



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΣΕ ΠΕΡΙΒΑΛΛΟΝ ROS ΓΙΑ ΕΛΕΓΧΟ ΕΜΠΕΔΗΣΗΣ ΣΕ ΔΙΑΣΤΗΜΙΚΟ ΕΞΟΜΟΙΩΤΗ

Μελέτη και υλοποίηση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

ΝΙΚΗΦΟΡΟΥ ΤΣΟΥΛΙΑ

Επιβλέπων: Κωνσταντίνος Τζαφέστας
Αναπληρωτής Καθηγητής, Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2025



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΣΕ ΠΕΡΙΒΑΛΛΟΝ ROS ΓΙΑ ΕΛΕΓΧΟ ΕΜΠΕΔΗΣΗΣ ΣΕ ΔΙΑΣΤΗΜΙΚΟ ΕΞΟΜΟΙΩΤΗ

Μελέτη και υλοποίηση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΝΙΚΗΦΟΡΟΥ ΤΣΟΥΛΙΑ

Επιβλέπων: Κωνσταντίνος Τζαφέστας
Αναπληρωτής Καθηγητής, Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 26 Φεβρουαρίου 2025.

(Υπογραφή)

(Υπογραφή)

(Υπογραφή)

.....
Κωνσταντίνος Τζαφέστας
Αναπληρωτής Καθηγητής, Ε.Μ.Π.

.....
Ιωάννης Γκόνος
Καθηγητής, Ε.Μ.Π.

.....
Ευάγγελος Παπαδόπουλος
Καθηγητής, Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2025



Copyright © – All rights reserved. Με την επιφύλαξη παντός δικαιώματος.
Νικηφόρος Τσούλιας, 2025.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα.

Το περιεχόμενο αυτής της εργασίας δεν απηχεί απαραίτητα τις απόψεις του Τμήματος, του Επιβλέποντα, ή της επιτροπής που την ενέκρινε.

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
Νικηφόρος Τσούλιας

22 Φεβρουαρίου 2025

Περίληψη

Η παρούσα διπλωματική εργασία αφορά την εξομοίωση μιας In-Orbit-Service αποστολής, όπως το capturing ενός δορυφόρου δίχως καύσιμα, στον Επίπεδο Διαστημικό Εξομοιωτή του Εργαστηρίου Αυτομάτου Ελέγχου του ΕΜΠ. Ο στόχος είναι η ανάπτυξη ενός ελεγκτή εμπέδησης σε C++ μέσω ROS, αρχικά στον προσομοιωτή Gazebo και στη συνέχεια στον εξομοιωτή.

Ο εξομοιωτής περιλαμβάνει μια τράπεζα γρανίτη, το Motion Capture σύστημα της Vicon, το ρομπότ "Κηφέας" και τον στόχο. Ο ελεγκτής βασίζεται στην επενέργηση των τριών αρθρώσεων του βραχίονα και του σφονδύλου αντίδρασης, ώστε το Τελικό Σημείο Δράσης (ΤΣΔ) να πλησιάσει και να πιάσει τον στόχο, διατηρώντας σταθερό τον προσανατολισμό της βάσης.

Αναπτύχθηκε λογισμικό ελέγχου ακολουθώντας το design pattern ενός native ελεγκτή σε C++, με ROS Wrapper, επιτρέποντας χρήση τόσο στην προσομοίωση όσο και στο πραγματικό ρομπότ. Ο "Κηφέας" μοντελοποιήθηκε στο Gazebo για δοκιμή του ελεγκτή, ο οποίος πέτυχε και στις δύο φάσεις (free space και contact).

Έπειτα, ο εξομοιωτής του εργαστηρίου επανήλθε σε λειτουργία, με αντικατάσταση ελαττωματικού εξοπλισμού και επίλυση καλωδιώσεων. Ο νέος gripper ενσωματώθηκε στο ΤΣΔ και πραγματοποιήθηκε πετυχημένο πείραμα ελέγχου για την πρώτη φάση (free space phase). Η προσομοίωση δείχνει ότι το σύστημα μπορεί να πετύχει αντίστοιχα αποτελέσματα στη δεύτερη φάση μετά την αναβάθμιση του εξοπλισμού.

Λέξεις Κλειδιά

ROS, C++, έλεγχος εμπέδησης, επίπεδος διαστημικός εξομοιωτής, Gazebo, Κηφέας

Abstract

This thesis concerns the emulation of an In-Orbit-Service mission, such as capturing a fuel-depleted satellite, in the Planar Space Robot Emulator of the Control Systems Lab at NTUA. The objective is to develop an impedance controller in C++ using ROS, initially in the Gazebo simulator and subsequently in the physical emulator.

The emulator consists of a granite table, the Vicon Motion Capture system, the "Cepheus" robot, and the target. The controller is based on the actuation of the three arm joints and the reaction wheel, enabling the end effector to approach and capture the target while maintaining the robot base's orientation.

A control software was developed following the design pattern of a native controller in C++, with a ROS Wrapper, allowing seamless integration both in simulation and in the real robot. "Cepheus" was modeled in Gazebo to test the controller, which successfully performed in both phases (free space and contact).

Subsequently, the laboratory's simulator was restored to operational status by replacing faulty equipment and resolving wiring issues. The newly developed gripper was integrated into the end effector, and a successful impedance control experiment was conducted for the first phase (free space phase). The simulation results indicate that the system can achieve similar performance in the second phase after equipment upgrades.

Keywords

ROS, C++, impedance control, space robot emulator, Gazebo, Cepheus

Ευχαριστίες

Θα ήθελα αρχικά να ευχαριστήσω τον καθηγητή κ.Ευάγγελο Παπαδόπουλο και τον κ.Κωνσταντίνο Τζαφέστα για την επίβλεψη αυτής της διπλωματικής εργασίας και για την ευκαιρία που μου δώσανε να την εκπονήσω στο εργαστήριο Συστημάτων Αυτομάτου Ελέγχου. Επίσης ευχαριστώ ιδιαίτερα τον Δρ.Κώστα Νάνο για την καθοδήγησή του και την εξαιρετική συνεργασία που είχαμε. Επιπλέον θα ήθελα να ευχαριστήσω όλα τα παιδιά από το εργαστήριο, ιδιαίτερα τον Λευτέρη Καντούνια και τον Νίκο Δούναβη, οι οποίοι μου παρείχαν καθοριστική βοήθεια κατά τη διεξαγωγή των πειραμάτων. Τέλος θα ήθελα να ευχαριστήσω τους γονείς μου για την στήριξη που μου παρείχαν όλα αυτά τα χρόνια κι εξακολουθούν να μου παρέχουν, καθώς επίσης και τους φίλους μου που ήταν, είναι και θα είναι δίπλα μου.

Αθήνα, Φεβρουάριος 2025

Νικηφόρος Τσούλιας

Περιεχόμενα

Περίληψη	1
Abstract	3
Ευχαριστίες	4
1 Εισαγωγή	11
1.1 Σκοπός Εργασίας	11
1.2 Βιβλιογραφική Ανασκόπηση	14
1.3 Δομή Εργασίας	16
2 Στοιχεία Θεωρίας Ελέγχου Εμπέδησης	17
2.1 Εισαγωγή	17
2.2 Έλεγχος Εμπέδησης	17
2.2.1 Κεντρική Ιδέα - Μονοδιάστατο Παράδειγμα	17
2.2.2 Έλεγχος Εμπέδησης Επίπεδου Διαστημικού Εξομοιωτή	20
3 Το ROS Framework	26
3.1 Εισαγωγή	26
3.2 Βασικά στοιχεία του ROS	26
3.2.1 ROS package	26
3.2.2 ROS node	27
3.2.3 Μηχανισμοί επικοινωνίας μεταξύ των nodes	28
3.2.4 Το εργαλείο RQT	30
3.2.5 ROS Control	32
3.2.6 Command line ROS commands	34
3.2.7 ROS Network	35
3.2.8 ROS Time και ROS Rate	36
3.3 Το ROS του Κηφέα	37
3.3.1 Το προϋπάρχον λογισμικό	37
3.3.2 Η προσθήκη για impedance control	38
4 Το περιβάλλον προσομοίωσης Gazebo	44
4.1 Εισαγωγή	44
4.2 Ο ODE physics engine	44
4.3 Διασύνδεση Gazebo με ROS	45

4.4 Η προσομοίωση του Κηφέα	47
4.5 Πείραμα στον προσομοιωτή	48
5 Πείραμα στον διαστημικό εξομοιωτή του εργαστηρίου	56
5.1 Εισαγωγή	56
5.2 Πείραμα του νόμου ελέγχου	60
5.3 Συμπεράσματα	64
6 Συμπεράσματα και Μελλοντική Εργασία	65
6.1 Συμπεράσματα	65
6.2 Μελλοντική Εργασία	65
Βιβλιογραφία	70
Παραρτήματα	71
A' Hardware Manuals	72
B' Γενικές οδηγίες Κηφέα	88
B'.1 Οδηγίες για γέμισμα και αποθήκευση φιάλης CO ₂	88
B'.2 Οδηγίες διεξαγωγής πειράματος	88
Γ' Κώδικες	92
Γ'.1 robot_functions.h	92
Γ'.2 robot_variables.h	116

Κατάλογος Σχημάτων

1.1	Απεικόνιση των διαστημικών σκουπιδιών γύρω από τη Γη σύμφωνα με τη NASA.	12
1.2	Προτεινόμενες in-orbit υπηρεσίες της ESA.	13
1.3	Free Flyers, Stanford University.	15
1.4	Στιγμιότυπο από πείραμα για το πρόγραμμα ASSIST, στον διαστημικό εξομοιωτή του Εργαστηρίου Αυτομάτου Ελέγχου, Ε.Μ.Π.	15
2.1	(α) Η κίνηση του κυνηγού στον ελεύθερο χώρο, (β) Η αρχή της επαφής με το στόχο, (γ) Ηφάση της επαφής.	18
2.2	Επίπεδος διαστημικός εξομοιωτής, κυνηγός και στόχος.	21
3.1	Η τυπική δομή ενός ROS workspace.	27
3.2	Σχεδιάγραμμα επικοινωνίας μεταξύ client και server της actionlib.	30
3.3	Παράδειγμα χρήσης του <code>rqt_graph</code> .	31
3.4	Παράδειγμα χρήσης του <code>rqt_plot</code> .	32
3.5	Η δομή του ROS Control.	33
3.6	Δυνατές καταστάσεις ενός ελεγκτή.	34
3.7	Η λειτουργία του ROS Rate.	36
3.8	Τα ηλεκτρονικά στοιχεία του Κηφέα.	37
3.9	Δομή του υλοποιημένου ελεγκτή.	39
3.10	Λογική ελεγκτή κατά το πείραμα.	41
3.11	Επικοινωνία των βασικών αρχείων κατά το πείραμα στον εξομοιωτή.	42
3.12	Επικοινωνία των βασικών αρχείων κατά το πείραμα στη προσομοίωση.	43
4.1	Τα περιεχόμενα του <code>gazebo_ros_pkgs</code> .	45
4.2	Προσομοίωση του Κηφέα στο Gazebo.	48
4.3	Αρχική θέση Κηφέα και στόχου στο προσομοιωμένο πείραμα ελέγχου εμπέδησης.	49
4.4	Στιγμιότυπο κατά τη διάρκεια της επαφής ΤΣΔ και στόχου, στο προσομοιωμένο πείραμα ελέγχου μηχανικής εμπέδησης.	50
4.5	Δεύτερο στιγμιότυπο κατά τη διάρκεια της διατηρημένης επαφής ΤΣΔ και στόχου, στο προσομοιωμένο πείραμα ελέγχου εμπέδησης.	51
4.6	Γραφικές παραστάσεις των x , y , θ και θ_0 στο προσομοιωμένο πείραμα ελέγχου εμπέδησης.	52
4.7	Γραφικές παραστάσεις των $\dot{x}$, $\dot{y}$, $\dot{\theta}$ και $\dot{\theta}_0$ στο προσομοιωμένο πείραμα ελέγχου εμπέδησης.	52

4.8	Γραφικές παραστάσεις των ροπών στις αρθρώσεις και στη βάση στο προσμοιωμένο πείραμα ελέγχου μηχανικής εμπέδησης.	53
4.9	Μεγεθυμένα διαγράμματα των ροπών στις αρθρώσεις και στη βάση στο προσμοιωμένο πείραμα ελέγχου μηχανικής εμπέδησης.	54
4.10	Η εξωτερική δύναμη f_{ext} που ασκείται στο ΤΣΔ κατά τον άξονα x του τοπικού frame στο προσμοιωμένο πείραμα ελέγχου μηχανικής εμπέδησης.	55
4.11	Εποπτεία των ενεργών ROS nodes και topics μέσω του εργαλείου RQT, κατά το προσμοιωμένο πείραμα ελέγχου μηχανικής εμπέδησης.	55
5.1	Η μορφή και η λειτουργία του αερόδρανου, με τη χρήση κατάλληλου αερίου υπό πίεση.	56
5.2	Ο Κηφέας κατά τη πειραματική διαδικασία.	57
5.3	Ο force-torque sensor, ο gripper και το παθητικό interface.	58
5.4	Οι κάμερες του συστήματος Vicon περιμετρικά της τράπεζας γρανίτη και οι markers.	59
5.5	Το gui της Vicon και η δημιουργία καταλλήλων frames.	59
5.6	Ο στόχος κατά τη πειραματική διαδικασία.	60
5.7	a) Η αρχική διάταξη του πειράματος. b) Η διάταξη πριν ξεκινήσει το softgrip.	61
5.8	a) Η διάταξη πριν ξεκινήσει το hardgrip (μετά το softgrip). b) Η τελική διάταξη μετά το hardgrip.	62
5.9	Γραφικές παραστάσεις των x , y , θ και θ_0 στο πείραμα ελέγχου εμπέδησης του εργαστηρίου.	63
5.10	Γραφικές παραστάσεις των $\dot{x}$, $\dot{y}$, $\dot{\theta}$ και $\dot{\theta}_0$ στο πείραμα ελέγχου εμπέδησης του εργαστηρίου.	63
5.11	Οι ροπές των αρθρώσεων και του reaction wheel στο πείραμα ελέγχου εμπέδησης του εργαστηρίου.	64
A'.1	Τα χαρακτηριστικά και οι ιδιότητες του κινητήρα που χρησιμοποιείται για την επενέργεια του βραχίονα [1].	73
A'.2	Τα χαρακτηριστικά και οι ιδιότητες του μειωτήρα που συνδέεται με τον κινητήρα και αποτελεί την έξοδο της ισχύος του συστήματος επενέργειας [1]	74
A'.3	Η παλμογεννήτρια (encoder) που συνδέεται με τον κινητήρα και μετρά τη γωνία περιστροφής της ατράκτου του [1]	75
A'.4	Εικόνα του κάθε driver για κάθε κινητήρα του βραχίονα. Είναι το μοντέλο ESCON 36/2 DC.	76
A'.5	Χαρακτηριστικά των drivers των κινητήρων του βραχίονα.	77
A'.6	Εικόνα του driver για τον κινητήρα του Reaction Wheel. Είναι το μοντέλο ESCON 50/5.	78
A'.7	Χαρακτηριστικά του driver για τον κινητήρα του Reaction Wheel.	79
A'.8	Πλακέτα ρύθμισης τάσης που είναι εγκατεστημένη στο stack του PC104 και τροφοδοτεί το Arduino shield με 12V [2].	80

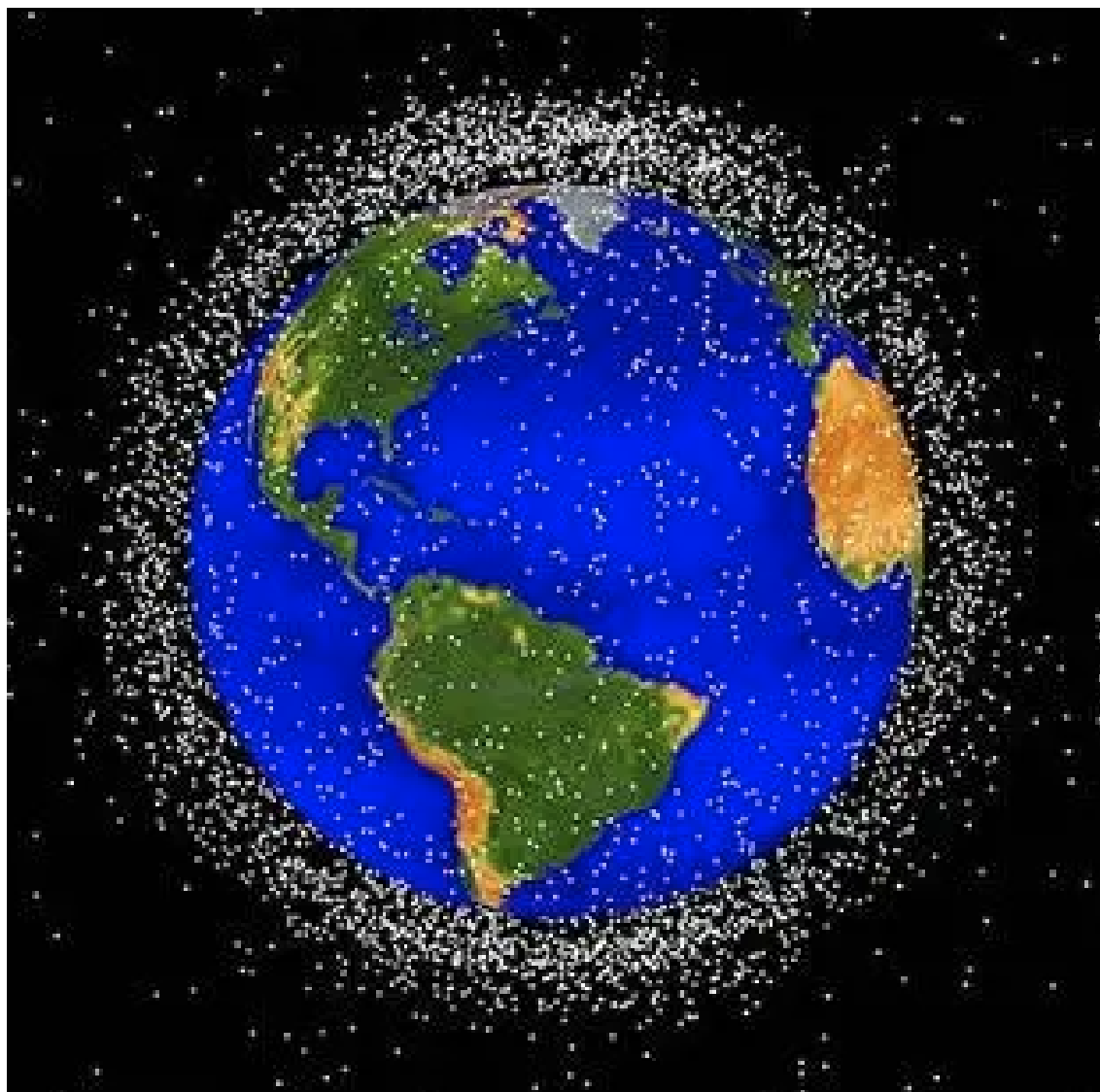
A'.9	Πλακέτα ψηφιακών εισόδων-εξόδων που είναι εγκατεστημένη στο stack του PC104. Είναι υπεύθυνη για την αποστολή σημάτων-εντολών στα shields των κινητήρων και λαμβάνει τις μετρήσεις από τις παλμογεννήτριες των κινητήρων [3].	81
A'.10	Ο ρυθμιστής υψηλής πίεσης που τοποθετείται αμέσως μετά τις φιάλες διοξειδίου για να ρίξει την πίεση των 50bar.	82
A'.11	Τα χαρακτηριστικά του ρυθμιστή πίεσεως που χρησιμοποιείται για τον έλεγχο του αερίου στα αεροέδρανα του ρομπότ.	83
A'.12	Τα χαρακτηριστικά του ρυθμιστή πίεσεως που χρησιμοποιείται στο νέο πνευματικό για τον έλεγχο του αερίου στα αεροέδρανα της αρπάγης.	84
A'.13	Το φίλτρο που χρησιμοποιείται στο πνευματικό κύκλωμα του ρομπότ και χρησιμοποιείται για την απομάκρυνση νερού από το κύκλωμα. Ο τύπος του είναι LF-M7-D-MICRO B + 1/8"-M7 adapter [4].	85
A'.14	Τα χαρακτηριστικά των αεροεδράνων της εταιρείας NEWWAY-AIRBEARINGS που χρησιμοποιούνται στο διαστημικό σύστημα με διάμετρο 25mm [5]. . . .	86
A'.15	Τα χαρακτηριστικά των αεροεδράνων της εταιρείας NEWWAY-AIRBEARINGS που χρησιμοποιούνται στο διαστημικό σύστημα με διάμετρο 40mm [5]. . . .	87

Κεφάλαιο **1**

Εισαγωγή

1.1 Σκοπός Εργασίας

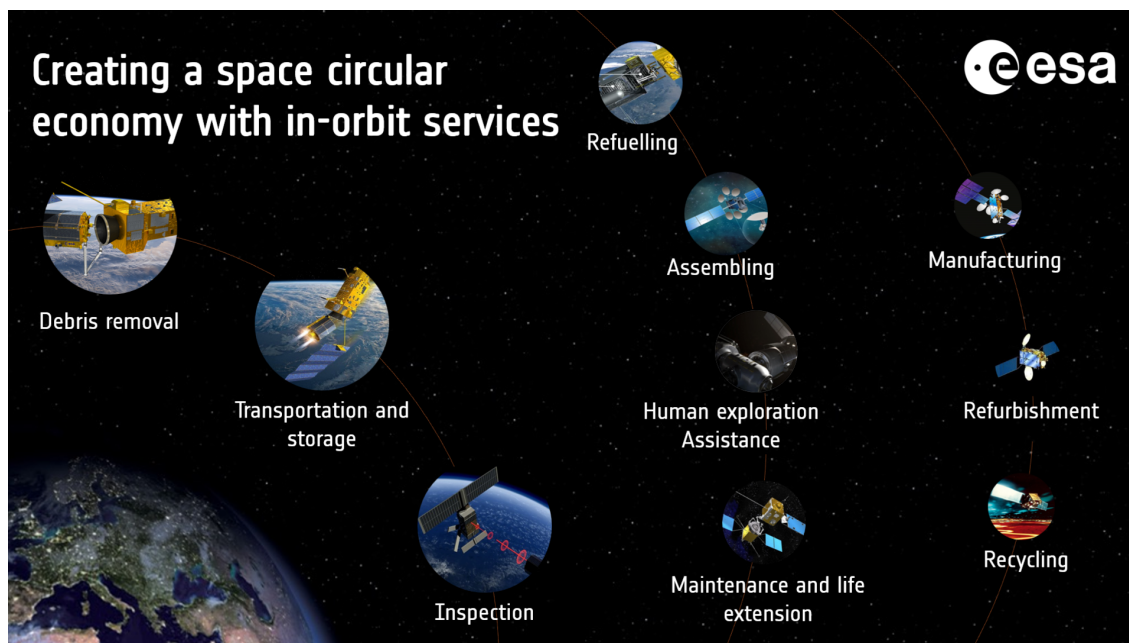
Τα τελευταία χρόνια η προσοχή της επιστημονικής κοινότητας έχει στραφεί στην ανάπτυξη της διαστημικής τεχνολογίας, τόσο με την έρευνα στα επίγεια εργαστήρια όσο και με την πραγματοποίηση διαστημικών αποστολών, επανδρωμένων ή δορυφόρων. Ωστόσο, παράπλευρη συνέπεια των αποστολών αυτών είναι η δημιουργία του προβλήματος των διαστημικών σκουπιδιών. Με τον όρο διαστημικά σκουπίδια (space debris) μπορούμε να περιγράψουμε ποικίλα αντικείμενα τα οποία κινούνται ανεξέλεγκτα στο διάστημα και αποτελούν κίνδυνο για τις μελλοντικές αποστολές, όπως αποκομμένα μέρη αεροσκαφών, κομμάτια βογιόγας, μέχρι και δορυφόρους χωρίς καύσιμα. Η πλειονότητα αυτών των σκουπιδιών βρίσκονται σε χαμηλή τροχιά γύρω από τη Γη (Low Earth Orbit - LEO), σε υψόμετρα 800-1400 km. Παρακάτω βλέπουμε μια απεικόνιση της NASA η οποία τονίζει τον όγκο αυτού του προβλήματος [6]:



Σχήμα 1.1: Απεικόνιση των διαστημικών σκουπιδιών γύρω από τη Γη σύμφωνα με τη NASA.

Για την αντιμετώπιση αυτού του προβλήματος έχουν αναπτυχθεί νέες τεχνολογικές εφαρμογές, όπως το In-Orbit Servicing (IOS) και το Space Debris Removal (SDR). Το IOS περιλαμβάνει την αποστολή ρομποτικών συστημάτων για την εκτέλεση εργασιών όπως η επισκευή δορυφόρων, η μεταφορά καυσίμου ή η επανατοποθέτησή τους σε τροχιά, ενώ το SDR στοχεύει στην απομάκρυνση μη λειτουργικών δορυφόρων ή διαστημικών σκουπιδιών που θέτουν σε κίνδυνο ενεργούς δορυφόρους [7][8].

Πιο συγκεκριμένα, ο Ευρωπαϊκός Οργανισμός Διαστήματος (European Space Agency - ESA) από το 2021 έχει ξεκινήσει το πρόγραμμα Clean Space, το οποίο αποσκοπεί στην απομάκρυνση και στην πρόληψη διαστημικών σκουπιδιών. Στο πρόγραμμα αυτό υπάγεται μια γκάμα in-orbit υπηρεσιών, όπως είναι η αποστολή δορυφόρων με ειδικά σχεδιασμένο gripping interface έτσι ώστε να γίνεται in-orbit grasping και refueling του δορυφόρου από ένα δεύτερο διαστημικό ρομπότ-εξυπηρετητή [9][10].



Σχήμα 1.2: Προτεινόμενες in-orbit υπηρεσίες της ESA.

Ένας από τους βασικούς περιορισμούς κατά την ανάπτυξη αυτών των τεχνολογιών είναι η αδυναμία αναπαραγωγής των συνθηκών έλλειψης βαρύτητας στη Γη, για την πραγματοποίηση της δοκιμαστικής διαδικασίας με σκοπό την αναγνώριση και επίλυση πιθανών δυσκολιών, πριν χρησιμοποιηθούν πραγματικά στο διάστημα. Για την αντιμετώπιση αυτού του προβλήματος, έχουν αναπτυχθεί διαστημικοί εξομοιωτές στη Γη που εξομοιώνουν συνθήκες μηδενικής βαρύτητας, παρέχοντας ένα ελεγχόμενο περιβάλλον για τη δοκιμή και βελτίωση αυτών των τεχνολογιών.

Το **Εργαστήριο Αυτομάτου Ελέγχου του Εθνικού Μετσόβιου Πολυτεχνείου (Ε.Μ.Π.)** αποτελεί σημείο αναφοράς στην ανάπτυξη τεχνολογιών για την εξομοίωση συνθηκών διαστήματος. Η συνεισφορά του στην έρευνα διαστημικών εξομοιωτών είναι ιδιαίτερα σημαντική, με έργα που προωθούν τη μελέτη και την ανάπτυξη καινοτόμων συστημάτων για αποστολές στο διάστημα. Στη φαρέτρα του εργαστηρίου βρίσκονται 2 ενεργητικά ρομπότ, ο **Κηφέας** και η **Κασσιόπη**, τα οποία εκτελούν εργασίες σε συνθήκες διαστήματος κι ένα παθητικό ρομπότ-στόχος το οποίο έχει τη δυνατότητα να συλλέγει δεδομένα από αισθητήρες κατά τη διάρκεια των πειραμάτων. Η πειραματική διάταξη του εργαστηρίου περιγράφεται αναλυτικά στα επόμενα κεφάλαια, καθώς επίσης και η δομή του Κηφέα, με τον οποίο θα ασχοληθούμε στην παρούσα εργασία.

Η παρούσα εργασία κινείται στο ίδιο πνεύμα της πρωτοβουλίας Clean Space της ESA, καθώς σκοπός της είναι η πραγματοποίηση πειράματος στον διαστημικό εξομοιωτή του εργαστηρίου κατά το οποίο το ρομπότ Κηφέας επιχειρεί να πιάσει το ρομπότ-στόχο, ο οποίος σε ένα πραγματικό σενάριο θα ήταν ένας δορυφόρος χωρίς καύσιμα, χρησιμοποιώντας το νεοσύστατο gripping interface του εργαστηρίου και εφαρμόζοντας τον κατάλληλο νόμο ελέγχου μηχανικής εμπέδησης. Ουσιαστικά, σκοπός του νόμου εμπέδησης είναι, χρησιμοποιώντας μόνο τους κινητήρες του βραχίονα και το reaction wheel, να πλησιάσει η αρπάγη του Κηφέα τον στόχο και να διατηρήσει επαφή μαζί του για ένα εύλογο χρονικό διάστημα, έτσι ώστε

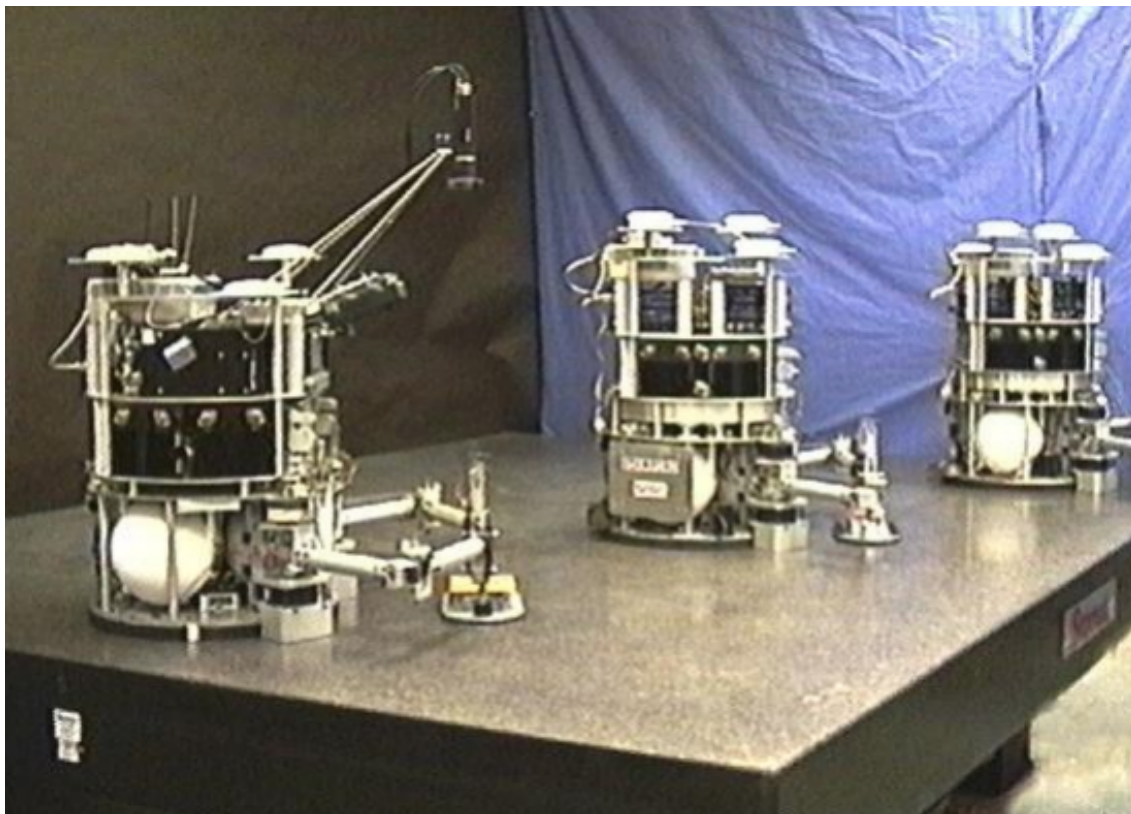
αξιοποιώντας το καινούργιο gripping interface να καταφέρει να τον πιάσει.

Για να επιτευχθεί ο παραπάνω σκοπός, η παρούσα εργασία χωρίζεται πρακτικά σε δύο στάδια: Σε πρώτο στάδιο στόχος είναι να υλοποιηθεί ο υπολογισμένος από το εργαστήριο νόμος ελέγχου μηχανικής εμπέδησης σε C++ και να δημιουργηθεί η κατάλληλη δομή έτσι ώστε ο ίδιος ελεγκτής να αλληλεπιδρά μέσω του ROS Framework τόσο με το προσομοιωμένο όσο και με το αληθινό ρομπότ. Γιαυτό, ταυτόχρονα σκοπός είναι να μοντελοποιηθεί ο Κηφέας με παραμετρικά γεωμετρικά και αδρανειακά μεγέθη στον προσομοιωτή Gazebo, ο οποίος υποστηρίζει επικοινωνία με ROS, έτσι ώστε να δοκιμαστεί ο νόμος ελέγχου πρώτα στη προσομοίωση.

Σε δεύτερο στάδιο σκοπός της εργασίας είναι η αναβάθμιση της πειραματικής διάταξης του εργαστηρίου, με την ανανέωση του συστήματος καμερών από το υπάρχον PhaseSpace στο καινούργιο σύστημα Vicon, με την προσθήκη και χρήση του αισθητήρα δύναμης-ροπής Botasys στον καρπό του βραχιόνα και την ενσωμάτωση του νεοσύστατου από το εργαστήριο gripping interface (στο τελικό σημείο δράσης του Κηφέα και στο σώμα του στόχου), χρησιμοποιώντας τον υλοποιημένο νόμο ελέγχου μηχανικής εμπέδησης, οποίος θα επικοινωνεί με το ήδη υπάρχον λογισμικό του Κηφέα μέσω ROS.

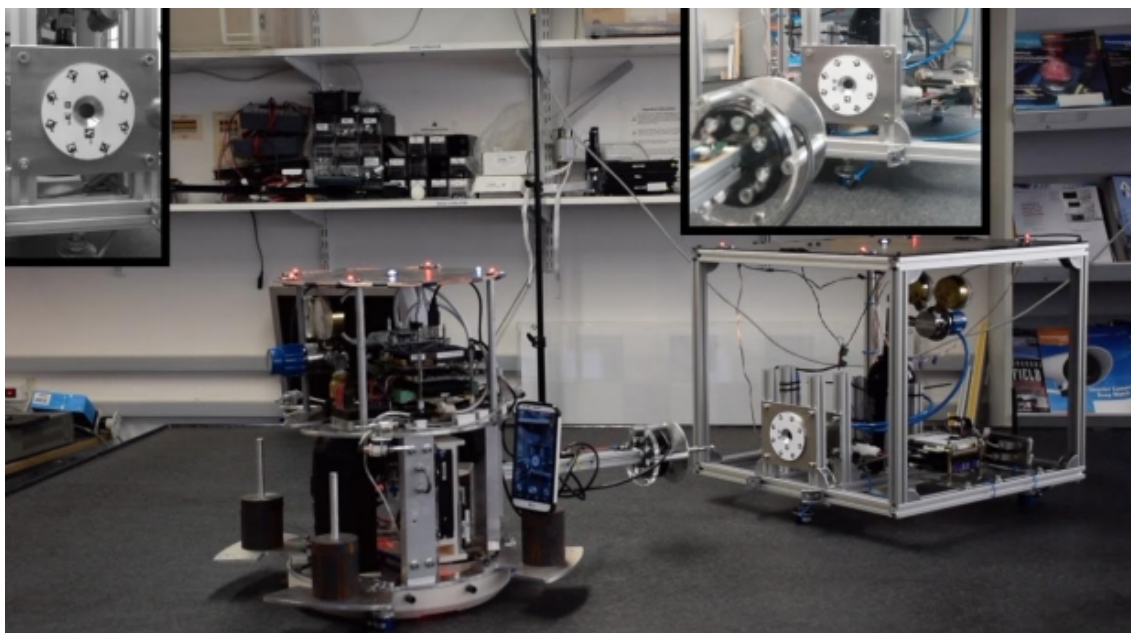
1.2 Βιβλιογραφική Ανασκόπηση

Ο πρώτος διαστημικός εξομοιωτής σε πανεπιστημιακό επίπεδο δημιουργήθηκε στο Aerospace Robotics Laboratory του πανεπιστημίου Stanford. Ο εξομοιωτής αυτός, γνωστός ως Free Flyers, χρησιμοποιούσε τρία ενεργητικά ρομπότ με προωθητήρες για την επίπεδη κίνηση και σφόνδυλο αντίδρασης για τον έλεγχο του προσανατολισμού. Τα ρομπότ, μαζί με το παθητικό, αιωρούνταν σε τράπεζα γρανίτη με τη χρήση αεροεδράνων. Για τον υπολογισμό της θέσης και του προσανατολισμού των ρομπότ χρησιμοποιήθηκε κάμερα πάνω από την τράπεζα γρανίτη και σύστημα βασισμένο σε Pseudolite GPS [11][12]:



Σχήμα 1.3: *Free Flyers, Stanford University.*

Όπως αναφέρθηκε προηγουμένως, παρόμοιος εξομοιωτής έχει αναπτυχθεί και στο Εργαστήριο Αυτομάτου Ελέγχου του Ε.Μ.Π., όπου ένα από τα πρόσφατα projects που έχουν πραγματοποιηθεί είναι το ASSIST, στο οποίο εξομοιώθηκε σενάριο πρόσδεσης (docking) και αναπλήρωσης καυσίμου (refueling) μεταξύ δύο ρομπότ [13]:



Σχήμα 1.4: Στιγμιότυπο από πείραμα για το πρόγραμμα ASSIST, στον διαστημικό εξομοιωτή του Εργαστηρίου Αυτομάτου Ελέγχου, Ε.Μ.Π. .

1.3 Δομή Εργασίας

Η παρούσα διπλωματική εργασία απαρτίζεται από τα εξής **6** κεφάλαια :

Στο **πρώτο κεφάλαιο** γίνεται η εισαγωγή στην εργασία. Σε αυτήν εξηγείται ο σκοπός της διπλωματικής, γίνεται σύντομη αναφορά στη διεθνή βιβλιογραφία περί διαστημικής ρομποτικής και περιγράφεται η δομή της εργασίας.

Στο **δεύτερο κεφάλαιο** αναλύονται τα απαραίτητα στοιχεία θεωρίας ελέγχου εμπέδησης. Αρχικά μελετάται ο έλεγχος εμπέδησης σε μονοδιάστατο παράδειγμα και στη συνέχεια ανάγεται στη περίπτωση του επίπεδου διαστημικού εξομοιωτή.

Το **τρίτο κεφάλαιο** αφορά το λογισμικό και συγκεκριμένα το περιβάλλον ROS. Αρχικά, περιγράφεται η τυπική δομή των ROS πακέτων και παρουσιάζονται οι βασικές λειτουργίες και χαρακτηριστικά του ROS Framework. Έπειτα, περιγράφεται το υπάρχον λογισμικό του Κηφέα και αναλύεται το νέο λογισμικό το οποίο περιέχει τον ελεγκτή εμπέδησης. Εξηγείται διεξοδικά η δομή του νέου λογισμικού καθώς και η λογική του ελεγκτή κατά τη διάρκεια του πειράματος.

Το **τέταρτο κεφάλαιο** πραγματεύεται τον προσομοιωτή Gazebo. Πρώτα εξηγείται ο physics engine του περιβάλλοντος προσομοίωσης και η διασύνδεση του με το ROS. Στη συνέχεια, παρουσιάζεται το προσομοιωμένο μοντέλο του Κηφέα και πραγματοποιείται προσομοιωμένο πείραμα του υλοποιημένου ελεγκτή εμπέδησης.

Στο **πέμπτο κεφάλαιο** παρουσιάζεται αναλυτικά η διάταξη του διαστημικού εξομοιωτή του Εργαστηρίου Αυτομάτου Ελέγχου του Ε.Μ.Π. και στη συνέχεια πραγματοποιείται πείραμα ελέγχου μηχανικής εμπέδησης στον εξομοιωτή.

Στο **έκτο κεφάλαιο** αναφέρονται τα συμπεράσματα που αποφαινόμαστε με το πέρας της διπλωματικής εργασίας. Επιπλέον, γίνονται προτάσεις για μελλοντική εργασία.

Είναι σημαντικό να σημειωθεί πως στο **παράρτημα** βρίσκονται χρήσιμες για το εργαστήριο πληροφορίες, όπως τα manual των κινητήρων που χρησιμοποιούνται, οδηγίες χρήσης του Κηφέα κ.ά.

Κεφάλαιο 2

Στοιχεία Θεωρίας Ελέγχου Εμπέδησης

2.1 Εισαγωγή

Στο κεφάλαιο αυτό αναλύεται η βασική ιδέα πίσω από τον έλεγχο εμπέδησης και περιγράφεται ο ελεγκτής εμπέδησης για επίπεδο διαστημικό εξομοιωτή, όπως έχει προταθεί από το Control Systems Lab του ΕΜΠ.

Ο έλεγχος εμπέδησης αποτελεί μια μορφή ελέγχου με ανατροφοδότηση ο οποίος είναι χρήσιμος σε διεργασίες όπου το ρομπότ δέχεται δυνάμεις και αλληλεπιδρά με άλλα σώματα. Είναι σημαντικό να σημειωθεί πως διαφέρει από τον έλεγχο δύναμης, καθώς στον έλεγχο δύναμης η είσοδος και η έξοδος του νόμου ελέγχου είναι μονάχα η δύναμη που εφαρμόζεται στο σύνδεσμο στο οποίο γίνεται ο έλεγχος. Αντιθέτως, κατά τον έλεγχο εμπέδησης ο στόχος του ρομπότ περιγράφεται όχι με όρους κίνησης ούτε με όρους δύναμης, αλλά με τις σχέσεις μεταξύ δύναμης και κίνησης. Δηλαδή, ο νόμος ελέγχου δεν ρυθμίζει επιθυμητή δύναμη ή θέση, αλλά επιθυμητή **εμπέδηση** [14]. Εμπέδηση ενός συνδέσμου ρομποτικού συστήματος ορίζουμε έναν τελεστή ο οποίος υπολογίζει ως έξοδο μια χρονική συνάρτηση δύναμης(ή ροπής), έχοντας ως είσοδο μια χρονική συνάρτηση ταχύτητας(ή γωνιακής ταχύτητας) [15].

