ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ«ΣΥΣΤΗΜΑΤΑ ΑΥΤΟΜΑΤΙΣΜΟΥ»

Μεταπτυχιακή Εργασία

«Ολοκλήρωση Συστήματος Ελέγχου Υποβρύχιου Ρομποτικού Βραχίονα και Ακροδέκτη»

«Integration of Underwater Robotic Arm and End Effectors Control System»

Αντώνιος Καραολάνης

ΕπιβλέπωνΚαθηγητής:
Κωνσταντίνος Κυριακόπουλος

ΑΘΗΝΑ 2021

Περίληψη

Η εργασία αυτή περιγράφει την ολοκλήρωση και βελτίωση ενός υποβρύχιου ρομποτικού βραχίονα, τόσο κατασκευαστικά όσο και προγραμματιστικά. Βασικός στόχος ήταν η αντικατάσταση των παλιών σερβοκινητήρων με νέους πιο εξελιγμένους,ικανούς να παρέχουν την επιθυμητή ανάδραση στο χρήστη, για ολοκλήρωση σύνθετων διαδικασιών ελέγχου .

Στα κατασκευαστικά πλαίσια αυτού του εγχειρήματος υπάγεται η συναρμογή των νέων μονάδων κίνησης και η δημιουργία στεγανού χώρου που θα φιλοξενεί όλα τα ηλεκτρονικά υποστήριξης. Για την τοποθέτηση των σερβοκινητήρων πάνω στη διάταξη, ήταν απαραίτητος ο επανασχεδιασμός των βάσεων που τους συγκρατούν, με χρήση σχεδιαστικών προγραμμάτων όπως το Solidworks. Απαραίτητη προϋπόθεση σε αυτόν τον σχεδιασμό, υπήρξε η διατήρηση της ήδη υπάρχουσας γεωμετρίας, ώστε να μην έχουμε αλλοίωση στο κινηματικό μοντέλο . Η τεχνική κατασκευής τους είναι αυτή της τρισδιάστατης εκτύπωσης (3D printing). Εν συνεχεία, συμπεριλήφθηκε αδιάβροχο housing που εσωκλείει τα ηλεκτρονικά και την μονάδα ελέγχου για την υποστήριξη των νέων κινητήρων. Ο συγκεκριμένος χώρος αποτελεί γέφυρα επικοινωνίας μεταξύ του βραχίονα και του ακροδέκτη αλλά και μεταξύ χρήστη – διάταξης.

Απαραίτητη προϋπόθεση στην ολοκλήρωση του σχήματος ελέγχου της διάταξης, ήταν η συνεργασία της με το λειτουργικό σύστημα ROS. Για το λόγο αυτό, η μονάδα ελέγχου που επιλέχθηκε για να οδηγεί τα νέα σέρβο ήταν της εταιρείας arduino. Αυτοί οι μικρο ελεγκτές είναι απόλυτα συμβατοί με το εν λόγω προγραμματιστικό περιβάλλον και επιπρόσθετα παρέχουν μεγάλη υποστήριξη στο χρήστη, καθώς η κοινότητα που τους χρησιμοποιεί είναι εξαιρετικά εκτεταμένη. Ο μικρο ελεγκτής arduino προγραμματίστηκε σε γλώσσα C++ με τέτοιο τρόπο ώστε ο μελλοντικός χρήστης να μην χρειαστεί να εμπλακεί ξανά με γλώσσα χαμηλού επιπέδου. Συγκεκριμένα έχει αναπτυχθεί κατάλληλο περιβάλλον ώστε να μπορεί κανείς να στέλνει εντολές ταχύτητας ή θέσης στον βραχίονα και να λαμβάνει πίσω την αντίστοιχη πληροφορία τόσο για τη θέση,ταχύτητα αλλά και το φορτίο που αναπτύσσουν οι σερβοκινητήρες.

Τέλος, για την επαλήθευση της λειτουργικότητας όπως περιγράφηκαν ως τώρα, ζητήθηκε η υλοποίηση ενός σχήματος ελέγχου με εφαρμογή Imaged Based Visual Servoing. Με το IBVS, λαμβάνοντας την σχετική θέση του τελικού στοιχείου δράσης ως προς κάποιο στόχο με τη χρήση κάμερας, είμαστε σε θέση να υπολογίζουμε ένα σφάλμα που σχετίζεται με την απόκλιση επιθυμητών και πραγματικών συντεταγμένων στο χώρο. Στόχος μας υπήρξε, με κατάλληλες εντολές ταχύτητας στους κινητήρες, να οδηγούμε αυτό το σφάλμα στο μηδέν. Με αυτή την διαδικασία επαληθεύουμε το σωστό κινηματικό μοντέλο του βραχίονα, την μηχανική του επιδεξιότητα, αλλά και τις σωστά προγραμματισμένες “αναμονές” των εντολών ελέγχου ταχύτητας, για το μελλοντικό χρήστη.

Abstract

This work describes the completion and improvement of an underwater robotic arm, both structurally and programmatically. The main goal was to replace the old servomotors with new more advanced ones, able to provide the desired feedback to the user to complete complex control procedures.

The construction of this project includes the assembly of the new drive units and the creation of a watertight space that will house all the support electronics. In order to place the servomotors on the device, it was necessary to redesign the bases that hold them, using design programs such as Solidworks. A necessary condition in this design was the preservation of the already existing geometry, so that we do not have alteration in the kinematic model. The manufacturing technique was the one of 3D Printing. Next, a waterproof housing was included, to enclose the electronics and the control unit which supports the new motors. This space is a bridge of communication between the arm and the end effector but also between user - device.

A necessary condition for the completion of the control scheme of this device was its cooperation with the ROS operating system. For this reason, the control unit chosen to drive the new servo was from the arduino company. These microcontrollers are fully compatible with this programming environment and in addition provide great support to the user, as the community that uses them is extremely extensive. The arduino microcontroller is programmed in C ++ language in such a way that the future user does not have to re-engage with low-level language. Specifically, a suitable environment has been developed so that one can send speed or position commands to the arm and receive back the corresponding information about the position, speed and the load developed by the servomotors.

Finally, to verify the functionality as described so far, the implementation of a control scheme with Imaged Based Visual Servoing application was requested. With IBVS, taking the relative position of the end effector in relation to a target using a camera, we are able to calculate an error related to the deviation of desired and actual coordinates in space. Our goal was, with proper speed commands on the servos, to drive this error to zero. With this process we verify the correct kinematic model of the arm, its mechanical dexterity, but also the correctly programmed speed control commands, for the future user.

Περιεχόμενα

Εισαγωγή

Το τμήμα υποβρύχιας ρομποτικής του Εργαστηρίου Αυτομάτου Ελέγχου (εφεξής: Ε.Α.Ε.) διαθέτει δύο υποβρύχια οχήματα απομακρυσμένου ελέγχου (ROV). Για τις ανάγκες των ερευνητικών δραστηριοτήτων του εργαστηρίου, είχαν αναπτυχθεί στο παρελθόν ρομποτικοί βραχίονες τεσσάρων (στροφικών) βαθμών ελευθερίας, οι οποίοι προσαρμόστηκαν στα εν λόγω οχήματα. Δυστυχώς αυτές οι διατάξεις ήταν περιορισμένης χρήσης σε εφαρμογές ελέγχου, λόγω της παλιάς τεχνολογίας των σερβοκινητήρων τους οποίους διέθεταν. Βασικό μέλημα ήταν η άμεση αντικατάσταση τους, με νέους ψηφιακούς, ικανούς να δίνουν στο χρήστη το απαιτούμενο feedback που χρειάζεται, για να ολοκληρώσει σύνθετες διεργασίες αυτομάτου ελέγχου. Στην παρακάτω εικόνα (εικόνα1) μπορεί κανείς να δει την τελική μορφή που απέκτησαν οι βραχίονες, ύστερα από αντικατάσταση των παλαιών επενεργητών. Σημειώνουμε ότι στις διατάξεις αυτές προστέθηκαν νέα τελικά στοιχεία δράσης (end effectors) τα οποία αποτέλεσαν αποτέλεσμα διπλωματικής εργασίας προηγούμενου φοιτητή και για αυτό τον λόγο δεν θα γίνει εκτενής αναφορά.

Εικόνα 1 *- Οι δύο βραχίονες στην τελική τους μορφή, όπως βρίσκονται στο εργαστήριο αυτομάτου ελέγχου του ΕΜΠ*

Για την εξελικτική αυτή διαδικασία επιλέχθηκαν τα servo SCS15 της εταιρείας Feetech(εικόνα 2). Έμφαση για την επιλογή τους δόθηκε στην ενσωμάτωση των χαρακτηριστικών που απουσιάζουν από τα προηγούμενα συστήματα, τα οποία μπορούν να συνοψισθούν ως εξής:

- Έτοιμοι controllers, προγραμματισμένοι από την εταιρεία τους να παρέχουν απόλυτο έλεγχο θέσης και ταχύτητας με μεγάλη ακρίβεια
- Feedback θέσης,ταχύτητας και φορτίου
- Ευελιξία στην αλληλεπίδραση με το χρήστη
- Εύκολο προγραμματιστικό περιβάλλον για το χρήστη

Εικόνα 2 *- Οι επενεργητές της εταιρείας Feetech, SCS15*

Βεβαίως, δεν θα πρέπει να αμεληθεί το σημαντικότατο χαρακτηριστικό της λειτουργίας εντός νερού ή υπό βύθιση, που οι συγκεκριμένοι επενεργητές δεν διέθεταν και αποτέλεσε μεγάλο τροχοπέδη στην πορεία της παρούσας εργασίας. Αυτό το εμπόδιο ξεπεράστηκε με την custom αδιαβροχοποίηση τους, με τεχνικές οι οποίες είναι ευρέως γνωστές.

Εφόσον είναι δυνατή η δημιουργία ενός συστήματος με τα ανωτέρω χαρακτηριστικά, οι δυνατότητες των ROV και των βραχιόνων, όσον αφορά στο χειρισμό αντικειμένων, θα βελτιωθούν σημαντικά, δίνοντας χώρο στα μέλη του εργαστηρίου για την εκτέλεση διαφορετικών και συνθετότερων πειραμάτων και την ενσωμάτωση πληρέστερων σχημάτων ελέγχου, εγγύτερων σε ρεαλιστικές εφαρμογές.

Οι συγκεκριμένοι σερβοκινητήρες είναι κατασκευασμένοι ώστε να μπορούν να ελέγχονται μέσω ενός μικροελεγκτή. Αν το σύστημα ελέγχεται τοπικά από κάποια λογική μονάδα, όλες οι εντολές προς τα υποσυστήματα μπορούν να κωδικοποιηθούν σε μορφή υψηλού επιπέδου, προς διευκόλυνση του χρήστη. Η χρήση ενός τέτοιου «ενδιάμεσου» θα επιτρέπει την αποδέσμευση της λειτουργίας του βραχίονα από τις ευρύτερες ενέργειες που επιθυμεί να εκτελέσει ο χρήστης και, συνεπακόλουθα, την κατάστρωση σχημάτων ελέγχου σε υψηλότερο επίπεδο. Η απαίτηση ύπαρξης μικροελεγκτή για τον χειρισμό αυτών των κινητήρων, εισάγει την ανάγκη ύπαρξης στεγανού χώρου που θα τον φιλοξενεί και θα βρίσκεται πάνω στον βραχίονα, εντός του υγρού στοιχείου, καθόλα την διάρκεια της λειτουργίας.

