλες οι επόμενες μεταβλητές ορίζονται με παρόμοιο τρόπο.
<xacro:property name="mass" value="3"/>

Μέγεθος μοντέλου = 1, χρησιμοποιείται για κλιμάκωση
<xacro:property name="body_s" value="1" />


Χαρακτηριστικά μεγέθη body link (body_FLU)
<xacro:property name="body_h" value="${body_s}" />
```

```
<xacro:property name="body_r" value="${0.14*body_s}" />
<xacro:property name="body_m" value="${0.3*mass}" />

    Δεκτό μόνο το όνομα [body_FLU] για το βασικό link
    <link name="body_FLU">
        <visual>
            <origin rpy="0 -${pi/2} 0 " />
            <geometry>
                <cylinder radius="${body_r}" length="${body_h}" />
            </geometry>
        </visual>

        <collision>
            <origin rpy="0 -${pi/2} 0 " />
            <geometry>
                <cylinder radius="${body_r}" length="${body_h}" />
            </geometry>
        </collision>

        <inertial>
            <origin rpy="0 -${pi/2} 0 " />
            Μάζα link
            <mass value="${body_m}" />
            Δήλωση 3x3 πίνακα αδρανείας link
            <inertia
                 ixx="${1/12*(3*body_r*body_r+body_h*body_h)*body_m}"
                 ixy="0" ixz="0" iyz="0"
                 iyy="${1/12*(3*body_r*body_r+body_h*body_h)*body_m}"
                  izz="${1/2*body_m*body_r*body_r}" />
        </inertial>
    </link>

         -------------------- Πτερύγια -----------------------
Παράμετροι πτερυγίου 1
<xacro:property name="airfoil_x" value="${0.15*body_s}" />
<xacro:property name="airfoil_y" value="${2*body_s}" />
<xacro:property name="airfoil_z" value="${0.04*body_s}"/>
<xacro:property name="airfoil_m" value="${0.2*mass}" />

 Μόνο το airfoil(Αριθμός_πτερυγίου) δεκτή μορφή ονόματος για πτερύγια
    <link name="airfoil1">
        <visual>
            <geometry>
                <box size="${airfoil_x} ${airfoil_y} ${airfoil_z}" />
            </geometry>
        </visual>

        <collision>
            <geometry>
```

```
                <box size="${airfoil_x} ${airfoil_y} ${airfoil_z}" />
            </geometry>
        </collision>

         <inertial>
            <mass value="${airfoil_m}" />
            <inertia
                ixx="${1/12*airfoil_m*(airfoil_y*airfoil_y+airfoil_z
                *airfoil_z)}" ixy="0" ixz="0" iyz="0"
                iyy="${1/12*airfoil_m*(airfoil_x*airfoil_x+airfoil_z
                *airfoil_z)}"
                izz="${1/12*airfoil_m*(airfoil_x*airfoil_x+airfoil_y
                *airfoil_y)}" />
        </inertial>
    </link>

    Δήλωση συνδέσμου μεταξύ δύο link
    <joint name="body_FLU_to_airfoil1" type="revolute">
        <origin xyz="0 0 0" rpy ="0 0 0"/>
        <parent link="body_FLU" />
        <child link="airfoil1" />
        Άνω και κάτω όριο γωνίας μηδέν, όταν δεν δηλώνονται
        <limit effort="1000.0" velocity="0.5"/>
    </joint>

<xacro:property name="airfoil2_x" value="${0.1*body_s}" />
<xacro:property name="airfoil2_y" value="${0.8*body_s}" />
<xacro:property name="airfoil2_z" value="${0.02*body_s}"/>
<xacro:property name="airfoil2_m" value="${0.05*mass}" />

    <link name="airfoil2">
      … όμοια με airfoil1 …
    </link>

    <joint name="body_FLU_to_airfoil2" type="revolute">
        <origin xyz="${-body_h/2+airfoil2_x/2} 0 0" rpy ="0 0 0"/>
        <parent link="body_FLU" />
        <child link="airfoil2" />
        <limit effort="1000.0" velocity="0.5"/>
    </joint>

<xacro:property name="airfoil3_x" value="${0.1*body_s}" />
<xacro:property name="airfoil3_y" value="${0.15*body_s}" />
<xacro:property name="airfoil3_z" value="${0.02*body_s}"/>
<xacro:property name="airfoil3_m" value="${0.05*mass}" />