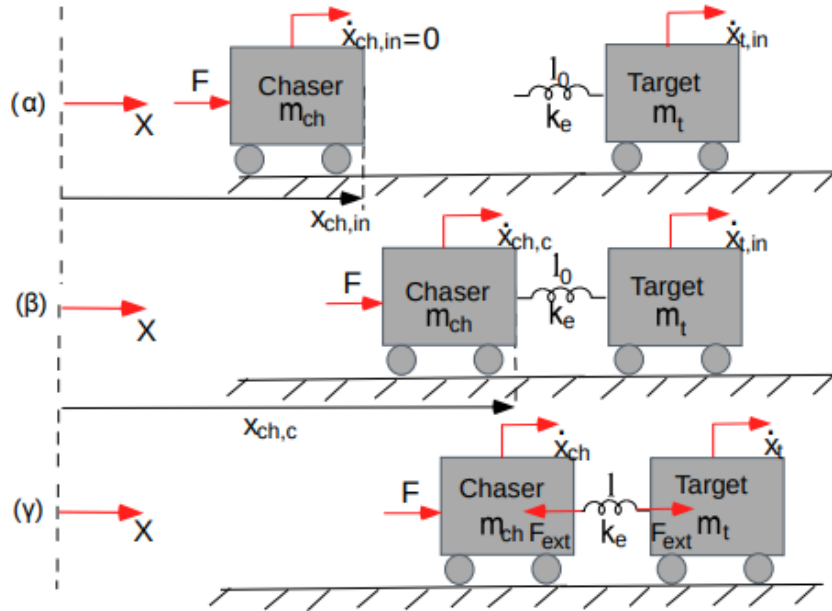
Ο έλεγχος εμπέδησης βρίσκει μεγάλο πλεονέκτημα σε μια εργασία όπου το ρομπότ χρειάζεται να ελέγξει τόσο τη θέση του όσο και την δύναμη με την οποία αλληλεπιδρά. Σε μια τυπική προσέγγιση θα μπορούσαμε να υπολογίσουμε δύο διαφορετικούς ελεγκτές, έναν θέσης κι έναν δύναμης και να εναλλάσσουμε μεταξύ των δύο όποτε αυτό χρειάζεται. Ωστόσο, ο ελεγκτής εμπέδησης είναι ένας ενιαίος ελεγκτής όπου πετυχαίνει την επιθυμητή συμπεριφορά του ρομπότ σε κάθε στάδιο της εργασίας, χωρίς να χρειάζεται να αλλάξει ο νόμος ελέγχου. Επιπλέον, σε In-Orbit-Service αποστολές είναι αναπόφευκτη η επαφή μεταξύ του ρομπότ-εξυπηρετητή και του στόχου, ο οποίος θα μπορούσε να είναι ένας δορυφόρος χωρίς κάυσιμα. Καθώς είναι σημαντικό να μην αναπτυχθούν μεγάλες κρουστικές δυνάμεις και χαθεί η επαφή μεταξύ των δύο, μια λύση που προτείνεται είναι η εφαρμογή νόμου ελέγχου εμπέδησης, με σκοπό να διατηρηθεί συνεχής επαφή [16][17].

2.2 Έλεγχος Εμπέδησης

2.2.1 Κεντρική Ιδέα - Μονοδιάστατο Παράδειγμα

Η κεντρική ιδέα του ελέγχου μηχανικής εμπέδησης γίνεται εύκολα κατανοητή αρχικά στο μονοδιάστατο παράδειγμα για ένα σώμα-κυνηγό: Κατά τη φάση της ελεύθερης κίνησης,

ο έλεγχος του σώματος ταυτίζεται με έναν model-based PD ελεγκτή θέσης. Ωστόσο, κατά τη φάση επαφής του σώματος με τον στόχο, ο ελεγκτής μηχανικής εμπέδησης υπολογίζεται κατάλληλα έτσι ώστε το σώμα να συμπεριφέρεται ως ένα σύστημα μάζας-ελατηρίου-αποσβεστήρα (mass-spring-damper system), με επιθυμητή προφανή μάζα m_d , σταθερά ελατηρίου k_d και σταθερά απόσβεσης b_d . Σχηματικά [18]:



Σχήμα 2.1: (α) Η κίνηση του κυνηγού στον ελεύθερο χώρο, (β) Η αρχή της επαφής με το στόχο, (γ) Η φάση της επαφής.

Για να κατανοήσουμε καλύτερα πως προκύπτει ο υπολογισμός του ενιαίου ελεγκτή, πρέπει πρώτα να δούμε την συμπεριφορά που θέλουμε να έχει σε κάθε φάση [18]:

Επιθυμητή δυναμική σφάλματος στο free space phase

Κατά τη διάρκεια της ελεύθερης κίνησης του, ο κυνηγός (chaser) θέλουμε να ακολουθεί μια επιθυμητή πολυωνυμική τροχιά, από την αρχική του θέση μέχρι την θέση της επαφής. Για αυτό, η επιθυμητή δυναμική σφάλματος δίνεται από την εξής σχέση:

$$m_{d,f}\ddot{e}_f + b_{d,f}\dot{e}_f + k_{d,f}e_f = 0 \quad (2.1)$$

Το σφάλμα μεταξύ πραγματικής κι επιθυμητής θέσης δίνεται από τη σχέση:

$$e_f(t) = x_{ch}(t) - x_{d,f}(t) \quad (2.2)$$

Η επιθυμητή τροχιά υπολογίζεται χρησιμοποιώντας τόξο παραμετροποίησης:

$$x_{d,f}(t) = x_{ch,in} + s(t)(x_{ch,c} - x_{ch,in}) \quad (2.3)$$

, όπου $x_{ch,in}$ η αρχική θέση του κυνηγού και $x_{ch,c}$ η θέση του κυνηγού όταν πραγματοποιηθεί η επαφή, που είναι ουσιαστικά η αρχική θέση του στόχου, καθώς θεωρούμε τον στόχο

ακίνητο. Επίσης θεωρούμε την αρχική ταχύτητα του κυνηγού μηδενική. Η συνάρτηση $s(t)$ ονομάζεται τόξο παραμετροποίησης και είναι ένα πολυώνυμο 5ου βαθμού:

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \quad (0 \leq s(t) \leq 1) \quad (2.4)$$

Οι συντελεστές $a_i (i = 0, \dots, 5)$ υπολογίζονται από τις αρχικές και τελικές συνθήκες θέσης, ταχύτητας κι επιτάχυνσης του ΤΣΔ πριν την επαφή.

Επιθυμητή δυναμική σφάλματος στο contact phase

Η εμπεδητική συμπεριφορά του κυνηγού παρουσιάζεται όταν εμφανιστεί επαφή με το σώμα του στόχου. Σε αυτή τη φάση η επιθυμητή δυναμική σφάλματος δίνεται από την σχέση:

$$m_{d,c}\ddot{e}_c + b_{d,c}\dot{e}_c + k_{d,c}e_c = -F_{ext} + F_d \quad (2.5)$$

Το σφάλμα ορίζεται ως:

$$e_c(t) = x_{ch}(t) - x_t(t) + l_0 \quad (2.6)$$

Η δύναμη επαφής F_{ext} μοντελοποιείται ως μια δύναμη ελατηρίου σταθεράς k_e και φυσικού μήκους l_0 :

$$F_{ext} = k_e e_c \quad (2.7)$$

Κοινή επιθυμητή δυναμική σφάλματος

Συνδυάζοντας τις επιθυμητές δυναμικές των δύο φάσεων προκύπτει η κοινή επιθυμητή δυναμική σφάλματος, για τον ενιαίο ελεγκτή εμπέδησης που θέλουμε να υπολογίσουμε:

$$m_d\ddot{e} + b_d\dot{e} + k_d e = -F_{ext} + F_{des} \quad (2.8)$$

Το σφάλμα ορίζεται ως:

$$e(t) = x_{ch}(t) - x_d(t) \quad (2.9)$$

Για να υπολογίζεται σωστά η επιθυμητή θέση, ανάλογα την φάση του πειράματος, το $x_d(t)$ ορίζεται ως:

$$x_d(t) = x_{d,f}(t) \frac{|1 - |F_{ext}|/a_1|}{1 + a_1 |F_{ext}|} + x_{d,c}(t) \frac{|F_{ext}|}{|F_{ext}| + a_2} \quad (2.10)$$

Με a_1, a_2 αρκετά μεγάλες και μικρές τιμές αντίστοιχα έτσι ώστε όταν η F_{ext} είναι 0 να ταυτίζεται το $x_d(t)$ με το $x_{d,f}(t)$ και όταν δεν είναι 0 τότε το $x_d(t)$ να ταυτίζεται με το $x_{d,c}(t)$. Με την ίδια λογική ορίζονται τα κέδη και για τις δύο φάσεις:

$$m_d = m_{d,f} \frac{|1 - |F_{ext}|/a_1|}{1 + a_1 |F_{ext}|} + m_{d,c} \frac{|F_{ext}|}{|F_{ext}| + a_2} \quad (2.11)$$

$$b_d = b_{d,f} \frac{|1 - |F_{ext}|/a_1|}{1 + a_1 |F_{ext}|} + b_{d,c} \frac{|F_{ext}|}{|F_{ext}| + a_2} \quad (2.12)$$

$$k_d = k_{d,f} \frac{|1 - |F_{ext}|/a_1|}{1 + a_1 |F_{ext}|} + k_{d,c} \frac{|F_{ext}|}{|F_{ext}| + a_2} \quad (2.13)$$

,όπου για $m_{d,f} > 0$ έχουμε:

$$b_{d,f} = 2\zeta_f \omega_{n,f} m_{d,f} \quad (2.14)$$

$$k_{d,f} = (\omega_{n,f})^2 m_{d,f} \quad (2.15)$$

και για $m_{c,f} > 0$

$$b_{d,c} = 2\zeta_f \omega_{n,c} m_{d,c} \quad (2.16)$$

$$k_{d,c} = (\omega_{n,c})^2 m_{d,c} \quad (2.17)$$

Ομοίως ορίζεται η δύναμη F_{des} , οποία ταυτίζεται με τη δύναμη F_d όταν υπάρχει επαφή και είναι 0 όταν δεν υπάρχει επαφή:

$$F_{des} = \frac{F_d |F_{ext}|}{|F_{ext}| + a_2} \quad (2.18)$$

Δυναμική chaser

Η δυναμική του chaser δίνεται από τη σχέση:

$$F = m_{ch} \ddot{x}_{ch} + F_{ext} \quad (2.19)$$

,όπου m_{ch} η μάζα του chaser, $\ddot{x}_{ch}$ η επιτάχυνση του και F η δύναμη επενέργησης στον chaser.

Αντικαθιστώντας στη σχέση 2.8 την δεύτερη παράγωγο του σφάλματος: $\ddot{e}(t) = \ddot{x}_{ch}(t) - \ddot{x}_d(t)$ λύνουμε ως προς $\ddot{x}_{ch}(t)$ και αντικαθιστούμε στη σχέση 2.19. Τελικά προκύπτει ο **νόμος ελέγχου εμπέδησης**:

$$F = (1 - \frac{m_{ch}}{m_d}) F_{ext} - \frac{m_{ch}}{m_d} (b_d \dot{e} + k_d e - F_{des}) + m_{ch} \ddot{x}_d \quad (2.20)$$

2.2.2 Έλεγχος Εμπέδησης Επίπεδου Διαστημικού Εξομοιωτή

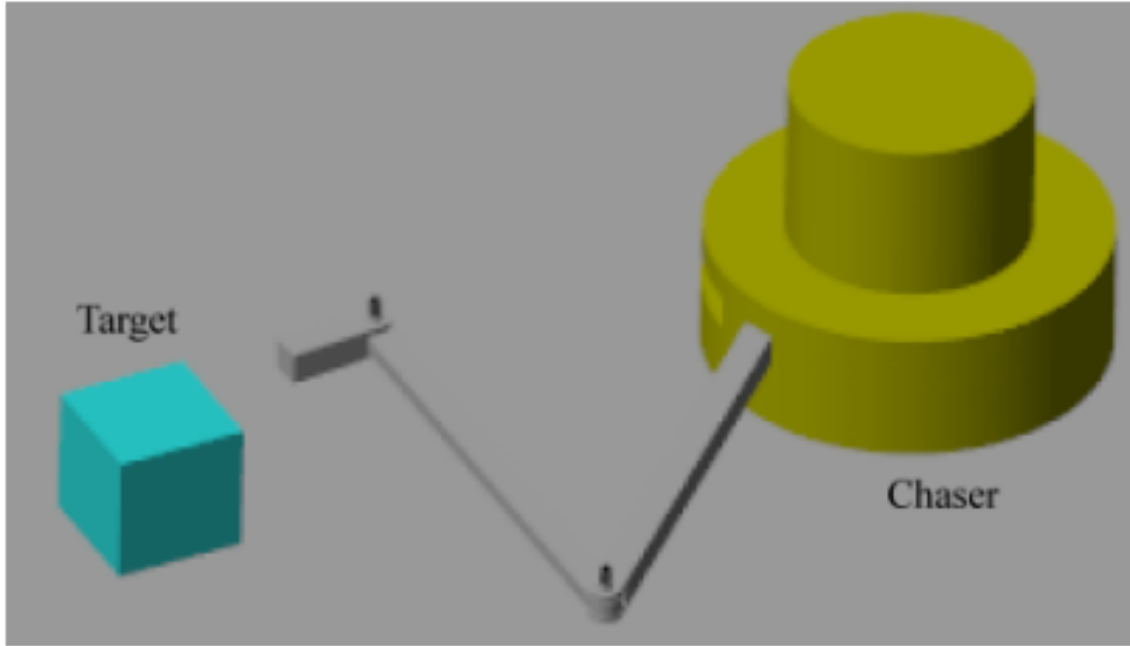
Ακολουθώντας την ίδια λογική με το μονοδιάστατο παράδειγμα, μπορούμε να περιγράψουμε τον νόμο ελέγχου εμπέδησης για τον επίπεδο διαστημικό εξομοιωτή, όπως προκύπτει από την δουλειά του Control Systems Lab [17]. Κρατώντας την ίδια ονοματολογία με το μονοδιάστατο παράδειγμα, θα θεωρούμε chaser το ρομπότ και target τον στόχο. Ο chaser έχει έναν manipulator με 3 joints κι ένα reaction wheel στη βάση του. Σχηματικά [17]:

Δυναμική chaser

Αποδεικνύεται ότι η δυναμική του chaser δίνεται από την εξής σχέση [17]:

$$\mathbf{M} \dot{\mathbf{v}}_{cont} + \mathbf{c} = \mathbf{J}_c^T \mathbf{Q}_c + \mathbf{J}_e^T \mathbf{F}_{ext} \quad (2.21)$$

,όπου $\mathbf{M}$: Το μητρώο αδράνειας του chaser το οποίο εξαρτάται από το configuration του, $\mathbf{c}$ οι μη-γραμμικοί όροι που οφείλονται σε coriolis και φυγοκεντρικές δυνάμεις, $\mathbf{J}_c^T$ και $\mathbf{J}_e^T$ κατάλληλες ιακωδιανές. Τα παραπάνω δίνονται στο παράρτημα, στον κώδικα του ελεγκτή.



Σχήμα 2.2: Επίπεδος διαστημικός εξομοιωτής, κυνηγός και στόχος.

Το διάνυσμα επενέργειας $\mathbf{Q}_c$ ορίζεται ως εξής:

$$\mathbf{Q}_c = \begin{bmatrix} \tau_{rw} \\ \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} \quad (2.22)$$

και περιέχει τις ροπές που ασκούνται στο reaction wheel και στα 3 joints αντίστοιχα.

Το ελεγχόμενο διάνυσμα ταχυτήτων ορίζεται ως εξής:

$$\mathbf{v}_{cont} = \begin{bmatrix} \dot{\theta}_b \\ \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{bmatrix} \quad (2.23)$$

, όπου θ_b ο προσανατολισμός της βάσης και x_e, y_e, θ_e η θέση και ο προσανατολισμός του τελικού σημείου δράσης (ΤΣΔ) του manipulator του chaser, σύμφωνα με το αδρανειακό σύστημα συντεταγμένων (world frame).

Επιπλέον, $\mathbf{F}_{ext}$ ορίζεται το (3×1) διάνυσμα εξωτερικών δυνάμεων που δέχεται το ΤΣΔ κατά την επαφή, σύμφωνα με το σύστημα συντεταγμένων του ΤΣΔ. Στα πλαίσια της εργασίας θεωρούμε ότι η επαφή πραγματοποιείται κατά τον άξονα x . Γιαυτό το διάνυσμα παίρνει τη μορφή:

$$\mathbf{F}_{ext} = \begin{bmatrix} f_{ext,x} \\ 0 \\ 0 \end{bmatrix} \quad (2.24)$$

Επιθυμητή δυναμική σφάλματος ΤΣΔ στο free space phase

Η επιθυμητή δυναμική σφάλματος του ΤΣΔ στο free space phase δίνεται από τη σχέση :

$$\mathbf{M}_{d,e}^f \ddot{\mathbf{e}}_e^f + \mathbf{B}_{d,e}^f \dot{\mathbf{e}}_e^f + \mathbf{K}_{d,e}^f \mathbf{e}_e^f = \mathbf{0}_{3 \times 1} \quad (2.25)$$

,όπου $\mathbf{M}_{d,e}^f, \mathbf{B}_{d,e}^f, \mathbf{K}_{d,e}^f$ κατάλληλοι 3×3 πίνακες κερδών :

$$\mathbf{M}_{d,e}^f = m_{d,e}^f \mathbf{I}_3 \quad (2.26)$$

$$\mathbf{B}_{d,e}^f = b_{d,e}^f \mathbf{I}_3 \quad (2.27)$$

$$\mathbf{K}_{d,e}^f = k_{d,e}^f \mathbf{I}_3 \quad (2.28)$$

Τα βαθμωτά κέρδη, για $m_{d,e}^f > 0$, υπολογίζονται ως εξής :

$$b_{d,e}^f = 2\zeta_e^f \omega_{n,e}^f m_{d,e}^f \quad (2.29)$$

και

$$k_{d,e}^f = (\omega_{n,e}^f)^2 m_{d,e}^f \quad (2.30)$$

Το (3×1) διάνυσμα σφάλματος θέσης και προσανατολισμού για το ΤΣΔ ορίζεται ως εξής :

$$\mathbf{e}_e^f = \mathbf{x}_e - \mathbf{x}_{e,d}^f \quad (2.31)$$

,όπου $\mathbf{x}_e$ το διάνυσμα θέσης και προσανατολισμού του ΤΣΔ και $\mathbf{x}_{e,d}^f$ το διάνυσμα επιθυμητής τροχιάς του ΤΣΔ, η οποία δίνεται από τη σχέση :

$$\mathbf{x}_{e,d}^f = \mathbf{x}_e^{in} + s(t)(\mathbf{x}_e^{fin} - \mathbf{x}_e^{in}) \quad (2.32)$$

, όπου $s(t)$ το πολυωνυμικό τόξο παραμετροποίησης και $\mathbf{x}_e^{in}, \mathbf{x}_e^{fin}$ η αρχική και τελική θέση και προσανατολισμός του ΤΣΔ πριν την επαφή.

Επιθυμητή δυναμική σφάλματος ΤΣΔ στο contact phase

Η επιθυμητή δυναμική σφάλματος του ΤΣΔ στο contact phase δίνεται από τη σχέση :

$$\mathbf{M}_{d,e}^c \ddot{\mathbf{e}}_e^c + \mathbf{B}_{d,e}^c \dot{\mathbf{e}}_e^c + \mathbf{K}_{d,e}^c \mathbf{e}_e^c = -\mathbf{F}_{ext} + \mathbf{F}_{des} \quad (2.33)$$

Ο ορισμός του διανύσματος $\mathbf{F}_{des}$ ακολουθεί παρακάτω και χρησιμοποιείται για να εξασφαλιστεί η διατήρηση της επαφής.

Τα κέρδη για το contact phase ορίζονται ομοίως :

$$\mathbf{M}_{d,e}^c = m_{d,e}^c \mathbf{I}_3 \quad (2.34)$$

$$\mathbf{B}_{d,e}^c = b_{d,e}^c \mathbf{I}_3 \quad (2.35)$$

$$\mathbf{K}_{d,e}^c = k_{d,e}^c \mathbf{I}_3 \quad (2.36)$$

Τα βαθμωτά κέρδη, για $m_{d,e}^c > 0$, υπολογίζονται ως εξής:

$$b_{d,e}^c = 2\zeta_e^c \omega_{n,e}^c m_{d,e}^c \quad (2.37)$$

και

$$k_{d,e}^c = (\omega_{n,e}^c)^2 m_{d,e}^c \quad (2.38)$$

Το σφάλμα του ΤΣΔ για το contact phase ορίζεται ομοίως:

$$\mathbf{e}_e^c = \mathbf{x}_e - \mathbf{x}_{e,d}^c \quad (2.39)$$

,όπου το $\mathbf{x}_{e,d}^c$ ταυτίζεται με τη θέση και προσανατολισμό του στόχου κατά τη διάρκεια της επαφής:

$$\mathbf{x}_{e,d}^c = \mathbf{x}_t = \begin{bmatrix} x_t \\ y_t \\ \vartheta_t \end{bmatrix} \quad (2.40)$$

Επιθυμητή δυναμική σφάλματος προσανατολισμού βάσης

Η επιθυμητή δυναμική σφάλματος του προσανατολισμού της βάσης και στις δύο φάσεις του πειράματος δίνεται από τη σχέση:

$$\ddot{e}_b + b_{d,b} \dot{e}_b + k_{d,b} e_b = 0 \quad (2.41)$$

Το σφάλμα ορίζεται ως εξής:

$$e_b = \vartheta_b - \vartheta_{b,d} \quad (2.42)$$

,όπου $\vartheta_{b,d}$ είναι ο αρχικός προσανατολισμός της βάσης. Ομοίως με πριν ορίζονται τα βαθμωτά κέρδη:

$$b_{d,b} = 2\zeta_b \omega_{n,b} \quad (2.43)$$

και

$$k_{d,b} = (\omega_{n,b})^2 \quad (2.44)$$

Επιθυμητή δυναμική σφάλματος κλειστού συστήματος

Συνδυάζοντας την επιθυμητή δυναμική του ΤΣΔ και της βάσης και για τις δύο φάσεις, προκύπτει τελικά η επιθυμητή δυναμική σφάλματος του κλειστού συστήματος:

$$\mathbf{M}_d \ddot{\mathbf{e}} + \mathbf{B}_d \dot{\mathbf{e}} + \mathbf{K}_d \mathbf{e} = -\mathbf{F}_{ext}^* + \mathbf{F}_{des}^* \quad (2.45)$$

,όπου :

$$\mathbf{F}_{ext}^* = \begin{bmatrix} 0 \\ \mathbf{F}_{ext} \end{bmatrix} \quad (2.46)$$

και

$$\mathbf{F}_{des}^* = \begin{bmatrix} 0 \\ \mathbf{F}_{des} \end{bmatrix} \quad (2.47)$$

Οι (4x4) πίνακες κερδών ορίζονται ως εξής:

$$\mathbf{M}_d = \begin{bmatrix} 1 & \mathbf{0}_{1 \times 3} \\ \mathbf{0}_{3 \times 1} & \mathbf{M}_{d,e} \end{bmatrix} \quad (2.48)$$

$$\mathbf{B}_d = \begin{bmatrix} b_{d,b} & \mathbf{0}_{1 \times 3} \\ \mathbf{0}_{3 \times 1} & \mathbf{B}_{d,e} \end{bmatrix} \quad (2.49)$$

$$\mathbf{k}_d = \begin{bmatrix} K_{d,b} & \mathbf{0}_{1 \times 3} \\ \mathbf{0}_{3 \times 1} & \mathbf{K}_{d,e} \end{bmatrix} \quad (2.50)$$

Το σφάλμα ορίζεται από τη σχέση:

$$\mathbf{e} = \begin{bmatrix} e_b \\ \mathbf{e}_e \end{bmatrix} = \begin{bmatrix} \vartheta_b - \vartheta_{b,d} \\ \mathbf{x}_e - \mathbf{x}_{e,d} \end{bmatrix} \quad (2.51)$$

Καθώς θέλουμε κοινή δυναμική σφάλματος και στις δύο φάσεις ώστε να υπολογιστεί ενιαίος ελεγκτής, τα παρακάτω μεγέθη ορίζονται ως εξής:

$$\mathbf{x}_{e,d}(t) = \mathbf{x}_{e,d}^f(t) \frac{|1 - \|\mathbf{F}_{ext}\|/a_1|}{1 + a_1 \|\mathbf{F}_{ext}\|} + \mathbf{x}_{e,d}^c(t) \frac{\|\mathbf{F}_{ext}\|}{\|\mathbf{F}_{ext}\| + a_2} \quad (2.52)$$

Όπου τα αδιάστατα μεγέθη a_1, a_2 είναι αρκετά μεγάλα και μικρά αντίστοιχα, έτσι ώστε η επιθυμητή τροχιά να ταυτίζεται με αυτή του free space phase όταν δεν υπάρχει επαφή, ενώ όταν υπάρχει επαφή να ταυτίζεται με αυτή του contact phase. Με την ίδια λογική ορίζονται οι πίνακες κερδών $\mathbf{M}_{d,e}, \mathbf{B}_{d,e}, \mathbf{K}_{d,e}$:

$$\mathbf{M}_{d,e} = \mathbf{M}_{d,e}^f \frac{|1 - \|\mathbf{F}_{ext}\|/a_1|}{1 + a_1 \|\mathbf{F}_{ext}\|} + \mathbf{M}_{d,e}^c \frac{\|\mathbf{F}_{ext}\|}{\|\mathbf{F}_{ext}\| + a_2} \quad (2.53)$$

$$\mathbf{B}_{d,e} = \mathbf{B}_{d,e}^f \frac{|1 - \|\mathbf{F}_{ext}\|/a_1|}{1 + a_1 \|\mathbf{F}_{ext}\|} + \mathbf{B}_{d,e}^c \frac{\|\mathbf{F}_{ext}\|}{\|\mathbf{F}_{ext}\| + a_2} \quad (2.54)$$

$$\mathbf{K}_{d,e} = \mathbf{K}_{d,e}^f \frac{|1 - \|\mathbf{F}_{ext}\|/a_1|}{1 + a_1 \|\mathbf{F}_{ext}\|} + \mathbf{K}_{d,e}^c \frac{\|\mathbf{F}_{ext}\|}{\|\mathbf{F}_{ext}\| + a_2} \quad (2.55)$$

Επίσης, όταν υπάρχει επαφή το διάνυσμα $\mathbf{F}_{des}$ ταυτίζεται με το επιθυμητό διάνυσμα $\mathbf{F}_d$ ενώ όταν δεν υπάρχει επαφή ταυτίζεται με το μηδενικό (3x1) διάνυσμα.

$$\mathbf{F}_{des} = \mathbf{F}_d \frac{\|\mathbf{F}_{ext}\|}{\|\mathbf{F}_{ext}\| + a_2} \quad (2.56)$$

, όπου $\mathbf{F}_d$ το (3x1) διάνυσμα για να μην μηδενίζεται η δύναμη επαφής, όπου στα πλαίσια

της εργασίας ορίζεται μόνο για τον x άξονα :

$$\mathbf{F}_d = \begin{bmatrix} f_{d,x} \\ 0 \\ 0 \end{bmatrix} \quad (2.57)$$

Υλοποιημένος ενιαίος ελεγκτής εμπέδησης

Έχοντας ορίσει όλα τα αναγκαία μεγέθη, μπορούμε τώρα να περιγράψουμε τον ενιαίο ελεγκτή εμπέδησης, με τον οποίο το κλειστό μας σύστημα θα έχει την επιθυμητή συμπεριφορά, όπως έχει υπολογιστεί από το Εργαστήριο Αυτομάτου Ελέγχου του ΕΜΠ [17]:

$$\mathbf{Q}_{contr} = \mathbf{M}\mathbf{u} + \mathbf{c} - \mathbf{J}_e^T \mathbf{F}_{ext} \quad (2.58)$$

,όπου :

$$\mathbf{u} = \ddot{\mathbf{x}}_d + \mathbf{M}_d^{-1}(-\mathbf{B}_d \dot{\mathbf{e}} - \mathbf{K}_d \mathbf{e} - \mathbf{F}_{ext}^* + \mathbf{F}_{des}^*) \quad (2.59)$$

Το διάνυσμα ελέγχου $\mathbf{Q}_{contr}$ με το διάνυσμα επενέργειας $\mathbf{Q}_c$ συνδέονται με τη σχέση :

$$\mathbf{Q}_{contr} = \mathbf{J}_c^T \mathbf{Q}_c \quad (2.60)$$

Τελικά, το **υπολογισθέν διάνυσμα επενέργειας** δίνεται από τη σχέση :

$$\mathbf{Q}_c = \begin{bmatrix} \tau_{rw} \\ \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = (\mathbf{J}_c^T)^{-1} \mathbf{Q}_{contr} \quad (2.61)$$

Κεφάλαιο **3**

To ROS Framework

3.1 Εισαγωγή

Ο διαστημικός εξομοιωτής του εργαστηρίου Κηφέας είναι ένα σύνθετο ρομποτικό σύστημα το οποίο αποτελείται από μια πληθώρα ηλεκτρονικών στοιχείων, ελεγκτών κι ενός βασικού υπολογιστή στη βάση του. Συνεπώς προκύπτει εύλογα το ερώτημα του πως μπορούν να επικοινωνούν μεταξύ τους όλα τα παραπάνω σε πραγματικό χρόνο, από τον αισθητήρα δύναμης που διαβάζει τις δυνάμεις που ασκούνται στο ΤΣΔ και τις μεταφέρει στον κεντρικό υπολογιστή, ο οποίος επεξεργάζεται τα δεδομένα και παράγει τον νόμο ελέγχου, μέχρι τους drivers στους κινητήρες κάθε άρθρωσης, οι οποίοι περιμένουν εντολή ώστε να κουνηθεί ο βραχίονας. Στο πρόβλημα αυτό καταλύτης είναι το Robot Operating System ή εν συντομία ROS.

Με λίγα λόγια, το ROS αποτελεί ένα framework (ή αλλιώς περιβάλλον) στο οποίο διευκολύνεται η επικοινωνία μεταξύ πολλαπλών προγραμμάτων (π.χ. εκτελέσιμων αρχείων C++, Python) τα οποία τρέχουν ταυτόχρονα, σε έναν ή παραπάνω υπολογιστές. Επιπλέον, προσφέρει μια πληθώρα βιβλιοθηκών οι οποίες παρέχουν υλοποιημένους αλγορίθμους και συναρτήσεις, κομβικές για πολλές διαδικασίες στο κλάδο της ρομποτικής [19].

3.2 Βασικά στοιχεία του ROS

Σε αυτή την ενότητα θα αναλύσουμε τη δομή και τα βασικά συστατικά ενός ρομποτικού συστήματος το οποίο χρησιμοποιεί το ROS framework.

3.2.1 ROS package

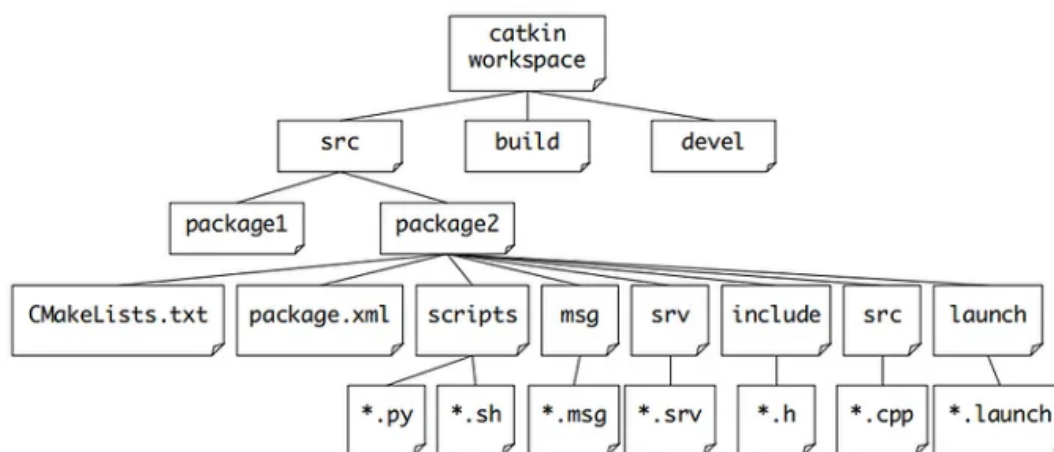
Αρχικά, το ROS λογισμικό οργανώνεται σε βασικούς φακέλους τα οποία ονομάζονται ROS packages. Τα πακέτα αυτά βρίσκονται όλα μέσα στον src φάκελο ενός μητρικού φακέλου, οποίος αποτελεί το ROS workspace μας και συνήθως ονομάζεται catkin_workspace. Μέσα στα πακέτα γίνεται να υπάρχει μια πληθώρα αρχείων τα οποία οργανώνονται σε υποφακέλους, ανάλογα με την χρησιμότητα τους [20].

Δομή ROS πακέτου

Μια τυπική αρχιτεκτονική ενός ROS πακέτου αποτελείται τα εξής αρχεία και υποφακέλους [20]:

- `...\src` : Υποφάκελος ο οποίος περιέχει όλα τα executable αρχεία (σε C++ ή Python), τα οποία “τρέχουν” και λειτουργούν ως ROS nodes.
- `...\msg`: Υποφάκελος ο οποίος περιέχει διάφορα μηνύματα τα οποία ανταλλάσσουν τα nodes μεταξύ τους.
- `...\srv`: Υποφάκελος ο οποίος περιέχει διάφορα είδη υπηρεσιών (services) τις οποίες εκτελούν τα nodes.
- **CMakeLists.txt**: Αρχείο το οποίο περιέχει εντολές για το build του πακέτου.
- **package.xml**: Αρχείο το οποίο περιέχει πληροφορίες για το πακέτο και τα επιμέρους στοιχεία που το απαρτίζουν.

Η παραπάνω αρχιτεκτονική μπορεί να εμπλουτιστεί με επιπλέον αρχεία και υποφάκελους ή να απλουστευτεί, ανάλογα με τις ανάγκες του χρήστη [20]. Με bold γράφονται τα αρχεία τα οποία είναι απαραίτητα σε κάθε ROS πακέτο και βρίσκονται στον root folder [21][22]. Σχηματικά, η τυπική δομή ενός ROS workspace είναι ως εξής [23]:



Σχήμα 3.1: Η τυπική δομή ενός ROS workspace.

3.2.2 ROS node

ROS nodes (κόμβοι) είναι processes (διεργασίες) οι οποίες εκτελούν υπολογισμούς. Ουσιαστικά, είναι (C++ ή Python) αρχεία τα οποία τρέχουν τον κώδικα τους είτε στιγμιαία είτε ανά τακτά χρονικά διαστήματα και χρησιμοποιώντας την πληθώρα των βιβλιοθηκών εκτελούν μια συγκεκριμένη εργασία [24].

Η δομή του λογισμικού ενός ρομπότ βασίζεται στην αφαιρετικότητα που προσφέρει το ROS. Δηλαδή, όπως το hardware ενός ρομπότ μπορεί να χωρίζεται σε πολλά υποσυστήματα, έτσι και το software του χωρίζεται σε πολλά ROS πακέτα, τα οποία περιέχουν ROS nodes. Αυτά τα nodes μπορούν να τρέχουν παράλληλα και το καθένα να επιτελεί έναν συγκεκριμένο στόχο. Επιπλέον, τα nodes μπορούν να επικοινωνούν και να ανταλλάζουν πληροφορίες μεταξύ τους μέσω διαφόρων πρωτοκόλλων επικοινωνίας, τα οποία θα εξετάσουμε παρακάτω. [24].

Για παράδειγμα, σε ένα κινητό ρομπότ γίνεται ένα node να ελέγχει έναν laser scanner, ένα δεύτερο node να επικοινωνεί με το πρώτο node και να παίρνει τις μετρήσεις του laser και στη συνέχεια να εκτελεί υπολογισμούς για το localization του ρομπότ. Έπειτα, ένα τρίτο node μπορεί να επικοινωνεί με το δεύτερο node και να ελέγχει τους κινητήρες για τις ρόδες ώστε το ρομπότ τελικά να κινηθεί προς την επιθυμητή θέση [24].

3.2.3 Μηχανισμοί επικοινωνίας μεταξύ των nodes

Όπως έχει αναφερθεί, το μεγάλο πλεονέκτημα του ROS framework είναι ο καταμερισμός των εργασιών σε επιμέρους nodes, τα οποία μπορούν να ανταλλάσσουν πληροφορίες μεταξύ τους [24]. Η επικοινωνία τους εξασφαλίζεται με τους εξής μηχανισμούς:

1. ROS Topics

Τα ROS topics αποτελούν τον βασικό τρόπο επικοινωνίας μεταξύ των nodes, καθώς χρησιμοποιούνται κατά κόρον σε κάθε ρομπωτική εφαρμογή που χρησιμοποιεί ROS. Ο λόγος για αυτό είναι η ευχρηστία και η απλότητα στη χρήση τους. Ουσιαστικά, τα ROS topics είναι named data buses, δηλαδή δίαυλοι δεδομένων με συγκεκριμένο όνομα, το οποίο είτε καθορίζεται από τον χρήστη ή είναι προκαθορισμένο. Σε αυτούς τους δίαυλους περνάνε **ROS messages** (μηνύματα) τα οποία έχουν συγκεκριμένο τύπο. Ο τύπος του μηνύματος καθορίζει και τον τύπο το topic, καθώς επίσης και τα fields (πεδία) τα οποία θα περιέχει το μήνυμα αυτό. Να σημειωθεί πως το πρωτόκολλο επικοινωνίας που χρησιμοποιείται για την μεταφορά των μηνυμάτων είναι το TCP/IP, ενώ μπορεί να χρησιμοποιηθεί και το UDP [25].

Για παράδειγμα, ένα std_msgs/Float64 message περιέχει μόνο ένα field τύπου float64 με το όνομα data [26]. Αντιθέτως, ένα geometry_msgs/Vector3 message περιέχει 3 float64 πεδία με τα ονόματα x, y και z [27]. Επιπλέον, γίνεται ένα ROS message να περιέχεται ως πεδίο σε ένα άλλο message, όπως για παράδειγμα το geometry_msgs/Pose message, το οποίο έχει ως πεδία ένα geometry_msgs/Point message με όνομα position κι ένα geometry_msgs/Quaternion message με όνομα orientation [28].

Για κάθε ROS topic ένα ROS node μπορεί είτε να γράψει μήνυμα σε αυτό το topic είτε να διαβάσει μήνυμα από αυτό. Δηλαδή, κάθε node μπορεί να έχει **publishers** και **subscribers** σε διάφορα topics [25].

Ένας publisher μπορεί να γράφει ένα συγκεκριμένο τύπο μηνύματος σε ένα topic όποτε του ζητηθεί. Για παράδειγμα, στη C++ ένα object της κλάσης `ros::Publisher` κατασκευάζεται με την κλήση της συνάρτησης `nodehandle.advertise()`. Αυτή η συνάρτηση είναι συνάρτηση της κλάσης `ros:: nodehandle`, η οποία είναι υπεύθυνη για τις λειτουργίες του ROS node. Κατά τη κλήση της παραπάνω συνάρτησης, τα βασικά ορίσματα που πρέπει να δώσει ο χρήστης είναι ο τύπος του μηνύματος που θα γράφει ο publisher καθώς και το topic στο οποίο θα γράφει. Φυσικά, πρέπει ο τύπος του topic (αν είναι ήδη καθορισμένος) να συνάδει με το τύπο του μηνύματος που θα κάνει publish. Αν το topic δεν είχε καθοριστεί προηγουμένως, τότε ο τύπος του θα είναι ίδιος με τον τύπο του μηνύματος. Επιπλέον, υπάρχουν και άλλα ορίσματα, όπως το μέγιστο πλήθος των μηνυμάτων που μπορεί να κρατήσει ο publisher μέχρι να τα κάνει publish, τα οποία παίρνουν default τιμές αν δεν τις ορίσει ο χρήστης.

Για να κάνει publish το μήνυμα ο publisher καλείται η συνάρτηση `publisher.publish()` με όρισμα το μήνυμα [29].

Στο άλλο άκρο, ένας subscriber κάνει subscribe σε ένα topic και όποτε του ζητηθεί ελέγχει αν έχουν φτάσει νέα μηνύματα στο topic (από οποιονδήποτε publisher). Ο έλεγχος για μηνύματα μπορεί να γίνει π.χ. στη C++ με τη χρήση της συνάρτησης `ros::spin()` ή `ros::spinOnce()`. Αν όντως αντιληφθεί κάποιο νέο μήνυμα, τότε καλεί αυτόματα την δική του callback συνάρτηση, η οποία παίρνει ως όρισμα έναν pointer στο μήνυμα αυτό και το επεξεργάζεται καταλλήλως. Είναι σημαντικό να σημειωθεί εδώ πως, όταν ο χρήστης ορίζει έναν subscriber, πρέπει επίσης να ορίσει την σιαμαία callback συνάρτηση του. Στη C++ ένα object της κλάσης `ros::Subscriber` κατασκευάζεται με τη χρήση της συνάρτησης `nodehandle.subscribe()`. Κατά τη κλήση αυτής της συνάρτησης ο χρήστης πρέπει να ορίσει το όνομα του topic στο οποίο κάνει subscribe καθώς και το όνομα της callback συνάρτησης. Ένα επιπλέον όρισμα το οποίο παίρνει ως default τιμή την μονάδα είναι το πλήθος των μηνυμάτων που μπορεί να δρομολογήσει προς επεξεργασία ο subscriber, σε περίπτωση που δεν μπορεί να τα επεξεργαστεί με την ίδια ταχύτητα με την οποία γίνονται published [29].

Το σημαντικό με αυτόν τον μηχανισμό επικοινωνίας είναι ότι η επικοινωνία επιτυγχάνεται ασύγχρονα και είναι δυνατόν να υπάρχουν πολλαπλοί publishers και subscribers για το ίδιο topic από διαφορετικά nodes [25].

2. ROS Services

Οι υπηρεσίες (ROS services) αποτελούν έναν από τους βασικούς μηχανισμούς επικοινωνίας στο ROS, παρέχοντας έναν απλό αλλά ισχυρό τρόπο για την ανταλλαγή δεδομένων μεταξύ των nodes μέσω ενός request-response μοντέλου (αιτήματος-απόκρισης). Αυτό το μοντέλο είναι ιδανικό για περιπτώσεις όπου ένας κόμβος χρειάζεται να ζητήσει την εκτέλεση μιας συγκεκριμένης εργασίας από έναν άλλο κόμβο και να λάβει μια συγκεκριμένη απάντηση [30].