Για την προσαρμογή των νέων servo στους ήδη υπάρχοντες βραχίονες, έπρεπε να γίνουν τροποποιήσεις στις βάσεις που τους συγκρατούν και αποτελούν αναπόσπαστα κομμάτια αυτών των διατάξεων. Για το σκοπό αυτό, έπρεπε να εισαχθώ στο κομμάτι του σχεδιασμού μέσω προγραμμάτων όπως το Solidworks και να εξοικειωθώ με τεχνικές τρισδιάστατης εκτύπωσης(3D printing). Ένα τέτοιο εγχείρημα αποτέλεσε πρόκληση για τον συγγραφέα, καθώς δεν υπήρχε προηγούμενη εμπειρία αλλά και καθώς έπρεπε η ήδη υπάρχουσα γεωμετρία να μην αλλοιωθεί για λόγους διατήρησης του κινηματικού μοντέλου.

Επιπρόσθετα το σύστημα του βραχίονα έπρεπε να εξοπλιστεί με την ιδιότητα της όρασης. Σχεδόν όλες οι σύγχρονες ρομποτικές εφαρμογές χρησιμοποιούν κάμερα στα τελικά στοιχεία δράσης για διεκπεραίωση σχημάτων ελέγχου. Αν και από προηγούμενο φοιτητή αυτή η ιδιότητα είχε καλυφθεί μέσω χρήσης αυτοσχέδιων αδιάβροχων καμερών, αυτές αποδείχθηκαν ανεπαρκείς κατά την πειραματική διαδικασία. Συγκεκριμένα τόσο η κακή ανάλυση τους όσο και η στεγανότητα τους έκανε αδύνατη την ολοκλήρωση των πειραμάτων. Για τον λόγο αυτό, αντικαταστάθηκαν με αδιάβροχες κάμερες υψηλής ανάλυσης ενδοσκοπικού τύπου.

Όσον αφορά στη δομή του κειμένου που ακολουθεί, έχει ως έξης:

Στο πρώτο κεφάλαιο παρουσιάζεται η κινηματική ανάλυση του μοντέλου του βραχίονα, πάνω στην οποία βασίστηκαν όλοι οι αλγόριθμοι και εκτέλεση των πειραματικών διαδικασιών

Στο *δεύτερο κεφάλαιο* της εργασίας διατυπώνονται οι προδιαγραφές που ορίστηκαν για το σύστημα, από τις ιδιότητες του μηχανισμού έως τη διασύνδεσή του με το χρήστη και τα παρελκόμενα στοιχεία.

Στο *τρίτο κεφάλαιο* γίνεται η περιγραφή της σχεδιαστικής διαδικασίας και των μερών του μηχανισμού, με σκοπό την αιτιολόγηση των επιλογών και των λύσεων που υιοθετήθηκαν στην ανάπτυξή των μερών του.

Το σύνολο του ηλεκτρολογικού υλικού, καθώς και του λογισμικού ελέγχου, περιγράφεται σε τεχνικό επίπεδο στο *τέταρτο κεφάλαιο*.

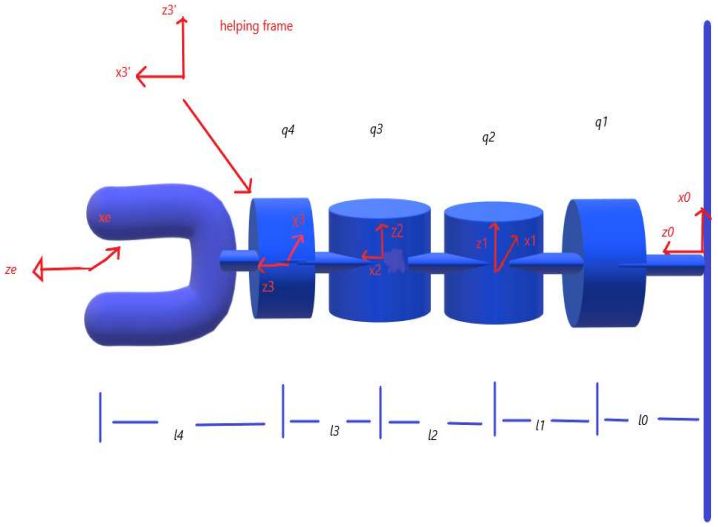
Στο *πέμπτο κεφάλαιο* περιγράφονται οι πειραματικές διαδικασίες βασισμένες στο Imaged Based Visual Servoing και τον έλεγχο της στεγανότητας του μηχανισμού υπό βύθιση.

Στο *έκτο κεφάλαιο* παραθέτω έναν οδηγό χρήσης της διάταξης, με τα συγκεκριμένα βήματα που πρέπει να ακολουθήσει κανείς, τι πρέπει να προσέξει και που να αναζητήσει την πληροφορία που χρειάζεται για να θέσει την διάταξη σε λειτουργία.

1 Κινηματική Ανάλυση

1.1 Θεωρητική Ανάλυση Manipulator's Kinematics

Η κινηματική περιγραφή του υποβρύχιου χειριστή αναπτύσσεται με βάση την παράμετρο Denavit-Hartenberg (D-H). Το μηδενικό πλαίσιο που ακολουθεί τη διαδικασία D-H, προσαρτάται στο όχημα. Η κεντρική ιδέα είναι να περιγράψουμε την κίνηση του τελικού στοιχείου δράσης στη βάση του και εν συνεχεία στο σταθερό πλαίσιο του ROV (base rigid body frame). Στο τέλος, γνωρίζοντας τη θέση του οχήματος ως προς το αδρανειακό πλαίσιο και τη θέση του end effector σε σχέση με το σταθερό πλαίσιο του αμαξώματος, θα είμαστε σε θέση να υπολογίσουμε το τελικό διάνυσμα που περιγράφει τη θέση και την περιστροφή του τελικού στοιχείου δράσης στο αδρανειακό πλαίσιο.



DH-TABLE

Link i	θi	di	αi	ai
1	q1+π/2	lo+l1	+π/2	0
2	q2+π/2	0	0	l2
3'	q3	0	0	l3
3	-π/2	0	-π/2	0

e	q4	l4	0	0

Με βάση τις παραμέτρους DH στον προηγούμενο πίνακα, το μητρώο ομογενούς μετασχηματισμού που καθορίζει τη θέση του e.e σε σχέση με το μηδενικό σύστημα συντεταγμένων της βάσης του δίνεται παρακάτω:

$$A_e^0 = A_1^0 A_2^1 A_{3'}^2 A_3^{3'} A_e^3$$

$$A_1^0 = \begin{bmatrix} -s_1 & 0 & c_1 & 0 \\ c_1 & 0 & s_1 & 0 \\ 0 & 1 & 0 & l_0 + l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2^1 = \begin{bmatrix} -s_2 & -c_2 & 0 & -l_2 s_2 \\ c_2 & -s_2 & 0 & l_2 c_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{3'}^2 = \begin{bmatrix} c_3 & -s_3 & 0 & l_3 c_3 \\ s_3 & c_3 & 0 & l_3 s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^{3'} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4^3 = \begin{bmatrix} c_4 & -s_4 & 0 & 0 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 1 & l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2^0 = A_1^0 A_2^1 = \begin{bmatrix} -s_1 & 0 & c_1 & 0 \\ c_1 & 0 & s_1 & 0 \\ 0 & 1 & 0 & l_0 + l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -s_2 & -c_2 & 0 & -l_2 s_2 \\ c_2 & -s_2 & 0 & l_2 c_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow$$

$$A_2^0 = \begin{bmatrix} s_1 s_2 & s_1 c_2 & c_1 & l_2 s_1 s_2 \\ -c_1 s_2 & -c_1 c_2 & s_1 & -l_2 c_1 s_2 \\ c_2 & -s_2 & 0 & l_2 c_2 + l_0 + l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{3'}^{0} = A_{2}^{0}A_{3'}^{2} = \begin{bmatrix} s_1s_2 & s_1c_2 & c_1 & l_2s_1s_2 \\ -c_1s_2 & -c_1c_2 & s_1 & -l_2c_1s_2 \\ c_2 & -s_2 & 0 & l_2c_2 + l_0 + l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_3 & -s_3 & 0 & l_3c_3 \\ s_3 & c_3 & 0 & l_3s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow$$

$$A_{3'}^{0} = \begin{bmatrix} s_1s_2c_3 + s_1c_2s_3 & -s_1s_2s_3 + s_1c_2c_3 & c_1 & s_1s_2l_3c_3 + s_1c_2l_3s_3 + l_2s_1s_2 \\ -c_1s_2c_3 - c_1c_2s_3 & c_1s_2s_3 - c_1c_2c_3 & s_1 & -c_1s_2l_3c_3 - c_1c_2l_3s_3 - l_2c_1s_2 \\ c_2c_3 - s_2s_3 & -c_2s_3 - s_2c_3 & 0 & c_2l_3c_3 - s_2l_3s_3 + l_2c_2 + l_0 + l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{3}^{0} = A_{3}^{0}A_{3}^{3'}$$

$$= \begin{bmatrix} s_1s_2c_3 + s_1c_2s_3 & -s_1s_2s_3 + s_1c_2c_3 & c_1 & s_1s_2l_3c_3 + s_1c_2l_3s_3 + l_2s_1s_2 \\ -c_1s_2c_3 - c_1c_2s_3 & c_1s_2s_3 - c_1c_2c_3 & s_1 & -c_1s_2l_3c_3 - c_1c_2l_3s_3 - l_2c_1s_2 \\ c_2c_3 - s_2s_3 & -c_2s_3 - s_2c_3 & 0 & c_2l_3c_3 - s_2l_3s_3 + l_2c_2 + l_0 + l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{3}^{0} = A_{3}^{0}A_{3}^{3'} = \begin{bmatrix} s_1s_{23} & s_1c_{23} & c_1 & s_1l_3s_{23} + l_2s_1s_2 \\ -c_1s_{23} & -c_1c_{23} & s_1 & -c_1l_3s_{23} - l_2c_1s_2 \\ c_{23} & -s_{23} & 0 & l_3c_{23} + l_2c_2 + l_0 + l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{3}^{0} = \begin{bmatrix} -s_1c_{23} & -c_1 & s_1s_{23} & s_1l_3s_{23} + l_2s_1s_2 \\ c_1c_{23} & -s_1 & -c_1s_{23} & -c_1l_3s_{23} - l_2c_1s_2 \\ s_{23} & 0 & c_{23} & l_3c_{23} + l_2c_2 + l_0 + l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{e}^{0} = A_{3}^{0}A_{4}^{3}$$

$$A_{e}^{0} = \begin{bmatrix} -s_1c_{23} & -c_1 & s_1s_{23} & s_1l_3s_{23} + l_2s_1s_2 \\ c_1c_{23} & -s_1 & -c_1s_{23} & -c_1l_3s_{23} - l_2c_1s_2 \\ s_{23} & 0 & c_{23} & l_3c_{23} + l_2c_2 + l_0 + l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_4 & -s_4 & 0 & 0 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 1 & l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{e}^{0} = \begin{bmatrix} -s_1c_{23}c_4 - c_1s_4 & s_1c_{23}s_4 - c_1c_4 & s_1s_{23} & s_1s_{23}l_4 + s_1l_3s_{23} + l_2s_1s_2 \\ c_1c_{23}c_4 - s_1s_4 & -c_1c_{23}s_4 - s_1c_4 & -c_1s_{23} & -c_1s_{23}l_4 - c_1l_3s_{23} - l_2c_1s_2 \\ s_{23}c_4 & -s_{23}s_4 & c_{23} & l_4c_{23} + l_3c_{23} + l_2c_2 + l_0 + l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_e^0 = \begin{bmatrix} -s_1c_{23}c_4 - c_1s_4 & s_1c_{23}s_4 - c_1c_4 & s_1s_{23} & s_1s_{23}(l_3 + l_4) + l_2s_1s_2 \\ c_1c_{23}c_4 - s_1s_4 & -c_1c_{23}s_4 - s_1c_4 & -c_1s_{23} & -c_1s_{23}(l_3 + l_4) - l_2c_1s_2 \\ s_{23}c_4 & -s_{23}s_4 & c_{23} & c_{23}(l_3 + l_4) + l_2c_2 + l_0 + l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Οπότε καταλήγουμε για τον προσανατολισμό και τη θέση του εργαλείου ως προς το μηδενικό frame της βάσης του(το οποίο είναι προσκολλημένο πάνω στο Rov):