    <link name="airfoil3">
        … όμοια με airfoil1 …
```

```
    </link>

    <joint name="body_FLU_to_airfoil3" type="revolute">
        <origin xyz="${-body_h/2+airfoil3_x/2} 0
                        ${body_r+airfoil3_y/2}" rpy ="1.57 0 0"/>
        <parent link="body_FLU" />
        <child link="airfoil3" />
        <limit effort="1000.0" velocity="0.5"/>
    </joint>


          -------------------- Motors --------------------
```

Παράμετροι 1ου κινητήρα

```
<xacro:property name="motor_x" value="${0.02*body_s}" />
<xacro:property name="motor_y" value="${0.04*body_s}" />
<xacro:property name="motor_z" value="${0.04*body_s}" />
<xacro:property name="motor_m" value="${0.15*mass}" />
```

Μόνο το motor(Αριθμός_κινητήρα) δεκτή μορφή ονόματος για κινητήρες

```
    <link name="motor1">
        <visual>
            <origin xyz="${0.5*motor_x} 0 0"/>
            <geometry>
                <box size="${motor_x} ${motor_y} ${motor_z}" />
            </geometry>
        </visual>

        <collision>
            <origin xyz="${0.5*motor_x} 0 0"/>
            <geometry>
                <box size="${motor_x} ${motor_y} ${motor_z}" />
            </geometry>
        </collision>

        <inertial>
            <origin xyz="${0.5*motor_x} 0 0"/>
            <mass value="${motor_m}" />
            <inertia
              ixx="${1/12*motor_m*(motor_y*motor_y+motor_z*motor_z)}"
              ixy="0" ixz="0" iyz="0"
              iyy="${1/12*motor_m*(motor_x*motor_x+motor_z*motor_z)}"
              izz="${1/12*motor_m*(motor_x*motor_x+motor_y*motor_y)}"
              />
        </inertial>
    </link>


    <joint name="airfoil1_to_motor1" type="revolute">
        <origin xyz="${airfoil_x/2} ${0.3*airfoil_y} 0" />
```

```
        <limit effort="1000.0" velocity="0.5"/>
        <parent link="airfoil1" />
        <child link="motor1" />
    </joint>

    <link name="motor2">
        … όμοια με motor1 …
    </link>

    <joint name="airfoil1_to_motor2" type="revolute">
        <origin xyz="${airfoil_x/2} -${0.3*airfoil_y} 0" />
        <limit effort="1000.0" velocity="0.5"/>
        <parent link="airfoil1" />
        <child link="motor2" />
    </joint>

---------------------- Άξονες προπέλας ----------------------
Παράμετροι προπέλας
<xacro:property name="axle_h" value="${0.02*body_s}" />
<xacro:property name="axle_r" value="${0.01*body_s}" />
<xacro:property name="axle_m" value="${0.005*mass}" />

    <link name="axle1">
        <visual>
            <origin xyz="${axle_h/2} 0 0" rpy="0 ${pi/2} 0"/>
            <geometry>
                <cylinder radius="${axle_r}" length="${axle_h}" />
            </geometry>
        </visual>

        <collision>
            <origin xyz="${axle_h/2} 0 0" rpy="0 ${pi/2} 0"/>
            <geometry>
                <cylinder radius="${axle_r}" length="${axle_h}" />
            </geometry>
        </collision>

        <inertial>
            <origin xyz="${axle_h/2} 0 0" />
            <mass value="${axle_m}" />
            <inertia
          ixx="${1/12*axle_m*axle_h*axle_h+1/4*axle_m*axle_r*axle_r}"
          ixy="0" ixz="0" iyz="0"
          iyy="${1/12*axle_m*axle_h*axle_h+1/4*axle_m*axle_r*axle_r}"
           izz="${1/2*axle_m*axle_r*axle_r}" />
        </inertial>
    </link>

    <joint name="motor1_to_axle1" type="continuous">
```

```
        <origin xyz="${motor_x} 0 0" rpy="0 0 0" />
        <axis xyz="1 0 0" />
        <parent link="motor1" />
        <child link="axle1" />
    </joint>