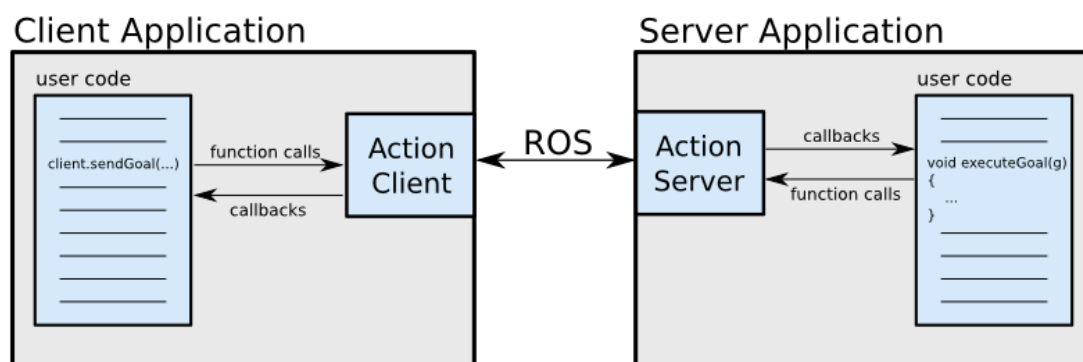
Η λειτουργία μιας υπηρεσίας στο ROS περιλαμβάνει δύο βασικά στοιχεία: τον **server** και τον **client**. Ο server είναι ο κόμβος που προσφέρει την υπηρεσία και περιμένει αιτήματα από άλλους κόμβους. Ο client είναι ο κόμβος που κάνει τα αιτήματα και αναμένει μια απάντηση από τον server. Όταν ένας client στέλνει ένα αίτημα, ο server το επεξεργάζεται και επιστρέφει μια απάντηση. Αυτό το είδος επικοινωνίας είναι συγχρονισμένο, δηλαδή ο client μπλοκάρει (δηλαδή περιμένει) μέχρι να λάβει την απάντηση του server πριν συνεχίσει την εκτέλεση των υπόλοιπων εντολών του [30].

Τα services είναι κατάλληλα για περιπτώσεις όπου απαιτείται η εκτέλεση μιας συγκεκριμένης ενέργειας που δεν είναι συνεχής, όπως η ανάγνωση ενός αισθητήρα κατά απαίτηση, η ενεργοποίηση ενός μηχανισμού, ή η αποστολή μιας συγκεκριμένης εντολής σε ένα ρομπότ. Ωστόσο, η χρήση των services απαιτεί την προσεκτική διαχείριση του χρόνου, καθώς η φύση της συγχρονισμένης επικοινωνίας μπορεί να οδηγήσει σε καθυστερήσεις εάν ο server χρειάζεται πολύ χρόνο για να επεξεργαστεί ένα αίτημα. Παρόλα αυτά, η απλότητα του μοντέλου αιτήματος-απόκρισης το καθιστά ιδανικό για εφαρμογές όπου η καθυστέρηση δεν είναι κρίσιμη ή για διεργασίες που απαιτούν άμεση και συγκεκριμένη αλληλεπίδραση [31].

3. ROS Actions

Οι ενέργειες (actions) στο ROS αποτελούν ένα μηχανισμό επικοινωνίας που επιτρέπει την εκτέλεση πιο σύνθετων και μακροχρόνιων λειτουργιών σε σχέση με τις υπηρεσίες (services). Σε αντίθεση με τις υπηρεσίες που παρέχουν ένα μοντέλο αιτήματος-απάντησης με άμεση απόκριση, οι ενέργειες επιτρέπουν την εκτέλεση εργασιών που μπορεί να διαρκέσουν μεγάλο χρονικό διάστημα, προσφέροντας παράλληλα δυνατότητα ανατροφοδότησης κατά τη διάρκεια της εκτέλεσης και δυνατότητα ακύρωσης της ενέργειας [32].

Αυτός ο μηχανισμός επικοινωνίας υλοποιείται μέσω τριών μηνυμάτων: το αίτημα της ενέργειας (**goal**), η ανατροφοδότηση (**feedback**), και το αποτέλεσμα (**result**). Ο κόμβος που ζητά την εκτέλεση της ενέργειας ονομάζεται **client** και αποστέλλει το αίτημα στον εξυπηρετητή της ενέργειας, ο οποίος ονομάζεται **server**. Ο server εκτελεί την εργασία και ανά τακτά χρονικά διαστήματα στέλνει feedback στον client για την πρόοδο της ενέργειας. Όταν η εργασία ολοκληρωθεί, αποστέλλει το result στον client. Σχηματικά [32]:



Σχήμα 3.2: Σχεδιάγραμμα επικοινωνίας μεταξύ client και server της actionlib.

Αυτός ο μηχανισμός είναι ιδιαίτερα χρήσιμος σε εφαρμογές όπου η εργασία μπορεί να είναι χρονοβόρα ή να απαιτεί συνεχή παρακολούθηση της προόδου ή ακόμα και ακύρωση της, όπως για παράδειγμα κατά τη μετακίνηση ενός ρομπότ σε έναν μακρινό στόχο [32].

3.2.4 Το εργαλείο RQT

Το εργαλείο rqt αποτελεί ένα από τα βασικότερα γραφικά εργαλεία στο οικοσύστημα του ROS για την οπτικοποίηση, την παρακολούθηση, και τη διαχείριση των διαφόρων στοιχείων ενός ρομποτικού συστήματος. Το rqt είναι ένα γραφικό περιβάλλον βασισμένο στο Qt, το οποίο παρέχει μια συλλογή από plugins που επιτρέπουν στους χρήστες να διαχειρίζονται και να αλληλεπιδρούν με τα διάφορα nodes, topics, services και άλλους πόρους ενός ROS συστήματος [33].

Η αρχιτεκτονική του rqt είναι επεκτάσιμη, επιτρέποντας την ανάπτυξη και προσθήκη νέων plugins που μπορούν να παρέχουν επιπλέον λειτουργικότητες ανάλογα με τις ανάγκες του χρήστη. Μέσα από το rqt, ο χρήστης έχει τη δυνατότητα να επιλέγει ποια plugins θέλει να ενεργοποιήσει, να τα οργανώνει σε διάφορες καρτέλες (tabs) και να τα προσαρμόζει ανάλογα με τις απαιτήσεις της εφαρμογής του [33].

Μεταξύ των πιο κοινών plugins που προσφέρει το rqt είναι τα εξής [34]:

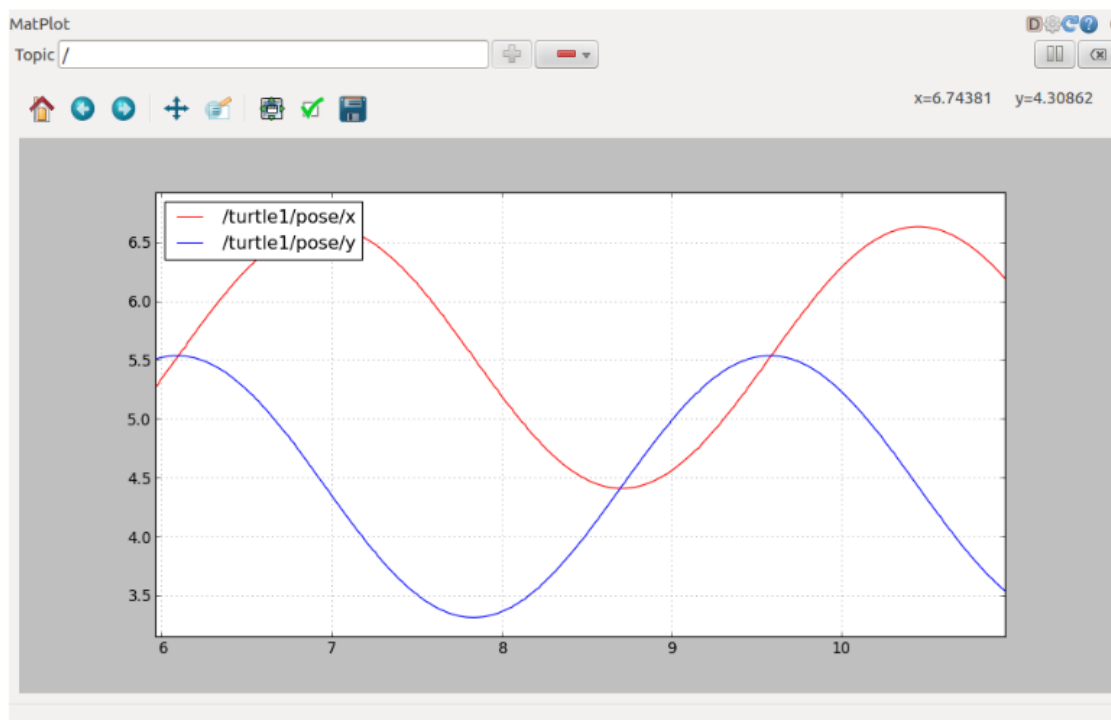
- **rqt_graph**: Επιτρέπει την οπτικοποίηση του γραφήματος επικοινωνίας των nodes σε ένα σύστημα ROS, όπου απεικονίζονται οι σχέσεις μεταξύ των nodes, topics, και services. Αυτή η οπτική αναπαράσταση βοηθά στην κατανόηση της ροής δεδομένων στο σύστημα και στην αναγνώριση πιθανών προβλημάτων στην επικοινωνία. Ένα απλό παράδειγμα που δίνεται στη βιβλιογραφία κι εξηγεί τη σχέση μεταξύ των nodes μέσω των topics είναι το εξής [35]:



Σχήμα 3.3: Παράδειγμα χρήσης του rqt_graph.

Στο παραπάνω παράδειγμα βλέπουμε ότι το node με όνομα joint_state_publisher κάνει publish μηνύματα στο topic joint_states, στο οποίο έχει κάνει subscribe και διαβάζει τα μηνύματα το node με όνομα robot_state_publisher.

- **rqt_plot**: Επιτρέπει την γραφική αναπαράσταση των δεδομένων που δημοσιεύονται σε διάφορα topics σε πραγματικό χρόνο. Αυτή η δυνατότητα είναι ιδιαίτερα χρήσιμη για την παρακολούθηση των εξόδων αισθητήρων ή των αποτελεσμάτων από ελεγκτές και αλγόριθμους. Ένα παράδειγμα που δίνεται στη βιβλιογραφία και αναπαριστά γραφικά τις (x,y) συντεταγμένες ενός ρομπότ με όνομα turtle1 είναι το εξής [36]:

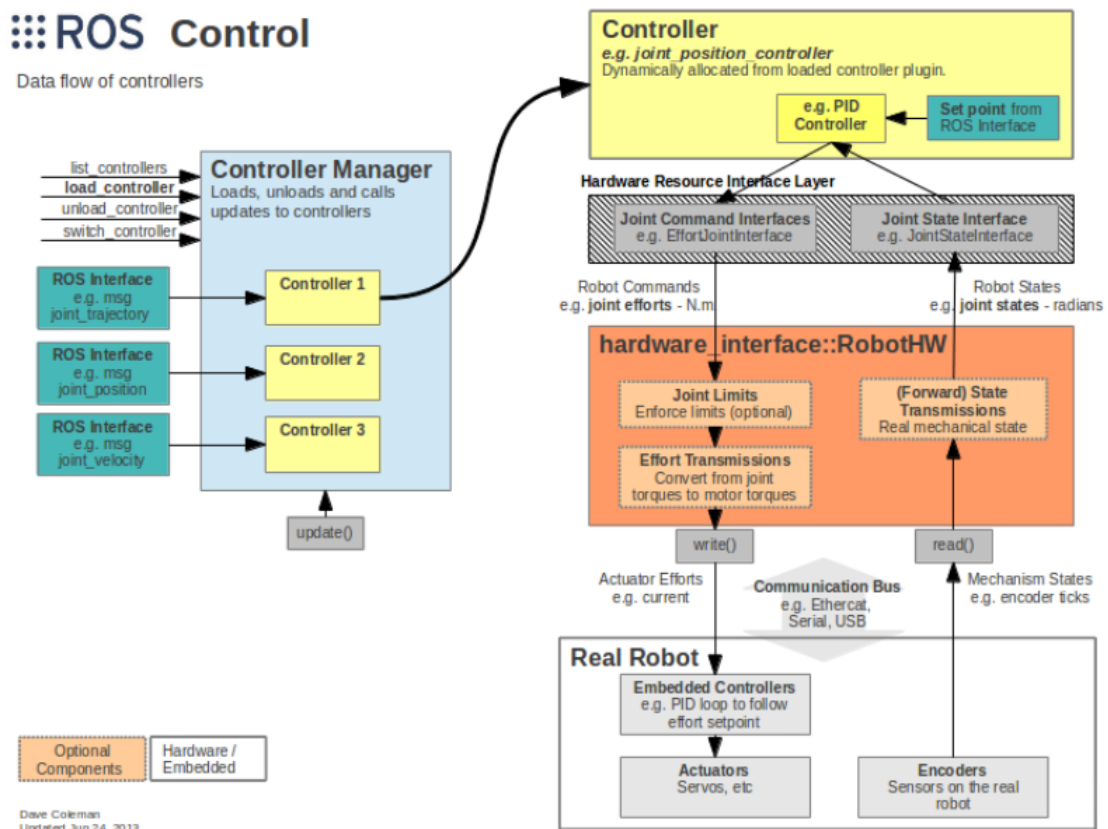


Σχήμα 3.4: Παράδειγμα χρήσης του `rqt_plot`.

- **rqt_bag**: Επιτρέπει την οπτικοποίηση και ανάλυση αρχείων `.bag`, τα οποία περιέχουν δεδομένα που έχουν καταγραφεί από τοπικούς κατά την εκτέλεση ενός συστήματος ROS. Το plugin αυτό είναι ιδιαίτερα χρήσιμο για την ανασκόπηση προηγούμενων εκτελέσεων και την ανάλυση της συμπεριφοράς του ρομποτικού συστήματος.
- **rqt_console**: Παρέχει μια κονσόλα όπου εμφανίζονται όλα τα μηνύματα καταγραφής (log messages) που παράγονται από τα διάφορα nodes του συστήματος. Μέσω του `rqt_console`, ο χρήστης μπορεί να φιλτράρει, να αναζητά και να μελετά τα μηνύματα αυτά για να εντοπίσει σφάλματα και να βελτιστοποιήσει τη λειτουργία του συστήματος.

3.2.5 ROS Control

Το `ros_control` αποτελεί μια από τις βασικότερες υποδομές του ROS για τη διαχείριση και τον έλεγχο ρομποτικών συστημάτων σε πραγματικό χρόνο. Πρόκειται για ένα σύνολο από πακέτα και διεπαφές λογισμικού που επιτρέπει την υλοποίηση και τον συντονισμό διαφορετικών ελεγκτών σε ρομποτικά συστήματα, όπως βραχίονες, κινητές πλατφόρμες, και άλλα είδη ρομπότ. Παρακάτω βλέπουμε σχηματικά και αναλύουμε τα βασικά στοιχεία που δομούν το ROS Control [37]:



Σχήμα 3.5: Η δομή του ROS Control.

Controllers

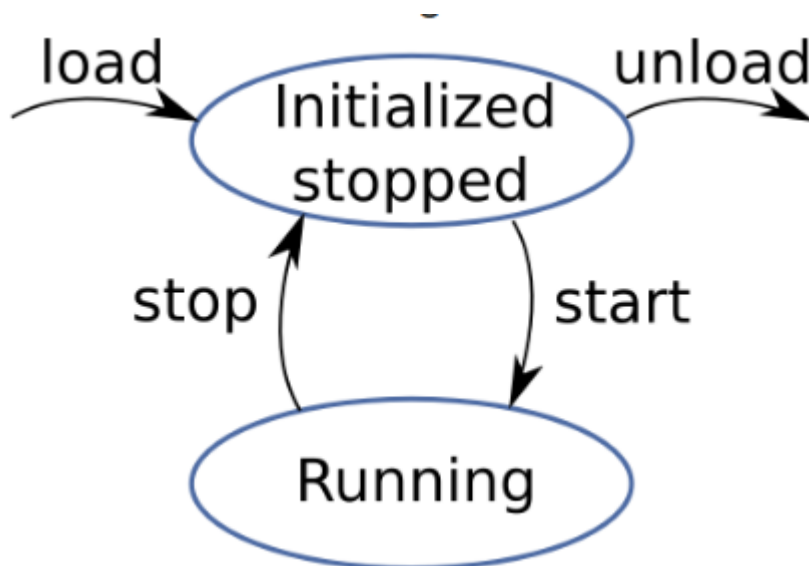
Οι controllers (ελεγκτές) αποτελούν το κεντρικό στοιχείο του πακέτου `ros_control` και είναι υπεύθυνοι για την εφαρμογή των αλγορίθμων ελέγχου που κατευθύνουν τη λειτουργία των ρομποτικών συστημάτων. Ένας ελεγκτής μπορεί να είναι υπεύθυνος για τον έλεγχο της θέσης, της ταχύτητας, της ροπής ή οποιουδήποτε άλλου φυσικού μεγέθους που σχετίζεται με την κίνηση ενός ρομπότ. Οι βασικές κατηγορίες ελεγκτών είναι οι εξής:

- **joint_state_controller**: Κάνει publish το state όλων των resources που είναι καταγεγραμμένα σε ένα `hardware_interface::JointStateInterface`, σε ένα topic τύπου `sensor_msgs/JointState`.
- **position_controllers**: Δίνουν εντολή επιθυμητής θέσης στο `HardwareInterface`.
- **velocity_controllers**: Δίνουν εντολή επιθυμητής ταχύτητας στο `HardwareInterface`.
- **effort_controllers**: Δίνουν εντολή επιθυμητού effort (ροπής ή/και δύναμης) στο `HardwareInterface`.

Controller Manager

Ο `controller_manager` είναι το σύστημα διαχείρισης των ελεγκτών στο `ros_control`. Αυτός ο διαχειριστής αναλαμβάνει την ενεργοποίηση, την απενεργοποίηση και τη διαχείριση πολλα-

πλών ελεγκτών που λειτουργούν παράλληλα στο ρομποτικό σύστημα. Μέσω του `controller_manager`, είναι δυνατόν να γίνεται δυναμική εναλλαγή μεταξύ διαφορετικών ελεγκτών, να προστίθενται νέοι ελεγκτές ή να αφαιρούνται υφιστάμενοι, ανάλογα με τις απαιτήσεις της εκάστοτε εφαρμογής, καθώς επίσης και να αλλάζουν καταστάσεις. Σχηματικά, οι δυνατές καταστάσεις στι οποίες μπορεί να βρεθεί ένας ελεγκτής είναι οι εξής [38]:



Σχήμα 3.6: Δυνατές καταστάσεις ενός ελεγκτή.

Hardware Interface

Το `hardware_interface` αποτελεί τον συνδετικό κρίκο μεταξύ των ελεγκτών και του φυσικού υλικού του ρομπότ. Πρόκειται για μια αφαιρετική διεπαφή (abstract interface) που επιτρέπει στους ελεγκτές να επικοινωνούν με τους ενεργοποιητές και τους αισθητήρες χωρίς να χρειάζεται να γνωρίζουν τις λεπτομέρειες της υλοποίησης του υλικού [37].

Το `hardware_interface` χωρίζεται σε διάφορους τύπους, όπως `PositionJointInterface`, `VelocityJointInterface`, και `EffortJointInterface`, ανάλογα με το είδος του ελέγχου που θέλουμε να ασκήσουμε. Αυτές οι διεπαφές παρέχουν μεθόδους για την ανάγνωση των τρεχουσών καταστάσεων (π.χ. θέσεις, ταχύτητες) και για την αποστολή εντολών στους ενεργοποιητές.

3.2.6 Command line ROS commands

Το ROS παρέχει μια πληθώρα εντολών οι οποίες εκτελούνται στο `command line` και είναι απαραίτητες για τη διαχείριση και η αλληλεπίδραση με τα διάφορα στοιχεία του συστήματος. Παρακάτω βλέπουμε μερικές από τις βασικές εντολές που χρησιμοποιούνται συχνά [39]:

roscore

Η εντολή `roscore` είναι η κεντρική διαδικασία που απαιτείται για την λειτουργία ενός ROS συστήματος, καθώς ξεκινά και διαχειρίζεται τις βασικές υπηρεσίες και διεργασίες που χρειάζονται για την επικοινωνία μεταξύ των κόμβων. Όταν εκτελείται αυτή η εντολή, ξεκινούν τα εξής στοιχεία [40]:

- **ROS Master:** Αποτελεί τον κεντρικό διαχειριστή των nodes στο ROS. Παρέχει έναν πίνακα καταλόγου, μέσω του οποίου οι κόμβοι μπορούν να βρουν και να επικοινωνήσουν μεταξύ τους. Κάθε node που εκτελείται σε ένα ROS σύστημα συνδέεται πρώτα με τον ROS Master για να δηλώσει τα messages, services και topics που προσφέρει [41].
- **Parameter Server:** Διαθέτει έναν διακομιστή παραμέτρων, όπου αποθηκεύονται και διαχειρίζονται διάφορες παράμετροι που χρειάζονται για την εκτέλεση των nodes. Αυτές οι παράμετροι είναι διαθέσιμες σε όλα τα nodes που συνδέονται με το σύστημα [42].
- **rosout:** Είναι το node που συλλέγει τα αρχεία καταγραφής (log files) από τους άλλους κόμβους και τα εμφανίζει στο console ή τα αποθηκεύει για μελλοντική ανάλυση [43].

roslaunch

Η εντολή `roslaunch` χρησιμοποιείται για την εκτέλεση ενός κόμβου που έχει δημιουργηθεί σε ένα συγκεκριμένο ROS πακέτο. Η σύνταξη της εντολής είναι:

```
roslaunch [package_name] [node_name]
```

Για παράδειγμα, αν θέλουμε να εκτελέσουμε το node `talker` από το πακέτο `beginner_tutorials`, η εντολή θα είναι:

```
roslaunch beginner_tutorials talker
```

rostopic

Η εντολή `rostopic` επιτρέπει την αλληλεπίδραση με τα ROS topics. Μερικές κοινές χρήσεις περιλαμβάνουν [44]:

- `rostopic list`: Εμφανίζει μια λίστα με όλα τα διαθέσιμα θέματα στον τρέχοντα ROS Master.
- `rostopic echo [topic_name]`: Εμφανίζει τα messages που δημοσιεύονται ζωντανά στο συγκεκριμένο topic.
- `rostopic pub [topic_name] [msg_type] [data] [args]`: Δημοσιεύει ένα message σε ένα topic. Τα [args] παρέχουν στον χρήστη μια πληθώρα (προαιρετικών) επιλογών, όπως για παράδειγμα το να μπορεί να δημοσιεύσει το μήνυμα στο topic στιγμιαία ή περιοδικά.

3.2.7 ROS Network

Σε ένα ρομποτικό σύστημα είναι πολύ πιθανό να τρέχουν ROS nodes παράλληλα σε μια πληθώρα υπολογιστών, όπως ο υπολογιστής που εκτελεί motion capture, ο υπολογιστής που τρέχει τον νόμο ελέγχου και ο υπολογιστής που μετατρέπει τις εντολές ροπών σε ρεύματα για τους κινητήρες των αρθρώσεων. Είναι λοιπόν αναγκαίο τα nodes των διαφορετικών υπολογιστών να επικοινωνούν σε ένα κοινό περιβάλλον ROS, με κοινά ROS Topics, Services και Actions. Η επικοινωνία αυτή εδραιώνεται με την δημιουργία ενός ROS Network, όπου ουσιαστικά ο ROS Master τρέχει μόνο σε έναν υπολογιστή (τρέχοντας την εντολή `roscore`)

και οι υπόλοιποι υπολογιστές, δεδομένου ότι βρίσκονται σε κοινό δίκτυο, επικοινωνούν στο περιβάλλον ROS του κεντρικού υπολογιστή. Για να γίνει αυτό, στο `bashrc` αρχείο κάθε υπολογιστή (συμπεριλαμβανομένου και του κεντρικού) ορίζεται η μεταβλητή `ROS_MASTER_URI` και `ROS_HOSTNAME` να είναι η IP διεύθυνση του κεντρικού υπολογιστή που τρέχει το ROS Master. Επιπλέον, ορίζεται η μεταβλητή `ROS_IP` να είναι η προσωπική IP διεύθυνση του κάθε υπολογιστή. Με αυτά τα απλά βήματα εξασφαλίζεται η *real-time* επικοινωνία των υπολογιστών σε ένα κοινό δίκτυο ROS [45] [46].

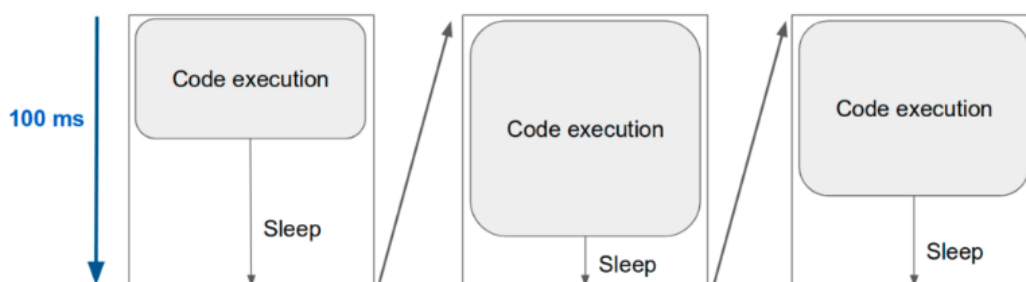
3.2.8 ROS Time και ROS Rate

ROS Time

Το περιβάλλον ROS παρέχει στον χρήστη το πακέτο `rostime`, το οποίο περιέχει κρίσιμα εργαλεία τόσο σε C++ αλλά και σε Python με σκοπό τη διαχείριση του χρόνου, δηλαδή των χρονικών στιγμών και της χρονικής διάρκειας [47]. Στο ρομποτικό μας σύστημα χρησιμοποιήθηκαν σε C++ αντικείμενα των κλάσεων `ros::Time` και `ros::Duration`, έτσι ώστε μέσω του `ros::Time` να γνωρίζουμε σημαντικές χρονικές στιγμές, όπως τη στιγμή της εκκίνησης του πειράματος και κάνοντας πράξεις με αυτές τις στιγμές να μπορούμε μέσω του `ros::Duration` να υπολογίζουμε τα σωστά ορίσματα για τις συναρτήσεις που εξαρτώνται από τον χρόνο, όπως το πολυώνυμο παρεμβολής που δίνεται από την εξίσωση 2.4 [48].

ROS Rate

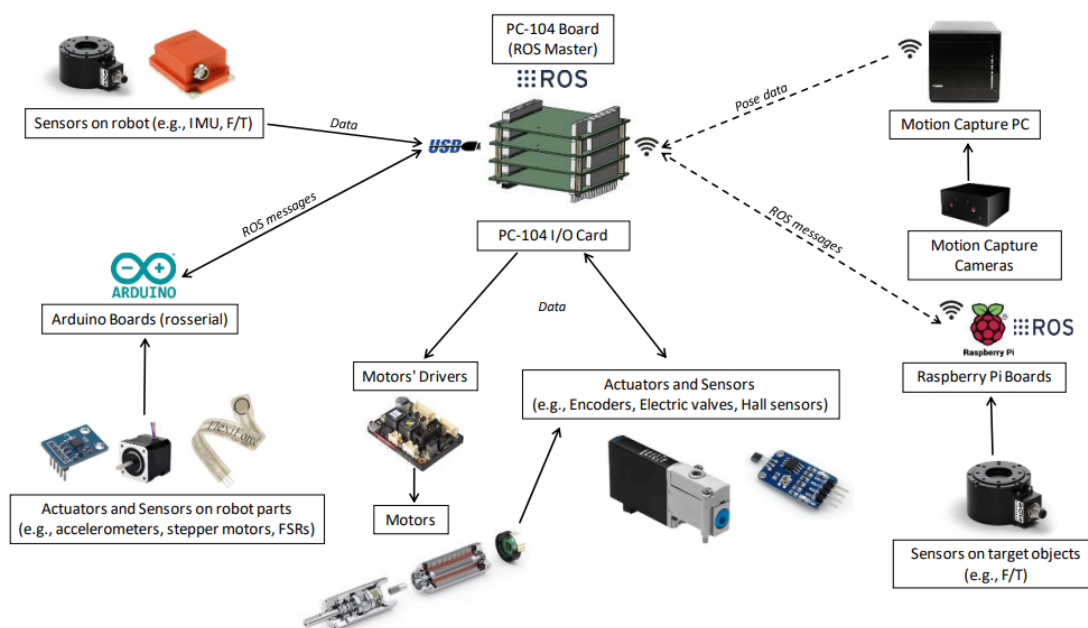
Για την ομαλή επικοινωνία και συνεργασία των ποικίλων ROS nodes σε ένα ROS Network, είναι θεμιτή η χρήση της κλάσης `ros::Rate`, η οποία παρέχεται από το ROS. Η κλάση αυτή χρησιμοποιείται για τον ακριβή έλεγχο του ρυθμού επανάληψης ενός βρόχου, επιτρέποντας τον καθορισμό της συχνότητας εκτέλεσης ενός ROS node [49]. Η μεγάλη χρησιμότητα του ROS Rate έγκειται στην χρήση της απλής συνάρτησης `sleep`, η οποία, αφού έχεις ορίσει την επιθυμητή συχνότητα του βρόχου, εξασφαλίζει πως ο χρόνος εκτέλεσης του κώδικα, μαζί με το “sleep” θα παραμένει σταθερός. Για παράδειγμα, αν αρχικοποιήσουμε ένα αντικείμενο της κλάσης `ros::Rate` ως εξής: `ros::Rate loop_rate(10)`, δηλαδή θέλουμε ο βρόχος να τρέχει στα 10 Hz και καλούμε την συνάρτηση `loop_rate.sleep()` στο τέλος του βρόχου, τότε ο βρόχος θα εκτελείται 1 φορά ανά 100 ms, ανεξάρτητα αν χρειάζεται 1 ms ή 99 ms για να ολοκληρωθεί. Το παραπάνω παράδειγμα γίνεται εύκολα κατανοητό με το εξής σχήμα [50]:



Σχήμα 3.7: Η λειτουργία του ROS Rate.

3.3 Το ROS του Κηφέα

Ο διαστημικός εξομοιωτής του εργαστηρίου CSL-NTUA Κηφέας είναι ένα σύνθετο ρομποτικό σύστημα το οποίο αποτελείται από πολλά επιμέρους υποσυστήματα και ηλεκτρονικά στοιχεία. Στην ενότητα αυτή θα προσεγγίσουμε τον Κηφέα από την πλευρά του ROS λογισμικού. Τα ηλεκτρονικά στοιχεία του Κηφέα και η μεταξύ τους επικοινωνία παρουσιάζεται συνοπτικά στο εξής σχήμα [51]:



Σχήμα 3.8: Τα ηλεκτρονικά στοιχεία του Κηφέα.

Είναι εμφανές πως ο Κηφέας και τα περιφερειακά περιλαμβάνουν μια πληθώρα ηλεκτρονικών στοιχείων τα οποία λαμβάνουν πληροφορίες και μέσω του ROS τα διαβιβάζουν στον κεντρικό υπολογιστή (PC-104), ο οποίος εκτελεί τους αλγορίθμους ελέγχου κι εξάγει τις κατάλληλες εντολές στους κινητήρες και τους actuators [51]. Παρακάτω θα δούμε τα απαραίτητα ROS nodes που χρειάζονται να τρέχουν ώστε να καταστεί δυνατή η εύρωστη επικοινωνία των υποσυστημάτων και λειτουργία του Κηφέα γενικότερα.

3.3.1 Το προϋπάρχον λογισμικό

Τα ROS nodes που τρέχουν κατά τη διάρκεια λειτουργίας του Κηφέα ποικίλουν σε λειτουργίες και προέρχονται από διαφορετικές πηγές. Πιο συγκεκριμένα [52]:

Nodes υλοποιημένα από το ROS

Οι βασικές λειτουργίες των ελεγκτών καλύπτονται από τα έτοιμα nodes που παρέχονται από το πακέτο controller manager. Τα renamed nodes που χρησιμοποιούνται από τον Κηφέα είναι τα εξής:

- **cepheus_controller_spawner**: Είναι το υπεύθυνο node για την εκκίνηση και διαχείριση των ελεγκτών του Κηφέα.

- **cephesus_state_publisher**: Είναι το υπεύθυνο node για το publish των καταστάσεων (states) των αρθρώσεων του Κηφέα.

Nodes υλοποιημένα από το Vicon

Το Vicon είναι ένα σύστημα καταγραφής κίνησης που χρησιμοποιεί κάμερες υπερύθρων για την παρακολούθηση markers στο χώρο, παρέχοντας ακριβείς μετρήσεις θέσης και προσανατολισμού. Όπως θα δούμε σε επόμενο κεφάλαιο που αφορά τη πειραματική διάταξη, ο Κηφέας χρησιμοποιεί το Vicon για το localization του κατά τη διάρκεια του πειράματος. Η ενσωμάτωσή του στο ROS γίνεται μέσω της βιβλιοθήκης `vicon_bridge`, της οποίας βασικό node είναι το συνονόματο **vicon_bridge node**. Αυτό είναι υπεύθυνο για την ανάκτηση των δεδομένων κίνησης από το hardware του Vicon και την αποστολή τους ως messages στα αντίστοιχα topics [53].

Nodes από υλοποιημένα την ομάδα του εργαστηρίου CSL-NTUA

Ο high level έλεγχος του Κηφέα μεταφράζεται σε πραγματική κίνηση του ρομπότ λόγω της επικοινωνίας της κεντρικής μονάδας ελέγχου με το υπόλοιπο hardware. Η επικοινωνία αυτή έχει υλοποιηθεί μέσω ROS nodes από την ομάδα του εργαστηρίου. Πιο συγκεκριμένα [52]:

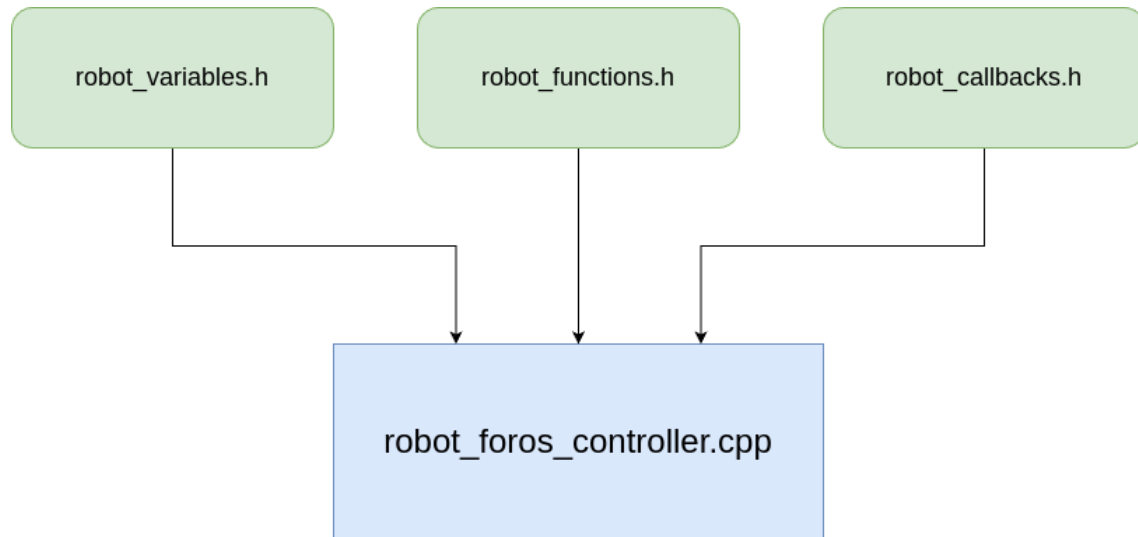
- **cephesus_interface_node**: Το `cephesus_interface_node` είναι το node το οποίο πλαισιώνει την επικοινωνία μεταξύ του software και του hardware του Κηφέα. Κάνει subscribe στα topics **cmd_thrust** και **cmd_torque**, στα οποία κάνει publish τις εντολές κίνησης ο εκάστοτε ελεγκτής. Επεξεργάζεται τα μηνύματα αυτά και τα μετατρέπει σε εντολές κατάλληλες για να εκτελεστούν από το hardware. Κατά την εκκίνηση του node, δημιουργείται ένα αντικείμενο της κλάσης **CepheusHW**, το οποίο περιλαμβάνει κρίσιμα στοιχεία για το ρομπότ, όπως η επιθυμητή ροπή για κάθε άρθρωση, οι γωνίες των αρθρώσεων, οι εντολές ρεύματος για τους κινητήρες και descriptors για τις πλακέτες του PC104. Κάποια από αυτά τα στοιχεία είναι οργανωμένα σε πίνακες, με κάθε θέση να αντιστοιχεί σε ένα εξάρτημα ή άρθρωση του ρομπότ. Ουσιαστικά, σε κάθε iteration, το `cephesus_interface` node διαβάζει τις επιθυμητές ροπές για κάθε κινητήρα (στην αντίστοιχη άρθρωση). Στη συνέχεια, μετατρέπει τη ροπή αυτή σε επιθυμητό ρεύμα κινητήρα, σύμφωνα με την σχέση $\tau = Ki$, όπου τ η ροπή, i το ρεύμα και K η σταθερά ροπής κάθε κινητήρα (δίνεται στα manuals). Τέλος, μετατρέπει το ρεύμα σε width ενός PWM, τον οποίο παλμό στέλνει στους drivers. Επιπλέον, διαβάζει τα counts από τους encoders και τους μετατρέπει σε γωνίες των αρθρώσεων.

3.3.2 Η προσθήκη για impedance control

Για την επίτευξη του impedance control είναι αναγκαία η ανάπτυξη ενός high level αλγορίθμου ελέγχου. Τον σκοπό αυτό επιτελεί ένα καινούργιο node με το όνομα **robot_foros_controller_node**, το οποίο βρίσκεται στο αρχείο **robot_foros_controller.cpp**.

Δομή του controller

Για την καλύτερη οργάνωση και αναγνωσιμότητα του κώδικα, ο έλεγχος του Κηφέα χωρίζεται στα εξής αρχεία:



Σχήμα 3.9: Δομή του υλοποιημένου ελεγκτή.

- **robot_foros_controller.cpp:** Είναι το βασικό αρχείο το οποίο περιέχει το ROS node. Εδώ ορίζονται όλοι οι ROS subscribers, οι οποίοι κάνουν subscribe σε ROS topics για να αντλήσουν πληροφορίες για την θέση και τη ταχύτητα του ρομπότ, καθώς επίσης και για την δύναμη επαφής που ασκείται στον end effector. Επίσης, ορίζονται οι ROS publishers οι οποίοι σε κάθε βήμα ελέγχου κάνουν publish στα κατάλληλα topics τις επιθυμητές ροπές στις αρθρώσεις και στο reaction wheel. Επιπλέον, εδώ δομείται η λογική του controller καθώς τρέχει το main control loop, στο οποίο, ανάλογα με τη φάση του πειράματος, ανάβουν ή σβήνουν κατάλληλα boolean flags και καλούνται οι συναρτήσεις υπολογισμού τροχιάς, σφάλματος και ροπής. Τέλος, δίνεται η εντολή για το κλείσιμο της αρπάγης και η ολοκλήρωση και καταγραφή του πειράματος σε bag files, για μετέπειτα ανάλυση.
- **robot_variables.cpp:** Εδώ ορίζονται όλες οι global μεταβλητές, απαραίτητες για τον έλεγχο του ρομπότ.
- **robot_callbacks.cpp:** Εδώ ορίζονται όλες οι callback συναρτήσεις, οι οποίες καλούνται κάθε φορά που ένας ROS subscriber του main αρχείου διαβάσει ένα νέο μήνυμα στο αντίστοιχο topic.
- **robot_functions.cpp:** Εδώ ορίζονται όλες οι απαραίτητες συναρτήσεις που χρησιμοποιούνται κατά τον έλεγχο του ρομπότ. Η initialiseParameters() αρχικοποιεί τις μεταβλητές που περιγράφουν τα αδρανειακά και γεωμετρικά χαρακτηριστικά του Κηφέα. Η calculateTrajectoryPolynomials() υπολογίζει τους συντελεστές του τόξου παραμετροποίησης (εξίσωση 2.4), σύμφωνα με τις επιθυμητές αρχικές και τελικές συνθήκες. Η finalTrajectories() υπολογίζει το επόμενο σημείο της επιθυμητής τροχιάς για κάθε

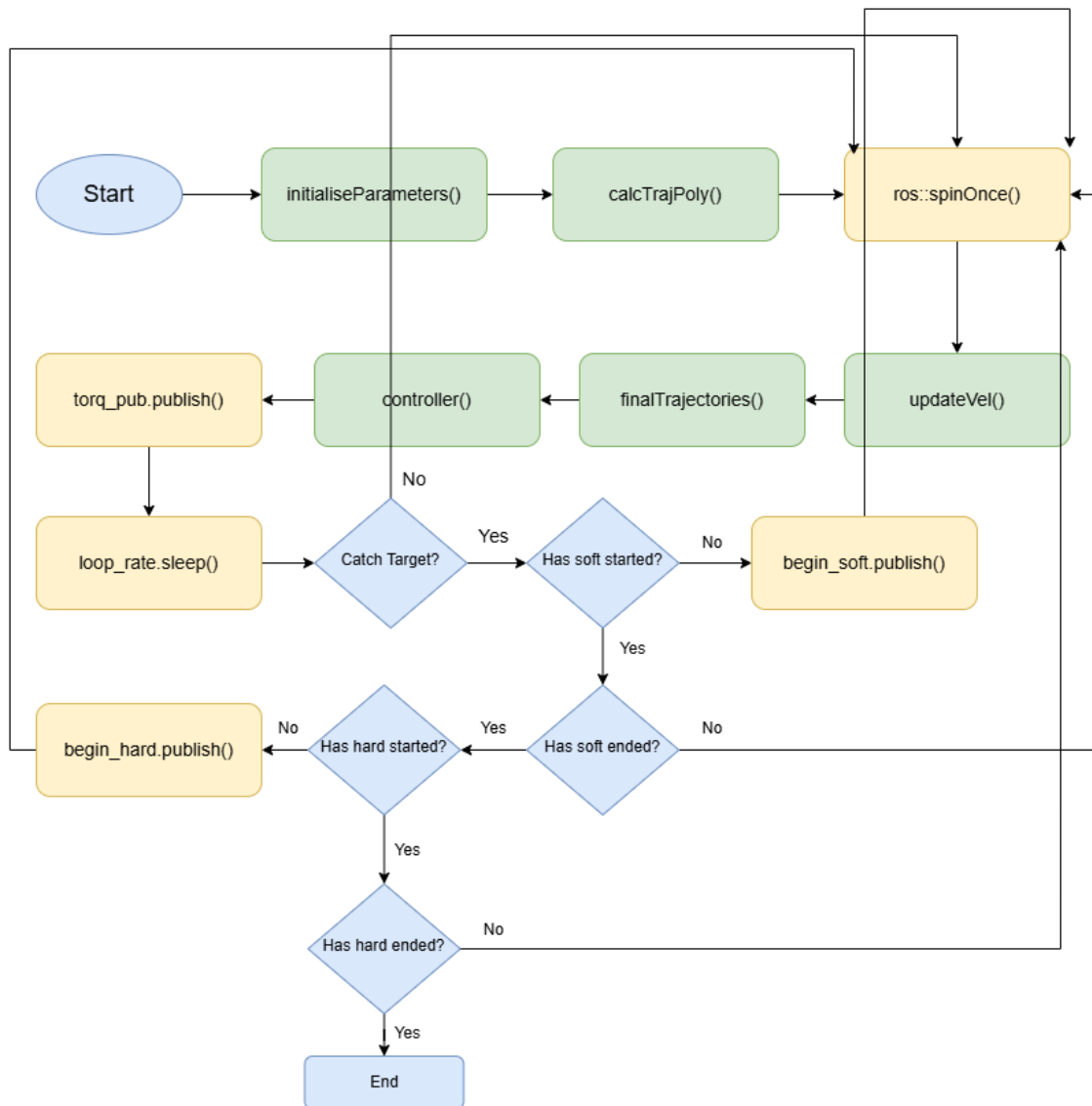
χρονική στιγμή . Τέλος, η `controller()` είναι η συνάρτηση που σε κάθε χρονικό βήμα εφαρμόζει το `impedance control` και υπολογίζει τις ροπές που θα ασκηθούν στις αρθρώσεις και στο `reaction wheel`.