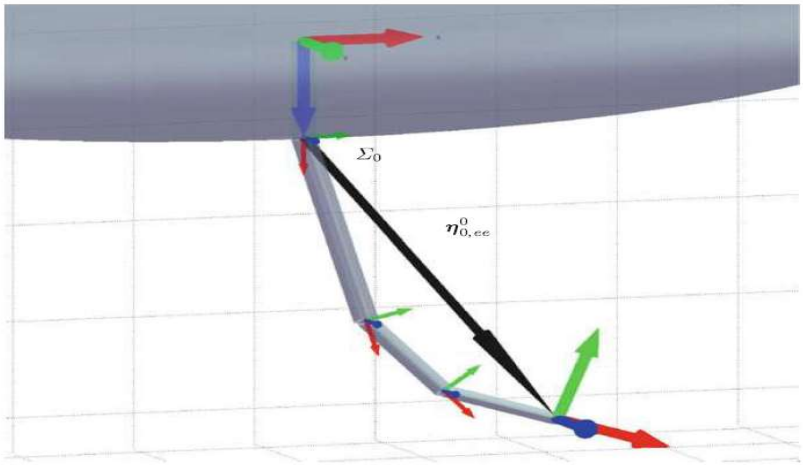
Θέση end effector ως προς το Manipulator base frame

$$\boldsymbol{p_e^0} = [\boldsymbol{p}_{e,x} \quad \boldsymbol{p}_{e,y} \quad \boldsymbol{p}_{e,z}]^T$$

$$\boldsymbol{p}_{e,x} = \boldsymbol{s_1}[\boldsymbol{s_{23}}(\boldsymbol{l_3} + \boldsymbol{l_4}) + \boldsymbol{l_2}\boldsymbol{s_2}]$$

$$\boldsymbol{p}_{e,y} = -\boldsymbol{c_1}[\boldsymbol{s_{23}}(\boldsymbol{l_3} + \boldsymbol{l_4}) + \boldsymbol{l_2}\boldsymbol{s_2}]$$

$$\boldsymbol{p}_{e,z} = \boldsymbol{c_{23}}(\boldsymbol{l_3} + \boldsymbol{l_4}) + \boldsymbol{l_2}\boldsymbol{c_2} + \boldsymbol{l_0} + \boldsymbol{l_1}$$



Rotational matrix από το base frame στο end effector frame

$$\boldsymbol{R_e^0} = [\hat{\boldsymbol{n}}_{e,x} \quad \hat{\boldsymbol{o}}_{e,y} \quad \hat{\boldsymbol{a}}_{e,z}]^T$$

$$R_e^0 = \begin{bmatrix} -s_1c_{23}c_4 - c_1s_4 & s_1c_{23}s_4 - c_1c_4 & s_1s_{23} \\ c_1c_{23}c_4 - s_1s_4 & -c_1c_{23}s_4 - s_1c_4 & -c_1s_{23} \\ s_{23}c_4 & -s_{23}s_4 & c_{23} \end{bmatrix}$$

Inverse Kinematics

$$\frac{p_{e,x}}{p_{e,y}} = -tan(q_1) \rightarrow \boldsymbol{q_1 = -atan2(p_{e,x}, p_{e,y})}$$

Με γνωστό πλέον το q_1:

$$k = p_{e,x}/s_1 = -n_{e,y}/c_1 = s_{23}(l_3 + l_4) + l_2 s_2$$

$$m = p_{e,z} - l_0 - l_1 = c_{23}(l_3 + l_4) + l_2 c_2$$

Υψώνοντας στο τετράγωνο και αθροίζοντας τους δύο όρους προκύπτει :

$$k^2 + m^2 = (l_3 + l_4)^2 + l_2^2 + 2(l_3 + l_4) l_2 (s_{23} s_2 + c_{23} c_2)$$

$$k^2 + m^2 = (l_3 + l_4)^2 + l_2^2 + 2(l_3 + l_4) l_2 c_3$$

Οπότε

$$\boldsymbol{q_3 = \pm \arccos\left[\frac{k^2 + m^2 - (l_3 + l_4)^2 - l_2^2}{2(l_3 + l_4) l_2}\right]}$$

Για την εύρεση της q2 γίνεται αναδιατύπωση των k,m :

$$k = (c_2 s_3 + s_2 c_3)(l_3 + l_4) + l_2 s_2$$

$$m = (c_2 c_3 - s_2 s_3)(l_3 + l_4) + l_2 c_2$$

Άρα

$$k = s_3 l_{34} c_2 + (c_3 l_{34} + l_2) s_2$$

$$m = -(s_3 l_{34}) s_2 + (c_3 l_{34} + l_2) c_2$$

Υποθέτοντας τις αποστάσεις

$d_1 = s_3 l_{34} = dsa$

$d_2 = c_3 l_{34} + l_2 = dca$

Με

$a = atan2(d_1, d_2)$

Τελικά έχουμε

$k = dcos(q_2 + a)$

$m = dsin(q_2 + a)$

Οπότε

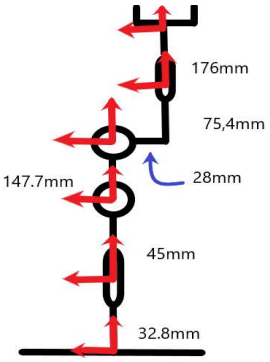
$\boldsymbol{q_2 = atan2(m, k) - \alpha}$

Τέλος με γνωστά πλέον τα q1,q2,q3, αντικαθιστώντας στον Rotational Matrix βρίσκουμε εύκολα την τιμή του **q4** για δοσμένο γνωστό orientation.

1.2 Προγραμματιστική απλούστευση με Python και Pykdl

Όπως είδαμε στο προηγούμενο κεφάλαιο, η κινηματική προσέγγιση μέσω της ανάλυσης, είναι μία επίπονη διαδικασία, που όμως ένας μηχανικός Ρομποτικής πρέπει να κατέχει ώστε να μπορεί να βγάλει ακριβή και ορθά συμπεράσματα για τα μοντέλα τα οποία μελετά. Ευτυχώς στα χέρια μας κατέχουμε εργαλεία που μας απαλλάσσουν από αυτό το χρονοβόρο υπολογισμό, όπως αυτό της βιβλιοθήκης της Python και συγκεκριμένα της PyKdl. Εδώ, τοποθετώντας αυθαίρετα τα πλαίσια αναφοράς σε κάθε joint, είμαστε σε θέση να υπολογίζουμε κάθε φορά την κινηματική αλυσίδα μέσω απλών εντολών και χωρίς να είμαστε υποχρεωμένοι να ακολουθούμε την σύμβαση DH. Παρακάτω παρουσιάζω τον τρόπο με τον οποίο τοποθετήθηκαν τα πλαίσια αναφοράς και ένα σύντομο κομμάτι κώδικα που υπολογίζει την ζητούμενη κινηματική αλυσίδα.

```
#!/usr/bin/env python

#ROS msgs
from sensor_msgs.msg import JointState
from geometry_msgs.msg import Vector3Stampe


# Python imports
from PyKDL import *

# Python Libs
import numpy as np
# Ros Libs
import rospy
import math
from math import *

# Math Utils
from math_utils import *

arm_chain=Chain()


#From Platform Vehicle Center Frame to Arm Base Frame (Sxhmatismos alysidas braxiona)

joint0 = Joint(Joint.None)
frame0 = Frame(Vector(0.0, 0.0, 32.8e-3))
segment0 = Segment(joint0, frame0)
arm_chain.addSegment(segment0)

joint1 = Joint(Joint.RotZ)
frame1 = Frame(Vector(0.0,0.0,45e-3 ))
segment1 = Segment(joint1, frame1)
arm_chain.addSegment(segment1)

joint2 = Joint(Joint.RotY)
frame2 = Frame(Vector(0.0, , 0.0 ,147.7e-3))
segment2 = Segment(joint2, frame2)
arm_chain.addSegment(segment2)

joint3 = Joint(Joint.RotY)
frame3 = Frame(Vector(-28e-3,0.0 ,75.4e-3 ))
segment3 = Segment(joint3, frame3)
arm_chain.addSegment(segment3)

joint4 = Joint(Joint.RotZ)
frame4 = Frame(Vector(0.0, 0.0, 176e-3))
segment4 = Segment(joint4, frame4)
arm_chain.addSegment(segment4)
```



Το παραπάνω κομμάτι κώδικα δεν κάνει τίποτα περισσότερο απο το να υπολογίζει την κινηματική μας αλυσίδα. Στη συνέχεια παραθέτω μία κλάση η οποία αναλαμβάνει να υπολογίζει τόσο την ορθή, ανάστροφη κινηματική αλλά και την Ιακωβιανή του βραχίονα.

```
''' ARM Kinematics Class '''
class arm_uvms_kinematics:

	def __init__(self):
		print "ARM UVMS CLASS INITIALIZED"
		#Subscribers
		rospy.Subscriber("/lbv150_uvms/joint_states", JointState, self.updateArmStates, queue_size=1)
		#Publishers
		self.pub_ee_pos = rospy.Publisher("/lbv150_uvms/ee/position", Vector3Stamped, queue_size=1)

		self.arm_states = np.array([0.0, 0.0, 0.0, 0.0])


	''' Calculate Forward Kinemtics '''
	def fk(self, jointAngles):
			if isinstance (jointAngles, JntArray):
				joints = jointAngles
			elif isinstance (jointAngles, np.ndarray):
					joints = JntArray(4)
					for i in range (1,4):
						joints[i] = jointAngles[i]
			fk=ChainFkSolverPos_recursive(arm_chain)
			finalFrame=Frame()
			fk.JntToCart(joints,finalFrame)
			return finalFrame

	''' Calculate Inverse Kinemtics
	inputs: joint vector, position vector, orientation vector
	output: joint vector
	'''
	def ik(self, jointAngles, desVector,desRot):
			if isinstance (jointAngles, JntArray):
				joints = jointAngles
			elif isinstance (jointAngles, np.ndarray):
				joints = JntArray(4)
				for i in range (1,4):
					joints[i] = jointAngles[i]
			fk=ChainFkSolverPos_recursive(arm_chain)
			vik=ChainIkSolverVel_pinv(arm_chain)
			ik=ChainIkSolverPos_NR(arm_chain,fk,vik)
			desiredFrame = Frame(desRot,desVector)
			q_out=JntArray(4)
			ik.CartToJnt(joints,desiredFrame,q_out)
			return q_out


	''' Calculate Jacobian Matrix'''
	def jac(self, jointAngles):
			if isinstance (jointAngles, JntArray):
				joints = jointAngles
			elif isinstance (jointAngles, np.ndarray):
				joints = JntArray(4)
				for i in range (1,4):
					joints[i] = jointAngles[i]
			jacobian = Jacobian(4)
			solver = ChainJntToJacSolver(arm_chain)
			solver.JntToJac(joints,jacobian)
			return jacobian
```

Μέσω αυτής της απλής προγραμματιστικής διαδικασίας έχουμε άμεσα στα χέρια μας όλα εκείνα τα κινηματικά εφόδια που χρειαζόμαστε για να περιγράψουμε πλήρως το μοντέλο του βραχίονα. Δεν πρέπει όμως κανείς να αποδοκιμάζει την ανάλυση όπως την δείξαμε στο κεφάλαιο 1.1 καθώς καλλιεργεί στον μηχανικό την αντίληψη και την βαθιά κατανόηση του τι ακριβώς συμβαίνει στο υπό μελέτη σύστημα.