    <link name="axle2">
        … όμοια με axle1 …
    </link>

    <joint name="motor2_to_axle2" type="continuous">
        <origin xyz="${motor_x} 0 0" rpy="0 0 0" />
        <axis xyz="1 0 0" />
        <parent link="motor2" />
        <child link="axle2" />
    </joint>

          -------------------- Προπέλες --------------------
Παράμετροι προπέλας 1
<xacro:property name="propeller_x" value="${0.26*body_s}" />
<xacro:property name="propeller_y" value="${0.04*body_s}" />
<xacro:property name="propeller_z" value="${0.01*body_s}" />
<xacro:property name="propeller_m" value="${0.015*mass}" />

     <link name="prop1">
        <visual>
        <origin xyz="0 -0.03 0" rpy="0 0 0" />
            <geometry>
            Αρχείο COLLADA για το σχέδιο της προπέλας
               <mesh filename="package://last_letter_2/config/param/
                             meshes/propeller.dae"
                  scale="${0.2*body_s} ${0.2*body_s} ${0.2*body_s}"/>
            </geometry>
        </visual>

        <collision>
            <geometry>
                <box size="${propeller_x} ${propeller_y}
                          ${propeller_z}" />
            </geometry>
        </collision>

        <inertial>
            <mass value="${propeller_m}" />
           <inertia     ixx="${1/12*propeller_m*(propeller_y*
                        propeller_y+propeller_z*propeller_z)}"
                        ixy="0" ixz="0" iyz="0"
                        iyy="${1/12*propeller_m*(propeller_x*propelle
                        r_x+propeller_z*propeller_z)}"
```

```
                        izz="${1/12*propeller_m*(propeller_x*propelle
                        r_x+propeller_y*propeller_y)}" />
        </inertial>
    </link>

    <joint name="axle1_to_prop1" type="fixed">
        <origin xyz="${axle_h} 0 0" rpy="0 ${pi/2} 0" />
        <parent link="axle1" />
        <child link="prop1" />
    </joint>

    <link name="prop2">
        … όμοια με prop1 …
    </link>

    <joint name="axle2_to_prop2" type="fixed">
        <origin xyz="${axle_h} 0 0" rpy="0 ${pi/2} 0" />
        <parent link="axle2" />
        <child link="prop2" />
    </joint>

---------------------- Μπροστινοί τροχοί ----------------------

Παράμετροι βάσης τροχών
<xacro:property name="wheel_base_x" value="${0.01*body_s}" />
<xacro:property name="wheel_base_y" value="${0.5*body_s}" />
<xacro:property name="wheel_base_z" value="${0.01*body_s}" />
<xacro:property name="wheel_base_m" value="${0.05*mass}" />

    <link name="wheel_base">
        <visual>
            <geometry>
                <box size="${wheel_base_x} ${wheel_base_y}
                          ${wheel_base_z}" />
            </geometry>
        </visual>

        <collision>
            <geometry>
                <box size="${wheel_base_x} ${wheel_base_y}
                          ${wheel_base_z}" />
            </geometry>
        </collision>

        <inertial>
            <mass value="${wheel_base_m}" />
            <inertia    ixx="${1/12*wheel_base_m*(wheel_base_y*
                        wheel_base_y+wheel_base_z*wheel_base_z)}"
                        ixy="0" ixz="0" iyz="0"
```

```
                        iyy="${1/12*wheel_base_m*(wheel_base_x*wheel_
                        base_x+wheel_base_z*wheel_base_z)}"
                        izz="${1/12*wheel_base_m*(wheel_base_x*wheel_
                        base_x+wheel_base_y*propeller_y)}" />
        </inertial>
    </link>

    <joint name=" body_FLU_to_wheel_base" type="fixed">
        <origin xyz="${0.3*body_h} 0 -${body_r}" rpy="0 0 0" />
        <parent link="body_FLU" />
        <child link="wheel_base" />
    </joint>
```

Παράμετροι αριστερού τροχού

```
<xacro:property name="left_wheel_r" value="${0.06*body_s}" />
<xacro:property name="left_wheel_h" value="${0.02*body_s}" />
<xacro:property name="left_wheel_m" value="${0.01*mass}" />
    <link name="left_wheel">
        <visual>
            <origin xyz="0 0 ${left_wheel_h/2}" />
            <geometry>
                <cylinder radius="${left_wheel_r}"
                     length="${left_wheel_h}" />
            </geometry>
        </visual>