Η δομή του υλοποιημένου λογισμικού ακολουθάει το design pattern ενός **native controller** σε C++ και ενός **ROS Wrapper**. Δηλαδή, οι συναρτήσεις στο αρχείο `robot_functions.h` και οι μεταβλητές στο `robot_variables.h` μπορούν να οριστούν και να τρέξουν ανεξάρτητα από το ROS. Ωστόσο, το αρχείο `robot_foros_controller.cpp` και το αρχείο `robot_callbacks.h` λειτουργούν ως ROS Wrapper, καθώς εδραιώνουν την επικοινωνία με τα υπόλοιπα ROS λογισμικά και καθορίζουν από που έρχονται τα `inputs` και που θα στέλνονται τα `outputs`. Αξίζει να σημειωθεί ότι οι ίδιες συναρτήσεις στο `robot_functions.h` μπορούν να τρέξουν αν αλλάξει ο τρόπος που έρχονται τα `inputs` και φεύγουν τα `outputs`, ακόμα και χωρίς τη χρήση του ROS.

Η λογική του πειράματος είναι η εξής: Πρώτα, αρχικοποιούνται οι παράμετροι του εκάστοτε πειράματος, όπως είναι τα αδρανειακά και τα γεωμετρικά χαρακτηριστικά του Κηφέα και υπολογίζεται το τόξο παραμετροποίησης, σύμφωνα με τις δοθείσες αρχικές και τελικές συνθήκες. Στη συνέχεια, σε κάθε χρονικό βήμα, ο ελεγκτής: ανανεώνει από το περιβάλλον ROS τις τιμές των `input` από τους αισθητήρες, υπολογίζει τις τρέχουσες ταχύτητες, υπολογίζει την τιμή της επιθυμητής τροχιάς για θέση και ταχύτητα και σύμφωνα με αυτά εφαρμόζει τον νόμο ελέγχου εμπέδησης ώστε να υπολογίσει τις ροπές που πρέπει να ασκηθούν από τους κινητήρες. Οι ροπές αυτές γίνονται `published` στο κατάλληλο `topic` και ο ελεγκτής περιμένει ένα χρονικό διάστημα.

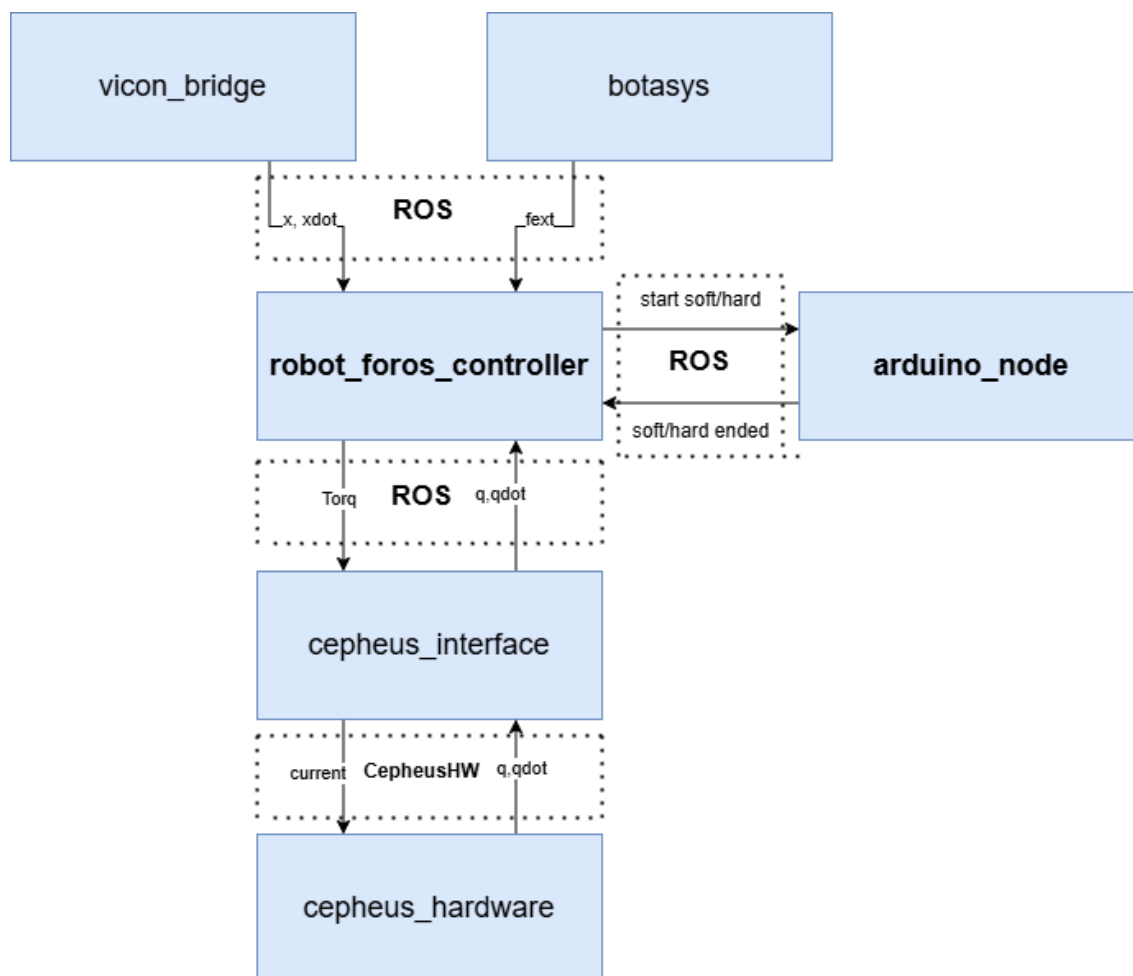
Έπειτα ακολουθεί μια σειρά λογικών εντολών: Αν το ΤΣΔ ικανοποιεί την συνθήκη που έχουμε ορίσει εμείς για να πιάσει τον στόχο (στο προκειμένο παράδειγμα η απόσταση του από τον στόχο να είναι κάτω από ένα δοθέν `threshold` για ένα δοθέν χρονικό διάστημα), ξεκινάει η διαδικασία σύλληψης του στόχου. Δηλαδή, πρώτα στέλνει εντολή στο `arduino` του `gripper` μέσω ROS να ξεκινήσει το `softgrip` και περιμένει μέχρι να λάβει μήνυμα από το `arduino` ότι το `softgrip` τελείωσε. Στη συνέχεια, στέλνει εντολή στο `arduino` να ξεκινήσει το `hardgrip` και περιμένει μέχρι να λάβει μήνυμα ότι το `hardgrip` τελείωσε. Τότε, το πείραμα έχει ολοκληρωθεί.

Σχηματικά:



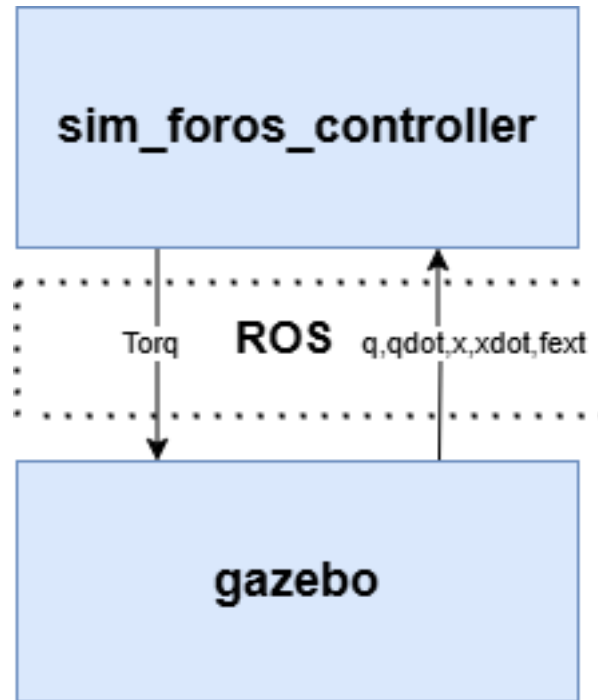
Σχήμα 3.10: Λογική ελεγκτή κατά το πείραμα.

Παρακάτω φαίνεται σχηματικά η επικοινωνία των βασικών αρχείων κατά τη διάρκεια του πειράματος:



Σχήμα 3.11: Επικοινωνία των βασικών αρχείων κατά το πείραμα στον εξομοιωτή.

Αξίζει να σημειωθεί για τα πειράματα του ελεγκτή στον προσομοιωτή Gazebo (θα τον δούμε στο επόμενο κεφάλαιο), ο native controller μένει ίδιος και αυτό που αλλάζει στην ουσία είναι ο ROS Wrapper, δηλαδή η επικοινωνία με τον προσομοιωτή (σε αντίθεση με το αληθινό hardware). Με το παρακάτω σχήμα τονίζεται το πόσο διευκολύνεται η επικοινωνία του ελεγκτή με την προσομοίωση, σε σχέση με το αληθινό hardware:



Σχήμα 3.12: Επικοινωνία των βασικών αρχείων κατά το πείραμα στη προσομοίωση.

Το περιβάλλον προσομοίωσης Gazebo

4.1 Εισαγωγή

Το Gazebo είναι ένας ισχυρός και ευρέως χρησιμοποιούμενος προσομοιωτής στον τομέα της ρομποτικής, σχεδιασμένος για να παρέχει ένα ρεαλιστικό περιβάλλον προσομοίωσης τρισδιάστατων μοντέλων ρομπότ και άλλων φυσικών συστημάτων. Με την ενσωμάτωση ακριβών physics engines, όπως ο ODE (Open Dynamics Engine), το Gazebo επιτρέπει την προσομοίωση της κίνησης και της αλληλεπίδρασης μεταξύ διαφορετικών αντικειμένων, λαμβάνοντας υπόψη δυνάμεις όπως η βαρύτητα, οι τριβές και οι συγκρούσεις. Έτσι, αποτελεί ένα δυνατό εργαλείο για την ανάπτυξη και την επαλήθευση αλγορίθμων ελέγχου καθώς επίσης ένα προστάδιο πριν το πραγματικό πείραμα. Δηλαδή, χωρίς την ανάγκη χρήσης του αληθινού ρομπότ και της πιθανής φθοράς ή δυσλειτουργίας του hardware, δοκιμάζεται η λειτουργικότητα των high level αλγορίθμων ελέγχου σε ένα πιο ρεαλιστικό και real time responsive περιβάλλον [54].

4.2 Ο ODE physics engine

Ο ODE υποστηρίζει την προσομοίωση των αλληλεπιδράσεων μεταξύ άκαμπτων σωμάτων, λαμβάνοντας υπόψη δυνάμεις όπως τη βαρύτητα, τις τριβές και τις συγκρούσεις. Χρησιμοποιείται ευρέως για την προσομοίωση της κίνησης και αλληλεπίδρασης ρομποτικών συστημάτων, επιτρέποντας την ακριβή αναπαράσταση της δυναμικής συμπεριφοράς των ρομπότ και των μηχανικών τους υποσυστημάτων [55]. Επιπλέον, για την ενσωμάτωση του στον προσομοιωτή Gazebo, ο ODE υποστηρίζει μια σειρά από παραμέτρους που ρυθμίζουν τη φυσική συμπεριφορά των αντικειμένων και των αλληλεπιδράσεών τους. Παρακάτω παρατίθενται μερικές από τις βασικές κατηγορίες παραμέτρων που μπορούν να ρυθμιστούν από τον χρήστη [56]:

Παράμετροι τριβής

- μ_1 : Συντελεστής τριβής μεταξύ των επιφανειών για την πρώτη κατεύθυνση.
- μ_2 : Συντελεστής τριβής μεταξύ των επιφανειών για την δεύτερη κατεύθυνση (συνήθως ισούται με το μ_1).

Παράμετροι επαφής

- k_p : Συντελεστής δυσκαμψίας (stiffness) για τις επαφές μεταξύ των σωμάτων.
- k_d : Συντελεστής απόσβεσης (damping) για τις επαφές μεταξύ των σωμάτων.

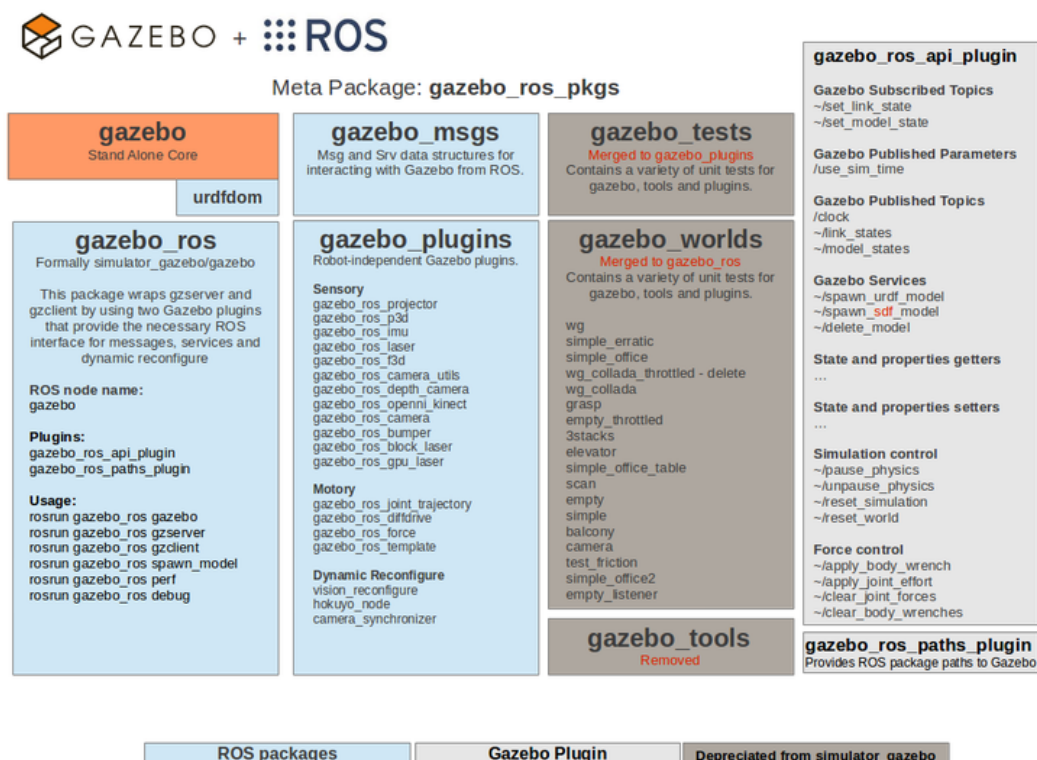
Άλλες παράμετροι

- `max_step_size`: Μέγιστο μέγεθος βήματος του χρονικού βρόχου της προσομοίωσης.
- `max_contacts`: Μέγιστος αριθμός επαφών που μπορεί να παραχθεί μεταξύ δύο οντοτήτων κατά τη σύγκρουση τους.

4.3 Διασύνδεση Gazebo με ROS

Η χρησιμότητα του προσομοιωτή Gazebo έγκειται στο γεγονός ότι μπορεί να συνεργάζεται άψογα και να ενσωματώνεται στο περιβάλλον ROS. Δηλαδή, το προσομοιωμένο ρομπότ δέχεται εντολές από το χρήστη (ή τον αλγόριθμο ελέγχου) μέσα από τα ROS topics και ο προσομοιωτής εξάγει σε πραγματικό χρόνο δεδομένα όπως η θέση και η ταχύτητα του ρομπότ, καθώς επίσης και τις δυνάμεις που δέχεται, σε μορφή μηνυμάτων σε αντίστοιχα ROS topics. Αυτό είναι δυνατό με τη χρήση του ROS μεταπακέτου (δηλαδή συνόλου από πακέτα) **gazebo_ros_pkgs**, το οποίο περιέχει χρήσιμα plugins, αλλά κι εξαιτίας των **URDF** αρχείων, τα οποία θα εξηγηθούν παρακάτω [57].

Παρακάτω φαίνονται σχηματικά τα περιεχόμενα του μεταπακέτου gazebo_ros_pkgs [57]:



Σχήμα 4.1: Τα περιεχόμενα του gazebo_ros_pkgs.

Unified Robot Description Format Αρχεία

Τα **URDF** (Unified Robot Description Format) αρχεία χρησιμοποιούνται ευρέως για την περιγραφή ρομποτικών συστημάτων στο ROS περιβάλλον αλλά και την προσομοίωση τους στο προσομοιωτή Gazebo. Η σύνταξή τους βασίζεται στην XML (Extensible Markup Language), μια ευέλικτη γλώσσα σήμανσης που επιτρέπει την οργάνωση και τη δομημένη αναπαράσταση δεδομένων. Μέσα από τα αρχεία URDF, περιγράφονται λεπτομερώς τα links και τα joints ενός ρομπότ καθώς και οι φυσικές ιδιότητες αυτών, όπως οι μάζες, τα κέντρα βάρους και οι αδράνειες [58].

Η περιγραφή των στοιχείων ενός ρομπότ στο URDF αρχείο βασίζεται στα διάφορα tags (ετικέτες) και την ενθλάκωση αυτών. Δηλαδή, για κάθε ιδιότητα ενός ρομποτικού στοιχείου γράφεται ένα tag όπου την περιγράφει. Όσον αφορά τα links ενός ρομπότ, τα κύρια tags τα οποία το περιγράφουν τις βασικές του ιδιότητες είναι τα εξής [59]:

- **visual tag:** Χρησιμοποιείται για την περιγραφή της οπτικής εμφάνισης ενός link στο περιβάλλον της προσομοίωσης. Στην ετικέτα αυτή, περιλαμβάνονται υποετικέτες που ορίζουν τη γεωμετρία (geometry tag) (π.χ., σφαίρα, κύλινδρος, κουτί) καθώς και το υλικό του link (material tag), η οποία καθορίζει το χρώμα ή την υφή της επιφάνειας.
- **inertial tag:** Καθορίζει τις αδρανειακές ιδιότητες του link, που περιλαμβάνουν τη μάζα, το κέντρο βάρους και το τανυστή αδράνειας. Αυτά τα στοιχεία είναι ζωτικής σημασίας για την ακριβή δυναμική προσομοίωση του ρομπότ, καθώς επηρεάζουν τη συμπεριφορά του κατά την κίνηση και τις αλληλεπιδράσεις του με το περιβάλλον.
- **collision tag:** Καθορίζει τις ιδιότητες σύγκρουσης ενός link στο περιβάλλον της προσομοίωσης. Περιλαμβάνει τις ίδιες γεωμετρικές πληροφορίες με το visual tag, αλλά χωρίς να απαιτείται η περιγραφή υλικού, αφού αφορά κυρίως τον υπολογισμό των επαφών και των αλληλεπιδράσεων με άλλα αντικείμενα στον προσομοιωτή.

Όσον αφορά την περιγραφή των αρθρώσεων (joints) μεταξύ των links, μερικά από τα βασικά tags που χρησιμοποιούνται είναι τα εξής [60]:

- **parent tag:** Ορίζει το γονικό link του joint. Είναι το σημείο αναφοράς στο οποίο συνδέεται το θοινι και μέσω του οποίου γίνεται η μετάδοση των δυνάμεων και των ροπών.
- **child tag:** Ορίζει το θυγατρικό link του joint. Είναι το link που επηρεάζεται άμεσα από τις κινήσεις του joint.
- **origin tag:** Καθορίζει τη θέση και τον προσανατολισμό του joint σε σχέση με το parent link. Περιλαμβάνει παραμέτρους όπως η μετατόπιση (xyz) και η περιστροφή (rpy - roll, pitch, yaw).
- **type tag:** Καθορίζει τον τύπο του joint, όπως revolute, prismatic, fixed, floating, continuous και planar. Ο τύπος του joint καθορίζει την κίνηση που μπορεί να εκτελεί το child link σε σχέση με το parent link, ορίζοντας έτσι τη συμπεριφορά του μηχανισμού.

- **dynamics tag**: Περιλαμβάνει παραμέτρους όπως το damping και το friction, οι οποίες καθορίζουν τις δυνάμεις που αντιστέκονται στην κίνηση του joint.

Αξίζει να σημειωθεί ότι για την ρεαλιστική προσομοίωση του κάθε κινητήρα που κινεί τα joints, χρησιμοποιείται το **transmission tag** το οποίο περιέχει διάφορα tags, όπως το joint στο οποίο συνδέεται, το **mechanicalReduction tag** που προσομοιώνει τις απώλειες του κινητήρα καθώς και το **hardwareInterface tag**, το οποίο είναι απαραίτητο για να δέχονται εντολές οι κινητήρες των joints από τους ROS controllers [61].

Gazebo plugins

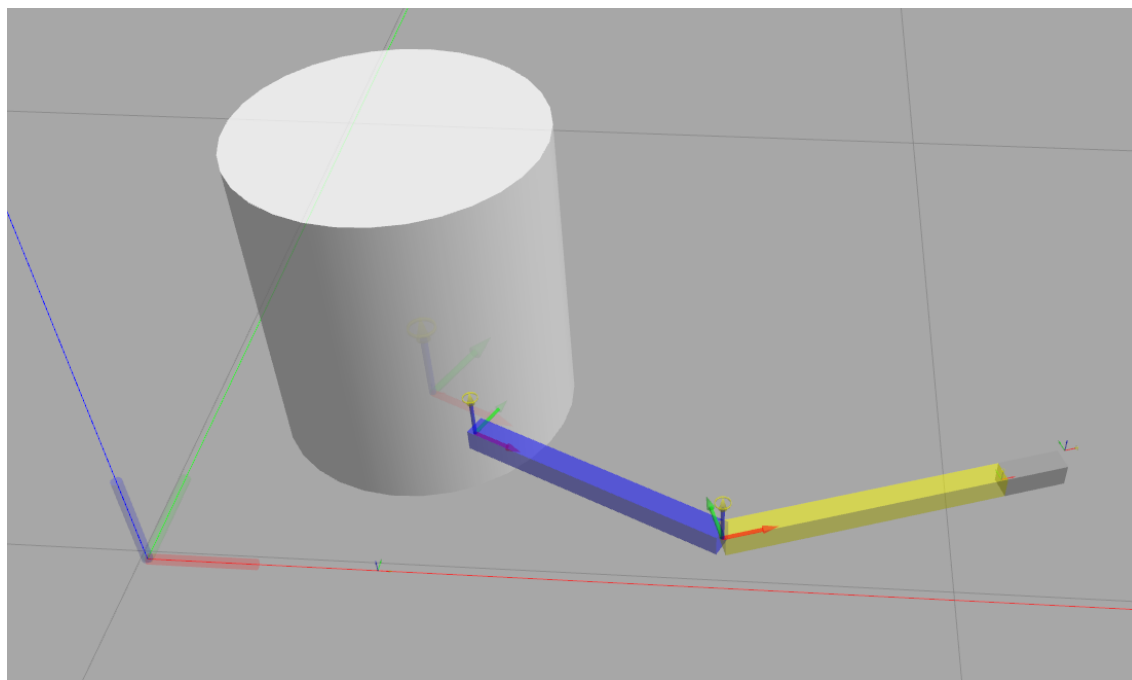
Τα plugins είναι αρχεία τα οποία ενσωματώνονται στο URDF αρχείο του ρομπότ και προσφέρουν μια πληθώρα δυνατοτήτων, οι οποίες αφορούν την επικοινωνία της προσομοίωσης με το ROS περιβάλλον. Χωρίζονται σε διάφορες κατηγορίες, όπως τα ModelPlugins που επηρεάζουν την κίνηση και τη συμπεριφορά των προσομοιωμένων μοντέλων και τα SensorPlugins που επεκτείνουν τις δυνατότητες των προσομοιωμένων αισθητήρων [62]. Για την προσομοίωση και τον έλεγχο του Κηφέα μέσω ROS στο Gazebo χρησιμοποιήθηκαν στο URDF αρχείο του τα εξής plugins:

- **libgazebo_ros_ft_sensor.so**: Προσομοιώνει έναν force-torque sensor, τοποθετημένο στο επιθυμητό joint και τυπώνει τις real time τιμές δυνάμεων και ροπών στα αντίστοιχα ROS topics [63].
- **libgazebo_ros_force.so**: Προσομοιώνει ένα εξωτερικό wrench (δύναμη και ροπή), το οποίο ασκείται σε επιθυμητό link και οι τιμές για τις δυνάμεις και τις ροπές που ασκούνται real time διαβάζονται από το αντίστοιχο ROS topic [64].
- **libgazebo_ros_control.so**: Παρέχει τη δυνατότητα σύνδεσης των controllers του ROS με τις αρθρώσεις των ρομπότ που προσομοιώνονται στο Gazebo [65].

4.4 Η προσομοίωση του Κηφέα

Για την απλοποίηση του URDF του Κηφέα καθώς και για πολλές συντομεύσεις τις οποίες προσφέρει, στο εργαστήριο χρησιμοποιήσαμε το εργαλείο **XACRO**. Όπως προδίδει το όνομα (XML Macros), το XACRO χρησιμοποιείται για την εισαγωγή μακροεντολών (macros) και παραμέτρων στο URDF αρχείο, ορίζοντας έτσι παραμετρικές τιμές π.χ. για τις μάζες και τα μήκη των links, με σκοπό τη διεξαγωγή διαφορετικών πειραμάτων [66].

Η προσομοίωση του Κηφέα σε ένα empty Gazebo world είναι η εξής:



Σχήμα 4.2: Προσομοίωση του Κηφέα στο Gazebo.

Κατά τη προσομοίωση του Κηφέα, τόσο στο visual όσο και στο inertial και collision tag προτιμήθηκαν απλά γεωμετρικά σχήματα (κύλινδρος, κουτί) με σκοπό την εύκολη παραμετροποίηση των μεγεθών αλλά και την οικονομία υπολογιστικών πόρων.

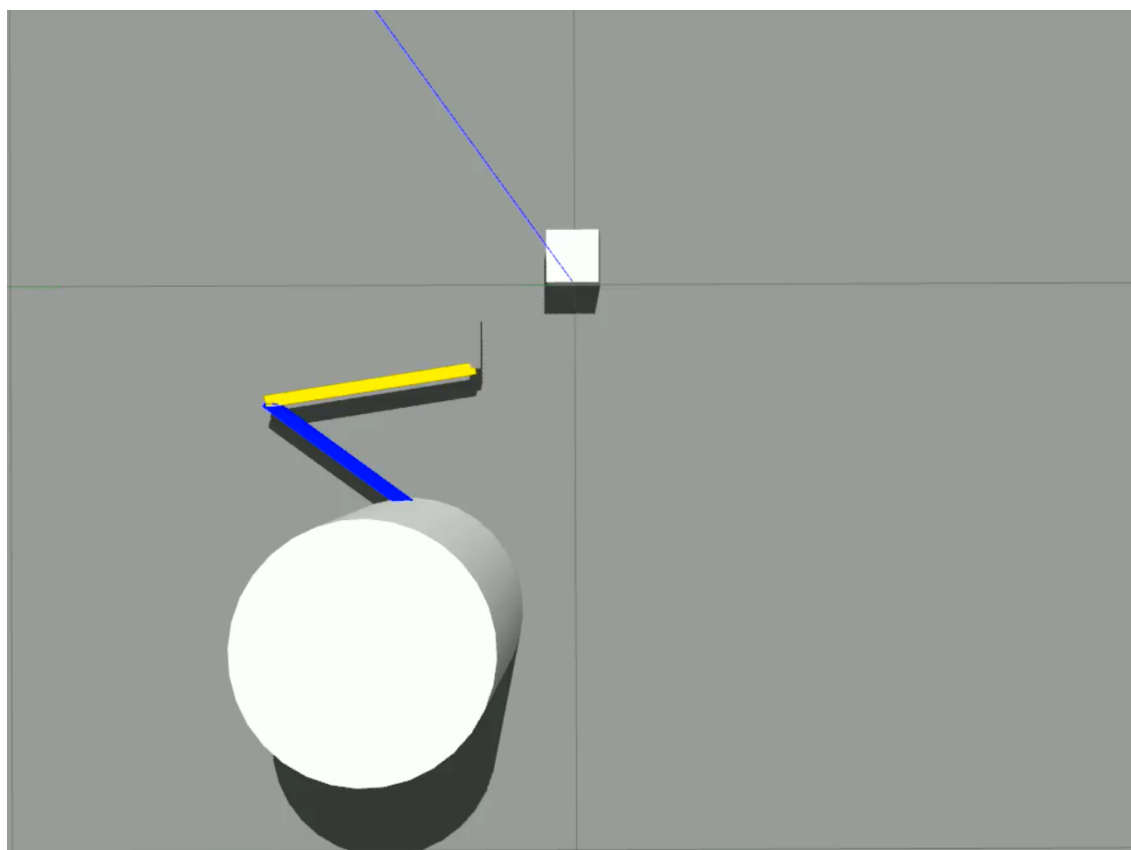
4.5 Πείραμα στον προσομοιωτή

Κατά την πειραματική διαδικασία πρώτο στάδιο αποτέλεσε η διεξαγωγή πειραμάτων στον προσομοιωτή Gazebo. Χρησιμοποιήθηκε το μοντέλο προσομοίωσης του Κηφέα, όπως περιγράφηκε στη προηγούμενη παράγραφο. Σε όλα τα πειράματα, τόσο στη προσομοίωση όσο και στο εργαστήριο, χρησιμοποιήσαμε τη βιβλιοθήκη `rosbag` για να αποθηκεύσουμε σε ένα `.bag` αρχείο τις χρονοσειρές διαφόρων μεγεθών, όπως τη θέση και προσανατολισμό του ΤΣΔ του Κηφέα. Στη συνέχεια αναπτύξαμε ένα `.py` script το οποίο επεξεργάζεται το δοθέν `.bag` αρχείο και παρουσιάζει τις γραφικές παραστάσεις των επιθυμητών μεγεθών.

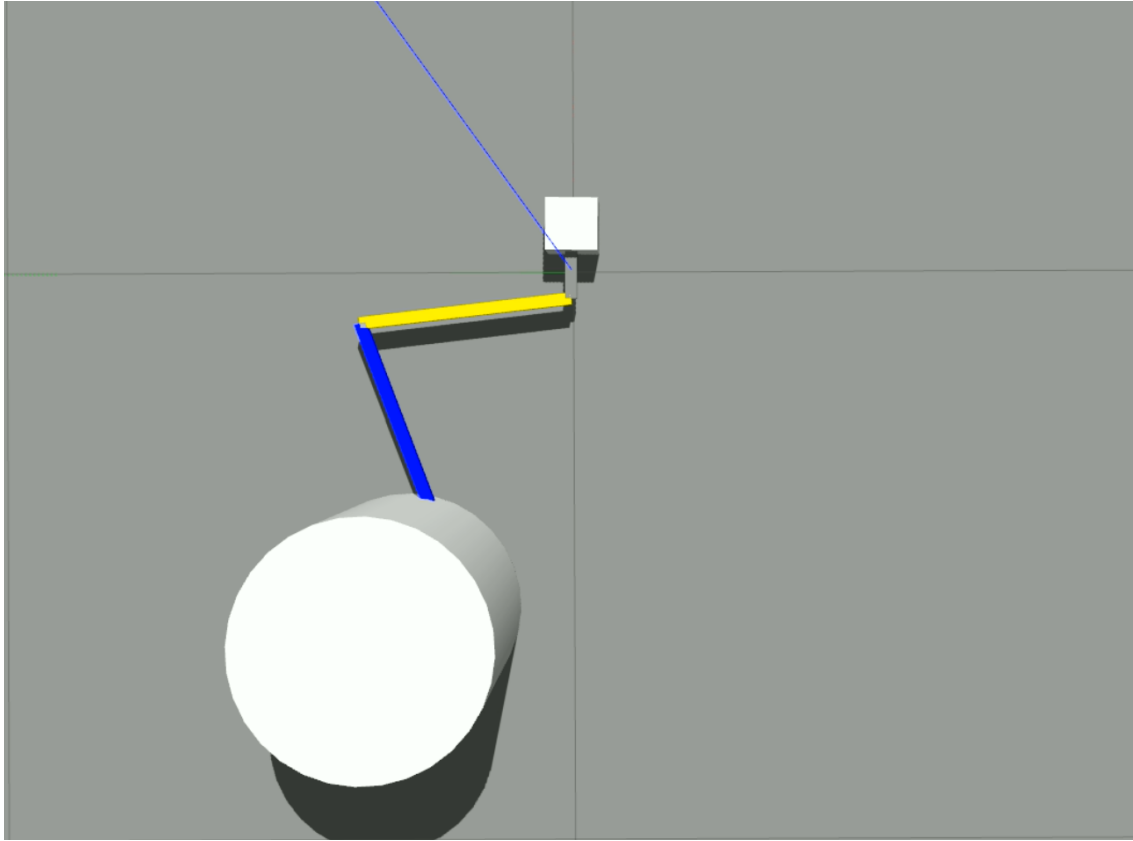
Πείραμα του ελεγκτή εμπέδησης στην προσομοίωση

Αφού στο προηγούμενο πείραμα ελέγχθηκε η λειτουργία του PD ελεγκτή σε ένα διπτό σενάριο ελεύθερης κίνησης και επαφής, προχωρήσαμε στη δοκιμή για το ίδιο σενάριο του νόμου ελέγχου μηχανικής εμπέδησης που έχουμε υλοποιήσει στο ROS framework, ξανά στο προσομοιωμένο μας μοντέλο. Για αυτό το πείραμα επιλέξαμε ο χρόνος κίνησης μέχρι να φτάσει τον στόχο (`t_free`) να είναι τα 50 δευτερόλεπτα. Ο βασικός βρόχος ελέγχου επιλέχτηκε (μέσω του `ros::rate`) να είναι στα 100 Hz. Σκοπός του πειράματος είναι με τον ενιαίο impedance ελεγκτή να ακολουθήσει αρχικά την πολυωνυμική τροχιά, και στη συνέχεια να ακουμπήσει τον στόχο και να διατηρήσει την επαφή μεταξύ ΤΣΔ και στόχου για ένα εύλογο χρονικό διάστημα.

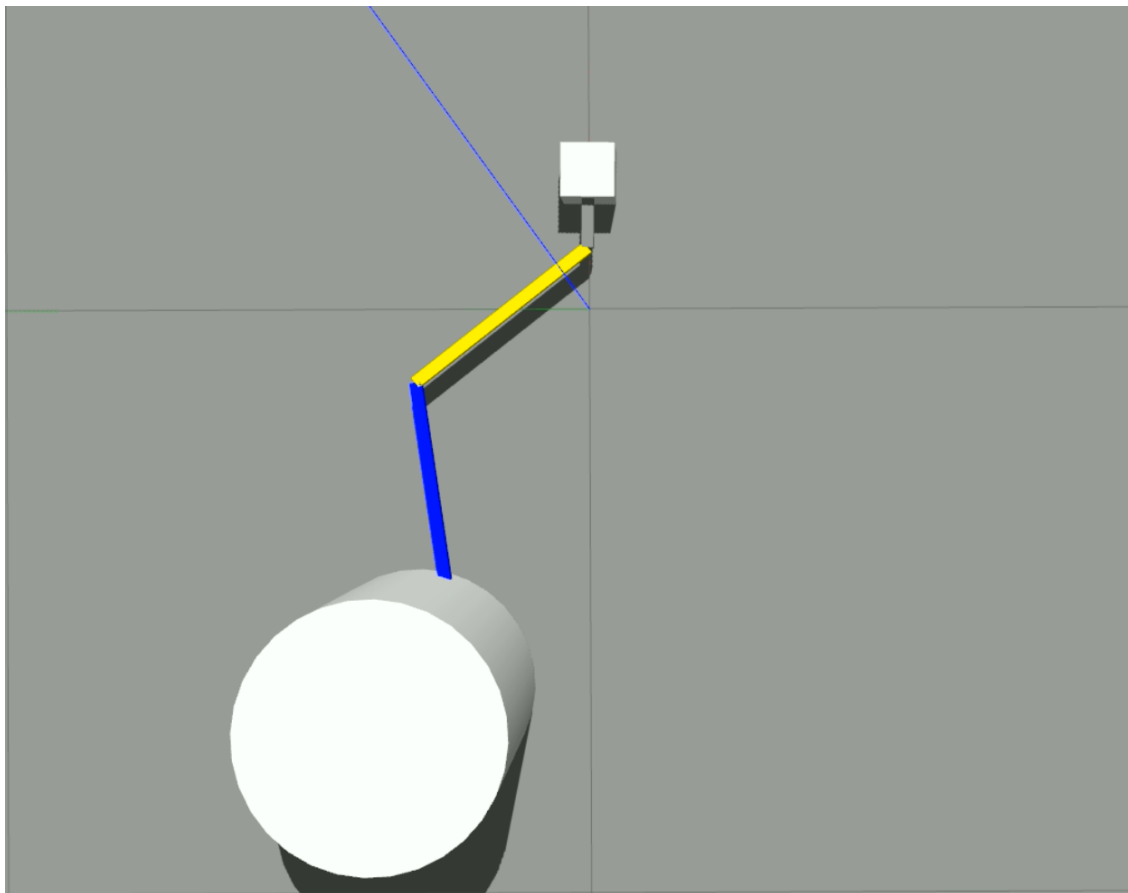
Παρακάτω βλέπουμε στιγμιότυπα από διάφορα στάδια κίνησης του Κηφέα :



Σχήμα 4.3: Αρχική θέση Κηφέα και στόχου στο προσομοιωμένο πείραμα ελέγχου εμπέδησης.

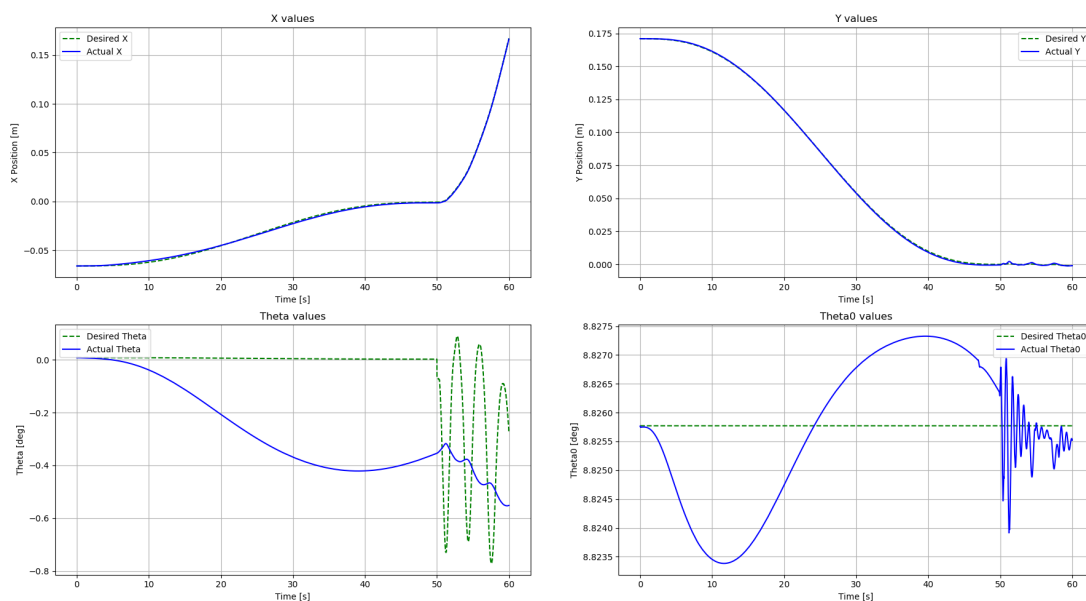


Σχήμα 4.4: Στιγμιότυπο κατά τη διάρκεια της επαφής ΤΣΔ και στόχου, στο προσομοιωμένο πείραμα ελέγχου μηχανικής εμπέδησης.

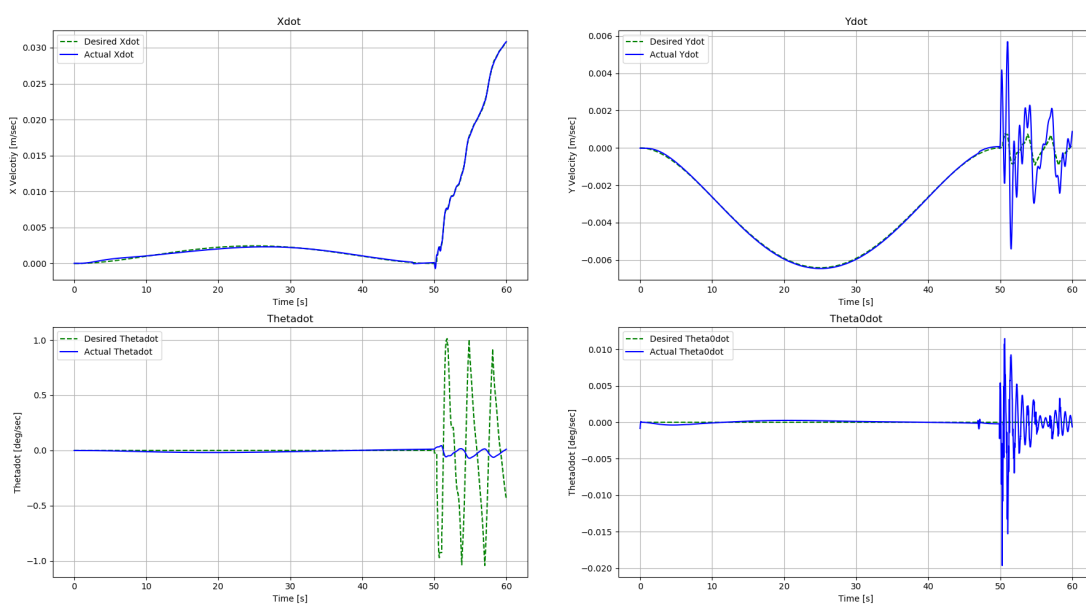


Σχήμα 4.5: Δεύτερο στιγμιότυπο κατά τη διάρκεια της διατηρημένης επαφής ΤΣΔ και στόχου, στο προσομοιωμένο πείραμα ελέγχου εμπέδησης.