2 Τεχνική περιγραφή του προβλήματος

Οι προδιαγραφές συνοψίζονται στην ακόλουθη λίστα και αναλύονται περαιτέρω στη συνέχεια.

- Συμβατότητα με τους υπάρχοντες βραχίονες (συναρμολόγηση, κινητικότητα)
- Feedback θέσης,ταχύτητας και φορτίου
- Ενσωμάτωση ενός ή και δύο οπτικών μέσων (cameras), ανάλογα με την ικανότητα κάλυψης του πεδίου δράσης
- Τροφοδοσία από εξωτερική πηγή
- Έλεγχος λειτουργίας (high-level) από εξωτερικό ηλεκτρονικό υπολογιστή (H/Y) και συνεργασία με το ROS
- Η υψηλότερη δυνατή προστασία των ευαίσθητων μερών από την επαφή με το νερό.

2.1 Προδιαγραφή 1: Συμβατότητα με τους υπάρχοντες βραχίονες

Ο σχεδιασμός των νέων βάσεων πρέπει να υπακούει στην ήδη υπάρχουσα γεωμετρία. Όσον αφορά τη συμβατότητα συναρμογής, το βασικότερο στοιχείο ήταν η δυνατότητα προσάρτησης των νέων βάσεων με τις όποιες σχεδιαστικές μεταβολές, πάνω στον βραχίονα, χωρίς να προκύπτει μεταβολή στο κινηματικό μοντέλο που ήδη έχει αναλυθεί και μελετηθεί.

2.2 Προδιαγραφή 2: Feedback

Κομβικής σημασίας ανάγκη υπήρξε η δυνατότητα παροχής ανάδρασης θέσης, ταχύτητας και φορτίου. Όλη αυτή η πληροφορία παρέχεται από συναρτήσεις προκατασκευασμένες από την εν λόγω εταιρεία μέσω της βιβλιοθήκης SCServo ειδικά σχεδιασμένη με χρήση μικροελεγκτών arduino. Χρειάστηκαν ελάχιστες μεταβολές στις ήδη υπάρχουσες υλοποιήσεις,

για παράδειγμα στη συνάρτηση ανάδρασης της ταχύτητας και φορτίου. Πιο συγκεκριμένα, τέτοια συνάρτηση δεν υπήρχε δομημένη για το συγκεκριμένο μοντέλο και χρειάστηκε να προγραμματιστεί εκ νέου. Την συγκεκριμένη τροποποιημένη βιβλιοθήκη την παρέχω στο επισυναπτόμενο με αυτή την διπλωματική usb, ώστε ο αναγνώστης να την εγκαταστήσει στο σύστημα του και να τρέξει του κώδικες χαμηλού επιπέδου στο arduino χωρίς προβλήματα.

2.3 Προδιαγραφή 3: Οπτικό μέσο (camera)

Η προδιαγραφή αυτή τέθηκε αφενός διότι πολλά από τα ερευνητικά αντικείμενα του εργαστηρίου περιλαμβάνουν τεχνικές MachineVision, αφετέρου γιατί ο μηχανισμός (όπως και τα ROV) δεν είναι αυτόνομα συστήματα. Σημειώνεται ότι καθοριστικός παράγοντας αποδείχθηκε και η ικανότητα αναγνώρισης/ανάγνωσης του οπτικού μέσου στο πακέτο λογισμικού RobotOperatingSystem. Τέλος, ενώ δεν δόθηκε κάποια επιπλέον προδιαγραφή ως προς την ποιότητα της εικόνας (της ανάλυσης του αισθητήρα του οπτικού μέσου), θεωρήθηκε ως ελάχιστη η συνήθης ανάλυση *640x480 pixels*, με συχνότητα *30 fps*. Για να καλυφθεί και η ανάγκη της χρήσης στο νερό επιλέχθηκαν κάμερες ενδοσκοπικού τύπου, νέας τεχνολογίας υψηλής ανάλυσης. Αυτές οι κάμερες πέραν της αδιάβροχης ιδιότητας τους , είναι πολύ ελαφριές με αποτέλεσμα να μην προσθέτουν επιπρόσθετο φορτίο στη διάταξη.

2.4 Προδιαγραφή 4: Ηλεκτρική τροφοδοσία & έλεγχος λειτουργίας

Τόσο η ηλεκτρική τροφοδοσία του συστήματος, όσο και ο έλεγχος της λειτουργίας του δεν θα μπορούσαν να γίνουν από το όχημα στο οποίο εγκαθίσταται ή από την γραμμή (tether) που χρησιμοποιείται γι’ αυτό - το ίδιο ίσχυε και για τους βραχίονες. Η μεν τροφοδοσία θα έπρεπε να γίνει μέσω εξωτερικής πηγής (τροφοδοτικό Η/Υ), με τάση $+12V$:

$$V_{supply} = +12\ V$$

ο δε έλεγχος λειτουργίας μέσω εξωτερικού Η/Υ και κάποιου συνήθη διαύλου επικοινωνίας (USB, RS232). Σημειώνω εδώ ότι τα 12 V απαιτούνται από τον επενεργητή του gripper, ενώ τα

servo που επιλέχθηκαν απαιτούν τάση 7.5 V. Έχοντας ως απαίτηση την ελαχιστοποίηση των καλωδίων επιλέξαμε την καθολική παροχή τροφοδοσίας στα 12 V και με χρήση ενός ολοκληρωμένου πτώσης τάσης ακριβώς πριν την τροφοδοσία των σερβοκινητήρων επιτυγχάνουμε με μία γραμμή παροχής, να ικανοποιούμε τόσο τις απαιτήσεις των servo όσο και του επενεργητή του gripper.

Οι δίαυλοι αυτοί δεν είναι οι πλέον πρακτικοί από άποψη αριθμού απαιτούμενων καλωδίων, ωστόσο είναι ιδιαίτερα διαδεδομένοι και σχετικά εύχρηστοι. Δεδομένου μάλιστα ότι η ανάπτυξη του γενικότερου συστήματος είναι στο πλέον πρώιμο στάδιο, θα ήταν δυνατόν να επωφεληθεί κανείς από ευρέως χρησιμοποιούμενα, ολοκληρωμένα υποσυστήματα (development boards, cameraσκ.α.), τα οποία χρησιμοποιούν συνήθως ακριβώς αυτούς τους διαύλους.

2.5 Προδιαγραφή 5: Προστασία από επαφή με το νερό

Τέλος, σχετικά με την αξιόπιστη λειτουργία εντός νερού, ως κριτήριο έπρεπε να ληφθεί η μέγιστη πιθανή διάρκεια ενός πειράματος. Η παράμετρος αυτή μπορεί να χωριστεί

- στο μέγιστο χρόνο παραμονής των ROV εντός της πισίνας και
- στο μέγιστο χρονικό διάστημα πριν την ολοκλήρωση μιας πειραματικής διαδικασίας, η οποία μπορεί να περιλαμβάνει πολλαπλούς κύκλους εντός-εκτός πισίνας.

Το πρώτο χρονικό διάστημα– έστω υπό το συμβολισμό $\Delta t_{underwater}$ – είναι καθοριστικής σημασίας για την αξιόπιστη λειτουργία του μηχανισμού. Είναι προφανές ότι, για το σύνολο αυτού του χρονικού διαστήματος, δεν θα πρέπει να υπάρχει καμία δυσλειτουργία από την επαφή με το νερό σε κατάσταση βύθισης.

Ο χρόνος αυτός ορίστηκε εμπειρικά σε περίπου 30 λεπτά :

$$\Delta t_{underwater} \sim 30\ min$$

Το δεύτερο χρονικό διάστημα – έστω $\Delta t_{experiment}$ – σχετίζεται μεν με την αξιόπιστη λειτουργία του μηχανισμού, αλλά έμμεσα. Στον αρχικό σχεδιασμό συμπεριλήφθηκε η πιθανότητα να χρειαστούν ενδιάμεσες περίοδοι συντήρησης του μηχανισμού (στέγνωμα & έλεγχος), εφόσον αυτό θα ήταν αναγκαίο. Έτσι στα πλαίσια του συνολικού χρόνου ενός πειράματος, θα έπρεπε να υπάρχει η δυνατότητα πρόσβασης σε τυχόν ευάλωτα μέρη, με όσο το δυνατόν ευκολότερη και ταχύτερη αποσυναρμολόγηση/συναρμολόγηση και το μικρότερο δυνατό κίνδυνο βλάβης ή ζημιάς από αυτή τη διαδικασία.

Επίσης εμπειρικά, ο χρόνος αυτός ορίστηκε σε περίπου 4 ώρες :

$$\Delta t_{experiment} \sim 4\ h$$

Για το συστηματικό χαρακτηρισμό του βαθμού προστασίας για περιβλήματα (enclosures) χρησιμοποιείται ο κωδικός **IP**κατά το πρότυπο EN (ή IEC) 60529 [1], ο οποίος χαρακτηρίζεται από 2 κύρια αρηθμητικά ψηφία και 2 προαιρετικά:

Το *πρώτο* αριθμητικό ψηφίο αφορά στην προστασία από δυνητικά επικίνδυνα υλικά, βάσει διαστάσεων (π.χ. σκόνη). Το *δεύτερο* ψηφίο αφορά στην προστασία από την εισροή νερού, βάσει του χρόνου έκθεσης και της ποσότητας του ύδατος. Τα δύο επίπεδα στα οποία εμπίπτουν οι χρόνοι που αναφέρθηκαν παραπάνω είναι το **7** και το **8**, τα οποία αφορούν σε συνθήκες προσωρινής και συνεχούς βύθισης αντίστοιχα και ορίζονται ως εξής:

Επίπεδο	Προστασία από	Αποτελεσματικό σε	Λεπτομέρειες δοκιμής
7	Βύθιση, έως 1 m βάθος	Εισροή νερού σε επιβλαβή ποσότητα δεν θα είναι δυνατή, ενόσω το περίβλημα θα είναι βυθισμένο σε νερό, υπό ορισμένες συνθήκες πίεσης και χρόνου (βύθιση έως 1 m).	Διάρκεια δοκιμής: 30 min. Δοκιμή με το χαμηλότερο σημείο του περιβλήματος 1000 mm κάτω από την επιφάνεια του νερού ή το υψηλότερο σημείο 150 mm κάτω από την επιφάνεια - το βαθύτερο εκ των δύο.
8	Βύθιση, 1 m βάθος ή περισσότερο	Το περίβλημα είναι κατάλληλο για συνεχή βύθιση σε νερό, υπό συνθήκες οριζόμενες από τον κατασκευαστή. Ωστόσο, για συγκεκριμένους τύπους περιβλημάτων, σημαίνει ότι είναι	Διάρκεια δοκιμής: οριζόμενη από κατασκευαστή

		πιθανή η εισροή νερού, αλλά με τρόπο που δεν έχει επιβλαβείς συνέπειες. Το βάθος και η διάρκεια της δοκιμής αναμένονται μεγαλύτερα από τις συνθήκες του επιπέδου IPx7, και άλλου είδους επιδράσεις μπορούν να προστεθούν, όπως κύκλοι θέρμανσης-ψύξης πριν τη βύθιση.	Βάθος οριζόμενο από κατασκευαστή, γενικά έως 3 m

2.6 Προδιαγραφή 6: Συνεργασία με το ROS

Ίσως η πιο ζωτικής σημασίας προδιαγραφή, ήταν η επικοινωνία των νέων σερβοκινητήρων με το Robotic Operating System (ROS). Αυτό διασφαλίστηκε μέσω της δυνατότητας επικοινωνίας του μικροελεγκτή που οδηγεί τα servo (arduino) με το ROS, με χρήση της βιβλιοθήκης rosserial. Η rosserial αποτελεί μία ευρέως διαδεδομένη βιβλιοθήκη για τους χρήστες αυτής της ρομποτικής πλατφόρμας σε συνεργασια με ελεγκτές arduino και μεγάλη υποστήριξη στην διαδικτυακή κοινότητα.