        <collision>
            <origin xyz="0 0 ${left_wheel_h/2}" />
            <geometry>
                <cylinder radius="${left_wheel_r}"
                     length="${left_wheel_h}" />
            </geometry>
        </collision>

        <inertial>
            <origin xyz="0 0 ${left_wheel_h/2}" />
            <mass value="${left_wheel_m}" />
           <inertia     ixx="${1/12*left_wheel_m*left_wheel_h*
                        left_wheel_h+1/4*left_wheel_m*left_wheel_r*
                        left_wheel_r}" ixy="0" ixz="0" iyz="0"
                        iyy="${1/12*left_wheel_m*left_wheel_h*left_wh
                        eel_h+1/4*left_wheel_m*left_wheel_r*left_whee
                        l_r}"
                        izz="${1/2*left_wheel_m*left_wheel_r*left_whe
                        el_r}" />
        </inertial>
    </link>



    <joint name="wheel_base_to_left_wheel" type="continuous">
```

```
        <origin xyz="0 -${wheel_base_y/2} 0" rpy="${pi/2} 0 0" />
        <parent link="wheel_base" />
        <child link="left_wheel" />
        <axis xyz=" 0 0 1" />
    </joint>
```

Παράμετροι δεξιού τροχού

```
<xacro:property name="right_wheel_r" value="${left_wheel_r}" />
<xacro:property name="right_wheel_h" value="${left_wheel_h}" />
<xacro:property name="right_wheel_m" value="${left_wheel_m}" />

    <link name="right_wheel">
      … όμοια με left_wheel …
    </link>

    <joint name="wheel_base_to_right_wheel" type="continuous">
        <origin xyz="0 ${wheel_base_y/2} 0" rpy="${pi/2} 0 0" />
        <parent link="wheel_base" />
        <child link="right_wheel" />
        <axis xyz=" 0 0 1" />
    </joint>
```

---------- Πίσω σφαιρικός τροχός ----------

Παράμετροι πίσω τροχού

```
<xacro:property name="rear_wheel_r" value="${body_r*0.2}" />
<xacro:property name="rear_wheel_m" value="${body_m/2}" />

    <link name="rear_wheel">
        <visual>
            <geometry>
                <sphere radius="${rear_wheel_r}" />
            </geometry>
        </visual>

        <collision>
            <geometry>
                <sphere radius="${rear_wheel_r}" />
            </geometry>
        </collision>

        <inertial>
            <mass value="${rear_wheel_m}" />
            <inertia
                  ixx="${2/5*rear_wheel_m*rear_wheel_r*rear_wheel_r}"
                  ixy="0" ixz="0" iyz="0"
                  iyy="${2/5*rear_wheel_m*rear_wheel_r*rear_wheel_r}"
                  izz="${2/5*rear_wheel_m*rear_wheel_r*rear_wheel_r}"
                  />
        </inertial>
```

```
    </link>

    <joint name="body_FLU_to_rear_wheel" type="continuous">
        <origin xyz="-${0.4*body_h} 0 -${body_r}" />
        <axis xyz="0 1 0"/>
        <parent link="body_FLU" />
        <child link="rear_wheel" />
    </joint>


Δήλωση βιβλιοθηκών Gazebo
    <gazebo>
        Model plugin
        <plugin name="model_plugin" filename="libmodel_plugin.so"/>
        Joint state publisher
        <plugin name="joint_state_publisher"
                 filename="libgazebo_ros_joint_state_publisher.so">
        Δήλωση παραμέτρων του joint_state_publisher
             <jointName>
        Δήλωση joint για τα οποία θα δημοσιεύει την κατάστασή τους ο
                  joint state publisher. Δηλώνονται όλα τα revolute
                       body_FLU_to_airfoil1,
                       body_FLU_to_airfoil2,
                       body_FLU_to_airfoil3,
                       airfoil1_to_motor1,
                       airfoil1_to_motor2,
                       motor1_to_axle1,
                       motor2_to_axle2,
                       wheel_base_to_left_wheel,
                       wheel_base_to_right_wheel,
                       body_FLU_to_rear_wheel
            </jointName>
            <robotNamespace>last_letter_2</robotNamespace>
            <updateRate>0</updateRate>
        </plugin>
    </gazebo>