Παρακάτω φαίνονται οι γραφικές παραστάσεις της θέσης του στόχου, της επιθυμητής θέσης (desired trajectory) αλλά και της πραγματικής θέσης του end effector για τις συνεταγμένες x, y και για τον προσανατολισμό θ του end effector. Επιπλέον φαίνεται η γωνία της βάσης του Κηφέα (θ_0): Γωνία-στόχος(που είναι ουσιαστικά η αρχική γωνία κατά την εκκίνηση του πειράματος, καθώς θέλουμε να διατηρήσουμε την βάση του Κηφέα σταθερή), επιθυμητή γωνία σύμφωνα με το πολυώνυμο παρελμβολής και πραγματική γωνία κάθε χρονική στιγμή. Επιπλέον βλέπουμε τις χρονικές παραγώγους όλων των παραπάνω μεγεθών:



Σχήμα 4.6: Γραφικές παραστάσεις των x , y , θ και θ_0 στο προσμοιωμένο πείραμα ελέγχου εμπέδησης.

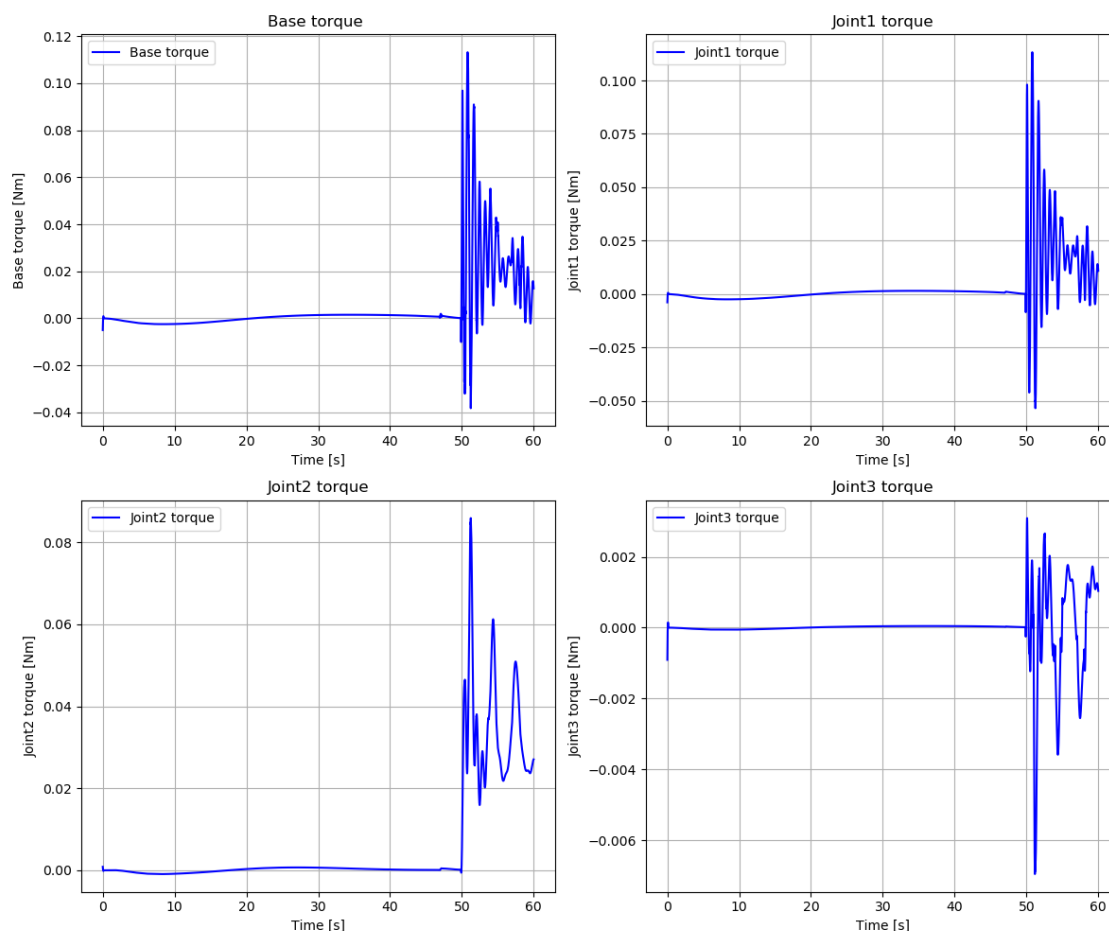


Σχήμα 4.7: Γραφικές παραστάσεις των $\dot{x}$, $\dot{y}$, $\dot{\theta}$ και $\dot{\theta}_0$ στο προσμοιωμένο πείραμα ελέγχου εμπέδησης.

Κατά την ελεύθερη κίνηση, παρατηρούμε σχεδόν πλήρη επικάλυψη μεταξύ της επιθυμητής και της πραγματικής τροχιάς για τα x , y του end effector. Επιπλέον, για τον προσανατολισμό του end effector παρατηρούμε απόκλιση της τάξης 0.4 deg και για τον προσανατολισμό της βάσης απόκλιση της τάξης 0.004 deg, οι οποίες πρακτικά είναι αμελητέες. Επιπλέον παρατηρούμε επαφή τη χρονική στιγμή $t = 50$ sec η οποία διατηρείται και για τα επόμενα 10 δευτερόλεπτα, ικανοποιητικό χρονικό διάστημα ώστε η αρπάγη της πειραματικής διάταξης να πιάσει τον στόχο. Κατά τη διάρκεια της επαφής, βλέπουμε ότι τα x , y και $\dot{x}$, $\dot{y}$ του ΤΣΔ ταυτίζονται με αυτά του στόχου. Επίσης, παρατηρούμε πάλι μικρή και πρακτικά

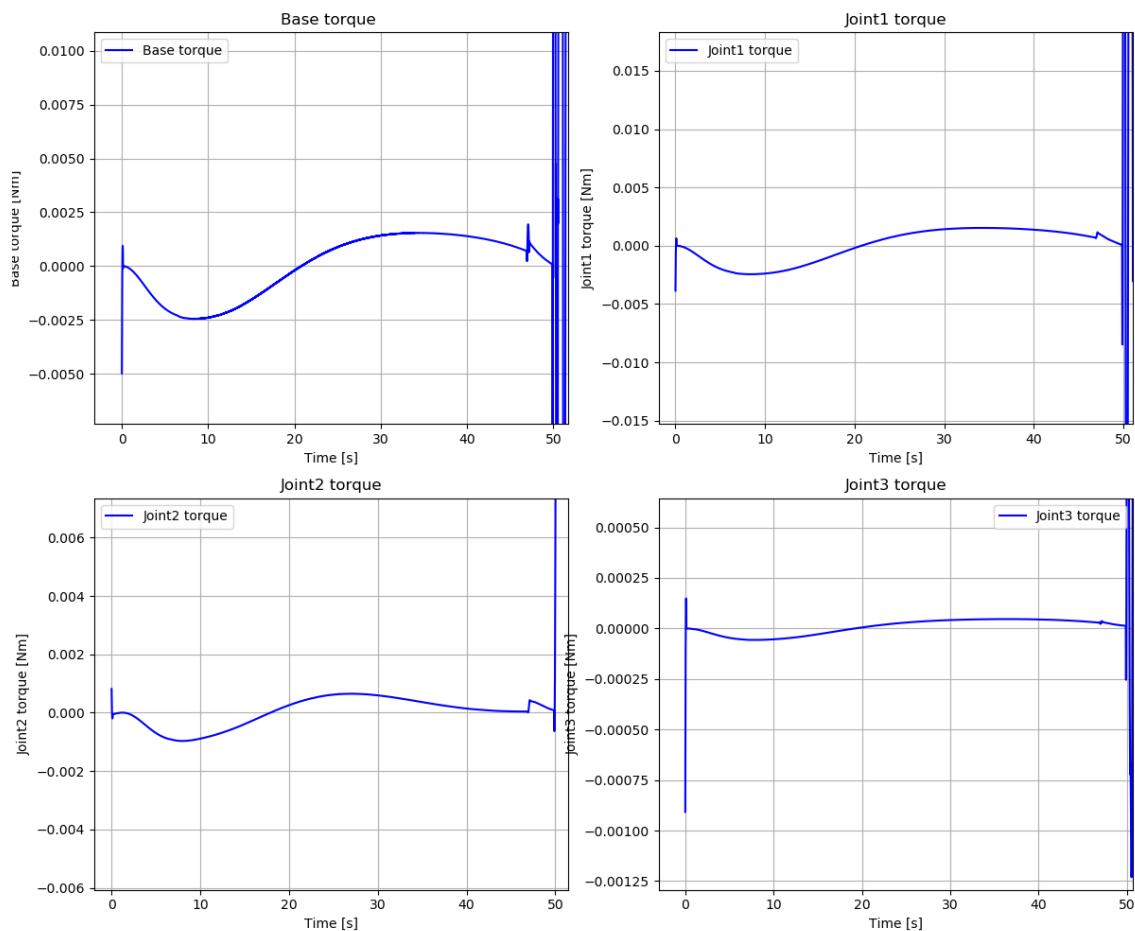
αμελητέα απόκλιση μεταξύ της γωνίας θ του ΤΣΔ και του στόχου. Το ίδιο βλέπουμε και για τις χρονικές παραγώγους τους. Επίσης, η βάση παραμένει πρακτικά σταθερή καθώς οι αποσβεσνύμενες ταλαντώσεις που βλέπουμε στη γωνία θ_0 κατά τη διάρκεια της επαφής έχουν μέγιστο πλάτος 0.003 deg και οι αντίστοιχες αποσβεσνύμενες ταλαντώσεις του $\dot{\theta}_0$ έχουν μέγιστο πλάτος 0.02 deg/sec .

Παρακάτω βλέπουμε τις ροπές που ασκήθηκαν στις αρθρώσεις από τους προσομοιωμένους κινητήρες καθώς και την προσομοιωμένη ροπή που ασκήθηκε στη βάση, χρησιμοποιώντας κατάλληλο Gazebo plugin:



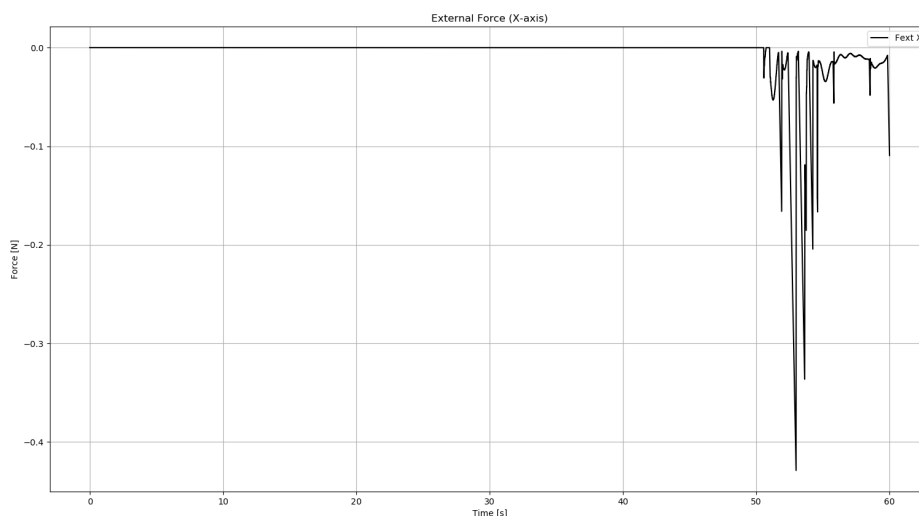
Σχήμα 4.8: Γραφικές παραστάσεις των ροπών στις αρθρώσεις και στη βάση στο προσομοιωμένο πείραμα ελέγχου μηχανικής εμπέδησης.

Καταλαβαίνουμε ότι οι ροπές που ασκούνται κατά τη διάρκεια του πειράματος είναι πολύ χαμηλότερες κατά την ελεύθερη κίνηση σε σχέση με τη στιγμή της επαφής και κατά τη διάρκεια διατήρησης της. Όμως, σε καμία χρονική στιγμή δεν ξεπερνούνται τα όρια ασφαλείας, καθώς οι μέγιστες ροπές είναι της τάξης 0.1 Nm . Μεγεθύνοντας τα διαγράμματα έχουμε καλύτερη εποπτεία των ροπών πριν την επαφή:



Σχήμα 4.9: Μεγεθυμένα διαγράμματα των ροπών στις αρθρώσεις και στη βάση στο προσομοιωμένο πείραμα ελέγχου μηχανικής εμπέδησης.

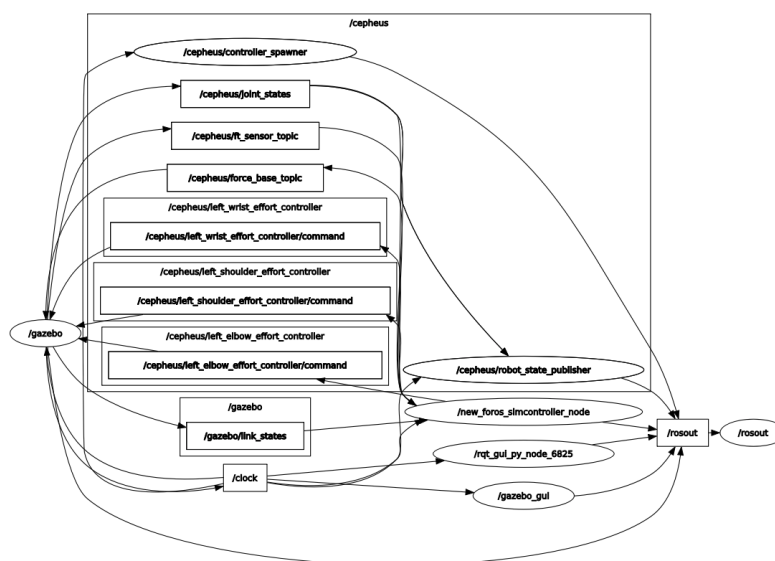
Βλέποντας το παρακάτω διάγραμμα της F_{ext} που δέχεται ο gripper κατά τον άξονα x του τοπικού του frame, παρατηρούμε την στιγμή της πρώτης επαφής με τον στόχο και την προσπάθεια του Κηφέα να διατηρήσει την επαφή, το οποίο καταφέρνει λίγα δευτερόλεπτα αργότερα:



Σχήμα 4.10: Η εξωτερική δύναμη f_{ext} που ασκείται στο ΤΣΔ κατά τον άξονα x του τοπικού $frame$ στο προσομοιωμένο πείραμα ελέγχου μηχανικής εμπέδησης.

Από το διάγραμμα της F_{ext} επαληθεύεται η αρχή της επαφής λίγο μετά την χρονική στιγμή $t = 50$ sec και η διατήρησή της για το επόμενο χρονικό διάστημα. Επιπλέον, βλέπουμε ένα πολύ μικρό spike κοντά στα 46 sec, το οποίο υποδεικνύει μια πρόωμη επαφή η οποία όμως διακόπτεται αμέσως. Αυτό διασταυρώνεται και από τα εξίσου μικρά spikes τα οποία παρατηρούμε στα διαγράμματα ροπών την ίδια χρονική στιγμή.

Κατά τη διάρκεια του πειράματος στη προσομοίωση, μέσω του εργαλείου RQT έχουμε εποπτεία για τα ROS nodes που τρέχουν και για τα ROS topics μέσω των οποίων επικοινωνούν:



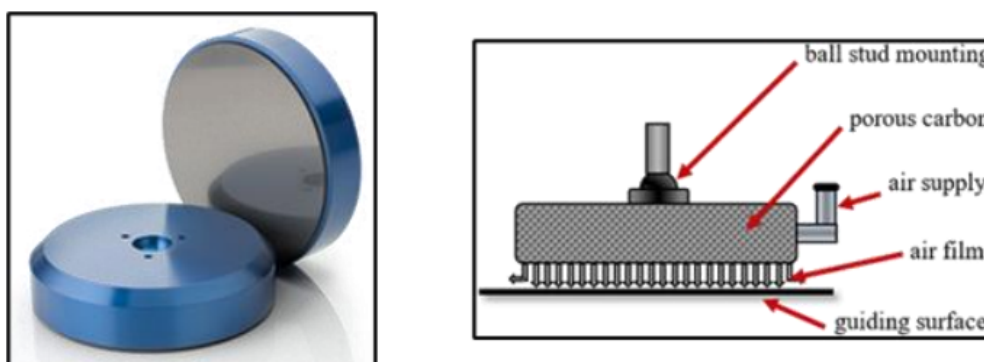
Σχήμα 4.11: Εποπτεία των ενεργών ROS nodes και topics μέσω του εργαλείου RQT, κατά το προσομοιωμένο πείραμα ελέγχου μηχανικής εμπέδησης.

Κεφάλαιο **5**

Πείραμα στον διαστημικό εξομοιωτή του εργαστηρίου

5.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα περιγράψουμε αναλυτικά την δομή του διαστημικού εξομοιωτή του Εργαστηρίου Αυτομάτου Ελέγχου του ΕΜΠ. Στον πυρήνα του εξομοιωτή βρίσκεται το ρομπότ Κηφέας, ο οποίος αποτελείται από έναν βραχίονα 3 βαθμών ελευθερίας και μια κινητή βάση. Κάτω από τη βάση και κάτω από το ΤΣΔ βρίσκονται αερόδρανα ή αλλιώς air bearings, τα οποία στηρίζουν τα σώματα πάνω στην τράπεζα από γρανίτη και επιτυγχάνονται κινήσεις με σχεδόν μηδενική τριβή. Αυτό συμβαίνει διότι χρησιμοποιώντας τα air bearings, μέσα από τον πορώδη γραφίτη περνά αέρας υπό πίεση δημιουργώντας ένα λεπτό φιλμ αέρος ανάμεσα στις δύο ξένες επιφάνειες. Συνεπώς, το σώμα που φέρει σαν βάσεις τα αεροέδρανα δεν έρχεται σε άμεση επαφή με την επιφάνεια της τράπεζας, αλλά με το φιλμ του αέρα, γεγονός που επιφέρει τη μηδενική τριβή ολίσθησης. Με αυτόν τον τρόπο επιτυγχάνεται εξομοίωση συνθηκών διαστήματος. Σχηματικά :

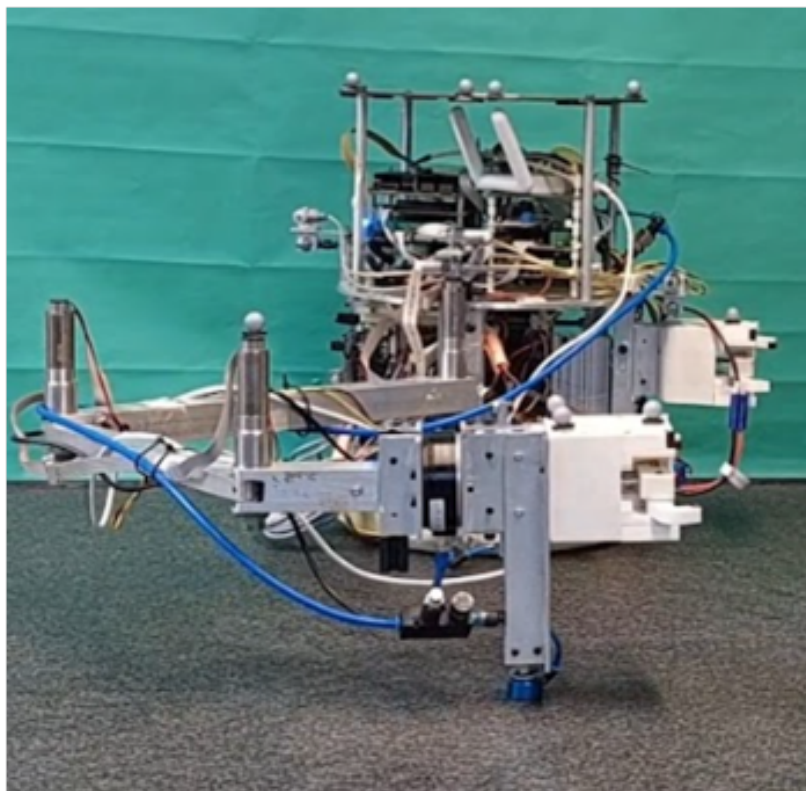


Σχήμα 5.1: Η μορφή και η λειτουργία του αερόδρανου, με τη χρήση κατάλληλου αερίου υπό πίεση.

Επιπλέον, για την μεταφορική κίνηση της βάσης πάνω στο επίπεδο, χρησιμοποιούνται 3 σει προωθήρων (thrusters). Επίσης, η βάση του Κηφέα μπορεί να περιστρέφεται γύρω από τον κατακόρυφο άξονα χάρη στον σφόνδυλο αντίδρασης (reaction wheel) που περιέχει. Στη παρούσα διπλωματική θα αξιοποιήσουμε τους 3 κινητήρες του βραχίονα καθώς και το

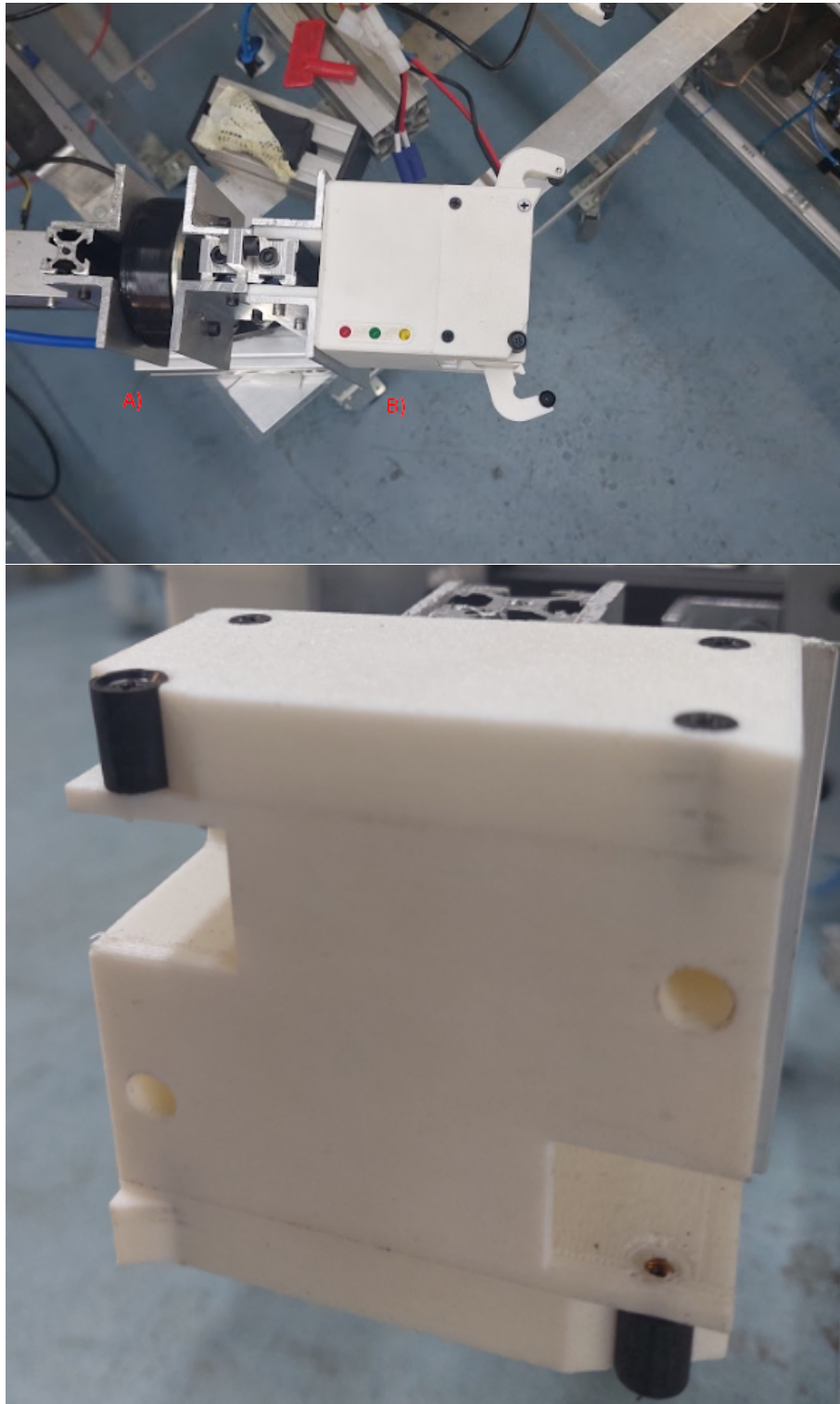
reaction wheel της βάσης[51].

Ο Κηφέας κατά τη πειραματική διαδικασία φαίνεται στην εξής εικόνα :



Σχήμα 5.2: Ο Κηφέας κατά τη πειραματική διαδικασία.

Πιο αναλυτικά, κεντρική μονάδα επεξεργασίας του Κηφέα είναι ο υπολογιστής PC104 σε μορφή πύργου (stack), οποίος τρέχει στο λειτουργικό linux και τρέχει το ROS του ρομπότ. Εκεί γίνεται η επεξεργασία και (με τη χρήση καρτών εισόδου-εξόδου) η ανταλλαγή κατάλληλων εντολών και σημάτων με τα περιφερειακά συστήματα του ρομπότ, όπως είναι οι drivers και οι encoders των κινητήρων. Για την τροφοδότηση του υπολογιστή και των κινητήρων χρησιμοποιούνται 2 μπαταρίες σε σειρά. Στο ΤΣΔ του βραχίονα είναι εγκαταστημένος ο botasys force torque sensor και στη συνέχεια το νεοσύστατο ενεργητικό gripping interface, το οποίο περιέχει δικό του arduino για την επεξεργασία και εκτέλεση εντολών και θα χρησιμοποιηθεί στο πείραμα για να πιάσει το αντίστοιχο παθητικό interface του στόχου [67]. Σχηματικά :



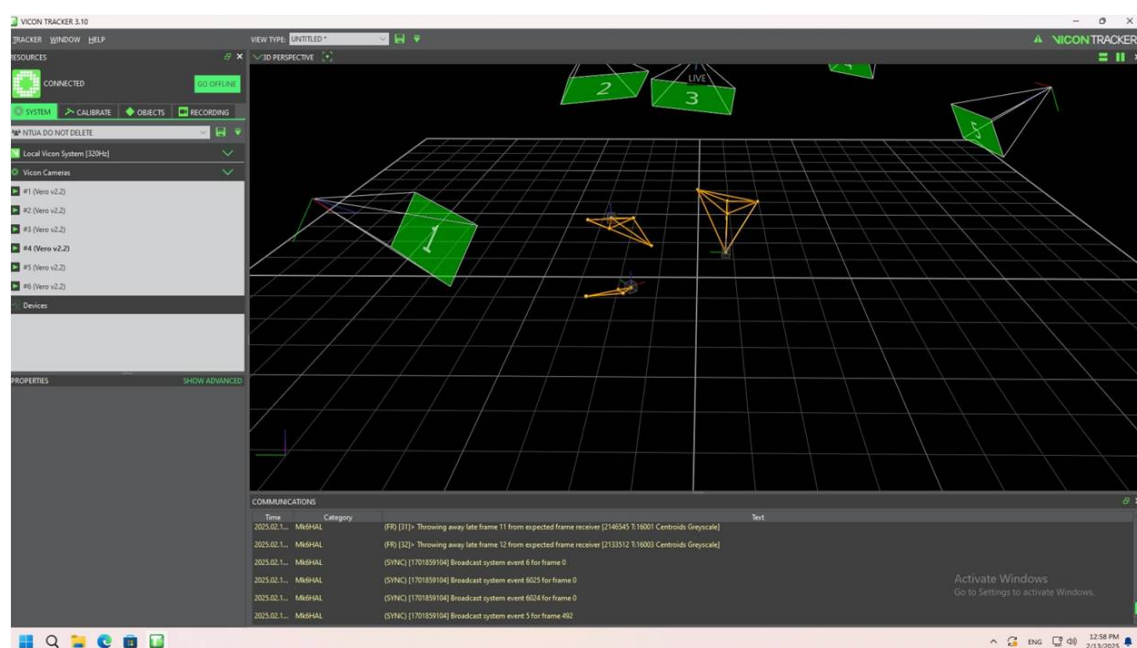
Σχήμα 5.3: Ο *force-torque sensor*, ο *gripper* και το *παθητικό interface*.

Για τα air bearings χρησιμοποιείται πνευματικό κύκλωμα στο οποίο συνδέεται μια φιάλη CO₂ με regulator, έτσι ώστε η πίεση στα bearings να είναι στα 5-6 bar. Επιπλέον, φαίνονται τοποθετημένοι στη βάση και στον βραχίονα του Κηφέα οι markers του Motion Capture συστήματος της Vicon. Χρησιμοποιώντας αυτούς μέσα από το gui της Vicon δημιουργήσαμε

κατάλληλα frames στη βάση και στο ΤΣΔ του Κηφέα, αναφορικά με το αδρανειακό world frame που έχουμε ορίσει.

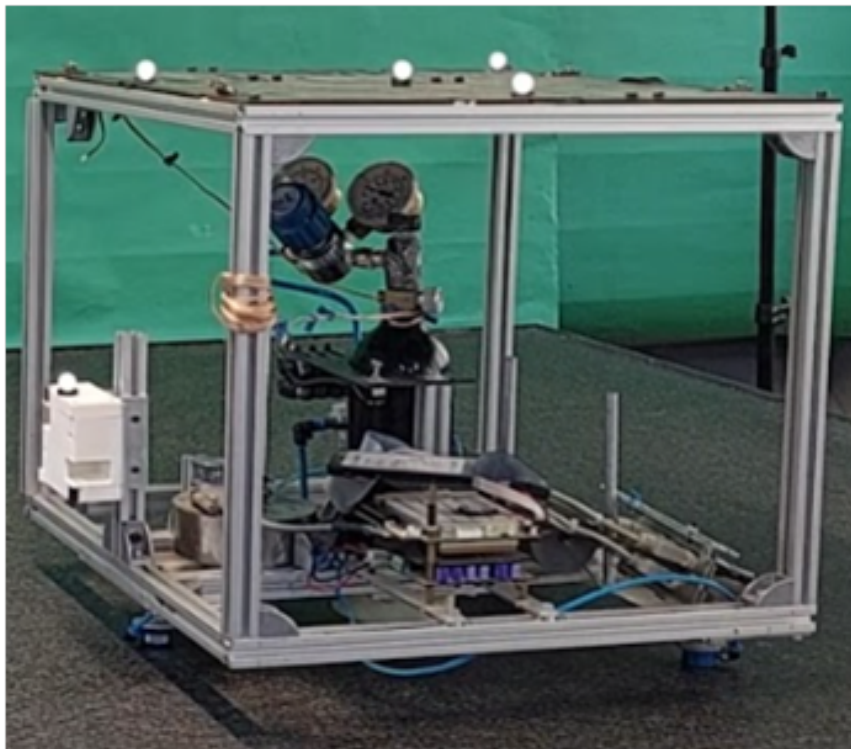


Σχήμα 5.4: Οι κάμερες του συστήματος Vicon περιμετρικά της τράπεζας γρανίτη και οι markers.



Σχήμα 5.5: Το gui της Vicon και η δημιουργία καταλλήλων frames.

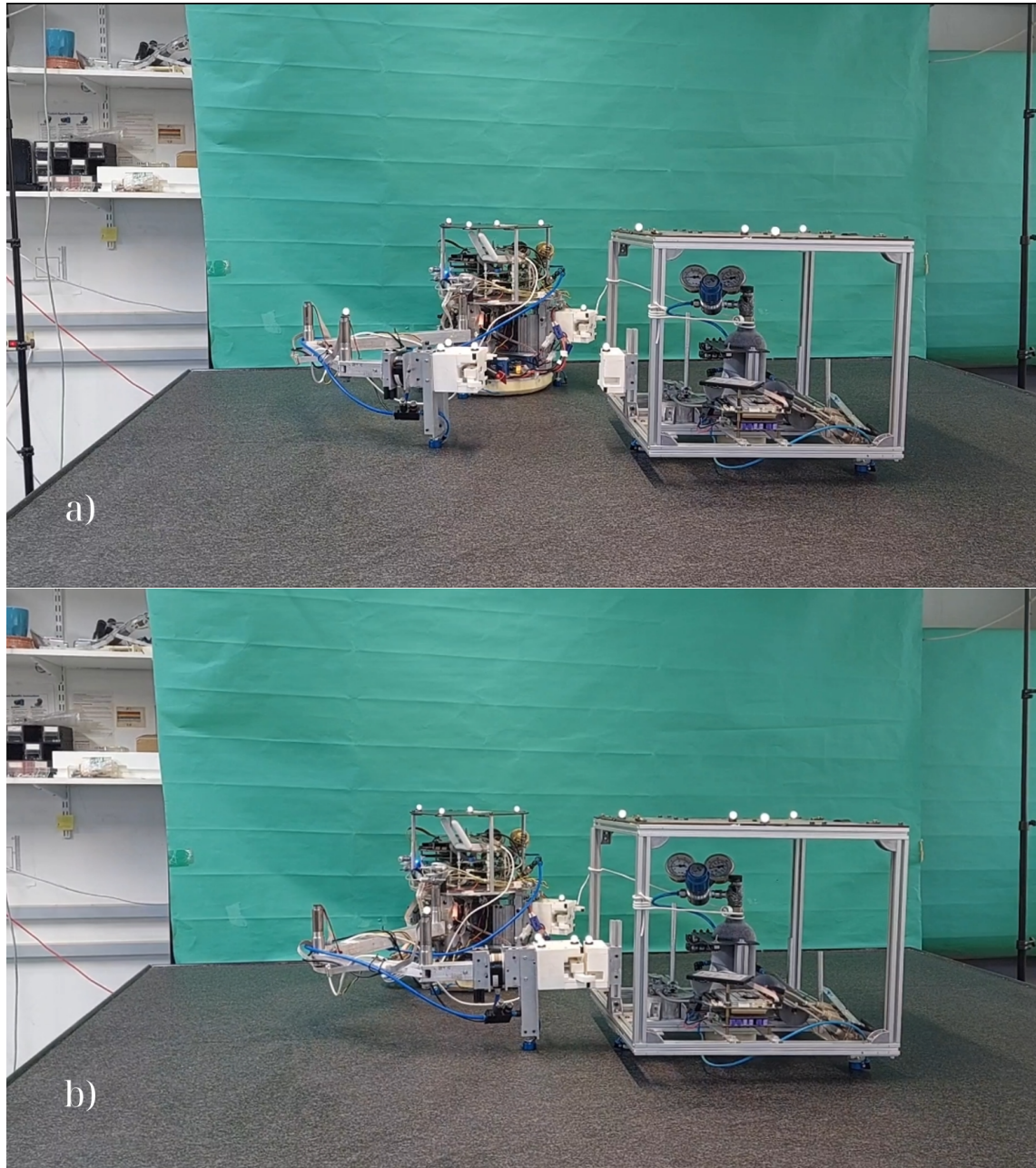
Ο στόχος αποτελείται από τα air bearings, το πνευματικό κύκλωμα, τους markers της Vicon και το παθητικό gripping interface:



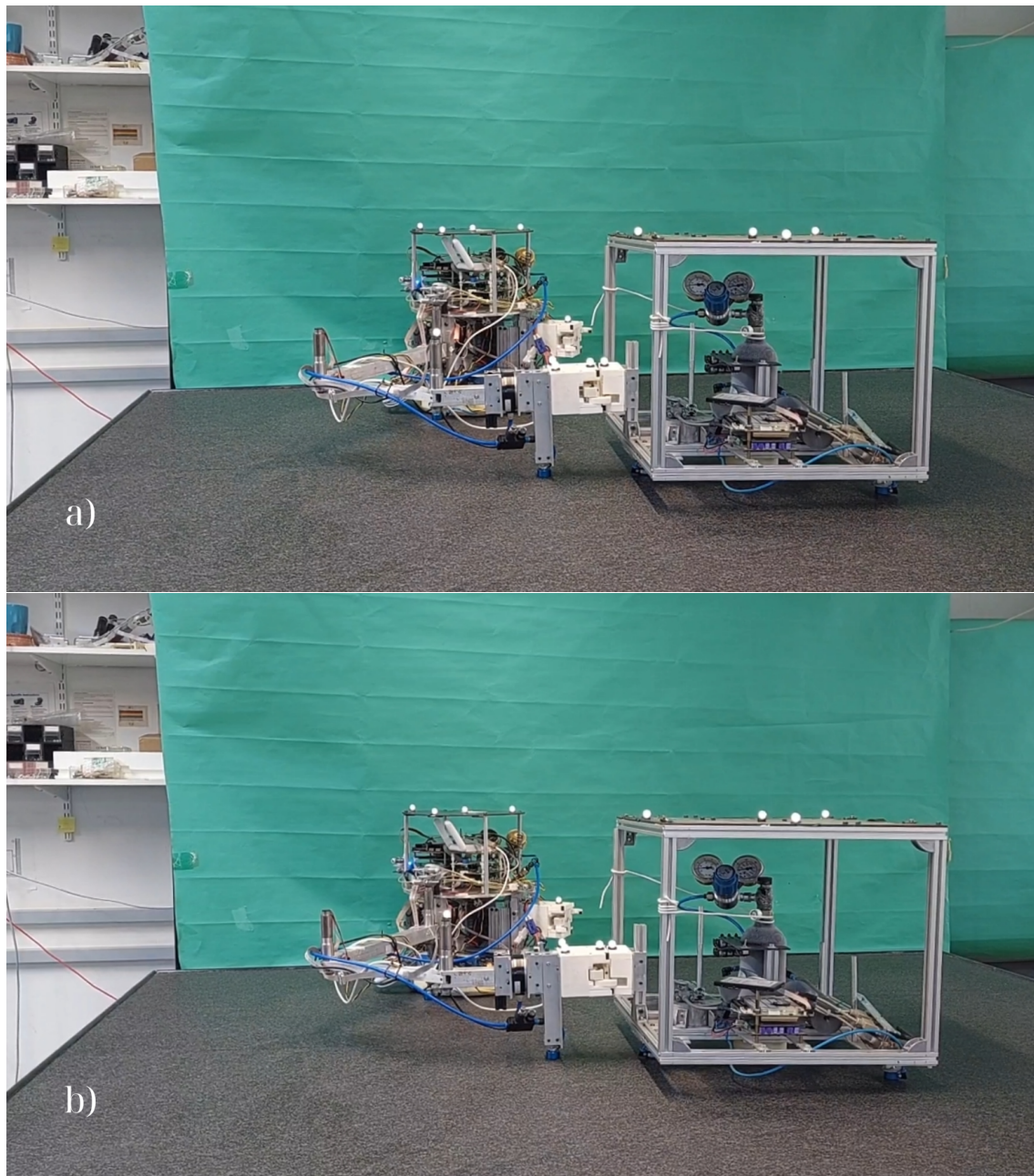
Σχήμα 5.6: Ο στόχος κατά τη πειραματική διαδικασία.

5.2 Πείραμα του νόμου ελέγχου

Στη παρακάτω παράγραφο θα αναλυθεί το πείραμα που πραγματοποιήθηκε στον διαστημικό εξομοιωτή του εργαστηρίου. Πιο συγκεκριμένα, πραγματοποιήθηκε πείραμα για την πρώτη φάση του ελεγκτή (free space phase). Σκοπός του πειράματος είναι να πλησιάσει το ΤΣΔ τον στόχο και στη συνέχεια να τον πιάσει ο gripper ευθυγραμμίζοντας όποιο missalignment μπορεί να υπάρχει. Ταυτόχρονα, ο προανατολισμός της βάσης καθόλη τη διάρκεια του πειράματος πρέπει να μείνει σταθερός και κατά το gripping ο βραχίονας να είναι επίσης σταθερός. Η λογική του ελεγκτή περιγράφεται λεπτομερώς στο κεφάλαιο 3. Παρακάτω φαίνονται στιγμιότυπα από το πείραμα:

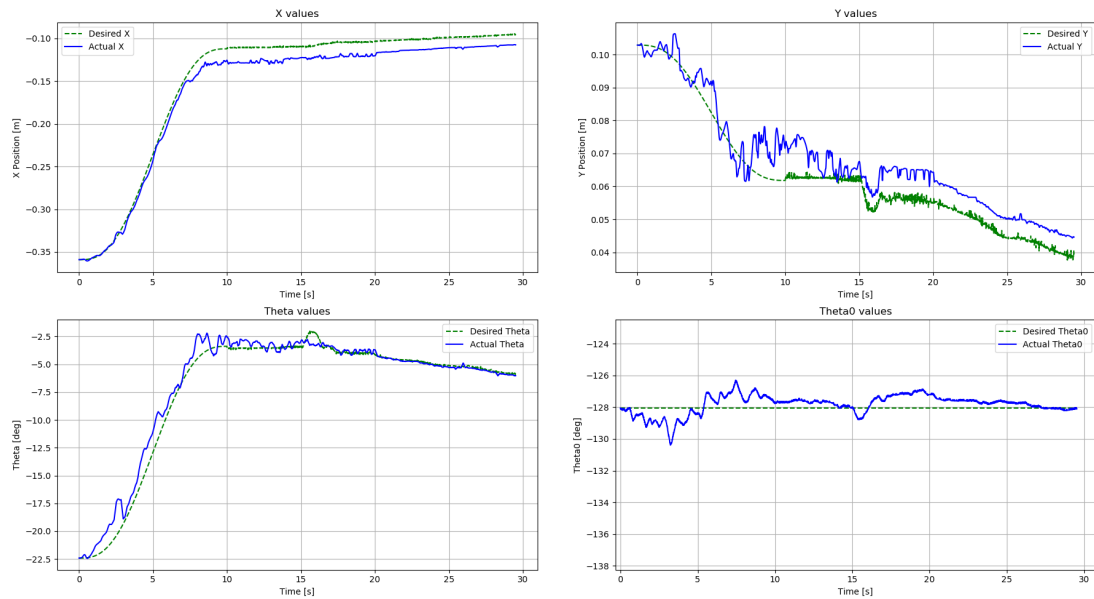


Σχήμα 5.7: a) Η αρχική διάταξη του πειράματος. b) Η διάταξη πριν ξεκινήσει το *softgrip*.