3 Σχεδιασμός και Ανάπτυξη

Στο παρών κεφάλαιο αναλύω όλη την πορεία που ακολούθησα για το κομμάτι της υλοποίησης του hardware. Η τελική λύση ήταν αποτέλεσμα πολλών δοκιμών, στην αναζήτηση της βέλτιστης δυνατής προσέγγισης του προβλήματος.

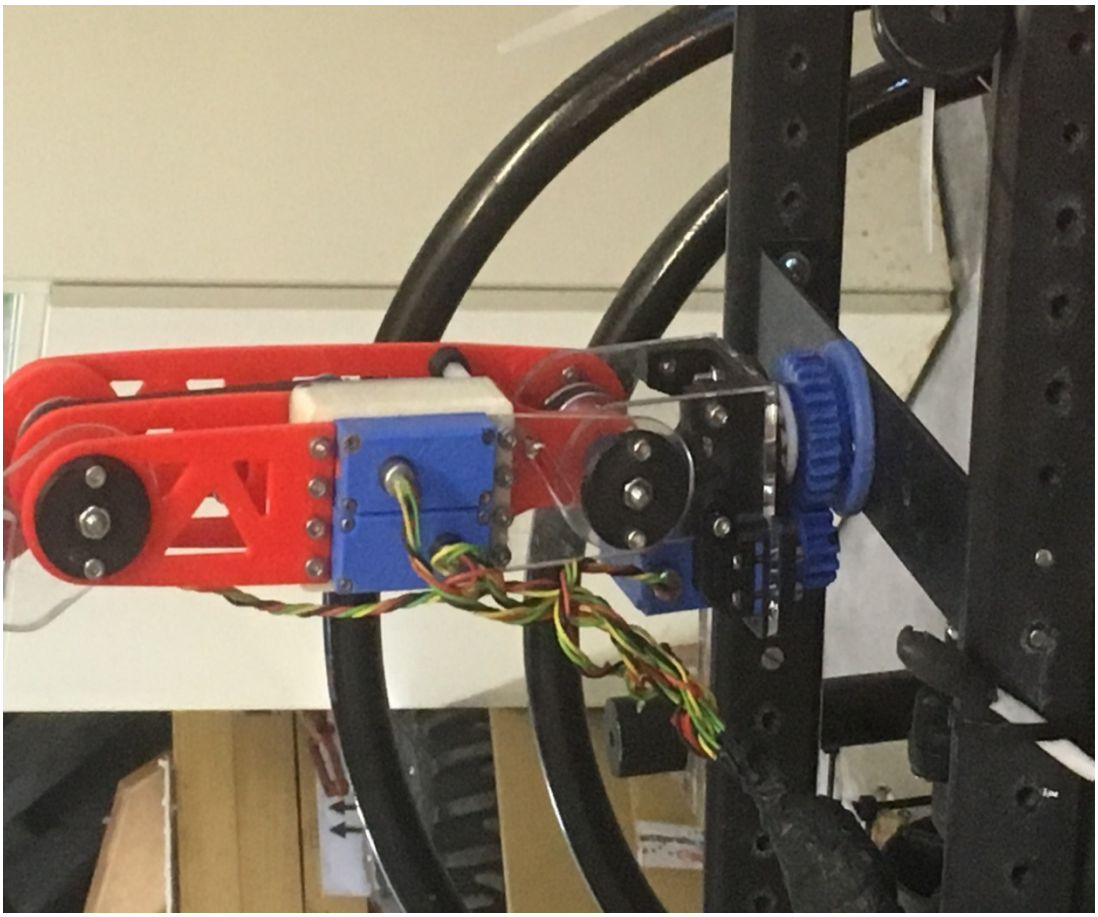
Ζητήματα

- Αντικατάσταση των σερβοκινητήρων με νέους ψηφιακούς, τύπου SCS15 και δημιουργία κατάλληλων βάσεων
- Δημιουργία μίας κομψής,αδιάβροχης και λειτουργικής διάταξης που θα εσωκλείει τις καλωδιώσεις και τη μονάδα ελέγχου των κινητήρων.
- Οπτικό μέσο
-Αδιαβροχοποίηση των σερβοκινητήρων

3.1 Βάσεις για τους νέους σερβοκινητήρες

Η αντικατάσταση των παλαιών σερβοκινητήρων κρίθηκε απαραίτητη λόγω της ανάγκης για παροχή feedback, κάτι που η τεχνολογία τους δεν επέτρεπε. Στην Εικόνα 3.1.1 διακρίνουμε την διάταξη του βραχίονα με τα προυπάρχοντα servo.

Η πρώτη πρόκληση με την οποία ήρθα αντιμέτωπος, ήταν ο σχεδιασμός κατάλληλων βάσεων στήριξης που θα φιλοξενούσαν τους νέους αυτούς κινητήρες(μιας και οι ήδη υπάρχουσες δεν ταίριαζαν με τα νέα servo) στην ήδη υλοποιημένη διάταξη του βραχίονα. Αναλογιζόμενοι ότι δεν υπήρχε καμία προηγούμενη εμπειρία στον σχεδιασμό, αυτό αποτέλεσε σημαντική πρόκληση.



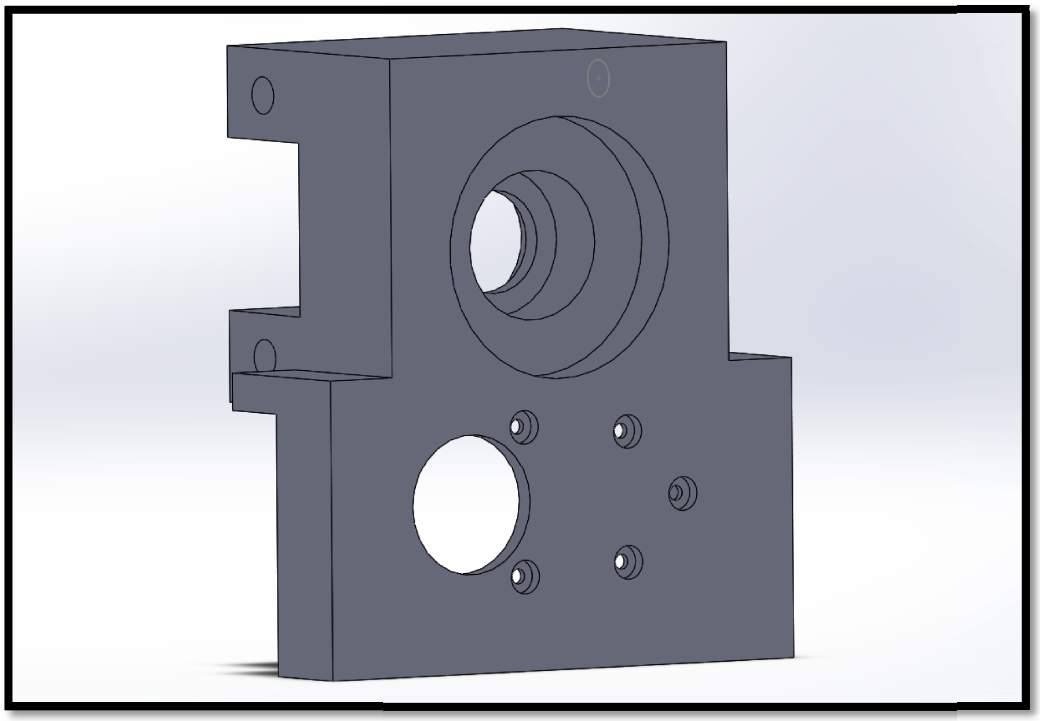
Εικόνα 3.1.1 *– Η διάταξη του βραχίονα με τα προυπάρχοντα servo*

Αφιερώθηκαν δύο εβδομάδες για την εκμάθηση των βασικών στοιχείων σχεδιασμού μέσω του προγράμματος SolidWorks και συνολικά περίπου ένας μήνας για την ολοκλήρωση του σχεδιασμού των νέων βάσεων.

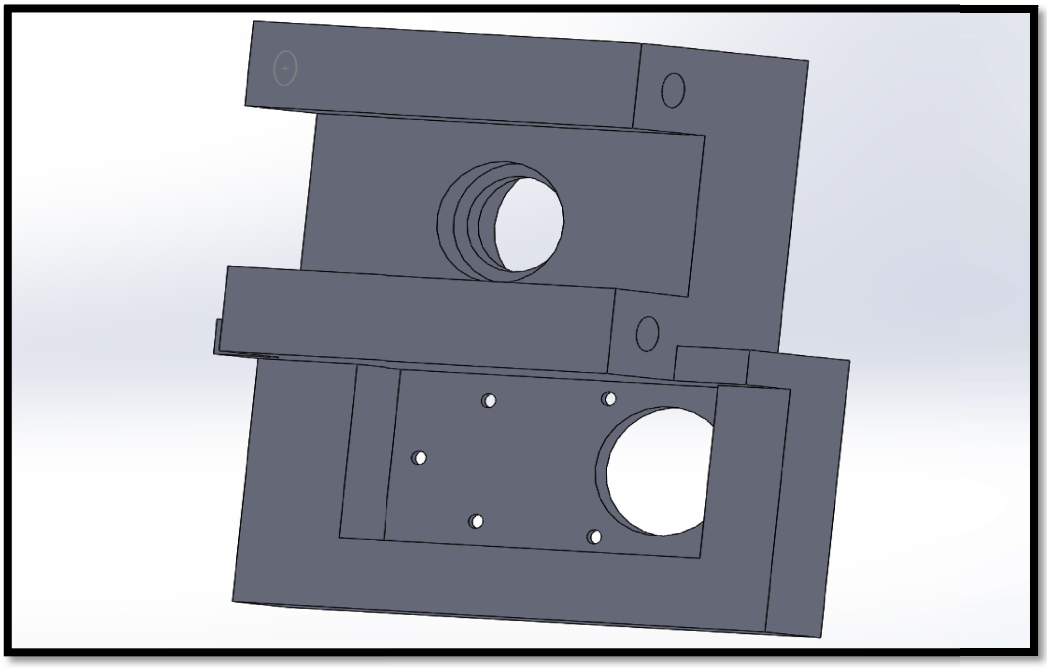
Η τεχνική κατασκευής των βάσεων ήταν αυτή της τρισδιάστατης εκτύπωσης (3D Printing). Και εδώ υπήρχαν ζητήματα εξοικείωσης μιας και για να ολοκληρωθεί μία επιτυχημένη εκτύπωση, η εμπειρία, η γνώση των ανοχών του εκτυπωτή καθώς και η καταλληλότητα των υλικών ήταν μέγιστης σημασίας.