</robot>
```

Παράρτημα Ε

Οδηγίες εγκατάστασης

Ε.1 Απαιτήσεις σε Hardware

Επειδή ο Last_letter_2 βασίζεται στο ROS και στο Gazebo, οι απαιτήσεις του προσομοιωτή για υλικό είναι αυτές των παραπάνω εργαλείων. Το Documentation του ROS προτείνει συγκεκριμένη έκδοση του Gazebo για κάθε έκδοση του Ros που χρησιμοποιείται. Για την συγκεκριμένη εφαρμογή, χρησιμοποιήθηκαν τα πιο πρόσφατα πακέτα, δηλαδή Ros melodic με Gazebo 9.x series.

Έτσι οι απαιτήσεις για hardware του Last letter 2 πηγάζουν από τα αντίστοιχα πακέτα και είναι τα εξής:

Operating System:	Linux Ubuntu 18.04 (Bionic)
Memory:	1Gb or more
CPU:	at least Intel i5
Disk space:	at least 500MB free space
Languages:	C++14, Python 2.7 (Python >= 3.5 not required, but testing against it is recommended)
Cmake:	3.10.2
Gazebo:	9.x series
Ros:	Melodic
Boost:	1.65.1

E.2 Οδηγός εγκατάστασης

Ο προσομοιωτής έχει αναπτυχθεί σε Linux 18.04, ROS melodic και Gazebo 9.x series. Οπότε ο οδηγός εγκατάστασης αφορά τις συγκεκριμένες εκδόσεις των λογισμικών.

E.2.1 Εγκατάσταση ROS – Gazebo

Σύμφωνα με τις οδηγείς του site http://wiki.ros.org/melodic/Installation/Ubuntu, σε ένα terminal γράψτε τις παρακάτω σειρές.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-
key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

sudo apt update

sudo apt install ros-melodic-desktop-full

sudo rosdep init

rosdep update

echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc

source ~/.bashrc

sudo apt install python-rosinstall python-rosinstall-generator
python-wstool build-essential
```

Με τις παραπάνω εντολές κατεβάζεται και το Gazebo μαζί. Οπότε δεν χρειάζονται επιπλέον γραμμές για το Gazebo.

Προσθέστε επιπλέον τα παρακάτω πακέτα.

```
sudo apt install python-catkin-tools


sudo apt install ros-melodic-joy
```

E.2.2 Δημιουργία ROS workspace

Σύμφωνα με τις οδηγίες της σελίδας http://wiki.ros.org/catkin/Tutorials/create_a_workspace προσθέστε τις παρακάτω εντολές, για την δημιουργία ενός workspace όπου θα βρίσκεται ο κώδικας του προσομοιωτή, θα τροποποιείται και θα γίνεται build.

```
mkdir -p ~/catkin_ws/src
```

E.2.3 Clone Last letter 2 κώδικα

```
cd ~/catkin_ws/src
```

```
git clone https://github.com/MiliStefanos/last_letter_2.git
cd ~/catkin_ws
catkin build
```

Στην συνέχεια πρέπει να γίνει source το παρακάτω αρχείο, είτε κάνοντας source στο τρέχον μόνο terminal γράφοντας:

```
source ~/catkin_ws/devel/setup.bash
```

είτε προσθέτοντας την παρακάτω γραμμή στο .bashrc αρχείο ώστε να φορτώνει αυτόματα σε κάθε νέο terminal.

```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

E.2.4 Οδηγίες εκτέλεσης

Η προσομοίωση ξεκινάει τρέχοντας ένα αρχείο, το **simulation.launch**.

Σε ένα terminal (το οποίο βλέπει το workspace του Last letter 2) τρέξτε την εντολή