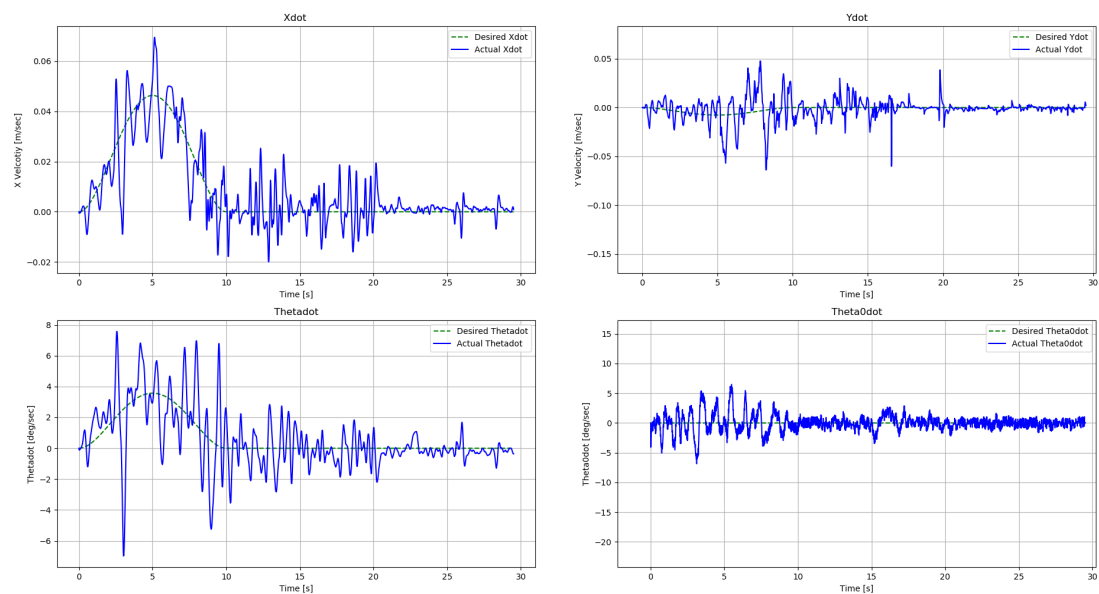
Σχήμα 5.8: *a) Η διάταξη πριν ξεκινήσει το hardgrip (μετά το softgrip). b) Η τελική διάταξη μετά το hardgrip.*

Παρακάτω βλέπουμε τις γραφικές παραστάσεις θέσης και προσανατολισμού του ΤΣΔ καθώς και προσανατολισμού της βάσης. Με διακεκομμένη γραμμή φαίνεται η επιθυμητή τροχιά για κάθε γραφική :



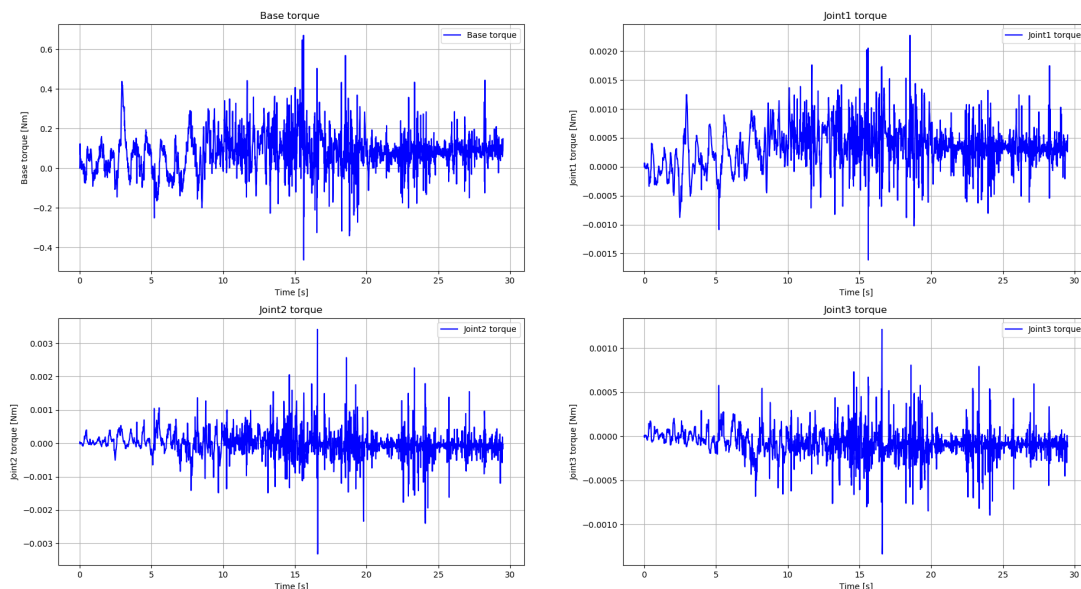
Σχήμα 5.9: Γραφικές παραστάσεις των x , y , θ και θ_0 στο πείραμα ελέγχου εμπέδησης του εργαστηρίου.

Στη συνέχεια βλέπουμε τις γραφικές παραστάσεις των πρώτων παραγώγων:



Σχήμα 5.10: Γραφικές παραστάσεις των $\dot{x}$, $\dot{y}$, $\dot{\theta}$ και $\dot{\theta}_0$ στο πείραμα ελέγχου εμπέδησης του εργαστηρίου.

Επιπλέον, παρακάτω βλέπουμε τις ροπές των αρθρώσεων και του reaction wheel:



Σχήμα 5.11: Οι ροπές των αρθρώσεων και του reaction wheel στο πείραμα ελέγχου εμπέδησης του εργαστηρίου.

5.3 Συμπεράσματα

Παρατηρώντας το πείραμα και τις γραφικές παραστάσεις καταλαβαίνουμε ότι στο free space phase ο έλεγχος της καρτεσιανής θέσης πραγματοποιήθηκε με επιτυχία. Η κίνηση έγινε κατά κόρον στο x άξονα, όπου ακολούθησε την επιθυμητή τροχιά. Μετά το τέλος του gripping, φαίνεται μια σταθερή απόσταση μεταξύ της επιθυμητής τροχιάς και της πραγματικής, η οποία οφείλεται στη σταθερή απόσταση μεταξύ των markers στο ΤΣΔ και στον στόχο. Επιπλέον, ο προσανατολισμός της βάσης καθόλη τη διάρκεια του πειράματος δεν αλλάζει για παραπάνω από ± 2 μοίρες. Επιπλέον, αξιοποιήθηκε για πρώτη φορά από το εργαστήριο το νέο σύστημα καμερών της Vicon αλλά και το νεοσύστατο gripping interface, το οποίο ευθυγράμμισε τον στόχο αποτελεσματικά. Επίσης, η λογική επικοινωνίας του βασικού αρχείου ελέγχου (robot_foros_controller) με το ROS του arduino ήταν αποτελεσματική, καθώς ακολουθήκαν αυτόματα τα βήματα για πρώτα softgrip και στη συνέχεια hardgrip. Ακόμη, κατά τη διάρκεια του gripping ο βραχίονας και ο προσανατολισμός της βάσης παρέμειναν σταθεροί.

Επιπλέον, βλέποντας τις παραστάσεις των ταχυτήτων είναι εμφανής η ανάγκη χρήσης imu για καλύτερη προσέγγιση της ταχύτητας, αντικαθιστώντας την παραγωγή των δεδομένων της Vicon. Έτσι η κίνηση θα γίνει ακόμα πιο ομαλή. Αξίζει να σημειωθεί ότι ο υπολογισμός της ταχύτητας έγινε αφού πρώτα φιλτραρίστηκε η θέση μέσω ενός moving average φίλτρου και στη συνέχεια η ταχύτητα μέσω ενός moving median φίλτρου. Επίσης, η χρήση του imu θα μας επέτρεπε τον υπολογισμό της αδρανειακής δύναμης του ΤΣΔ λόγω επιτάχυνσης και συνεπώς την εξαγωγή της δύναμης επαφής f_{ext} . Έτσι, μελλοντικά θα μπορούμε να κάνουμε πείραμα και για τη δεύτερη φάση του ελεγκτή (contact phase).

Συμπεράσματα και Μελλοντική Εργασία

6.1 Συμπεράσματα

Η εργασία αυτή είναι αποτέλεσμα ενδελεχούς προσπάθειας και προσεγγίζει τον διαστημικό εξομοιωτή από πολλές οπτικές. Πρώτα έγινε η θεωρητική μελέτη και κατανόηση για τον ελεγκτή που θέλουμε να υλοποιηθεί στο ρομπότ. Στη συνέχεια μελετήθηκε το περιβάλλον προγραμματισμού ROS και υλοποιήθηκε σε αυτό κατάλληλο software σε C++, το οποίο ακολουθεί σωστό design pattern έτσι ώστε είναι ικανό να δοκιμάζεται στη προσομοίωση αλλά και στο αληθινό hardware. Γιαυτό ταυτόχρονα μοντελοποιήθηκε ο Κηφέας στον προσομοιωτή Gazebo και πραγματοποιήθηκε πετυχημένο πείραμα ελέγχου εμπέδησης στη προσομοίωση. Στη συνέχεια, μεγάλο κομμάτι της διπλωματικής αφορούσε την επαναφορά του Κηφέα σε λειτουργία, ύστερα από 2 χρόνια ακινησίας. Αυτό προϋπέθετε αναγνώριση δυσλειτουργικού εξοπλισμού (παλιές μπαταρίες, PhaseSpace) και αντικατάσταση με καινούργιο (νέες μπαταρίες, Vicon). Επιπλέον, χρειάστηκε να αναγνωριστεί και να επανασυνδεθεί η παλιά καλωδίωση καθώς επίσης και η καλωδίωση παλαιότερου driver όπου δεν χρησιμοποιούταν για καιρό. Ακόμη, ενσωματώθηκε το νέο gripping interface τόσο στο ΤΣΔ όσο και στον στόχο. Αναφορικά με τον προϋπάρχον κώδικα του Κηφέα, χρειάστηκε εκτενής μελέτη για την αναγνώριση των λειτουργικών βασικών αρχείων, σε αντίθεση με τα αρχεία παλαιότερων πειραμάτων που έπρεπε να απενεργοποιηθούν και τη δημιουργία ενός βασικού documentation για την πειραματική διαδικασία.

Εν τέλει, ύστερα από αυτές τις αλλαγές καταφέραμε να πραγματοποιήσουμε πείραμα στο εργαστήριο που δείχνει την πετυχημένη λειτουργία του ελεγκτή για το πρώτο στάδιο. Επιπλέον, η προσομοίωση υπόσχεται παρόμοια αποτελέσματα για το δεύτερο στάδιο στο εργαστήριο, κατόπιν αναβάθμισης του εξοπλισμού.

Γιαυτό εργασία αυτή αποτελεί γέφυρα μεταξύ της παλιάς διάταξης, η οποία ήταν ανενεργή και της τωρινής, η οποία τρέχει και είναι έτοιμη, μαζί με λίγες προσθήκες, για τα μελλοντικά πρότζεκτ.

6.2 Μελλοντική Εργασία

Τα πρώτα βήματα που πρέπει να ακολουθήσουν είναι η καλύτερη μέτρηση ταχύτητας και η ακριβής μέτρηση της F_{ext} . Αυτά τα δύο πραγματοποιηθούν εύκολα με την ενσωμάτωση ενός νέου ακριβή αισθητήρα imu. Μια δεύτερη προσέγγιση για τον υπολογισμό της δύναμης

θα ήταν η ενσωμάτωση ενός FSR (Force-Sensitive-Resistor) στο tip του ΤΣΔ. Στη συνέχεια θα είναι δυνατό να διεξαχθούν επιπλέον πειράματα, με τη φάση επαφής, αλλά και με διαφορετικές αρχικές συνθήκες. Επίσης μπορεί να αξιοποιηθεί ο δεύτερος gripper που είναι τοποθετημένος στη βάση του ρομπότ.

Επιπλέον, ως μελλοντική δουλειά θα μπορούσε να αποτελέσει η αναβάθμιση του base code του Κηφέα από ROS σε ROS2 αλλά και η δημιουργία ενός εύχρηστου gui για την πειραματική διαδικασία, το οποίο θα γεφυρώνει τον high end user με τις εντολές που τρέχουν στο ROS.

Βιβλιογραφία

- [1] Maxon Group. *Maxon Group - Precision Electric Motors*. <https://www.maxongroup.com/en>.
- [2] RTD Corporation. *RTD ATX3510 Power Module*. <https://www.rtd.com/PC104/UM/power/atx3510.htm>.
- [3] RTD Corporation. *RTD DM35820 Digital I/O Module*. <https://www.rtd.com/PC104/DM/digital%20IO/DM35820.htm>.
- [4] Festo. *D-Filters*. https://www.festo.com/cat/en-gb_gb/data/doc_ENGB/PDF/EN/D-FILTERS_EN.PDF.
- [5] New Way Air Bearings. *Round Bearings Catalog*. <https://www.newwayairbearings.com/catalog/round-bearings/>.
- [6] NASA. *Space Debris - NASA Bibliography*. <https://www.nasa.gov/headquarters/library/find/bibliographies/space-debris/>.
- [7] NASA. *On-Orbit Servicing, Assembly, and Manufacturing*. <https://www.nasa.gov/mission/on-orbit-servicing-assembly-and-manufacturing-1/>.
- [8] European Space Agency. *Active Debris Removal*. https://www.esa.int/Space_Safety/Space_Debris/Active_debris_removal.
- [9] European Space Agency. *ESA Moves Ahead with In-Orbit Servicing Missions*. https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/ESA_moves_ahead_with_In-Orbit_Servicing_missions2.
- [10] European Space Agency. *Clean Space*. https://www.esa.int/Space_Safety/Clean_Space.
- [11] O. Khatib J. Russakow, S. Rock. *An Operational Space Formulation for a Free-flying Multi-arm Space Robot*,. *Experimental Robotics VII*, LNCIS 223, 1987.
- [12] C. Stewart. *GPS Pseudolites: Theory, Design and Applications*,. Διδακτορική Διατριβή, Stanford University, 1997.
- [13] CSL-EP, National Technical University of Athens. *Space Robotics Research*. <https://csl-ep.mech.ntua.gr/research-areas/space-robotics/>.
- [14] N.Hogan. *Stable execution of contact tasks using impedance control*. *IEEE International Conference on Robotics and Automation*, Raleigh,NC,USA, 1987.

- [15] Thomas R. Kurfess, επιμελητής. *Robotics and Automation Handbook*. CRC Press, 1η έκδοση, 2005.
- [16] S. Ali A. Moosavian, Rambod Rastegari και Evangelos Papadopoulos. *Multiple Impedance Control for Space Free-Flying Robots*. *Journal of guidance, control, and dynamics*, 2005.
- [17] Kostas Nanos και Evangelos Papadopoulos. *On Coordinated Compliant Control Design for On-Orbit Capture Operations by Space Manipulator Systems, Submitted to IEEE ICRA 2025*.
- [18] Φ. Ξύδη-Χρυσάφη. *Σχεδιασμός Ελέγχου Εμπέδησης Για Διαστημικό Ρομπότ Που Μεταβάλλει Την Κίνηση Δορυφόρου Με Συνεχή Επαφή*. Διπλωματική εργασία, Εθνικό Μετσόβιο Πολυτεχνείο, 2019.
- [19] Open Robotics Foundation. *ROS Wiki*. <https://wiki.ros.org/>.
- [20] Open Robotics Foundation. *ROS Packages*. <https://wiki.ros.org/Packages>.
- [21] Open Robotics Foundation. *ROS CMakeLists.txt*. <https://wiki.ros.org/catkin/CMakeLists.txt>.
- [22] Open Robotics Foundation. *Catkin package.xml - ROS Wiki*. <https://wiki.ros.org/catkin/package.xml>.
- [23] Gabriel Chollet. *What is ROS? - Medium Article*. https://medium.com/@Gabriel_Chollet/what-is-ros-c38493fe3eca.
- [24] Open Robotics Foundation. *Nodes - ROS Wiki*. <https://wiki.ros.org/Nodes>.
- [25] Open Robotics Foundation. *Topics - ROS Wiki*. <https://wiki.ros.org/Topics>.
- [26] ROS Community. *std_msgs/Float64 Message - ROS Documentation*. https://docs.ros.org/en/melodic/api/std_msgs/html/msg/Float64.html.
- [27] ROS Community. *geometry_msgs/Vector3 Message - ROS Documentation*. https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Vector3.html.
- [28] ROS Community. *geometry_msgs/Pose Message - ROS Documentation*. https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Pose.html.
- [29] Open Robotics Foundation. *Writing a Simple Publisher and Subscriber (C++) - ROS Wiki*. <https://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29>.
- [30] Open Robotics Foundation. *Services - ROS Wiki*. <https://wiki.ros.org/Services>.
- [31] The Robotics Backend. *What is a ROS Service?* <https://roboticsbackend.com/what-is-a-ros-service/>.
- [32] Open Robotics Foundation. *Actionlib - ROS Wiki*. <http://wiki.ros.org/actionlib>.

- [33] Open Robotics Foundation. *RQT - ROS Wiki*. <http://wiki.ros.org/rqt>.
- [34] Open Robotics Foundation. *RQT Plugins - ROS Wiki*. <https://wiki.ros.org/rqt/Plugins>.
- [35] Open Robotics Foundation. *RQT Graph - ROS Wiki*. https://wiki.ros.org/rqt_graph.
- [36] Open Robotics Foundation. *RQT Plot - ROS Wiki*. https://wiki.ros.org/rqt_plot.
- [37] Open Robotics Foundation. *ros_control - ROS Wiki*. https://wiki.ros.org/ros_control.
- [38] Open Robotics Foundation. *controller_manager - ROS Wiki*. https://wiki.ros.org/controller_manager.
- [39] Open Robotics Foundation. *ROS Command Line Tools - ROS Wiki*. <http://wiki.ros.org/ROS/CommandLineTools>.
- [40] Open Robotics Foundation. *roscore - ROS Wiki*. <https://wiki.ros.org/roscore>.
- [41] Open Robotics Foundation. *ROS Master - ROS Wiki*. <https://wiki.ros.org/Master>.
- [42] Open Robotics Foundation. *Parameter Server - ROS Wiki*. <https://wiki.ros.org/Parameter%20Server>.
- [43] Open Robotics Foundation. *rosout - ROS Wiki*. <https://wiki.ros.org/rosout>.
- [44] Open Robotics Foundation. *rostopic - ROS Wiki*. <http://wiki.ros.org/rostopic>.
- [45] Open Robotics Foundation. *ROS Network Setup - ROS Wiki*. <https://wiki.ros.org/ROS/NetworkSetup>.
- [46] Open Robotics Foundation. *Running ROS on Multiple Machines - ROS Wiki*. <https://wiki.ros.org/ROS/Tutorials/MultipleMachines>.
- [47] Open Robotics Foundation. *rostime - ROS Wiki*. <https://wiki.ros.org/rostime>.
- [48] Open Robotics Foundation. *ROS Time Overview - ROS Wiki*. <https://wiki.ros.org/roscpp/Overview/Time>.
- [49] Open Robotics Foundation. *ros::Rate Class - ROS Documentation*. https://docs.ros.org/en/lunar/api/rostime/html/classros_1_1Rate.html.
- [50] The Robotics Backend. *ROS Rate in Python and C++ - Robotics Backend*. <https://roboticsbackend.com/ros-rate-roscpy-roscpp/>.
- [51] O. Christidi Loumpasefski A. Konstantinidis, I. Pelekoudas και E. G. Papadopoulos. *The NTUA Space Robot Emulator for Validation of Closed-Loop On-Orbit Servicing Tasks. 16th Symposium on Advanced Space Technologies in Robotics and Automation, Noordwijk, The Netherlands, 2022.*

- [52] Π. Α. Μαυρίδης. *Ανάπτυξη λογισμικού σε περιβάλλον ROS για την κίνηση βραχιόνων ρομπότ διαστημικού εξομοιωτή και σχεδιασμός τροχιάς για την αυτόματη σύλληψη στόχων*. Διπλωματική εργασία, Εθνικό Και Καποδιστριακό Πανεπιστήμιο Αθηνών, 2019.
- [53] ETH Zurich Autonomous Systems Lab. *Vicon Bridge - GitHub Repository*. https://github.com/ethz-asl/vicon_bridge/tree/master.
- [54] Open Robotics Foundation. *Gazebo Simulator*. <https://classic.gazebosim.org/>.
- [55] ODE Community. *Open Dynamics Engine (ODE) User Guide*. <https://ode.org/ode-latest-userguide.html>.
- [56] Open Robotics Foundation. *Gazebo Physics Parameters - Tutorial*. https://classic.gazebosim.org/tutorials?tut=physics_params&cat=physics.
- [57] Open Robotics Foundation. *ROS Integration with Gazebo - Tutorial*. https://classic.gazebosim.org/tutorials?tut=ros_overview.
- [58] Open Robotics Foundation. *URDF - ROS Wiki*. <https://wiki.ros.org/urdf>.
- [59] Open Robotics Foundation. *URDF Link XML Specification - ROS Wiki*. <https://wiki.ros.org/urdf/XML/link>.
- [60] Open Robotics Foundation. *URDF Joint XML Specification - ROS Wiki*. <https://wiki.ros.org/urdf/XML/joint>.
- [61] Open Robotics Foundation. *URDF Transmission XML Specification - ROS Wiki*. <https://wiki.ros.org/urdf/XML/Transmission>.
- [62] Open Robotics Foundation. *ROS-Gazebo Plugins - Tutorial*. https://classic.gazebosim.org/tutorials?tut=ros_gzplugins.
- [63] Open Robotics Foundation. *Gazebo Force-Torque Sensor Plugin - ROS Documentation*. https://docs.ros.org/en/jade/api/gazebo_plugins/html/group__GazeboRosFTSensor.html.
- [64] Open Robotics Foundation. *Gazebo Wrench Plugin - ROS Documentation*. https://docs.ros.org/en/lunar/api/gazebo_plugins/html/group__GazeboRosForce.html.
- [65] Open Robotics Foundation. *Gazebo and ros_control Integration - Tutorial*. https://classic.gazebosim.org/tutorials?tut=ros_control.
- [66] Open Robotics Foundation. *Using Xacro to Clean Up a URDF File - ROS Wiki*. <https://wiki.ros.org/urdf/Tutorials/Using%20Xacro%20to%20Clean%20Up%20a%20URDF%20File>.
- [67] Ελευθέριος Καντούνιας. *Mechanical Interface for Space Emulator*. Διπλωματική εργασία, Εθνικό Μετσόβιο Πολυτεχνείο, 2024.

Παραρτήματα

Παράρτημα **A'**

Hardware Manuals

DCX22L EB KL 24V



Values at nominal voltage		
Nominal voltage		24 V
No load speed		5060 min ⁻¹
No load current		10.2mA
Nominal speed		4070 min ⁻¹
Nominal torque (max. continuous torque)		29.2 mNm
Nominal current (max. continuous current)		0.655 A
Stall torque		150 mNm
Stall current		3.31 A
Max. efficiency		89.3 %
Characteristics		
Max. output power continuous		19.8 W
Terminal resistance		7.25 Ω
Terminal inductance		0.746 mH
Torque constant		45.2 mNm A ⁻¹
Speed constant		211 min ⁻¹ V ⁻¹
Speed/torque gradient		33.9 min ⁻¹ mNm ⁻¹
Mechanical time constant		3.14ms
Rotor inertia		8.85 gcm ²
Thermal data		
Thermal resistance housing-ambient		13.6 KW ⁻¹
Thermal resistance winding-housing		4.57 KW ⁻¹
Thermal time constant of the winding		21 s
Thermal time constant of the motor		646 s
Ambient temperature		-40...85 °C
Max. winding temperature		100 °C
Mechanical data		
Max. permissible speed		7160 min ⁻¹
Axial play		0...0.1 mm
Preload		2.5 N
Radial backlash		0.02 mm
Max. axial load (dynamic)		2.5 N
Max. force for press fits (static)		30 N
Static, supported shaft		440 N
Max. radial load 5 mm from flange		16 N
Measurement from the flange		5 mm
Further specifications		
Number of pole pairs		1
Number of commutator segments		9
Motor weight		100.3 g
Motor length		46.7 mm
Typical noise level		52 dBA (min ⁻¹)

Σχήμα Α'.1: Τα χαρακτηριστικά και οι ιδιότητες του κινητήρα που χρησιμοποιείται για την επενέργηση του βραχίονα [1].

GPX26 LZ 186:1



Product specification		
Gearhead data		
Reduction		186:1
Absolute reduction		314226/1690
Number of stages		3
Max. continuous torque		4.50 Nm
Max. intermittent torque		6.2 Nm
Direction of rotation, drive to output		=
Max. efficiency		75 %
Average backlash no-load		0.9 *
Mass inertia		1.282 gcm ²
Max. transmittable power (continuous)		12 W
Max. short-time transferable output		15 W
Technical data		
Output shaft bearing		KL
Max. radial play, 5 mm from flange		max. 0.1 mm
Axial play		0...0.4 mm
Max. permissible radial load, 10 mm from flange		150 N
Max. permissible axial load		80 N
Max. permissible force for press fits		120 N
Max. continuous input speed		10000 min ⁻¹
Max. intermittent input speed		12500 min ⁻¹
Recommended temperature range		-40...100 °C

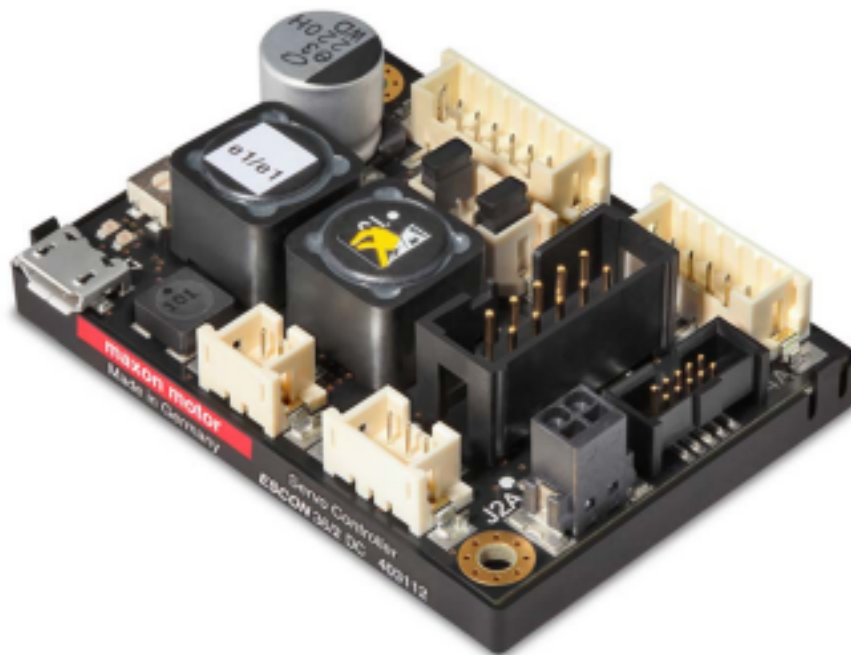
Σχήμα Α'.2: Τα χαρακτηριστικά και οι ιδιότητες του μειωτήρα που συνδέεται με τον κινητήρα και αποτελεί την έξοδο της ισχύος του συστήματος επενέργειας [1] .

ENX16 EASY 1000IMP



Product specification		
Sensor data		
	Counts per turn	1000
	Number of channels	3
	Line Driver	RS422
	Max. electrical speed	90000 min ⁻¹
	Max. mechanical speed	30000 min ⁻¹
Technical data		
	Supply voltage Vcc	5 V ±10 %
	Output signal	INC
	Output signal driver	Differential / EIA RS 422
	Output current per channel	-20...20 mA
	State length	10...170 °el
	Signal rise time/Signal fall time	20/20 ns
	Min. state duration	125 ns
	Direction of rotation	A before B CW
	Index position	A low & B low
	Index synchronously to AB	Yes
	Index pulse width	90 °el
	Typical current draw at standstill	22 mA
	Max. moment of inertia of code wheel	0.05 gcm ²
	Operating temperature range	-40...100 C°
	Number of autoclave cycles	0

Σχήμα Α'.3: Η παλμογεννήτρια (encoder) που συνδέεται με τον κινητήρα και μετρά τη γωνία περιστροφής της ατράκτου του [1] .



Σχήμα Α'.4: Εικόνα του κάθε driver για κάθε κινητήρα του βραχίονα. Είναι το μοντέλο ESCON 36/2 DC.

ESCON 36/2 DC (403112)		
Electrical Rating	Nominal operating voltage $+V_{CC}$	10...36 VDC
	Absolute operating voltage $+V_{CC \min} / +V_{CC \max}$	8 VDC / 38 VDC
	Output voltage (max.)	$0.98 \times +V_{CC}$
	Output current I_{cont} / I_{max} (<60 s)	2 A / 4 A
	Pulse Width Modulation frequency	53.6 kHz
	Sampling rate PI current controller	53.6 kHz
	Sampling rate PI speed controller	5.36 kHz
	Max. efficiency	95%
	Max. speed	limited by max. permissible speed (motor) and max. output voltage (controller)
	Built-in motor choke	300 μ H; 2 A
Inputs & Outputs	Analog Input 1 Analog Input 2	resolution 12-bit; -10...+10 V; differential
	Analog Output 1 Analog Output 2	resolution 12-bit; -4...+4 V; referenced to GND
	Digital Input 1 Digital Input 2	+2.4...+36 VDC ($R_i = 38.5 \text{ k}\Omega$)
	Digital Input/Output 3 Digital Input/Output 4	+2.4...+36 VDC ($R_i = 38.5 \text{ k}\Omega$) / max. 36 VDC ($I_L < 500 \text{ mA}$)
	Encoder signals	A, A $\bar{}$, B, B $\bar{}$, (max. 1 MHz)
Voltage Outputs	Auxiliary output voltage	+5 VDC ($I_L \leq 40 \text{ mA}$)
	Encoder supply voltage	+5 VDC ($I_L \leq 70 \text{ mA}$)
Potentiometer	Potentiometer P1 (on board)	210°; linear
Motor Connections	+ Motor	
	- Motor	
Interface	USB 2.0 / USB 3.0	full speed
Status Indicators	Operation	green LED
	Error	red LED
Physical	Weight	approx. 30 g
	Dimensions (L x W x H)	55 x 40 x 16.1 mm
	Mounting holes	for M2.5 screws (max. tightening torque 0.16 Nm)

Σχήμα Α'.5: Χαρακτηριστικά των drivers των κινητήρων του βραχίονα.



Σχήμα Α'.6: Εικόνα του driver για τον κινητήρα του Reaction Wheel. Είναι το μοντέλο ESCON 50/5.

ESCON 50/5 (409510)		
Electrical Rating	Nominal operating voltage V_{CC}	10...50 VDC
	Absolute operating voltage $V_{CC \min} / V_{CC \max}$	8 VDC / 56 VDC
	Output voltage (max.)	$0.98 \times V_{CC}$
	Output current I_{cont} / I_{max} (<20 s)	5 A / 15 A
	Pulse Width Modulation frequency	53.6 kHz
	Sampling rate PI current controller	53.6 kHz
	Sampling rate PI speed controller	5.36 kHz
	Max. efficiency	95%
	Max. speed DC motor	limited by max. permissible speed (motor) and max. output voltage (controller)
	Max. speed EC motor	150'000 rpm (1 pole pair)
	Built-in motor choke	3 x 30 μ H; 5 A
Inputs & Outputs	Analog Input 1 Analog Input 2	resolution 12-bit; -10...+10 V; differential
	Analog Output 1 Analog Output 2	resolution 12-bit; -4...+4 V; referenced to GND
	Digital Input 1 Digital Input 2	+2.4...+36 VDC ($R_i = 38.5 \text{ k}\Omega$)
	Digital Input/Output 3 Digital Input/Output 4	+2.4...+36 VDC ($R_i = 38.5 \text{ k}\Omega$) / max. 36 VDC ($I_L < 500 \text{ mA}$)
	Hall sensor signals	H1, H2, H3
	Encoder signals	A, A $\bar{}$, B, B $\bar{}$, (max. 1 MHz)
Voltage Outputs	Auxiliary output voltage	+5 VDC ($I_L \leq 10 \text{ mA}$)
	Hall sensor supply voltage	+5 VDC ($I_L \leq 30 \text{ mA}$)
	Encoder supply voltage	+5 VDC ($I_L \leq 70 \text{ mA}$)
Potentiometers	Potentiometer P1 (on board) Potentiometer P2 (on board)	240°; linear
Motor Connections	DC motor	+ Motor, - Motor
	EC motor	Motor winding 1, Motor winding 2, Motor winding 3
Interface	USB 2.0	full speed (12 Mbit/s)
Status Indicators	Operation	green LED
	Error	red LED

Σχήμα Α'.7: Χαρακτηριστικά του driver για τον κινητήρα του Reaction Wheel.

ATX3510HR-190W

PCIe/104 190 Watt Power Supply

Operating Temperature -40° to +85°C

Key Features

- PC/104 form factor
- PCIe/104 stackable bus structure
 - PCIe universal expansion bus (Type 1 or Type 2)
- PCIe x1 interface
- -40 to +85°C standard operating temperature
- Input Voltage Range: 8-36V
- 194.8 Watts maximum output power
- Outputs
 - +5 Vdc at 16.8 amps
 - +12 Vdc at 8.4 amps
 - +5 Vdc Standby at 2.0 amps
 - +3.3 Vdc at 2.0 amps
- ATX signals
 - +5VSB
 - PS_ON#
 - PWRGOOD
- Efficiency as high as 90%
- Reverse voltage protection to -40V
- Input over-current protection
- Input over voltage protection clamps to 36V
- Output over-current protection
- Remote ON/OFF function
- Synchronized supplies to reduce noise
- Spread spectrum capable
- No Electrolytic caps
- Includes pin fin heat sink
- Available in *stackable, rugged enclosures*



ATX3510HR 190 Watt Embedded Power Supply



(shown with pin fin heat sink installed)



Rugged IDAN configuration available

Description

The ATX3510HR is a 190 Watt embedded power supply with a PCIe/104 stackable bus structure. With 5V standby, 5V main, and 12V, this module is also input protected against over-voltage, over-current, and reverse voltage. This rugged power supply is synchronized to reduce noise and system stress. The small footprint and high power capabilities of the ATX3510HR-190W make it suitable for many technology-rich, high-reliability military, industrial, and research applications.

Physical Attributes

- Dimensions [view diagram]
 - Length (L): 3.775 inches (95.89 mm)
 - Width (W): 3.650 inches (90.17 mm)
 - Stand-off Height
 - Above the module: 0.866 inches (22.00 mm)
 - Below the module: 0.600 inches (15.24 mm)
- Weight: Approximately 0.48 lbs. with heat sink
- Standard Operating Temperature: -40 to +85°C, 90% humidity non-condensing
- Storage Temperature: -55 to +125°C
- MTBF: 3,693,562 hours
- Max Power Dissipation: 21.64 Watts

Σχήμα Α'.8: Πηλακέτα ρύθμισης τάσης που είναι εγκατεστημένη στο stack του PC104 και τροφοδοτεί το Arduino shield με 12V [2].

DM35820HR

PCIe/104 High-Speed Digital I/O DAQ Module
Operating Temperature -40° to +85°C

Key Features

- PCI/104 form factor
- PCIe/104 stackable bus structure [click here for PCIe/104-Express]
 - PCIe universal expansion bus (Type 1 or Type 2)
- PCIe x1 interface
- 48 Diode protected I/O lines
- 2 MB Input FIFO buffer onboard
- Standard outputs: -24mA / +24mA (source / sink)
- On-board 25 MHz clock
- Compatible with DMR & DOP expansion boards
- Mating Connector
 - Two Standard 0.1 inch spaced 2 x 25
- Available in stackable, rugged enclosures



DM35820 High-Speed DAQ Module

Description

The DM35820 is a high-speed digital I/O data module in a PCIe/104 format. This module includes many features such as pulse width modulators (PWM), incremental encoders, and advanced interrupts.

Technical Details

- Digital I/O
 - 48 Diode protected I/O lines
 - 24 mA source and sink current
 - Compatible with DMR and DOP expansion boards
 - Deep FIFOs with DMA
 - Two 2M-Word FIFOs
 - Each FIFO is attached to a separate DMA channel
 - 25 MHz burst throughput
 - 12.5 MHz continuous throughput
 - FIFO can be looped
- Pulse Width Modulators
 - Eight PWM outputs
 - Single-ended or Differential Outputs
 - 18-bit resolution
 - Separate period and width clocks provide full resolution at low duty cycles
- Optional Interrupt generations
- Incremental Encoders
 - Four Incremental Encoder channels
 - Single-ended or Pseudo-differential Inputs
 - Variable frequency input filtering
 - Max input speed of 40ns per transition
 - 18-bit resolution
 - Two channels can be combined for 32-bit resolution
 - Connect to FIFO for position sampling
- Advanced Interrupts
 - Two Advanced Interrupt Modules
 - Interrupt on Match, Change, or Strobe
 - All 48 bits are captured when the interrupt is generated
 - Any combination of the 48 bits can be monitored
- Programmable Clocks
 - Four programmable clocks
 - Maximum frequency of 25 MHz
 - Can be started and stopped by an interrupt or another clock
 - Continuous or One-Shot Operation
 - Can be cascaded
- 82C54 Timer/Counters
 - Six Timer/Counter Channels
 - Fully programmable
 - Input clock and gate driven from internal or external source
 - 10 MHz maximum input



Rugged DAQ configurations available
Product modifications available upon request

Software

- Includes software packages for the following Operating Systems:
 - DOS
 - Windows XP, Windows 7 (32/64 bit), and Windows 10 (32/64 bit)
 - Linux 2.4.x and 2.6.x
- Source code provided for easy porting to other platforms, including RTOSes
- Data acquisition toolkits such as LabVIEW and MATLAB may be used with the Windows drivers
- Example programs with source code provide a starting point for developing custom applications

Σχήμα Α.9: Πηλακία ψηφιακών εισόδων-εξόδων που είναι εγκατεστημένη στο stack του PC104. Είναι υπεύθυνη για την αποστολή σημάτων-εντολών στα shields των κινητήρων και λαμβάνει τις μετρήσεις από τις παλμογεννήτριες των κινητήρων [3].

MIR700 Series

Single Stage, Compact Regulator
Stainless Steel or Brass



Value Proposition:

The MIR700 Series regulator offers a compact single stage regulator designed for low to medium pressure applications.

Constructed from brass or stainless steel, this unit is capable of handling a broad range of media. Reliable performance and modest size make the MIR700 Series regulator ideal for applications that require pressure control in a compact space.



Contact Information:

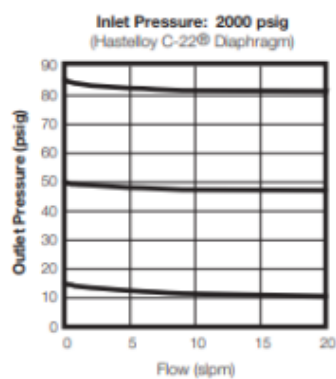
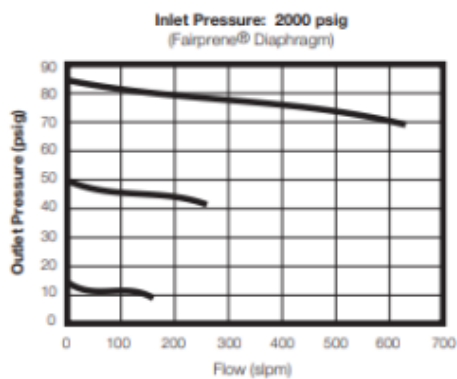
Parker Hannifin Corporation
Veriflo Division
250 Canal Blvd
Richmond, California 94804

phone 510 235 9590
fax 510 232 7396
veriflo.sales@parker.com

www.parker.com/veriflo
Mobile App: m.parker.com/veriflo

Product Features:

- Precise flexing Hastelloy C-22® diaphragm
- Low internal volume
- Cleaned for O₂ service is standard
- Machined from solid bar stock
- Proven valve seat assembly



Σχήμα Α.10: Ο ρυθμιστής υψηλής πίεσης που τοποθετείται αμέσως μετά τις φιάλες διοξειδίου για να ρίξει την πίεση των 50bar.

pressure regulator LRMA-M5-QS-4

Part number: 153488

Product to be discontinued

With pressure gauge.

Type to be discontinued. Available until 2021. See Support Portal for alternative products.

FESTO



Data sheet

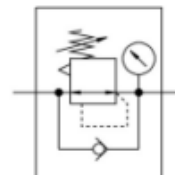
Feature	Value
Controller function	Output pressure constant with secondary exhaust
Pneumatic connection, port 1	M5
Pneumatic connection, port 2	QS-4
Mounting type	with through hole
Standard nominal flow rate	36 l/min
Inlet pressure 1	0 ... 9 bar
Ambient temperature	0 ... 60 °C
Material housing	PBT-reinforced
Operating medium	Compressed air in accordance with ISO8573-1:2010 [7:-:-]
Actuator lock	Knurled screw with lock nut
Assembly position	Any
Design structure	directly-controlled piston regulator with looped-through compressed air supply
Pressure gauge	with pressure gauge
Pressure regulation range	1 ... 8 bar
Product weight	27.5 g
Material of threaded plug	Brass
	Nickel plated
Material of threaded seal	PTFE

Σχήμα Α.11: Τα χαρακτηριστικά του ρυθμιστή πίεσης που χρησιμοποιείται για τον έλεγχο του αερίου στα αεροέδρανα του ρομπότ.

Pressure regulator LRMA-QS-6

Part number: 153496

FESTO



Data sheet

Feature	Value
Type code	LRMA
Controller function	Outlet pressure constant With secondary exhausting
Selection of additional function	Pressure indicator with pressure gauge
Pneumatic connection 1	QS-6
Pneumatic connection 2	QS-6
Type of mounting	With through-hole
Standard nominal flow rate	76 l/min
Inlet pressure 1	0 bar ... 9 bar
Ambient temperature	0 °C ... 60 °C
Housing material	PBT-reinforced
Operating medium	Compressed air as per ISO 8573-1:2010 [7:-:-]
Actuator lock	Counter-locking
Mounting position	Any
Design	with continuous compressed air supply Directly controlled piston regulator
Symbol	00991536
Pressure gauge	with pressure gauge
Pressure regulation range	1 bar ... 8 bar
Product weight	45 g
Material of screwed trunnion	Brass Nickel-plated
Material of threaded seal	PTFE

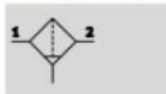
Σχήμα Α'.12: Τα χαρακτηριστικά του ρυθμιστή πίεσεως που χρησιμοποιείται στο νέο πνευματικό για τον έλεγχο του αερίου στα αεροέδρανα της αρπάγης.