Συγκεκριμένα αντικαταστάθηκαν 3 βάσεις , απο τις οποίες η μία εξ αυτών φιλοξενεί δύο servo motors. Ιδιαίτερη δυσκολία προέκυψε στην πρώτη βάση(βάση διασύνδεσης βραχίονα με το Rov) λόγω των αυξημένων ροπών που προέκυψαν στο σύστημα της βάσης του βραχίονα απο την προσθήκη του τελικού στοιχείου δράσης. Ο προηγούμενος σχεδιαστής έχοντας προσαρμόσει τη συγκεκριμένη βάση στις εκάστοτε μηχανικές ανοχές δεν είχε υπολογίσει μια μελλοντική προσθήκη ενός εξελιγμένου gripper, το οποίο θα προσέδιδε επιπρόσθετη ροπή στο σύστημα της βάσης του manipulator. Αυτή η παράλειψη οδήγησε στην ανεπάρκεια και τελικά την καταστροφή της υπάρχουσας βάσης.

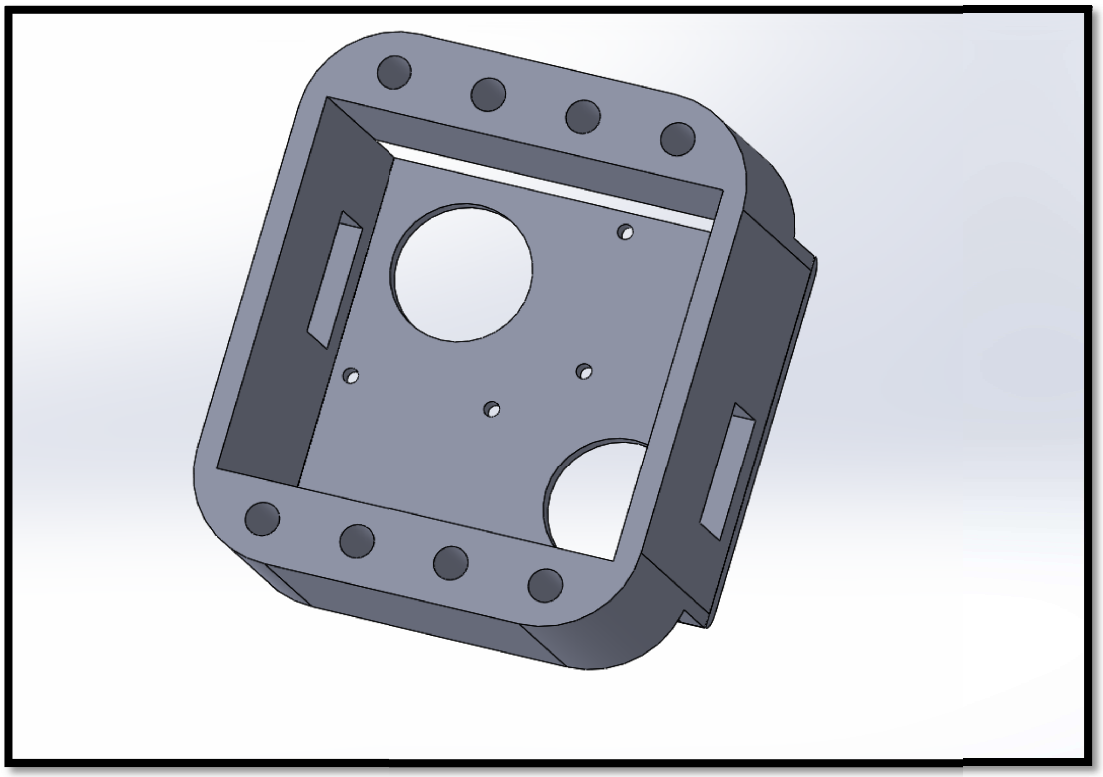
Ο σχεδιασμός των νέων βάσεων είχε ως απαίτηση να διατηρηθεί η γεωμετρία του βραχίονα με σκοπό να μην αλλάξει η υπάρχουσα κινηματική αλυσίδα. Παρακάτω βλέπει κανείς τις βάσεις όπως σχεδιάστηκαν στο πρόγραμμα SolidWorks.



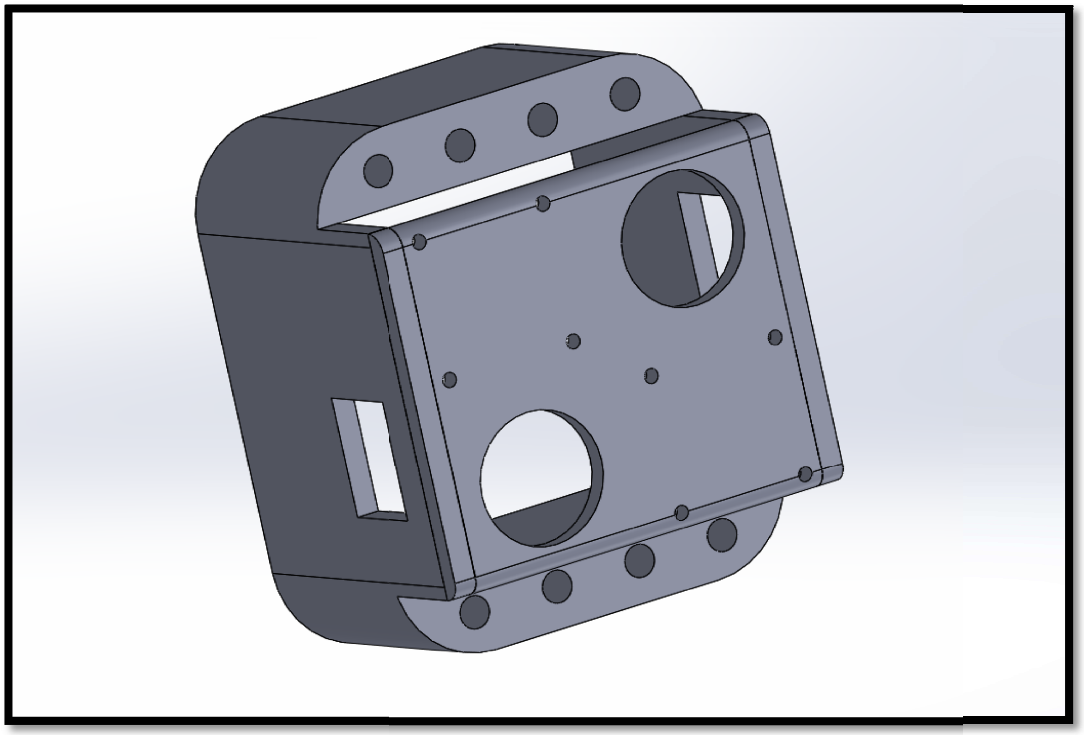
Εικόνα 3.1.2 *– Βάση σερβοκινητήρα, Νο1 (η αρίθμηση αρχίζει απο το πλαίσιο της βάσης και καταλήγει στο τελικό στοιχείο δράσης)*



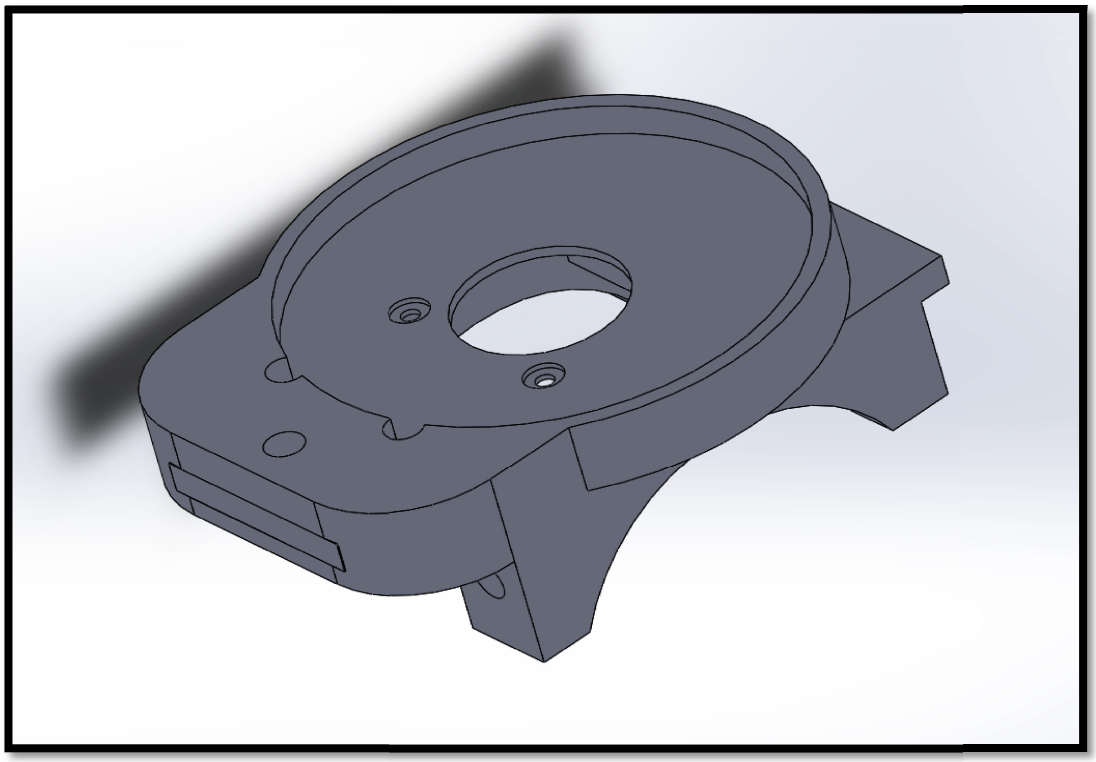
Εικόνα 3.1.3 *–Πίσω όψη βάσης σερβοκινητήρα, No1*



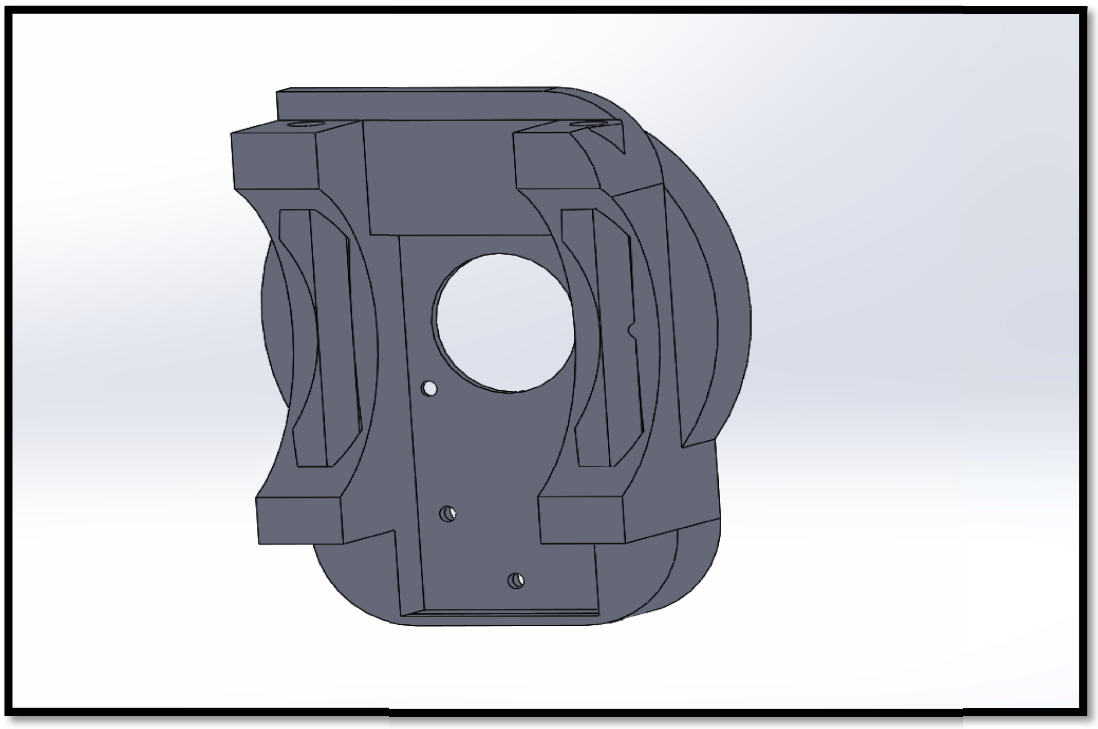
Εικόνα 3.1.4 *–Βάση(διπλή) σερβοκινητήρων, No2*