```
roslaunch last_letter_2 simulation.launch
```

αυτή ξεκινάει να τρέχει τον μοναδικό αρχείο launch του προσομοιωτή, το **simulation.launch** το οποίο σηκώνει όλα τα απαραίτητα node.

Ο χρήστης επιλέγει τον τύπο του μοντέλου που θέλει να προσομοιώσει φορτώνοντας στην μεταβλητή "uav_name" το ακριβές όνομα του μοντέλου, μέσα στο αρχείο simulation.launch. Όλα τα ονόματα είναι γραμμένα σε σχόλια για διευκόλυνση.

Για real time tuning χρειάζεται να σηκωθεί το node rqt_reconfigure. Για να γίνει αυτό αρκεί να γίνει uncomment η σχετική σειρά προς το τέλος του αρχείου. Οι controllers των μοντέλων επιλέγονται αυτόματα.

Εάν όλα έχουν πάει καλά, ο προσομοιωτής ξεκινάει.

Επίλογος

Σύνοψη και συμπεράσματα

Έχοντας αναλυθεί κάθε πτυχή του Last letter 2, γίνεται εύκολα αντιληπτό ότι αποτελεί ένα αρκετά ισχυρό εργαλείο για γρήγορη, εύκολη και αξιόπιστη δοκιμή αλγορίθμων ελέγχου σε ιπτάμενα πειραματικά μοντέλα. Παράλληλα δίνει την δυνατότητα προσαρμογής και βελτίωσης με την προσθήκη επιπλέον κώδικα από μελλοντικούς χρήστες. Τέλος, σε συνεργασία με τις επιπλέον δυνατότητα που δίνει το ROS (rqt_plot, dynamic_reconfigure), βελτιώνει την εμπειρία του χρήστη και τον έλεγχο της προσομοίωσης.

Μελλοντικές επεκτάσεις

Καθώς η συγκεκριμένη εφαρμογή έγινε στα πλαίσια διπλωματικής εργασίας, ο χρόνος ήταν περιορισμένος οπότε δημιουργήθηκαν όλα αυτά που καθιστούν τον Last letter 2 αυτόνομο, λειτουργικό και αποτελεσματικό εργαλείο προσομοίωσης. Αυτό σημαίνει πώς υπάρχουν περιθώρια βελτίωσης, όμως αφήνονται στην κρίση μελλοντικών χρηστών.

Ο Last letter 2, έχοντας δημιουργηθεί με σκοπό να τροποποιείται εύκολα, μπορεί κάλλιστα να βελτιωθεί και να επεκτείνει τις δυνατότητές του. Αρχικά θα ήταν χρήσιμο να κατασκευαστούν τα κατάλληλα plugin για την επικοινωνία του με άλλους προσομοιωτές ρομπότ σαν το Gazebo, ώστε να εκμεταλλευτεί τα πλεονεκτήματα που προσφέρουν. Επιπλέον θα ήταν αρκετά χρήσιμο να προστεθούν στην βιβλιοθήκη του Last letter 2 μοντέλα που υπάρχουν στην αγορά ώστε οι χρήστες που βασίζουν την έρευνά τους πάνω σε αυτά να μην χρειάζεται να τα κατασκευάσουν. Το ίδιο ισχύει και με τους αλγορίθμους ελέγχου. Θα βοηθούσε αρκετά να υπήρχαν διαθέσιμοι αλγόριθμοι ελέγχου ελεύθερου λογισμικού που flight controller που πωλούνται στην αγορά. Τέλος, μια άλλη βελτίωση του Last letter 2 είναι η προσθήκη περισσότερων επιλογών μοντέλων δυναμικής τόσο για τον υπολογισμό των αεροδυναμικών δυνάμεων όσο και για τις δυνάμεις των κινητήρων.

Βιβλιογραφία

[1] M. T. Beard R., Small Unmanned Aircraft Theory and Practice, Princeton University Press, 2012.

[2] G. Z. -. Papaliakos, «github.com,» 28 July 2016. [Ηλεκτρονικό]. Available: https://github.com/Georacer/uav-modeling/blob/master/preamble.pdf.

[3] S. Z. S., «Thermodynamics and Propulsion,» [Ηλεκτρονικό]. Available: https://web.mit.edu/16.unified/www/FALL/thermodynamics/notes/node86.html.

[4] «ROS wiki documentation,» [Ηλεκτρονικό]. Available: http://wiki.ros.org/.

[5] «Gazebo,» [Ηλεκτρονικό]. Available: http://gazebosim.org/.

[6] «Gazebo answers,» [Ηλεκτρονικό]. Available: http://answers.gazebosim.org/questions/.

[7] «ROS answers,» [Ηλεκτρονικό]. Available: https://answers.ros.org/questions/.