Filters LF, D series, metal design

Technical data

FESTO

Manual rotary condensate drain



Semi or fully automatic condensate drain



Flow rate
170 ... 5,300 l/min

Temperature range
-10 ... +60 °C

Operating pressure
0 ... 16 bar



- Good particle and condensate separation
- High flow rate with minimal pressure drop

- With or without connecting plates
- Choice of filter cartridges:
5 µm or 40 µm

- With manual or fully automatic condensate drain
- New filter cartridges → 37

General technical data				
Size	Micro	Mini	Midi	Maxi
Pneumatic connection 1, 2 ¹⁾	M5 ²⁾ , M7 ²⁾ , G1/8, QS-4 ³⁾ , QS-6 ³⁾	G1/8, G1/4, G3/8, - ⁴⁾	G1/4, G3/8, G1/2, G3/4, - ⁴⁾	G1/2, G3/4, G1, - ⁴⁾
Design	Sintered filter with centrifugal separator			
Type of mounting	Via accessories			
Mounting position	Vertical ±5°			
Grade of filtration [µm]	5	5 40		
Air purity class at the output	Compressed air according to ISO 8573-1:2010 [6:8:4] (grade of filtration 5 µm)			
	Compressed air according to ISO 8573-1:2010 [7:8:4] (grade of filtration 40 µm)			
	Inert gases			
Bowl guard	Metal bowl guard			
Condensate drain	Manual rotary			
	Semi-automatic			
	Fully automatic			
Max. condensate volume [cm ³]	3	22	43	80

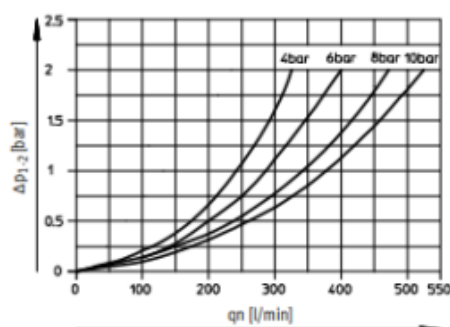
1) Connecting plates with connecting thread

2) Without connecting plates, connecting thread in housing

3) Connecting plates with push-in connector

4) Without connecting plates

Note: This product conforms to ISO 1179-1 and to ISO 228-1



Σχήμα Α'.13: Το φίλτρο που χρησιμοποιείται στο πνευματικό κύκλωμα του ρομπότ και χρησιμοποιείται για την απομάκρυνση νερού από το κύκλωμα. Ο τύπος του είναι LF-M7-D-MICRO B + 1/8"-M7 adapter [4].

SPECIFICATIONS: 25mm #S102501	
Input Pressure	.41 MPa (60psi)
Ideal Load N (lbs)	80 (18)
Stiffness N/micron (lbs/u in) ¹	18 (0.10)
Flow NLPM (SCFH) ¹	0.53 (1.2)
Fly Height ¹	5 microns
Bearing Size	25mm
Bearing Height mm (in)	13 (0.52)
Bearing Weight grams (oz.)	14 (0.5)
Housing Material/Finish	Aluminum/Anodized
Porous Media Material	Carbon
Bearing Face Surface Size - Carbon mm (in)	23mm (0.906) ϕ
Flatness mm (in)	0.0005 (0.00002)
Ball Socket Size	13mm
Pressure Port Thread	M3 X 0.5
Viable Pressure Range	.414-.552MPa (60psi - 80psi)
Maximum Allowable Pressure Supply	.689 MPa (100psi)
Resolution ³	Infinite
Maximum Speed ⁴	50m/sec
Common Guide Surfaces	Granite, hard-coated aluminum, ceramics, glass, stainless steel, plated steel
Suggested Guide Surface Finish ⁵	16 RMS

Σχήμα Α'.14: Τα χαρακτηριστικά των αεροεδράνων της εταιρείας NEWWAY-AIRBEARINGS που χρησιμοποιούνται στο διαστημικό σύστημα με διάμετρο 25mm [5].

SPECIFICATIONS: 40mm #S104001	
Input Pressure	.41 MPa (60psi)
Ideal Load N (lbs)	222 (50)
Stiffness N/micron (lbs/u in) ¹	28 (0.16)
Flow NLPM (SCFH) ¹	0.74 (1.6)
Fly Height ¹	5 microns
Bearing Size	40mm
Bearing Height mm (in)	13 (0.52)
Bearing Weight grams (oz.)	35 (1.2)
Housing Material/Finish	Aluminum/Anodized
Porous Media Material	Carbon
Bearing Face Surface Size - Carbon mm (in)	38 (1.49) ϕ
Flatness mm (in)	0.0005 (0.00002)
Ball Socket Size	13mm
Pressure Port Thread	M5 X 0.8
Viable Pressure Range	.414-.552MPa (60psi - 80psi)
Maximum Allowable Pressure Supply	.689 MPa (100psi)
Resolution ³	Infinite
Maximum Speed ⁴	50m/sec
Common Guide Surfaces	Granite, hard-coated aluminum, ceramics, glass, stainless steel, plated steel
Suggested Guide Surface Finish ⁵	16 RMS

Σχήμα Α'.15: Τα χαρακτηριστικά των αεροεδράνων της εταιρείας NEWWAY-AIRBEARINGS που χρησιμοποιούνται στο διαστημικό σύστημα με διάμετρο 40mm [5].

Παράρτημα **B'**

Γενικές οδηγίες Κηφέα

!!! Αναλυτικές οδηγίες φόρτισης των μπαταριών βρίσκονται στο αντίστοιχο pdf το οποίο παρέχεται στο USB stick της παρούσας διπλωματικής.

B'.1 Οδηγίες για γέμισμα και αποθήκευση φιάλης CO₂

Οι φιάλες αποθηκεύονται πάντα στην κατάψυξη, ώστε να μπορούν να αποθηκεύουν περισσότερο αέριο χωρίς να κινδυνεύουν από την υψηλή πίεση (50bar). Για γέμισμα, χρησιμοποιείται το tank με το fill station στο ισόγειο του κτιρίου M. Η μπουκάλα βιδώνεται στο fill station, βεβαιωνόμαστε ότι η έξοδος προς το περιβάλλον είναι κλειστή, ανοίγουμε τη μπουκάλα και στη συνέχεια το tank και περιμένουμε να γεμίσει. Κλείνουμε το tank, τη μπουκάλα, ανοίγουμε με προσοχή την έξοδο για να εκτονωθεί η αποθηκευμένη πίεση και ξεβιδώνουμε.

!!! Πολλή προσοχή και χρήση γαντιών γιατί το αέριο βρίσκεται σε πίεση 50 bar και αποθηκεύεται σε πολύ χαμηλή θερμοκρασία. Συνίσταται η επίβλεψη ενός υπεύθυνου κατά τη πρώτη φορά γεμίσματος. Συμβουλευτείτε και το σχετικό βίντεο επίδειξης γεμίσματος φιάλης που παρέχεται στο USB stick της παρούσας διπλωματικής.

B'.2 Οδηγίες διεξαγωγής πειράματος

ΓΙΑ ΠΕΙΡΑΜΑ ΜΕ ROS NETWORK

Αρχικά, βεβαιώνομαι ότι έχω κάνει τα εξής:

Στο PC:

1. `gedit .bashrc`

(α) **`export ROS_MASTER_URI=http://localhost:11311`**

(β) **`export ROS_HOSTNAME=192.168.1.38`**

(γ) **`export ROS_IP=192.168.1.38`**

, όπου **192.168.1.38** το IP Address του PC, ο οποίος θα είναι host του ros network Για να δεις το ip address σου πατάς `ifconfig` σε terminal.

Στο PC104:

1. `gedit .bashrc`

- (α) **export ROS_MASTER_URI=http://192.168.1.38:11311** (το IP address του host ,δηλαδή του υπολογιστή μας)
- (β) **export ROS_HOSTNAME=192.168.1.58** (το IP address του PC104)
- (γ) **export ROS_IP=192.168.1.58** (το IP address του PC104)

Παρακάτω για συντομία θα λέμε: [PC] ο υπολογιστής του εργαστηρίου, στη περίπτωση μου το δικό μου PC από το directory: /cephheus_impedance_tsoulas. [cephheus] Το PC104 του Κηφέα.

!ΠΑΝΤΑ source devel/setup.bash σε κάθε νέο terminal! (για να χρησιμοποιούμε τα ROS πακέτα)

ΠΡΟΕΤΟΙΜΑΣΙΑ ΠΕΙΡΑΜΑΤΟΣ

1. Προσέχουμε μην έχει υγρασία κοντά στο σημείο της φιάλης λόγω προηγούμενου πειράματος και σκουπίζουμε αν και όπου χρειαστεί.
2. Ανοίγουμε αρχικά τον πρώτο διακόπτη του PC104 για να φορτώσουν τα linux του PC104.
3. Ανοίγουμε τον υπολογιστή της vicon (password: 123), βάζουμε στην πρίζα το vicon hub, τρέχουμε το vicon tracker. (οδηγίες για την χρήση του vicon tracker υπάρχουν άφθονες στο διαδίκτυο).
4. Αφαιρούμε το κάλυμμα της τράπεζας και την καθαρίζουμε με το ειδικό πανί και υγρό.
5. Αφού έχουμε γεμίσει τις φιάλες, τις βάζουμε στα 2 ρομπότ και τις **ΔΕΝΟΥΜΕ**, ρυθμίζουμε τους regulators ώστε η πίεση στα air bairings των ρομπότ να είναι στα 5-6 bar. Φοράμε γάντια όταν πιάνουμε τις φιάλες.
6. Αν θέλουμε να τροποποιήσουμε την διάταξη χειροκίνητα (π.χ. για να αρχικοποιήσουμε τους encoders), δεν ανοίγουμε τους 2 διακόπτες των motors και των thrusters/gripper μέχρι να τελειώσουμε.
7. Ανοίγουμε και τους 2 διακόπτες και είμαστε έτοιμοι για πείραμα, όσον αφορά την προεργασία.

ΠΡΟΕΤΟΙΜΑΣΙΑ ΥΠΟΛΟΓΙΣΤΗ ΓΙΑ ΠΕΙΡΑΜΑ

Σε διαφορετικά terminals κάνουμε τα εξής:

1. [PC] **roscore**
2. [PC] **roslaunch cepheus_control robot_foros_controller_node** (στο τέλος αυτό, όταν όλα τα άλλα είναι έτοιμα, παρακάτω είναι όλα τα βήματα που προετοιμάζουν το πείραμα)
3. [PC] **roslaunch vicon_bridge vicon.launch**
4. [PC] σε ετοιμότητα το ./safeclose.sh (αν θέλω να μηδενίσω τις ροπές των αρθρώσεων και reaction wheel)

5. [PC] για plots:

(α') **cd pyscripts**

(β') **python3 angles_plotter.py bag_onoma**

6. [PC] **ssh cepheus@192.168.1.58** (pass: free!):

(α') [cepheus] **cd cepheus_ws**

(β') [cepheus] **roslaunch cepheus_robot launch_cepheus_robot** (για να ανοίξει το interface και οι ελεγκτές)

7. [PC] **ssh cepheus@192.168.1.58** (pass: free!):

(α') [cepheus] **cd cepheus_ws**

(β') [cepheus] **roslaunch botasys_force_torque_sensor calibration.launch** (περιμένεις να τελειώσει μόνο του)

(γ') [cepheus] **roslaunch botasys_force_torque_sensor driver.launch** (για τον ft sensor , βεβαιώσου ότι στα launch αρχεία η διεύθυνση του ft sensor είναι η σωστή καθώς με αποσυνδέσεις μπορεί να αλλάξει, η πιο πρόσφατη είναι /dev/ttyUSB0)

8. [PC] **ssh cepheus@192.168.1.58** (pass: free!):

(α') [cepheus] **cd cepheus_ws**

(β') [cepheus] **roslaunch rosserial_python serial_node.py /dev/ttyUSB1** (για το arduino του gripper, βεβαιώσου ότι η διεύθυνση /dev/ttyUSB1 είναι η σωστή , μπορεί να είναι /dev/ttyUSB0,1,2,3 κλπ ή /dev/ttyACM0,1,2 κλπ)

Μετά το πείραμα, καλό είναι πριν κλείσουμε τον διακόπτη του PC104 και τον διακόπτη ασφαλείας, να κλείσουμε το PC104 μέσω ενός terminal με την εντολή:

sudo poweroff

Αν κολλήσει το cepheus interface, απλά κλείνουμε το interface και το ξαναδοκιμάζουμε, ή κλείνουμε το roscore και το ξαναοιόγουμε. Αν δεν δουλέψουν αυτά, κλείνουμε το PC104 και ξαναμπαίνουμε.

ΣΥΝΤΟΜΗ ΠΕΡΙΓΡΑΦΗ

Ουσιαστικά όλα τα terminals εκτός του (2) είναι προετοιμασία του πειράματος, το βήμα (2) αποτελεί το πείραμα, το terminal(4) είναι ασφάλεια σε περίπτωση που θέλουμε να μηδενίσουμε ροπές (0.0001Nm για την ακρίβεια καθώς στα ακριβώς 0 Nm οι motor drivers βγάζουν error).

Το terminal (5) είναι για να plotάρουμε τα bags που αποθηκεύουμε σε κάθε πείραμα. Γενικά μπορούν να γίνουν λιγότερα τα terminals αν βάζαμε μακροεντολές (roslaunch αρχεία), δεν τα προτίμησα όμως για να έχω καλύτερη εποπτεία και πρόσβαση σε κάθε πρόγραμμα ξεχωριστά.

Ο χρόνος πτήσης για κάθε πείραμα είναι κάπου στα **5-10 λεπτά** ,γιαυτό προσέχουμε συνέχεια τις τιμές πίεσης των airbairings να μην πέφτουν κάτω από 5 bar. Επίσης προσέχουμε μην πιάσει **πάγο** η φιάλη και αφήσει υγρασία κοντά στα καλώδια.

Όταν τελειώσουμε το πείραμα, κλείνουμε τους κινητήρες και thrusters πρώτα και κατεβάζουμε το ρομπότ, προσοχή στη τράπεζα και στα air bearings. **ΔΕΝ ΞΕΧΝΑΜΕ ΝΑ ΚΛΕΙΣΟΥΜΕ ΤΗΝ ΦΙΑΛΗ ΚΑΙ ΝΑ ΤΗΝ ΒΑΛΟΥΜΕ ΑΜΕΣΩΣ ΚΑΤΑΨΥΞΗ**, ΜΕ ΓΑΝΤΙΑ ΠΑΝΤΑ. Επίσης δεν ξεχνάμε να κλείσουμε τον υπολογιστή της vicon και να βγάλουμε απο την πρίζα το hub.

ΟΠΤΙΚΑ INDICATIONS

Τα μπλε φωτάκια στις κάρτες εισόδου εξόδου δείχνουν καλή λειτουργία, αν σβήσει κάποιο χρειάζεται αλλαγή η αντίστοιχη ασφάλεια, έχουμε ασφάλειες κατάλληλες στο συρτάρι διπλά απο το PC(αριστερά).

Τα φωτάκια των motor drivers ανάβουν πράσινα αν τρέχει το cepheus interface και λειτουργούν σωστά. Αν ανάψουν κόκκινα σημαίνει οτι ασκήθηκε μεγάλη ροπή και σβήσανε, ή έγινε άλμα στη ροπή, ή το κουνήσαμε εμείς απότομα. Γιαυτό δεν ανάβουμε τα motors πριν κάνουμε όποια κίνηση θέλουμε με τα χέρια μας πάνω στο ρομπότ, ή αν χρειάζεται, το κάνουμε πολύ προσεκτικά (περισσότερα στο documentation του driver και στο αντίστοιχο pdf το οποίο παρέχεται στο USB stick της παρούσας διπλωματικής).

Κώδικες

Γ'.1 robot_functions.h

Ο κώδικας με τις βασικές συναρτήσεις ελέγχου:

```
1 #ifndef ROBOT_FUNCTIONS_H
2 #define ROBOT_FUNCTIONS_H
3 #include "includes.h"
4 #include "robot_variables.h"
5
6
7
8
9
10 void initialiseParameters(){
11     m0 = 12;
12     r0x=0.1425;
13     r0y =0;
14     mt = 10;
15     q01 = 0;
16     s01 = 0.5;
17     s02 = 0.2;
18     m1 = 0.4409;
19     m2 = 0.1304;
20     m3 = 1;
21
22     l1 = 0.269;
23     l2 = 0.15;
24     l3 = 0.15;
25
26     r1 = 0.1010;
27     r2 = 0.143;
28     r3 = 0.15;
29
30
31     ibzz = 0.27;
32     ilzz = 0.0068;
33     i2zz = 0.010;
```

```

34     i3zz = 0.007908;
35
36
37     double x1,x2,x3;
38     double radius = 0.2;
39     double y = 0.02;
40     x1 = l1+r1;
41     x2 = l2+r2;
42     x3 = l3+r3;
43
44
45
46 }
47
48 double movingMedian(double new_value, std::deque<double>& window, int window_size, double
    alpha, double smoothed_value) {
49
50     window.push_back(new_value);
51
52     if (window.size() > window_size) {
53         window.pop_front();
54     }
55
56     std::vector<double> sorted_window(window.begin(), window.end());
57     std::sort(sorted_window.begin(), sorted_window.end());
58
59
60     double median_value = sorted_window[sorted_window.size() / 2];
61     smoothed_value = alpha * median_value + (1 - alpha) * smoothed_value;
62     // return smoothed_value;
63     return median_value;
64 }
65
66 void calculateTrajecotryPolynomials(double tf){
67     Eigen::MatrixXd eq_matrix(3,3);
68     Eigen::VectorXd eq_bscale(3);
69     double ts = 0.1*tf;
70     double wn = 6/ts;
71     kprop_mb = wn*wn;
72     kder_mb = 2*wn;
73
74
75     eq_matrix << pow(tf,3), pow(tf,4), pow(tf,5),
76                 3*pow(tf,2), 4*pow(tf,3), 5*pow(tf,4),
77                 6*tf, 12*pow(tf,2), 20*pow(tf,3);
78
79     eq_bscale << 1 , 0, 0;
80

```

```

81 Eigen::VectorXd res = eq_matrix.colPivHouseholderQr().solve(eq_bscale);
82
83 a0 = a1 = a2 = 0; //from paper calculations, for t0 = 0
84 a3 = res(0);
85 a4 = res(1);
86 a5 = res(2);
87 }
88
89
90 void finalTrajectories(double t,double tf){
91
92     if(firstTime){ //initialize the position of chaser and target for the first time ONLY
93         xE_in = ee_x;
94         yE_in = ee_y;
95         xt_in = xt;
96         yt_in = yt;
97         thetaE_in = thetach;
98         thetat_in = thetat;// - M_PI/4; //για να γπαρχει μια διαφορά hehe
99         theta0in = theta0;
100        theta0fin = theta0;
101        firstTime = false;
102        ROS_INFO("[in_final_trajectories]:_First_positions_have_been_recorded_(xE_in_etc)._\n");
103    }
104    double s,sdot, sdotdot;
105
106    s = a0 + a1*t + a2*pow(t,2) + a3*pow(t,3) + a4*pow(t,4) + a5*pow(t,5);
107    sdot = a1 + 2*a2*t + 3*a3*pow(t,2) + 4*a4*pow(t,3) + 5*a5*pow(t,4);
108    sdotdot = 2*a2 + 6*a3*t + 12*a4*pow(t,2) + 20*a5*pow(t,3);
109    double xstepfr, ystepfr, thstepfr, theta0stepfr;
110    double xstepdotfr, ystepdotfr, thstepdotfr, theta0stepdotfr;
111    double xstepdotdotfr, ystepdotdotfr, thstepdotdotfr, theta0stepdotdotfr;
112    double xstepc, ystepc, thstepc, theta0stepc;
113    double xstepdotc, ystepdotc, thstepdotc, theta0stepdotc;
114    double xstepdotdotc, ystepdotdotc, thstepdotdotc, theta0stepdotdotc;
115    double a11 = pow(10,7);
116    double a22 = pow(10,-7);
117
118    xstepfr = xE_in + s*(xt_in - xE_in);
119    ystepfr = yE_in + s*(yt_in - yE_in);
120    thstepfr = thetaE_in + s*(thetat_in - thetaE_in);
121    theta0stepfr = theta0in + s*(theta0fin - theta0in);
122
123    xstepdotfr = sdot*(xt_in-xE_in);
124    ystepdotfr = sdot*(yt_in - yE_in);
125    thstepdotfr = sdot*(thetat_in - thetaE_in);
126    theta0stepdotfr = sdot*(theta0fin - theta0in);
127
128    xstepdotdotfr = sdotdot*(xt_in-xE_in);

```

```

129     ystepdotdotfr = sdotdot*(yt_in - yE_in);
130     thstepdotdotfr = sdotdot*(thetat_in - thetaE_in);
131     theta0stepdotdotfr = sdotdot*(theta0fin - theta0in);
132
133     xstepc = xt;
134     ystepc = yt;
135     thstepc = thetat;
136     theta0stepc = theta0fin;
137
138     xstepdotc = xtdot;
139     ystepdotc = ytdot;
140     thstepdotc = thetatdot;
141     theta0stepdotc = 0;
142
143     xstepdotdotc = 0;
144     ystepdotdotc = 0;
145     thstepdotdotc = 0;
146     theta0stepdotdotc = 0;
147
148     xstep = xstepfr;
149     ystep = ystepfr;
150     thstep = thstepfr;
151     theta0step = theta0stepfr;
152
153     xstepdot = xstepdotfr;
154     ystepdot = ystepdotfr;
155     thstepdot = thstepdotfr;
156     theta0stepdot = theta0stepdotfr;
157
158     xstepdotdot = xstepdotdotfr;
159     ystepdotdot = ystepdotdotfr;
160     thstepdotdot = thstepdotdotfr;
161     theta0stepdotdot = theta0stepdotdotfr;
162
163
164
165     if(t>tf){ //allios incontact isos kalytera me xrono
166         xstep = xstepc;
167         ystep = ystepc;
168         thstep = thstepc;
169         theta0step = theta0stepc;
170
171         xstepdot = xstepdotc;
172         ystepdot = ystepdotc;
173         thstepdot = thstepdotc;
174         theta0stepdot = theta0stepdotc;
175
176         xstepdotdot = xstepdotdotc;

```



```

177     ystepdotdot = ystepdotdotc;
178     thstepdotdot = thstepdotdotc;
179     theta0stepdotdot = theta0stepdotdotc;
180 }
181
182 }
183
184 void updateVel(double dt, double t, double tf){
185     double xdottemp, ydottemp, thetadottemp, theta0dottemp;
186     if(firstTime){
187         xeedot(0) = 0;
188         xeedot(1) = 0;
189         xeedot(2) = 0;
190         xtdot = 0;
191         ytdot = 0;
192         thetatdot =0;
193         xc0dot = 0;
194         yc0dot = 0;
195         theta0dot = 0;
196         xdotprev = 0;
197         ydotprev = 0;
198         thetadotprev =0;
199     }
200     else{
201         xdotprev = xeedot(0);
202         ydotprev = xeedot(1);
203         thetadotprev = xeedot(2);
204         xtdotprev = xtdot;
205         ytdotprev = ytdot;
206         thetatdotprev = thetatdot;
207         xc0dotprev = xc0dot;
208         yc0dotprev = yc0dot;
209         theta0dotprev = theta0dot;
210
211         ee_x = moving_average(ee_x, xwindow, 10,sumx);
212         ee_y = movingMedian(ee_y, ywindow, 3, 0.1, yE_prev); //3point median gia to y
213         thetach = moving_average(thetach, thetawindow, 15, sumtheta);
214
215         ydottemp = (ee_y-yE_prev)/dt;
216
217         xdottemp = (ee_x-xE_prev)/dt;
218
219         thetadottemp = (thetach-thetaE_prev)/dt;
220
221
222         xc0dot = (xc0-xc0_prev)/dt;
223         yc0dot = (yc0-yc0_prev)/dt;
224         theta0dottemp = (theta0-theta0_prev)/dt;

```

```

225
226
227     xtdot = (xt-xt_prev)/dt;
228     ytdot = (yt-yt_prev)/dt;
229     thetatdot = (thetat-thetat_prev)/dt;
230
231     xtdot = movingMedian(xtdot, xtdot_window, 3, 0.1, xtdotprev); //window size = 10
232     ytdot = movingMedian(ytdot, ytdot_window, 3, 0.1, ytdotprev); //window size = 10
233     thetatdot = movingMedian(thetatdot, thetatdot_window, 3, 0.1, thetatdotprev); //window size
        = 10
234
235     xeedot(0) = moving_average(xdottemp, xdot_window, 10, sumxdot); //window size = 10
236
237     xeedot(1) = movingMedian(ydottemp, ydot_window, 3, 0.1, ydotprev);
238
239
240     xeedot(2) = moving_average(thetadottemp, thetadot_window, 15, sumthetadot); //window size
        = 10
241     theta0dot = moving_average(theta0dottemp, theta0dot_window, 10, sumtheta0dot);
242
243
244
245
246     xc0dot = movingMedian(xc0dot, xc0dot_window, 5, 0.1, xc0dotprev); //window size = 10
247     yc0dot = movingMedian(yc0dot, yc0dot_window, 5, 0.1, yc0dotprev); //window size = 10
248
249 }
250 }
251
252 double filter_torque(double torq, double prev) {
253     if (torq == 0.0){
254         // torq = 0.00001;
255         torq = 0.00001;
256         if (prev < 0.0)
257             torq = torq * -1;
258         // printf("CHANGED ZERO TORQUE\n");
259     }
260     return torq;
261 }
262
263 void PDcontroller(double tf, double t){
264     double ts = 0.2*tf;
265     double z = 1;
266     double wn = 6/ts;
267     double jm = 8.85*pow(10,-7); //apo manual kinhthra braxiona
268     double tm = 3.4*pow(10,-3); //apo manual kinhthra braxiona
269     double bm = jm/tm;
270

```

```

271 double kp = wn*wn*jm;
272 double kd = 2*z*wn*jm - bm;
273
274 Eigen::MatrixXd kp_multiplier(4,4);
275 Eigen::MatrixXd bd_multiplier(4,4);
276
277
278 // kp_multiplier << 0.5, 0, 0, 0,
279 //           0, 100*kp, 0, 0,
280 //           0, 0, 100*kp, 0,
281 //           0, 0, 0, 100*kp;
282
283 // bd_multiplier << 20, 0, 0, 0,
284 //           0, kd, 0, 0,
285 //           0, 0, kd, 0,
286 //           0, 0, 0, kd;
287 kp_multiplier << 0.5, 0, 0, 0,
288                 0, 2, 0.5, 0,
289                 0, 0.5, 2.5, 0,
290                 0, 0, 0, 1.5;
291
292 bd_multiplier << 20, 0, 0, 0,
293                 0, 0.05, 0.001, 0,
294                 0, 0.001, 0.05, 0,
295                 0, 0, 0, 0.05;
296
297
298
299
300 Eigen::VectorXd error(4); //ta sxolia einai gia aplo PD xoris ton xrono
301
302 // error << (theta0 - theta0step), (ee_x - xstep), (ee_y - ystep), (thetach - thstep); //to
303 //           sosto eos 24-1
304 // error << (theta0in - theta0in), (ee_x - xt), (ee_y - yt), (thetach - thetat);
305 error << (theta0in - theta0), (xstep - ee_x), (ystep - ee_y), (thstep - thetach);
306
307 Eigen::VectorXd error_dot(4);
308 error_dot << (0 - theta0dot), (xstepdot - xeedot(0)), (ystepdot - xeedot(1)), (thstepdot -
309 //           xeedot(2));
310 // error_dot << (theta0dot - 0), (xeedot(0) - 0), (xeedot(1) - 0), (xeedot(2) - 0);
311 prev_tau(0) = tau(0);
312 prev_tau(1) = tau(1);
313 prev_tau(2) = tau(2);
314 prev_tau(3) = tau(3);
315
316 // tau(0) = 0.5*(theta0in - theta0) + 2*(0-theta0dot);
317 // tau(1) = -1.4*(0.75*error[1]+0.25*error[2]) - 0.0*(0.75*error_dot[1]+0.25*error_dot[2]);

```

```

    //q1 mono gia x
317 // tau(2) = -1.25*(0.25*error[1]+0.75*error[2]) - 0.0*(0.25*error_dot[1]+0.75*error_dot[2]);
    //q2 mono gia y
318 // tau(3) = -0.75*(thetach - thstep) - 0.0*(xeedot(2) - thstepdot); // dhladh q3 MONO gia
    orientation
319
320 tau = kp_multiplier*error + bd_multiplier*error_dot;
321
322 // tau(0) = 0.5*(theta0in - theta0) + 2*(0-theta0dot);
323 // tau(1) = -1.4*(0.75*error[1]+0.25*error[2]) - 0.0*(0.75*error_dot[1]+0.25*error_dot[2]);
    //q1 mono gia x
324 // tau(2) = -1.25*(0.25*error[1]+0.75*error[2]) - 0.0*(0.25*error_dot[1]+0.75*error_dot[2]);
    //q2 mono gia y
325 // tau(3) = -0.75*(thetach - thstep) - 0.0*(xeedot(2) - thstepdot); // dhladh q3 MONO gia
    orientation
326 // /*metatropi gia tous motors kai meiothres*/
327 tau(1) = -tau(1)/186;
328 tau(2) = tau(2)/186;
329 tau(3) = -tau(3)/186;
330 msg_RW.data = filter_torque(tau(0),prev_tau(0)); //tau(0);
331 // msg_RW.data = ns;
332 msg_LS.data = filter_torque(tau(1),prev_tau(1)); //tau(1);
333 msg_LE.data = filter_torque(tau(2),prev_tau(2)); //tau(2);
334 msg_LW.data = filter_torque(tau(3),prev_tau(3)); //tau(3);
335
336 if(!show){
337     std::cout << "Kp_array_is:"<< kp_multiplier<<std::endl;
338     std::cout << "Kd_array_is:"<< bd_multiplier<<std::endl;
339
340     show = !show;
341 }
342 }
343
344
345
346 void controller(int count, double tf, double t){
347     // force_x = 0;
348
349
350     /*Jacobian coefficients*/
351     double j13, j14, j15, j16, j23, j24, j25, j26;
352
353     /*Jacobiandot coefficients*/
354     double j13dot, j14dot, j15dot, j16dot, j23dot, j24dot, j25dot, j26dot;
355
356     /*c coefficients*/
357     double c11, c21, c31, c41, c51, c61;
358

```

```

359  /*h coefficients*/
360  double h11, h12, h13, h14, h15, h16,
361         h21, h22, h23, h24, h25, h26,
362         h31, h32, h33, h34, h35, h36,
363         h41, h42, h43, h44, h45, h46,
364         h51, h52, h53, h54, h55, h56,
365         h61, h62, h63, h64, h65, h66;
366
367  /*JE coefficients*/
368  double je11, je12, je13, je14, je15, je16;
369  double je21, je22, je23, je24, je25, je26;
370  double je31, je32, je33, je34, je35, je36;
371
372  /*JE DOT coefficients*/
373  double jedot11, jedot12, jedot13, jedot14, jedot15, jedot16;
374  double jedot21, jedot22, jedot23, jedot24, jedot25, jedot26;
375  double jedot31, jedot32, jedot33, jedot34, jedot35, jedot36;
376
377  Eigen::VectorXd jvw(2);
378  Eigen::MatrixXd jvq(2,3);
379  Eigen::MatrixXd jac1(6,6);
380  Eigen::MatrixXd jac1dot(6,6);
381  Eigen::VectorXd cstar(6);
382  Eigen::MatrixXd hstar(6,6);
383  Eigen::MatrixXd jstar(6,6);
384  Eigen::MatrixXd jestar(6,3);
385
386  Eigen::MatrixXd h11star(2,2);
387  Eigen::MatrixXd h12star(2,4);
388  Eigen::MatrixXd h21star(4,2);
389  Eigen::MatrixXd h22star(4,4);
390
391  Eigen::MatrixXd hbar(4,4);
392
393  Eigen::VectorXd clstar(2);
394  Eigen::VectorXd c2star(4);
395  Eigen::MatrixXd cbar(4,4);
396
397  Eigen::MatrixXd je11star(2,2);
398  Eigen::VectorXd je12star(2);
399  Eigen::MatrixXd je21star(4,2);
400  Eigen::VectorXd je22star(4);
401
402  Eigen::MatrixXd jebar(4,3);
403  Eigen::VectorXd qe(3);
404
405  Eigen::VectorXd xdotdot_des(4);
406  Eigen::VectorXd fdes(4);

```

```

407 Eigen::VectorXd v1(6);
408
409
410
411 double p1=M;
412 double p2=(m1+m2+m3)*r0x;
413 double p3=(m1+m2+m3)*r0y;
414 double p4=(m1+m2+m3)*l1+(m2+m3)*r1;
415 double p5=(m2+m3)*l2+m3*r2;
416 double p6=ibzz+(m1+m2+m3)*(r0x*r0x+r0y*r0y);
417 double p7=i1zz+(m1+m2+m3)*l1*l1+2*(m2+m3)*l1*r1+(m2+m3)*r1*r1;
418 double p8=i2zz+(m2+m3)*l2*l2+2*m3*l2*r2+m3*r2*r2;
419 double p9=i3zz+m3*l3*l3;
420 double p10=((m1+m2+m3)*l1+(m2+m3)*r1)*r0x;
421 double p11=((m1+m2+m3)*l1+(m2+m3)*r1)*r0y;
422 double p12=(l1+r1)*((m2+m3)*l2+m3*r2);
423 double p13=((m2+m3)*l2+m3*r2)*r0x;
424 double p14=((m2+m3)*l2+m3*r2)*r0y;
425 double p15=m3*l3;
426 double p16=m3*l3*r0x;
427 double p17=m3*l3*r0y;
428 double p18=(l1+r1)*m3*l3;
429 double p19=(l2+r2)*m3*l3;
430
431 double a11 = pow(10,20);
432 double a22 = pow(10,-20);
433 double z_free=1;
434 double ts_f=0.2*tf;
435 double wn_free=6/ts_f;
436 // double kdf=1;
437 // double mdf=kdf/pow(wn_free,2);
438 double mdf = 1;
439 double kdf = mdf*pow(wn_free,2);
440 double bdf=2*z_free*wn_free*mdf;
441
442
443 Eigen::MatrixXd md_f = mdf*Eigen::MatrixXd::Identity(4,4);
444 Eigen::MatrixXd bd_f = bdf*Eigen::MatrixXd::Identity(4,4);
445 Eigen::MatrixXd kd_f = kdf*Eigen::MatrixXd::Identity(4,4);
446 double ke = pow(10,6);
447 double z_contact=z_free*sqrt(kdf/(kdf+ke));
448 double wn_contact=wn_free*sqrt((kdf+ke)/kdf);
449 double fd = 1;
450 double kdc = 100;
451 double mdc=(kdc+ke)/pow(wn_contact,2);
452 double be = 0;
453 double bdc=2*z_contact*wn_contact*mdc-be;
454 Eigen::MatrixXd md_c = mdc*Eigen::MatrixXd::Identity(4,4);

```

```

455 Eigen::MatrixXd bd_c = bdc*Eigen::MatrixXd::Identity(4,4);
456 Eigen::MatrixXd kd_c = kdc*Eigen::MatrixXd::Identity(4,4);
457 Eigen::MatrixXd md(4,4);
458 Eigen::MatrixXd kd(4,4);
459 Eigen::MatrixXd bd(4,4);
460
461 q1 = q1+q01;
462
463 h11 = p1;
464 h12 = 0;
465 h13 = (-1)*p3*cos(theta0) + (-1)*p2*sin(theta0) + (-1)*p4*sin(q1 + theta0) + (-1)*p5*sin(q1 +
      q2 + theta0) + (-1)*p15*sin(q1 + q2 + q3 + theta0);
466 h14 = (-1)*p4*sin(q1 + theta0) + (-1)*p5*sin(q1 + q2 + theta0) + (-1)*p15*sin(q1 + q2 + q3 +
      theta0);
467 h15 = (-1)*p5*sin(q1 + q2 + theta0) + (-1)*p15*sin(q1 + q2 + q3 + theta0);
468 h16 = (-1)*p15*sin(q1 + q2 + q3 + theta0);
469
470 h21 = 0;
471 h22 = p1;
472 h23 = p2*cos(theta0) + p4*cos(q1 + theta0) + p5*cos(q1 + q2 + theta0) + p15*cos(q1 + q2 + q3
      + theta0) + (-1)*p3*sin(theta0);
473 h24 = p4*cos(q1 + theta0) + p5*cos(q1 + q2 + theta0) + p15*cos(q1 + q2 + q3 + theta0);
474 h25 = p5*cos(q1 + q2 + theta0) + p15*cos(q1 + q2 + q3 + theta0);
475 h26 = p15*cos(q1 + q2 + q3 + theta0);
476
477 h31 = (-1)*p3*cos(theta0) + (-1)* p2*sin(theta0)+(-1)*p4*sin(q1 + theta0)+(-1)*p5*sin(q1 +
      q2 + theta0) + (-1)*p15*sin(q1 + q2 + q3 + theta0);
478 h32 = p2*cos(theta0) + p4*cos(q1 + theta0) + p5*cos(q1 + q2 + theta0) + p15*cos(q1 + q2 + q3
      + theta0) + (-1)*p3*sin(theta0);
479 h33 = p6 + p7 + p8 + p9 + 2*p10*cos(q1) + 2*p12*cos(q2) + 2*p13*cos(q1 + q2) + 2*p19*cos(q3)
      + 2*p18*cos(q2 + q3) + 2*p16*cos(q1 + q2 + q3) + 2*p11*sin(q1) + 2*p14*sin(q1 + q2) +
      2*p17*sin(q1 + q2 + q3);
480 h34 = p7 + p8 + p9 + p10*cos(q1) + 2*p12*cos(q2)+p13*cos(q1 + q2) + 2*p19*cos(q3) + 2*p18*
      cos(q2 + q3) + p16* cos(q1 + q2 + q3) + p11*sin(q1) + p14*sin(q1 + q2) + p17*sin(q1 + q2
      + q3);
481 h35 = p8 + p9 + p12*cos(q2) + p13*cos(q1 + q2) + 2*p19*cos(q3) + p18*cos(q2 + q3) + p16*cos(
      q1 + q2 + q3) + p14*sin(q1 + q2) + p17*sin(q1 + q2 + q3);
482 h36 = p9 + p19*cos(q3) + p18*cos(q2 + q3) + p16*cos(q1 + q2 + q3) + p17*sin(q1 + q2 + q3);
483
484 h41 = (-1)*p4*sin(q1 + theta0) + (-1)*p5*sin(q1 + q2 + theta0) + (-1)*p15*sin(q1 + q2 + q3 +
      theta0);
485 h42 = p4*cos(q1 + theta0) + p5*cos(q1 + q2 + theta0) + p15*cos(q1 + q2 + q3 + theta0);
486 h43 = p7 + p8 + p9 + p10*cos(q1) + 2*p12*cos(q2) + p13*cos(q1 + q2) + 2*p19*cos(q3)+ 2*p18*
      cos(q2 + q3) + p16*cos(q1 + q2 + q3) + p11*sin(q1) + p14*sin(q1 +q2) + p17*sin(q1 + q2 +
      q3);
487 h44 = p7 + p8 + p9 + 2*p12*cos(q2) + 2*p19*cos(q3) + 2*p18*cos(q2 + q3);
488 h45 = p8 + p9 + p12*cos(q2) + 2*p19*cos(q3) + p18*cos(q2 + q3);
489 h46 = p9 + p19*cos(q3) + p18*cos(q2 + q3);

```

```

490
491 h51 = (-1)*p5*sin(q1 + q2 + theta0) + (-1)*p15*sin(q1 + q2 + q3 + theta0);
492 h52 = p5*cos(q1 + q2 + theta0) + p15*cos(q1 + q2 + q3 + theta0);
493 h53 = p8 + p9 + p12*cos(q2) + p13*cos(q1 + q2) + 2*p19*cos(q3) + p18*cos(q2 + q3) + p16*cos(
      q1 + q2 + q3) + p14*sin(q1 + q2) + p17*sin(q1 + q2 + q3);
494 h54 = p8 + p9 + p12*cos(q2) + 2*p19*cos(q3) + p18*cos(q2 + q3);
495 h55 = p8 + p9 + 2*p19*cos(q3);
496 h56 = p9 + p19*cos(q3);
497
498 h61 = (-1)*p15*sin(q1 + q2 + q3 + theta0);
499 h62 = p15*cos(q1 + q2 + q3 + theta0);
500 h63 = p9 + p19*cos(q3) + p18*cos(q2 + q3) + p16*cos(q1 + q2 + q3) + p17*sin(q1 + q2 + q3);
501 h64 = p9 + p19*cos(q3) + p18*cos(q2 + q3);
502 h65 = p9 + p19*cos(q3);
503 h66 = p9;
504
505 c11 = (-1)*p2*pow(theta0dot,2)*cos(theta0) + (-1)*p4*pow((q1dot + theta0dot),2)*cos(
506   q1 + theta0) + (-1)*p5*pow(q1dot,2)*cos(q1 + q2 + theta0) + (-2)*p5*q1dot*
507   q2dot*cos(q1 + q2 + theta0) + (-1)*p5*pow(q2dot,2)*cos(q1 + q2 + theta0) + (-2)*
508   p5*q1dot*theta0dot*cos(q1 + q2 + theta0) + (-2)*p5*q2dot*theta0dot*cos(q1 +
509   q2 + theta0) + (-1)*p5*pow(theta0dot,2)*cos(q1 + q2 + theta0) + (-1)*p15*pow(q1dot,2)*
510   cos(q1 + q2 + q3 + theta0) + (-2)*p15*q1dot*q2dot*cos(q1 + q2 + q3 + theta0) + (-1)
511   *p15*pow(q2dot,2)*cos(q1 + q2 + q3 + theta0) + (-2)*p15*q1dot*q3dot*cos(
512   q1 + q2 + q3 + theta0) + (-2)*p15*q2dot*q3dot*cos(q1 + q2 + q3 + theta0) + (-1)*
513   p15*pow(q3dot,2)*cos(q1 + q2 + q3 + theta0) + (-2)*p15*q1dot*theta0dot*cos(q1 +
514   q2 + q3 + theta0) + (-2)*p15*q2dot*theta0dot*cos(q1 + q2 + q3 + theta0) + (-2)*p15*
515   q3dot*theta0dot*cos(q1 + q2 + q3 + theta0) + (-1)*p15*pow(theta0dot,2)*cos(q1 + q2 + q3 +
      theta0) + p3*pow(theta0dot,2)*sin(theta0);
516
517 c21 = (-1)*p3*pow(theta0dot,2)*cos(theta0) + (-1)
518   *p2*pow(theta0dot,2)*sin(theta0) + (-1)*p4*pow(q1dot,2)*sin(q1 + theta0) + (-2)*
519   p4*q1dot*theta0dot*sin(q1 + theta0) + (-1)*p4*pow(theta0dot,2)*sin(q1 + theta0) + (
520   -1)*p5*pow(q1dot,2)*sin(q1 + q2 + theta0) + (-2)*p5*q1dot*q2dot*sin(q1 +
521   q2 + theta0) + (-1)*p5*pow(q2dot,2)*sin(q1 + q2 + theta0) + (-2)*p5*q1dot*
522   theta0dot*sin(q1 + q2 + theta0) + (-2)*p5*q2dot*theta0dot*sin(q1 + q2 + theta0) + (
523   -1)*p5*pow(theta0dot,2)*sin(q1 + q2 + theta0) + (-1)*p15*pow(q1dot,2)*sin(q1 + q2 +
524   q3 + theta0) + (-2)*p15*q1dot*q2dot*sin(q1 + q2 + q3 + theta0) + (-1)*p15*
525   pow(q2dot,2)*sin(q1 + q2 + q3 + theta0) + (-2)*p15*q1dot*q3dot*sin(q1 + q2 + q3 +
526   theta0) + (-2)*p15*q2dot*q3dot*sin(q1 + q2 + q3 + theta0) + (-1)*p15*
527   pow(q3dot,2)*sin(q1 + q2 + q3 + theta0) + (-2)*p15*q1dot*theta0dot*sin(q1 + q2 +
528   q3 + theta0) + (-2)*p15*q2dot*theta0dot*sin(q1 + q2 + q3 + theta0) + (-2)*p15*
529   q3dot*theta0dot*sin(q1 + q2 + q3 + theta0) + (-1)*p15*pow(theta0dot,2)*sin(q1 + q2 +
530   q3 + theta0);
531
532 c31 = p11*q1dot*(q1dot + 2*theta0dot)*cos(q1) + p14*(q1dot + q2dot)
533   *(q1dot + q2dot + 2*theta0dot)*cos(q1 + q2) + p17*pow(q1dot,2)*cos(q1 + q2 + q3)
534   + 2*p17*q1dot*q2dot*cos(q1 + q2 + q3) + p17*pow(q2dot,2)*cos(q1 + q2 + q3) +
535   2*p17*q1dot*q3dot*cos(q1 + q2 + q3) + 2*p17*q2dot*q3dot*cos(q1 +