Εικόνα 3.1.5 *–Πίσω όψη βάσης σερβοκινητήρων, No2*



Εικόνα 3.1.6 *–Βάση (διασύνδεσης με gripper) σερβοκινητήρα, No3*



Εικόνα 3.1.7 *–Πίσω όψη βάσης σερβοκινητήρα Νο3*

Ο σχεδιασμός ύστερα απο τοποθέτηση και μέτρηση διαστάσεων κρίθηκε αρκετά ικανοποιητικός διατηρώντας το αρχικό μήκος του βραχίονα αμετάβλητο. Επιπροσθέτως, οι συνδέσεις πάνω στην υπάρχουσα διάταξη έγιναν αβίαστα πράγμα που επαληθεύει την καλή προσέγγιση του σχεδιασμού. Όσον αφορά την ενίσχυση της πρώτης βάσης για να αντέξει το επιπρόσθετο φορτίο και εδώ ύστερα απο αρκετές δοκιμές δεν παρουσιάστηκε καμία επιπλοκή. Η διαδικασία που ακολουθήθηκε ήταν με τη σειρά, σχεδιασμός στο γραφικό περιβάλλον του Solidworks και export του αρχείου σε μορφή STL. Από εκεί , το αρχείο εισάγεται στο πρόγραμμα CURA για να μετατραπεί σε μορφή που μπορεί να γίνει αντιληπτή απο τον 3D printer και συγκεκριμένα μετατροπή απο STL σε gcode. Τέλος να αναφέρω ότι το υλικό που χρησιμοποιήθηκε ήταν PLA, ένα υλικό που είχε εξαιρετικά αποτελέσματα τόσο στην ακρίβεια της εκτύπωσης, όσο και στην ανοχή του σε μηχανικές καταπονήσεις.

3.2 Housing ηλεκτρονικών διατάξεων

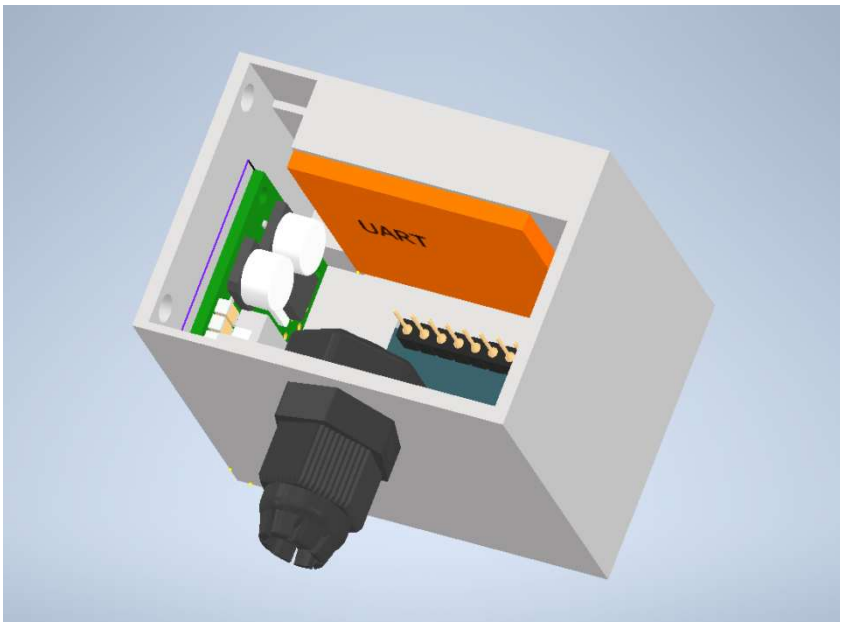
Πέραν του σχεδιασμού των κατάλληλων βάσεων, η κατασκευή κατάλληλου χώρου που θα εσωκλείει όλα τα ηλεκτρονικά (θα δοθεί εκτενής αναφορά για αυτά σε επόμενο κεφάλαιο), ήταν καθοριστικής σημασίας. Για την κατασκευή τέτοιου housing η τεχνική 3D printing απορρίφθηκε καθώς δεν παρέχει καμία στεγανότητα. Η επόμενη σκέψη ήταν η αναζήτηση στην αγορά και κάποιο έτοιμο προιόν που θα παρέχει στεγανότητα IP68 και θα μπορεί να φιλοξενεί τα παρακάτω:

-arduino pro micro

-ολοκληρωμένο κυκλωματικό ενισχυτή πτώσης τάσης

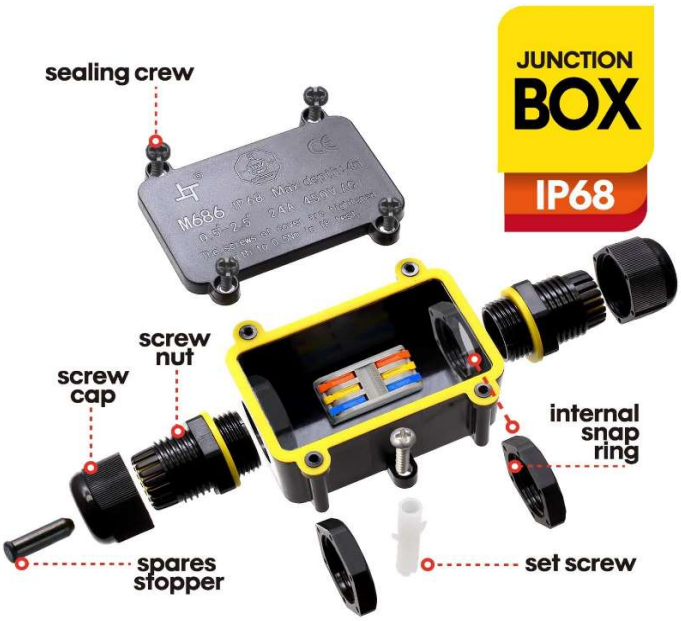
-UART interface

Ποιοτικά ο χώρος που χρειάζονται όλες αυτές οι ηλεκτρονικές διατάξεις φαίνεται στην Εικόνα 3.2.1. Σχεδιάστηκε μία γρήγορη προσομοίωση στο περιβάλλον του Solidworks για την αναγκαία διαστασιολόγηση και κρίθηκε ότι ένα κουτί διαστάσεων 42*37*55mm θα ήταν υπέραρκετο για να εξυπηρετήσει τον σκοπό μας.



Εικόνα 3.2.1- *Διαστασιολόγηση για την επιλογή κατάλληλου housing, διαστάσεις 42*37*55mm*

Ύστερα απο εκτενή αναζήτηση τέτοιου αδιάβροχου housing με τις παραπάνω ελάχιστες διαστάσεις, καταλήξαμε σε μία λύση της εταιρείας Peba. Πρόκειται για ένα αδιάβροχο κουτί ηλεκτρολογικού υλικού, με δύο στυπιοθλίπτες, παρεχόμενης στεγανότητας επιπέδου IP68. Το εν λόγω προιόν φαίνεται στην Εικόνα 3.2.2.



Εικόνα 3.2.2- *Αδιάβροχο Housing της εταιρείας PEBA, με IP68*

Η τοποθέτηση του έγινε στη βάση του βραχίονα διασφαλίζοντας έτσι ότι δεν θα προσθέσουμε επιπλέον ροπές στο τελικό στοιχείο δράσης, το οποίο έτσι κι αλλιώς βρίσκεται στα όρια της μηχανικής αντοχής που μπορεί να αντέξει ο βραχίονας.

3.3 Οπτικό μέσο

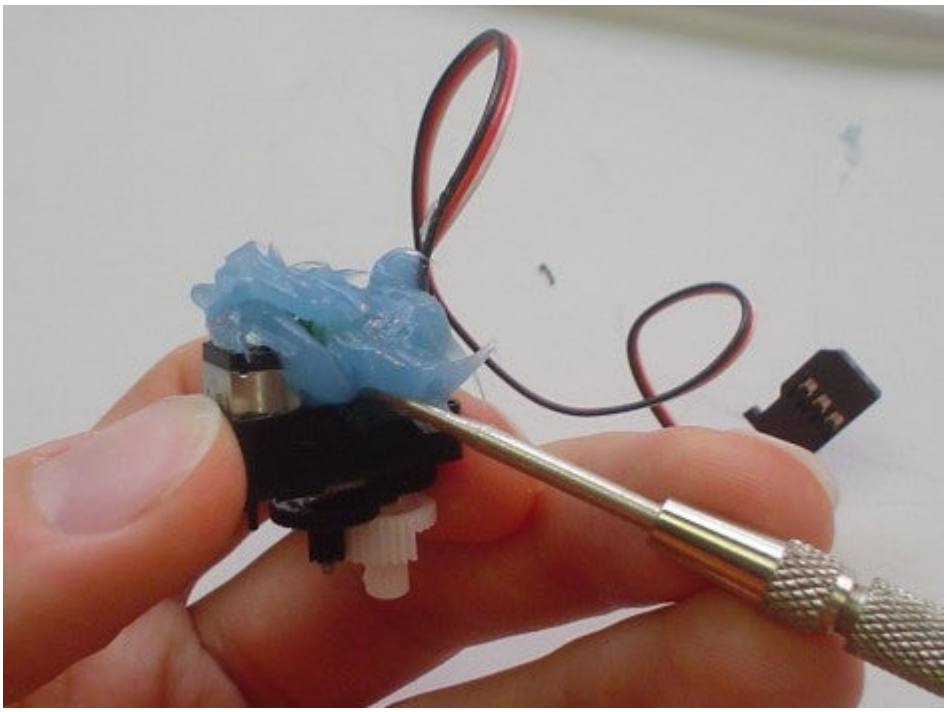
Αν και προηγούμενος φοιτητής είχε ασχοληθεί με την δημιουργία αδιάβροχων καμερών, αυτές κρίθηκαν ακατάλληλες τόσο σε επίπεδα ανάλυσης και στεγανότητας όσο και στο υπερβολικό βάρος που προσέδιδαν στη διάταξη. Η αναζήτηση μας περιορίστηκε στην εύρεση μίας κάμερας η οποία να συνεργάζεται με το ROS, να είναι αδιάβροχη και να έχει υψηλή ανάλυση. Δυστυχώς κάμερες που να πληρούν τα παραπάνω κριτήρια είναι πολύ περιορισμένες στην αγορά. Κατέληξα στην χρήση μίας ιδιαίτερης κατηγορίας καμερών ενδοσκοπικού τύπου. Αν και υπήρχαν αμφιβολίες για την ανάλυση που μπορεί να παρέχουν αυτές οι εξαιρετικά μικρές κάμερες, τα αποτελέσματα ήταν περισσότερο απο ικανοποιητικά. Παρακάτω παραθέτω την εν λόγω κάμερα με κωδικό μοντέλου AN99.

Εικόνα 3.3.1- *Ενδοσκοπική κάμερα, IP67, κωδικός μοντέλου AN99*

3.4 Αδιαβροχοποίηση των σερβοκινητήρων

Αν και τα σέρβο που επιλέχθηκαν για αυτό το εγχείρημα, κάλυπταν όλες τις ανάγκες μας τόσο σε μηχανική αλλά και προγραμματιστική υποστήριξη, δυστυχώς δεν ήταν αδιάβροχα. Αυτό αποτέλεσε σοβαρή πρόκληση μιας και οποιαδήποτε αστοχία, θα οδηγούσε στην καταστροφή τους. Η μέθοδος που ακολούθησα για αυτό το σκοπό ήταν ένας συνδυασμός από γράσο ναυτικού τύπου και υγρό καουτσούκ (Plastidip).