```



```

q2 + q3) + p17*pow(q3dot,2)*cos(q1 + q2 + q3) + 2*p17*q1dot*theta0dot*cos(q1 +
q2 + q3) + 2*p17*q2dot*theta0dot*cos(q1 + q2 + q3) + 2*p17*q3dot*
theta0dot*cos(q1 + q2 + q3) + (-1)*p10*pow(q1dot,2)*sin(q1) + (-2)*p10*
q1dot*theta0dot*sin(q1) + (-2)*p12*q1dot*q2dot*sin(q2) + (-1)*
p12*pow(q2dot,2)*sin(q2) + (-2)*p12*q2dot*theta0dot*sin(q2) + (-1)*
p13*pow(q1dot,2)*sin(q1 + q2) + (-2)*p13*q1dot*q2dot*sin(q1 + q2) + (-1)
*p13*pow(q2dot,2)*sin(q1 + q2) + (-2)*p13*q1dot*theta0dot*sin(q1 + q2) + (
-2)*p13*q2dot*theta0dot*sin(q1 + q2) + (-2)*p19*q1dot*q3dot*sin(
q3) + (-2)*p19*q2dot*q3dot*sin(q3) + (-1)*p19*pow(q3dot,2)*sin(q3) +
(-2)*p19*q3dot*theta0dot*sin(q3) + (-2)*p18*q1dot*q2dot*sin(q2 +
q3) + (-1)*p18*pow(q2dot,2)*sin(q2 + q3) + (-2)*p18*q1dot*q3dot*sin(
q2 + q3) + (-2)*p18*q2dot*q3dot*sin(q2 + q3) + (-1)*p18*pow(q3dot,2)*
sin(q2 + q3) + (-2)*p18*q2dot*theta0dot*sin(q2 + q3) + (-2)*p18*q3dot*
theta0dot*sin(q2 + q3) + (-1)*p16*pow(q1dot,2)*sin(q1 + q2 + q3) + (-2)*p16*
q1dot*q2dot*sin(q1 + q2 + q3) + (-1)*p16*pow(q2dot,2)*sin(q1 + q2 + q3) + (
-2)*p16*q1dot*q3dot*sin(q1 + q2 + q3) + (-2)*p16*q2dot*q3dot*
sin(q1 + q2 + q3) + (-1)*p16*pow(q3dot,2)*sin(q1 + q2 + q3) + (-2)*p16*
q1dot*theta0dot*sin(q1 + q2 + q3) + (-2)*p16*q2dot*theta0dot*sin(q1 + q2 +
q3) + (-2)*p16*q3dot*theta0dot*sin(q1 + q2 + q3);

c41 = (-1)*p11*pow(theta0dot,2)*
cos(q1) + (-1)*p14*pow(theta0dot,2)*cos(q1 + q2) + (-1)*p17*pow(theta0dot,2)*
cos(q1 + q2 + q3) + p10*pow(theta0dot,2)*sin(q1) + (-2)*p12*q1dot*q2dot*
sin(q2) + (-1)*p12*pow(q2dot,2)*sin(q2) + (-2)*p12*q2dot*theta0dot*
sin(q2) + p13*pow(theta0dot,2)*sin(q1 + q2) + (-2)*p19*q1dot*q3dot*sin(
q3) + (-2)*p19*q2dot*q3dot*sin(q3) + (-1)*p19*pow(q3dot,2)*sin(q3) +
(-2)*p19*q3dot*theta0dot*sin(q3) + (-2)*p18*q1dot*q2dot*sin(q2 +
q3) + (-1)*p18*pow(q2dot,2)*sin(q2 + q3) + (-2)*p18*q1dot*q3dot*sin(
q2 + q3) + (-2)*p18*q2dot*q3dot*sin(q2 + q3) + (-1)*p18*pow(q3dot,2)*
sin(q2 + q3) + (-2)*p18*q2dot*theta0dot*sin(q2 + q3) + (-2)*p18*q3dot*
theta0dot*sin(q2 + q3) + p16*pow(theta0dot,2)*sin(q1 + q2 + q3);

c51 = (-1)*p14*
pow(theta0dot,2)*cos(q1 + q2) + (-1)*p17*pow(theta0dot,2)*cos(q1 + q2 + q3) + p12*
pow(q1dot,2)*sin(q2) + 2*p12*q1dot*theta0dot*sin(q2) + p12*pow(theta0dot,2)*
sin(q2) + p13*pow(theta0dot,2)*sin(q1 + q2) + (-2)*p19*q1dot*q3dot*sin(
q3) + (-2)*p19*q2dot*q3dot*sin(q3) + (-1)*p19*pow(q3dot,2)*sin(q3) +
(-2)*p19*q3dot*theta0dot*sin(q3) + p18*pow(q1dot,2)*sin(q2 + q3) + 2*
p18*q1dot*theta0dot*sin(q2 + q3) + p18*pow(theta0dot,2)*sin(q2 + q3) + p16*
pow(theta0dot,2)*sin(q1 + q2 + q3);

c61 = (-1)*p17*pow(theta0dot,2)*cos(q1 + q2 + q3) + p19*
pow((q1dot + q2dot + theta0dot),2)*sin(q3) + p18*pow(q1dot,2)*sin(q2 + q3) + 2*
p18*q1dot*theta0dot*sin(q2 + q3) + p18*pow(theta0dot,2)*sin(q2 + q3) + p16*
pow(theta0dot,2)*sin(q1 + q2 + q3);

q1 = q1 - q01;

```

```

584 //to theta 1 einai to q1 ousiastika kai to th0 to theta0
585
586 je11 = 1;
587 je12 = 0;
588 je13 = (-1)*r0y*cos(theta0) + (-1)*r0x*sin(theta0) + (-1)*l1*sin(
589     q01 + theta0 + q1) + (-1)*r1*sin(q01 + theta0 + q1) + (-1)*l2*sin(
590     q01 + theta0 + q1 + q2) + (-1)*r2*sin(q01 + theta0 + q1 + q2) +
591     (-1)*l3*sin(q01 + theta0 + q1 + q2 + q3) + (-1)*r3*sin(q01 + theta0 + q1 + q2 + q3);
592 je14 = (-1)*(l1 + r1)*sin(q01 + theta0 +
593     q1) + (-1)*(l2 + r2)*sin(q01 + theta0 + q1 + q2) + (-1)*(l3 +
594     r3)*sin(q01 + theta0 + q1 + q2 + q3);
595 je15 = (-1)*(l2 + r2)*sin(
596     q01 + theta0 + q1 + q2) + (-1)*(l3 + r3)*sin(q01 + theta0 +
597     q1 + q2 + q3);
598 je16 = (-1)*(l3 + r3)*sin(q01 + theta0 + q1 + q2 + q3);
599
600 je21 = 0;
601 je22 = 1;
602 je23 = r0x*cos(theta0) + (l1 + r1)*cos(q01 + theta0 + q1) + l2*cos(q01 + theta0 + q1 + q2) +
        r2*cos(q01 + theta0 +
603     q1 + q2) + l3*cos(q01 + theta0 + q1 + q2 + q3) + r3*cos(q01 + theta0 + q1 + q2 + q3) +
        (-1)*r0y*sin(theta0);
604 je24 = (l1 +
605     r1)*cos(q01 + theta0 + q1) + (l2 + r2)*cos(q01 + theta0 + q1 + q2)
606     + (l3 + r3)*cos(q01 + theta0 + q1 + q2 + q3);
607 je25 = (l2 + r2)*cos(
608     q01 + theta0 + q1 + q2) + (l3 + r3)*cos(q01 + theta0 + q1 +
609     q2 + q3);
610 je26 = (l3 + r3)*cos(q01 + theta0 + q1 + q2 + q3);
611
612 je31 = 0;
613 je32 = 0;
614 je33 = 1;
615 je34 = 1;
616 je35 = 1;
617 je36 = 1;
618
619
620
621
622
623 jedot11 = 0;
624 jedot12 = 0;
625 jedot13 = (-1)*r0x*theta0dot*cos(theta0) + (-1)*l1*(theta0dot +
626     q1dot)*cos(q01 + theta0 + q1) + (-1)*r1*(theta0dot + q1dot)
627     *cos(q01 + theta0 + q1) + (-1)*l2*(theta0dot + q1dot + q2dot)
628     *cos(q01 + theta0 + q1 + q2) + (-1)*r2*(theta0dot + q1dot +
629     q2dot)*cos(q01 + theta0 + q1 + q2) + (-1)*l3*(theta0dot +

```

```

630     q1dot + q2dot + q3dot)*cos(q01 + theta0 + q1 + q2 +
631     q3) + (-1)*r3*(theta0dot + q1dot + q2dot + q3dot)*
632     cos(q01 + theta0 + q1 + q2 + q3) + r0y*theta0dot*sin(theta0);
633     jedot14 = (
634     -1)*(l1 + r1)*(theta0dot + q1dot)*cos(q01 + theta0 + q1) + (-1)*
635     (l2 + r2)*(theta0dot + q1dot + q2dot)*cos(q01 + theta0 + q1 +
636     q2) + (-1)*(l3 + r3)*(theta0dot + q1dot + q2dot +
637     q3dot)*cos(q01 + theta0 + q1 + q2 + q3);
638     jedot15 = (-1)*(l2 + r2)*(
639     theta0dot + q1dot + q2dot)*cos(q01 + theta0 + q1 + q2) + (-1)
640     *(l3 + r3)*(theta0dot + q1dot + q2dot + q3dot)*cos(q01 +
641     theta0 + q1 + q2 + q3);
642     jedot16 = (-1)*(l3 + r3)*(theta0dot +
643     q1dot + q2dot + q3dot)*cos(q01 + theta0 + q1 + q2 +
644     q3);
645
646     jedot21 = 0;
647     jedot22 = 0;
648     jedot23 = (-1)*r0y*theta0dot*cos(theta0) + (-1)*r0x*
649     theta0dot*sin(theta0) + (-1)*(l1 + r1)*(theta0dot + q1dot)*sin(
650     q01 + theta0 + q1) + (-1)*l2*(theta0dot + q1dot + q2dot)*sin(
651     q01 + theta0 + q1 + q2) + (-1)*r2*(theta0dot + q1dot +
652     q2dot)*sin(q01 + theta0 + q1 + q2) + (-1)*l3*(theta0dot +
653     q1dot + q2dot + q3dot)*sin(q01 + theta0 + q1 + q2 +
654     q3) + (-1)*r3*(theta0dot + q1dot + q2dot + q3dot)*
655     sin(q01 + theta0 + q1 + q2 + q3);
656     jedot24 = (-1)*(l1 + r1)*(theta0dot +
657     q1dot)*sin(q01 + theta0 + q1) + (-1)*(l2 + r2)*(theta0dot +
658     q1dot + q2dot)*sin(q01 + theta0 + q1 + q2) + (-1)*(l3 +
659     r3)*(theta0dot + q1dot + q2dot + q3dot)*sin(q01 + theta0 +
660     q1 + q2 + q3);
661     jedot25 = (-1)*(l2 + r2)*(theta0dot + q1dot +
662     q2dot)*sin(q01 + theta0 + q1 + q2) + (-1)*(l3 + r3)*(
663     theta0dot + q1dot + q2dot + q3dot)*sin(q01 + theta0 + q1 +
664     q2 + q3);
665     jedot26 = (-1)*(l3 + r3)*(theta0dot + q1dot + q2dot +
666     q3dot)*sin(q01 + theta0 + q1 + q2 + q3);
667
668     jedot31 = 0;
669     jedot32 = 0;
670     jedot33 = 0;
671     jedot34 = 0;
672     jedot35 = 0;
673     jedot36 = 0;
674
675
676
677     h(0,0) = h11;

```

```
678 h(0,1) = h12;
679 h(0,2) = h13;
680 h(0,3) = h14;
681 h(0,4) = h15;
682 h(0,5) = h16;
683 ///////////////
684 h(1,0) = h21;
685 h(1,1) = h22;
686 h(1,2) = h23;
687 h(1,3) = h24;
688 h(1,4) = h25;
689 h(1,5) = h26;
690 ///////////////
691 h(2,0) = h31;
692 h(2,1) = h32;
693 h(2,2) = h33;
694 h(2,3) = h34;
695 h(2,4) = h35;
696 h(2,5) = h36;
697 ///////////////
698 h(3,0) = h41;
699 h(3,1) = h42;
700 h(3,2) = h43;
701 h(3,3) = h44;
702 h(3,4) = h45;
703 h(3,5) = h46;
704 ///////////////
705 h(4,0) = h51;
706 h(4,1) = h52;
707 h(4,2) = h53;
708 h(4,3) = h54;
709 h(4,4) = h55;
710 h(4,5) = h56;
711 ///////////////
712 h(5,0) = h61;
713 h(5,1) = h62;
714 h(5,2) = h63;
715 h(5,3) = h64;
716 h(5,4) = h65;
717 h(5,5) = h66;
718
719 c(0) = c11;
720 c(1) = c21;
721 c(2) = c31;
722 c(3) = c41;
723 c(4) = c51;
724 c(5) = c61;
725
```

```

726
727   je(0,0) = je11;
728   je(0,1) = je12;
729   je(0,2) = je13;
730   je(0,3) = je14;
731   je(0,4) = je15;
732   je(0,5) = je16;
733   //////////
734   je(1,0) = je21;
735   je(1,1) = je22;
736   je(1,2) = je23;
737   je(1,3) = je24;
738   je(1,4) = je25;
739   je(1,5) = je26;
740   //////////
741   je(2,0) = je31;
742   je(2,1) = je32;
743   je(2,2) = je33;
744   je(2,3) = je34;
745   je(2,4) = je35;
746   je(2,5) = je36;
747
748
749   jedot(0,0) = jedot11;
750   jedot(0,1) = jedot12;
751   jedot(0,2) = jedot13;
752   jedot(0,3) = jedot14;
753   jedot(0,4) = jedot15;
754   jedot(0,5) = jedot16;
755   //////////
756   jedot(1,0) = jedot21;
757   jedot(1,1) = jedot22;
758   jedot(1,2) = jedot23;
759   jedot(1,3) = jedot24;
760   jedot(1,4) = jedot25;
761   jedot(1,5) = jedot26;
762   //////////
763   jedot(2,0) = jedot31;
764   jedot(2,1) = jedot32;
765   jedot(2,2) = jedot33;
766   jedot(2,3) = jedot34;
767   jedot(2,4) = jedot35;
768   jedot(2,5) = jedot36;
769
770   jvw << je13 , je23;
771
772   jvq << je14, je15, je16,
773       je24, je25, je26;

```

```

774
775
776
777 jac1 << 1, 0, 0, 0, 0, 0,
778         0, 1, 0, 0, 0, 0,
779         0, 0, 1, 0, 0, 0,
780         1, 0, je13, je14, je15, je16,
781         0, 1, je23, je24, je25, je26,
782         0, 0, 1, je34, je35, je36;
783
784
785
786
787
788 jacldot << 0, 0, 0, 0, 0, 0,
789           0, 0, 0, 0, 0, 0,
790           0, 0, 0, 0, 0, 0,
791           0, 0, jedot13, jedot14, jedot15, jedot16,
792           0, 0, jedot23, jedot24, jedot25, jedot26,
793           0, 0, 0, jedot34, jedot35, jedot36;
794
795
796
797
798 hstar = (jac1.transpose()).inverse()*h*(jac1.inverse());
799
800
801
802
803
804 v1 <<xc0dot, yc0dot, theta0dot, q1dot, q2dot, q3dot;
805
806
807 cstar = (jac1.transpose()).inverse()*(c-h*jac1.inverse()*jacldot*v1); //na ftiakso to v1
808
809
810 jstar = (jac1.transpose()).inverse();
811
812 Eigen::MatrixXd j12star(2,4);
813 Eigen::MatrixXd j22star(4,4);
814
815 j12star << jstar(0,2), jstar(0,3), jstar(0,4), jstar(0,5),
816           jstar(1,2), jstar(1,3), jstar(1,4), jstar(1,5);
817
818
819
820 j22star << jstar(2,2), jstar(2,3), jstar(2,4), jstar(2,5),
821           jstar(3,2), jstar(3,3), jstar(3,4), jstar(3,5),

```

```

822         jstar(4,2), jstar(4,3), jstar(4,4), jstar(4,5),
823         jstar(5,2), jstar(5,3), jstar(5,4), jstar(5,5);
824
825
826
827
828
829
830     jestar = jstar*(je.transpose());
831
832
833     h11star << hstar(0,0), hstar(0,1),
834             hstar(1,0), hstar(1,1);
835
836     ///std::cout<<"h11star check "<<std::endl;
837     //std::cout<<"h11star is: "<<h11star<<std::endl;
838
839
840     h12star << hstar(0,2), hstar(0,3), hstar(0,4), hstar(0,5),
841             hstar(1,2), hstar(1,3), hstar(1,4), hstar(1,5);
842
843     ///std::cout<<"h12star check "<<std::endl;
844     //std::cout<<"h12star is: "<<h12star<<std::endl;
845
846     h21star << hstar(2,0), hstar(2,1),
847             hstar(3,0), hstar(3,1),
848             hstar(4,0), hstar(4,1),
849             hstar(5,0), hstar(5,1);
850
851     ///std::cout<<"h21star check "<<std::endl;
852     //std::cout<<"h21star is: "<<h21star<<std::endl;
853
854
855     h22star << hstar(2,2), hstar(2,3), hstar(2,4), hstar(2,5),
856             hstar(3,2), hstar(3,3), hstar(3,4), hstar(3,5),
857             hstar(4,2), hstar(4,3), hstar(4,4), hstar(4,5),
858             hstar(5,2), hstar(5,3), hstar(5,4), hstar(5,5);
859
860     ///std::cout<<"h22star check "<<std::endl;
861     //std::cout<<"h22star is: "<<h22star<<std::endl;
862
863     hbar = h22star - h21star*(h11star.inverse())*h12star;
864
865
866     c1star << cstar(0), cstar(1);
867     //std::cout<<"c1star is: "<<c1star<<std::endl;
868
869     ///std::cout<<"c1star check "<<std::endl;

```

```

870
871 c2star << cstar(2), cstar(3), cstar(4), cstar(5);
872 //std::cout<<"c2star is: "<<c2star<<std::endl;
873 ///std::cout<<"c2star check "<<std::endl;
874
875 cbar=c2star-h21star*(h11star.inverse())*c1star;
876
877 jellstar << jestar(0,0), jestar(0,1),
878           jestar(1,0), jestar(1,1);
879
880 ///std::cout<<"jellstar check "<<std::endl;
881
882
883 je21star << jestar(2,0), jestar(2,1),
884           jestar(3,0), jestar(3,1),
885           jestar(4,0), jestar(4,1),
886           jestar(5,0), jestar(5,1);
887
888 ///std::cout<<"je21star check "<<std::endl;
889
890
891 je12star << jestar(0,2), jestar(1,2);
892 ///std::cout<<"je12star check "<<std::endl;
893
894
895 je22star << jestar(2,2), jestar(3,2), jestar(4,2), jestar(5,2);
896 ///std::cout<<"je22star check "<<std::endl;
897
898
899 // Jebar=[Je21star-H21star*inv(H11star)*Jellstar Je22star-H21star*inv(H11star)*Je12star);
900
901 Eigen::MatrixXd jbar=j22star-h21star*(h11star.inverse())*j12star;
902
903 jebar << je21star-h21star*(h11star.inverse())*jellstar, je22star-h21star*(h11star.inverse())*
           je12star;
904 ///std::cout<<"jebar check "<<std::endl;
905
906
907 // Qext=[0;0;Fext;0]; na to ftiakso
908
909 qe << 0, force_x, 0; //den eimai sigouros gia afto
910
911 // if(t<=tf){ //allios !incontact //t<=tf
912 //   force_x = 0;
913 //   bd = bd_f;
914 //   kd = kd_f;
915 //   md = md_f;
916 //   fdes << 0, 0, 0, 0;

```



```

917 // }
918 // else{
919 //   bd = bd_c;
920 //   kd = kd_c;
921 //   md = md_c;
922 //   fdes << 0, 0, fd, 0;
923 //   force_x = 0; //(0.08 + raw_force_x)/2;
924 // }
925
926
927 // fdes << 0, 0, fd*abs(force_x)/(abs(force_x)+a22), 0;
928
929 //std::cout<<"fdes check "<<std::endl;
930
931
932 // xdotdot_des=[theta0dotdot_des;xEdotdot_des;thetaEdotdot_des];
933 xdotdot_des<< theta0stepdotdot, xstepdotdot, ystepdotdot, thstepdotdot;
934
935
936 // if(count <1){ //giati to xstep[0] einai 0, akyro vrhka ton logo sto record
937 //   xstep = ee_x;
938 //   ystep = ee_y;
939 //   thstep = thetach;
940 //   theta0step = theta0;
941
942 //   xstepdot = 0;
943 //   ystepdot = 0;
944 //   thstepdot = 0;
945 //   theta0stepdot = 0;
946 // }
947
948
949
950 Eigen::VectorXd error(4);
951 error << (theta0 - theta0step), (ee_x - xstep), (ee_y - ystep), (thetach - thstep);
952
953
954 Eigen::VectorXd error_dot(4);
955 error_dot << (theta0dot - theta0stepdot), (xeedot(0) - xstepdot), (xeedot(1) - ystepdot), (
    xeedot(2) - thstepdot);
956
957
958
959 Eigen::VectorXd qext(4);
960 qext << 0, 0, force_x, 0;
961
962
963 Eigen::VectorXd u = xdotdot_des+(md.inverse())*(-kd*error-bd*error_dot-qext + fdes); //kd =

```

```

1, bd=2 kd=0.325 , bd = 0.2

964
965
966
967 Eigen::VectorXd qbar=hbar*u+cbar-jebar*qe;
968
969
970
971
972 tau = (jbar.inverse())*qbar;
973
974 // if(count <1){ //giati to xstep[0] einai 0
975 //   std::cout<<"frist torques zero"<<std::endl;
976 //   tau(0) = tau(1) = tau(2) = tau(3) = 0.00001;
977 // }
978
979 // tau(0) = 0.5*(theta0in - theta0) + 2*(0 - theta0dot);
980
981 prev_tau(0) = tau(0);
982 prev_tau(1) = tau(1);
983 prev_tau(2) = tau(2);
984 prev_tau(3) = tau(3);
985
986
987
988 /*metatropi gia tous motors kai meiothres*/
989 tau(1) = -tau(1)/186;
990 tau(2) = tau(2)/186;
991 tau(3) = -tau(3)/186;
992
993
994
995
996 //std::cout<<"/////////////////////"<<std::endl;
997 if(count%100 == 0){
998   show = !show; //giati einaI 200Hz kai thelo na deixnei ana 1 sec
999   if(show){
1000     std::cout<<"time_is:_"<<t<<"_sec"<<std::endl;
1001     std::cout<<"xstep_is:_"<<xstep<<std::endl;
1002     std::cout<<"ystep_is:_"<<ystep<<std::endl;
1003     std::cout<<"thstep_is:_"<<thstep<<std::endl;
1004
1005     // std::cout<<"md is: " <<md<<std::endl;
1006     std::cout<<"kd_f_is:_"<<kd_f<<std::endl;
1007     std::cout<<"bd_f_is:_"<<bd_f<<std::endl;
1008
1009     std::cout<<"kd_c_is:_"<<kd_c<<std::endl;
1010     std::cout<<"bd_c_is:_"<<bd_c<<std::endl;

```

```

1011
1012
1013
1014 // std::cout<<" ee x is: "<<ee_x<<std::endl;
1015 // std::cout<<" ee y is: "<<ee_y<<std::endl;
1016 // std::cout<<" ee theta is: "<<thetach<<std::endl;
1017 // std::cout<<"eex dot is: "<<xeedot(0)<<std::endl;
1018 // std::cout<<"eey dot is: "<<xeedot(1)<<std::endl;
1019 // std::cout<<"eetheta dot is: "<<xeedot(2)<<std::endl;
1020
1021 // std::cout<<"xt_in is: "<<xt_in<<std::endl;
1022 // std::cout<<"yt_in is: "<<yt_in<<std::endl;
1023 // std::cout<<"thetat_in is: "<<thetat_in<<std::endl;
1024
1025
1026 // std::cout<<"theta0 is: "<<theta0<<std::endl;
1027 // std::cout<<"theta0dot is: "<<theta0dot<<std::endl;
1028 // std::cout<<"xc0 is: "<<xc0<<std::endl;
1029 // std::cout<<"yc0 is: "<<yc0<<std::endl;
1030 // std::cout<<"xc0dot is: "<<xc0dot<<std::endl;
1031 // std::cout<<"yc0dot is: "<<yc0dot<<std::endl;
1032
1033 // std::cout<<" q1 is: "<<q1<<std::endl;
1034 // std::cout<<" q2 is: "<<q2<<std::endl;
1035 // std::cout<<" q3 is: "<<q3<<std::endl;
1036 // std::cout<<" q1dot is: "<<q1dot<<std::endl;
1037 // std::cout<<" q2dot is: "<<q2dot<<std::endl;
1038 // std::cout<<" q3dot is: "<<q3dot<<std::endl;
1039
1040 std::cout<<"fextx_is:␣"<<force_x<<"_N"<<std::endl;
1041
1042 std::cout<<"error_is:␣"<<error<<std::endl;
1043 std::cout<<"errordot_is:␣"<<error_dot<<std::endl;
1044
1045 std::cout<<"rw_torque_is:␣"<<tau(0)<<"_Nm.␣"<< std::endl;
1046 std::cout<<"q1_torque_is:␣"<<-tau(1)*186<<"_Nm.␣"<< std::endl;
1047 std::cout<<"q2_torque_is:␣"<<tau(2)*186<<"_Nm.␣"<< std::endl;
1048 std::cout<<"q3_torque_is:␣"<<-tau(3)*186<<"_Nm.␣"<< std::endl;
1049
1050
1051 // std::cout<<"h is: "<<h<<std::endl;
1052 // std::cout<<"c is: "<<c<<std::endl;
1053 // std::cout<<"je is: "<<je<<std::endl;
1054 // std::cout<<"jedot is: "<<jedot<<std::endl;
1055
1056 // std::cout <<"jvw is: "<<jvw<<std::endl;
1057 // std::cout <<"jvq is: "<<jvq<<std::endl;
1058 // std::cout <<"jac1 is: "<<jac1<<std::endl;

```

```

1059 // std::cout <<"jacldot is: "<<jacldot<<std::endl;
1060 // std::cout <<"hstar is: "<<hstar<<std::endl;
1061 // std::cout <<"v1 is: "<<v1<<std::endl;
1062 // std::cout<<"cstar is: "<<cstar<<std::endl;rate
1063 // std::cout<<"jstar is: "<<jstar<<std::endl;
1064 // std::cout<<"jestar is: "<<jestar<<std::endl;
1065 // std::cout<<"hbar is: "<<hbar<<std::endl;
1066 // std::cout<<"cbar is: "<<cbar<<std::endl;
1067 // std::cout<<"jbar is: "<<jbar<<std::endl;
1068 // std::cout<<"jebar is: "<<jebar<<std::endl;
1069 // std::cout<<"fes is: "<<fdes<<std::endl;
1070 // std::cout<<"xdotdot_des is: "<<xdotdot_des<<std::endl;
1071 // std::cout<<"qext is: "<<qext<<std::endl;
1072 // std::cout<<"u is: "<<u<<std::endl;
1073 // std::cout<<"qbar is: "<<qbar<<std::endl;
1074
1075 // std::cout<<"***STARS*****"<<std::endl;
1076 // std::cout<<"j12star is:"<<j12star<<std::endl;
1077 // std::cout<<"j22star is:"<<j22star<<std::endl;
1078 // std::cout<<"h11star is:"<<h11star<<std::endl;
1079 // std::cout<<"h12star is:"<<h12star<<std::endl;
1080 // std::cout<<"h21star is:"<<h21star<<std::endl;
1081 // std::cout<<"h22star is:"<<h22star<<std::endl;
1082 // std::cout<<"c1star is:"<<c1star<<std::endl;
1083 // std::cout<<"c2star is:"<<c2star<<std::endl;
1084 // std::cout<<"je11star is:"<<je11star<<std::endl;
1085 // std::cout<<"je12star is:"<<je12star<<std::endl;
1086 // std::cout<<"je21star is:"<<je21star<<std::endl;
1087 // std::cout<<"je22star is:"<<je22star<<std::endl;
1088
1089
1090
1091
1092
1093
1094 std::cout<<"/////////////////////"<<std::endl;
1095 std::cout<<"  "<<std::endl;
1096 }
1097 }
1098
1099 }
1100
1101
1102
1103 #endif

```

Γ'.2 robot_variables.h

Ο κώδικας με τον ορισμό των μεταβλητών:

```

1  #ifndef ROBOT_VARIABLES_H
2  #define ROBOT_VARIABLES_H
3  #include "includes.h"
4
5
6  ////////////////////////////////////////////////// GLOBAL VARIABLES DECLARATION START////////////////////////////////////
7  /*Boolean flags*/
8  bool reachedTarget;// = false;
9  bool start_movement;// = false;
10 // bool eefirstTime;// = true; //boolean for first time listening
11 // bool targetfirstTime;
12 // bool basefirstTime;
13 bool firstTime;
14
15 bool ls_initialized = false;
16 bool le_initialized = false;
17 bool re_initialized = false;
18
19 double q1_init = -60 * (M_PI / 180);
20 double q2_init = 105 * (M_PI / 180);
21 double q3_init = 45 * (M_PI / 180);
22
23 /*Cepheus' variables*/
24 double q1;      // angle of first joint [rad] from callback
25 double q2;      // angle of second joint [rad] from callback
26 double q3, q3dot;
27 double q1dot;   // rate of first joint [rad/s] from callback
28 double q2dot;   // rate of second joint [rad/s] from callback
29 double theta0dot; // reaction wheel velocity [rad/s] h allios theta0dot
30 double theta0, thetaee;
31 double ee_x, ee_y; //ee_Z not needed
32 double xc0, yc0, xc0dot, yc0dot; //center of mass of base
33 double thetach; //orientation of chaser (end effector)
34 double xE_prev, yE_prev, thetaE_prev;
35 double xt_prev, yt_prev, thetat_prev;
36 double xc0_prev, yc0_prev, theta0_prev;
37
38
39 /*Target's variables*/
40 double xt, yt; //xtarget, ytarget, pros to paron einai idia me to ring_x, ring_y
41 double thetat; //angle of target (orientation)
42 double rawxt, rawyt, rawthetat;
43 double xtdot, ytdot, thetatdot;
44 double rawxtdot, rawytdot, rawthetatdot;
45 double xt_in, yt_in, thetat_in; //arxikh thesh target (ring)

```

```

46
47 /*Messages to publish */
48 std_msgs::Float64 msg_RW; //reaction wheel
49 std_msgs::Float64 msg_LS; //left shoulder
50 std_msgs::Float64 msg_LE; //left elbow
51 std_msgs::Float64 msg_LW; //left wrist
52 std_msgs::Float64 msg_ex; //errorx
53 std_msgs::Float64 msg_ey; //errorry
54 std_msgs::Float64 msg_etheta; //errortheta
55 std_msgs::Float64 msg_fextx;
56 std_msgs::Float64 msg_fextx_raw;
57 // std_msgs::Float64 msg_TX; //thruster_x
58 // std_msgs::Float64 msg_TY; //thruster_y
59 geometry_msgs::Wrench base_wrench; //x,y force of thrusters
60 std_msgs::Float64 msg_xd_x; //xd_x
61 std_msgs::Float64 msg_xd_y; //xd_y
62 std_msgs::Float64 msg_xd_theta; //xd_y
63
64 std_msgs::Float64 msg_xt_x;
65 std_msgs::Float64 msg_xt_y;
66 std_msgs::Float64 msg_xt_theta;
67
68 std_msgs::Float64 msg_xt_x_raw;
69 std_msgs::Float64 msg_xt_y_raw;
70 std_msgs::Float64 msg_xt_theta_raw;
71
72 std_msgs::Float64 msg_xee_x;
73 std_msgs::Float64 msg_xee_y;
74 std_msgs::Float64 msg_xee_theta;
75
76 std_msgs::String arduino_msg;
77 std_msgs::Bool start_moving;
78 std_msgs::Bool start_grab_msg;
79
80
81
82
83
84
85 /*Kinematics variables From alex thesis*/
86 double r0,m0,l0; //l0 diko moy, to mhkos tou elathriou
87 double r1,m1,l1;
88 double r2,m2,l2;
89 double r3,m3,l3;
90 double M;
91 double mt;
92 double r0x,r0y;
93 //double a,b,c,d; not needed

```

```

94 double ibzz;
95 double ilzz, i2zz, i3zz, itzz;
96
97
98 Eigen::Vector3d fext(0,0,0);// = Eigen::VectorXd::Zero(3); // << (0,0,0); //fx,fy,n
99 Eigen::Vector3d xee(0,0,0);// = Eigen::VectorXd::Zero(3); //(0,0,0) //the actual trajecotry (x
    ,y,theta)
100 Eigen::Vector3d xeedot(0,0,0);// = Eigen::VectorXd::Zero(3); //(0,0,0)//the actual trajecotry
    (x,y,theta)
101 Eigen::VectorXd c(6);// = Eigen::VectorXd::Zero(6) ;
102 Eigen::MatrixXd h = Eigen::MatrixXd::Zero(6,6); //imp.thesis sel 62 eksisosi 5-31
103 Eigen::MatrixXd je = Eigen::MatrixXd::Zero(3,6); //update: jacobian for ee
104 Eigen::MatrixXd jedot = Eigen::MatrixXd::Zero(3,6); //update: jacobian for ee
105
106
107
108 double thetaE_in; //=30*(M_PI/180);
109 double xE_in, yE_in;
110 double q01 = 0; //to eftiaksa oste na einai mhden! // -27.88931 * M_PI/180; //gonia tou
    shoulder joint se sxesh me base
111 double s01 = 0.5, s02 = 0.2;
112
113 //gains
114 double kpx = 0.5,kdx = 0.6;
115 double kpy = 0.5,kdy = 0.6;
116 double kpth = 0.5,kdth = 0.6;
117 //double kprop = 0.5, kder = 10; AFTA DOULEVOUN GIA TO APLO PD XORIS TRAJECTORY gia tis palies
    mazes klp
118 double kprop = 0.5 ,kder = 20;
119 double kprop_mb,kder_mb; //model based
120
121 //for polynomial
122 double a0,a1,a2,a3,a4,a5;
123 double xstep,ystep,thstep,theta0step;
124 double xstepdot, ystepdot, thstepdot,theta0stepdot;
125 double xstepdotdot, ystepdotdot, thstepdotdot,theta0stepdotdot;
126 double sd = 0.25; //safety distance
127 double theta0in, theta0fin;
128 double xE_contact; // = x_target_in - l0;
129 double yE_contact; // = y_target_in;
130 double thetaE_contact;
131
132 bool beginGrab = false;
133 bool incontact =false;
134 bool beginSoft = false;
135 bool softFinished = false;
136 bool beginHard = false;
137 bool hardFinished = false;

```

```

138
139 bool gripperListenedSoft = false;
140 bool gripperListenedHard = false;
141
142 // int contactCounter = 0;
143
144 bool firstTimeq1 = true;
145 bool firstTimeq2 = true;
146 bool firstTimeq3 = true;
147 bool offsetsdone = false;
148 double offsetq1 = 0.0;
149 double offsetq2 = 0.0;
150 double offsetq3 = 0.0;
151 double q1known = 1.84447; //gonies analoga me tin gnosti diataksi, ego ekana omos thetiko
    akro, agonas karpos arnitika akra
152 double q2known = -0.9542;
153 double q3known = -0.2856;
154
155 std_msgs::Float64 msg_q1;
156 std_msgs::Float64 msg_q2;
157 std_msgs::Float64 msg_q3;
158 std_msgs::Float64 msg_theta0;
159
160 std_msgs::Float64 msg_q1dot;
161 std_msgs::Float64 msg_q2dot;
162 std_msgs::Float64 msg_q3dot;
163 std_msgs::Float64 msg_theta0dot;
164
165
166 std_msgs::Float64 msg_torqueq1;
167 std_msgs::Float64 msg_torqueq2;
168 std_msgs::Float64 msg_torqueq3;
169 std_msgs::Float64 msg_torquerw;
170
171 Eigen::VectorXd tau(4);
172 Eigen::VectorXd prev_tau(4);
173
174
175 std_msgs::Float64 msg_xd_x_dot; //xd_x
176 std_msgs::Float64 msg_xd_y_dot; //xd_y
177 std_msgs::Float64 msg_xd_theta_dot; //xd_y
178 std_msgs::Float64 msg_xd_theta0_dot; //xd_y
179
180
181 std_msgs::Float64 msg_xt_x_dot;
182 std_msgs::Float64 msg_xt_y_dot;
183 std_msgs::Float64 msg_xt_theta_dot;
184

```



```

185 std_msgs::Float64 msg_xt_x_dot_raw;
186 std_msgs::Float64 msg_xt_y_dot_raw;
187 std_msgs::Float64 msg_xt_theta_dot_raw;
188
189 std_msgs::Float64 msg_xt_theta0_dot;
190
191 std_msgs::Float64 msg_xee_x_dot;
192 std_msgs::Float64 msg_xee_y_dot;
193 std_msgs::Float64 msg_xee_theta_dot;
194 std_msgs::Float64 msg_xee_theta0_dot;
195
196 std_msgs::Float64 msg_xt_theta0, msg_xd_theta0, msg_xee_theta0;
197
198 bool safeclose = false;
199 double maxtorque = 10;
200 double theta0safeclose;// = 0; //telikes synthikes gia safeclose
201 double q1safeclose ;//= 45*M_PI/180;
202 double q2safeclose ;//= 45*M_PI/180;
203 double q3safeclose ;//= 10*M_PI/180;
204 double xsafeclose, ysafeclose,thetasafeclose;
205
206 double sumq1 = 0, sumq2 = 0, sumq3 =0;
207 double sumq1dot = 0, sumq2dot = 0, sumq3dot = 0;
208 double force_x, raw_force_x;
209 double forcesum = 0;
210 int force_window_size = 10;
211 int q_window_size = 10;
212 double sumxdot = 0, sumydot = 0, sumthetadot = 0;
213 double sumtheta0dot = 0;
214 double sumxtdot =0, sumytdot =0, sumthetatdot = 0;
215
216
217 std::deque<double> q1_window; // Stores the last N values
218 std::deque<double> q2_window; // Stores the last N values
219 std::deque<double> q3_window; // Stores the last N values
220
221 std::deque<double> q1dot_window; // Stores the last N values
222 std::deque<double> q2dot_window; // Stores the last N values
223 std::deque<double> q3dot_window; // Stores the last N values
224
225 std::deque<double> xdot_window; // Stores the last N values
226 std::deque<double> ydot_window; // Stores the last N values
227 std::deque<double> thetadot_window; // Stores the last N values
228
229 std::deque<double> xtdot_window; // Stores the last N values
230 std::deque<double> ytdot_window; // Stores the last N values
231 std::deque<double> thetatdot_window; // Stores the last N values
232

```

```

233 std::deque<double> xc0dot_window; // Stores the last N values
234 std::deque<double> yc0dot_window; // Stores the last N values
235 std::deque<double> theta0dot_window;
236
237
238 std::deque<double> force_window;
239
240
241
242 Eigen::MatrixXd kp_multiplier(4,4);
243 Eigen::MatrixXd bd_multiplier(4,4);
244
245 bool targetcheck, eecheck, basecheck;
246
247 bool show = false;
248
249 double xdotprev = 0, ydotprev = 0, thetadotprev = 0;
250 double xtdotprev = 0, ytdotprev = 0, thetatdotprev = 0;
251 double xc0dotprev = 0, yc0dotprev = 0, theta0dotprev = 0;
252
253 std::deque<double> xwindow; // Stores the last N values
254 std::deque<double> ywindow; // Stores the last N values
255 std::deque<double> thetawindow; // Stores the last N values
256 std::deque<double> theta0window; // Stores the last N values
257
258 double sumx = 0, sumy = 0, sumtheta = 0, sumtheta0 = 0;
259
260 std::deque<double> xhistory; // Stores the last N values
261 std::deque<double> yhistory; // Stores the last N values
262 std::deque<double> thetahistory;
263
264 double sumxc0dot = 0, sumyc0dot = 0 ;
265 double filtered_ydot = 0;
266 bool stopMotors = false;
267 /////////////// GLOBAL VARIABLES DECLARATION END////////////////////
268
269 #endif

```