Ως γνωστόν τα σέρβο αποτελούνται από δύο χώρους, ο ένας περιλαμβάνει τα μηχανικά μέρη και ο δεύτερος τα ηλεκτρονικά και τον controller. Στο πρώτο τμήμα με τα μηχανικά στοιχεία, ο χώρος πληρώθηκε με γράσο ναυτικού τύπου. Ο λόγος είναι η αντιστάθμιση της πίεσης που προκαλείται λόγω της ύπαρξης του νερού εξωτερικά και του αέρα εσωτερικά της διάταξης. Επίσης με αυτή την τεχνική προστατεύονται τα μέρη από την πρόκληση σκουριάς και ελαχιστοποιούνται οι τριβές που προκαλούνται κατά την λειτουργία, παρέχοντας μεγαλύτερο χρόνο ζωής.



Εικόνα 3.4.1- *Μόνωση του μηχανικού τμήματος του σέρβο με χρήση Marine grease*

Το πιο σημαντικό τμήμα που έπρεπε να προστατευθεί όμως, ήταν ο χώρος που εσωκλείει τα ηλεκτρονικά. Και εδώ θα μπορούσα να εφαρμόσω την τεχνική του γράσου, όμως τα τελευταία χρόνια παράγεται ένα πολύ έξυπνο υλικό που αποφέρει πολύ καλύτερα αποτελέσματα σε εφαρμογές νερού. Πρόκειται για το προιόν Plastidip, αλλιώς υγρό καουτσούκ. Έρχεται σε υγρή μορφή ή σπρέι και όταν στεγνώσει δίνει την γνωστή αίσθηση του καουτσούκ. Με αυτό το υλικό καλύφθηκαν όλες οι ηλεκτρονικές διατάξεις και το εξωτερικό κέλυφος του σερβοκινητήρα, προσδίδοντας εξαιρετικά επίπεδα στεγανότητας. Συγκεκριμένα καθόλα την διάρκεια των πειραμάτων στο χώρο της πισίνας, δεν παρουσιάστηκε καμία επιπλοκή στη λειτουργία τους.

Εικόνα 3.4.2- *Αδιαβροχοποίηση των ηλεκτρονικών μερών και του εξωτερικού κελύφους με υγρό καουτσούκ*

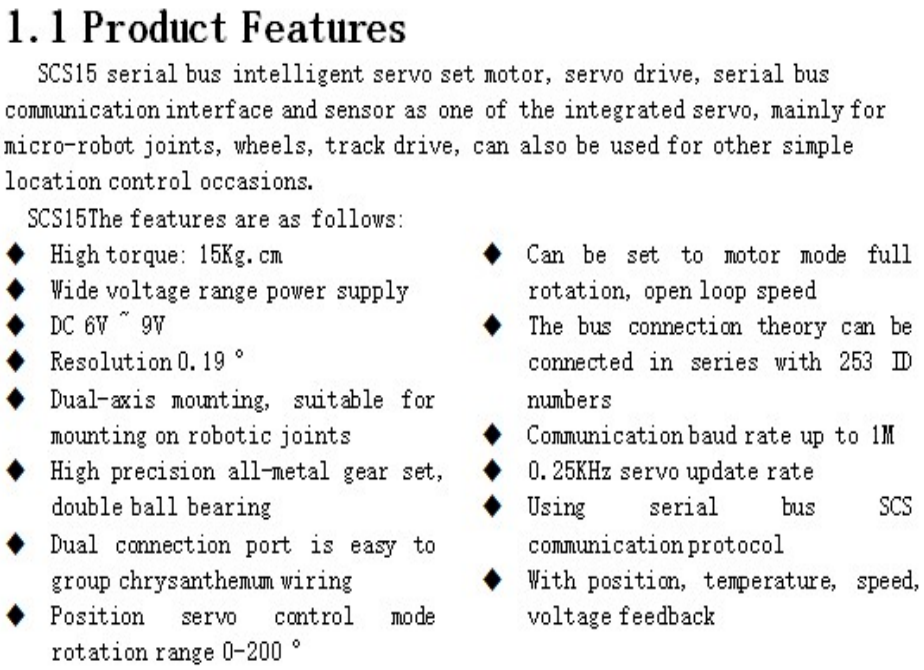
4 Ηλεκτρολογικό Υλικό & Λογισμικό Ελέγχου

Το κεφάλαιο αυτό χωρίζεται σε δύο τμήματα: α) την αναλυτική περιγραφή των ηλεκτρικών/ηλεκτρονικών μερών β) το λογισμικό λειτουργίας και ελέγχου του μηχανισμού.

4.1 Ηλεκτρικά/Ηλεκτρονικά μέρη

1. SCS15 by Feetech

Οι νέοι κινητήρες που επιλέχθηκαν είναι της εταιρείας Feetech τύπου SCS15. Πρόκειται για έναν ψηφιακό σερβοκινητήρα ικανό να παρέχει πληροφορία σχετική με την θέση, την ταχύτητα, την θερμοκρασία καθώς και την τάση. Παρακάτω παρουσιάζονται συνοπτικά τα χαρακτηρηστικά τους.



1.1 Product Features

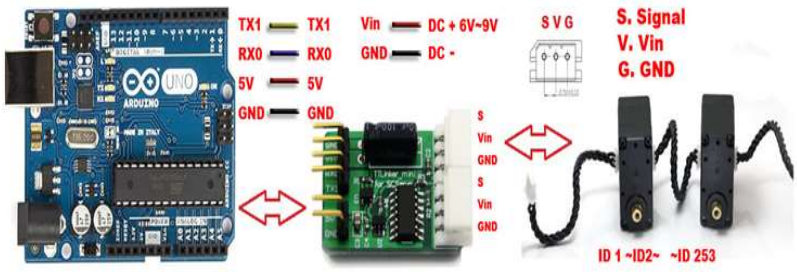
SCS15 serial bus intelligent servo set motor, servo drive, serial bus communication interface and sensor as one of the integrated servo, mainly for micro-robot joints, wheels, track drive, can also be used for other simple location control occasions.

SCS15The features are as follows:

- High torque: 15Kg.cm
- Wide voltage range power supply
- DC 6V ~ 9V
- Resolution 0.19 °
- Dual-axis mounting, suitable for mounting on robotic joints
- High precision all-metal gear set, double ball bearing
- Dual connection port is easy to group chrysanthemum wiring
- Position servo control mode rotation range 0-200 °
- Can be set to motor mode full rotation, open loop speed
- The bus connection theory can be connected in series with 253 ID numbers
- Communication baud rate up to 1M
- 0.25KHz servo update rate
- Using serial bus SCS communication protocol
- With position, temperature, speed, voltage feedback

Εικόνα 4.1.1 *Γενικά χαρακτηριστικά των σερβοκινητήρων SCS15*

Ο συγκεκριμένος σερβοκινητήρας έρχεται με μία UART interface όπως φαίνεται παρακάτω για επικοινωνία με τον μικρουπολιστή. Η απαιτούμενη διασύνδεση είναι απόλυτα ξεκάθαρη μέσω της ακόλουθης Εικόνας 4.1.2 . Συγκεκριμένα αυτά τα servo απαιτούν για την πλήρη διασύνδεση τους με τον μικρουπολογιστή την χρήση των pin Tx/Rx για την σειριακή τους επικοινωνία και οποιοδήποτε ground και 5V pins για την τροφοδοσία.



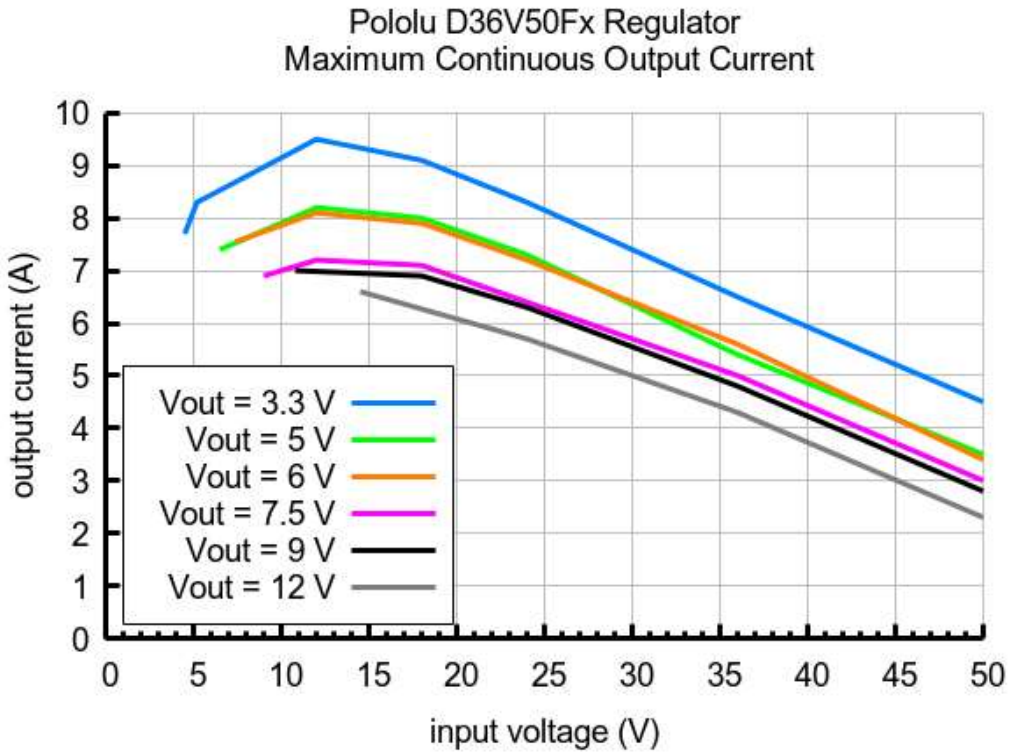
Εικόνα 4.1.2*-Τρόπος Διασύνδεσης servos με το arduino*

2. D36V50F7 POLOLU

Για το κομμάτι της **τροφοδοσίας** των σερβοκινητήρων , η απαίτηση είναι τάση 6-9 volt. Μέσω της μοναδικής καλωδίωσης μεταξύ χρήστη και υποβρύχιας διάταξης και συγκεκριμένα μέσω τροφοδοτικού ηλεκτρονικού υπολογιστή στα +12 volt, γίνεται αντιληπτό ότι θα χρειαστεί κάποιο DC-DC voltage regulator απο τα 12 στα 7.5 volt (για να είμαστε εντός ασφαλών ορίων λειτουργίας). Η παράλειψη μίας τέτοιας προσθήκης θα ήταν καταστροφική για τους επενεργητές.

'Υστερα απο σχετική αναζήτηση ενός τέτοιου voltage regulator, κατέληξα στην επιλογή του D36V50F7 POLOLU στα 23 δολάρια. Αυτό το ολοκληρωμένο δέχεται εισόδους τάσης έως 50V και δίνει την ζητούμενη έξοδο των 7.5V. Επιπροσθέτως πρέπει να λάβουμε υπόψιν τα 4 servo της διάταξης μας με μέγιστες ανάγκες σε ρεύμα συνολικά κοντά στα 6 A. Στην παρακάτω γραφική φαίνεται οτι για μία είσοδο της τάξης των 12volt (για την επιλογή του ολοκληρωμένου με τάση εξόδου στα 7 volt) το ρεύμα αντιστοιχεί στα 7 A και υπερκαλύπτει τις ανάγκες μας.

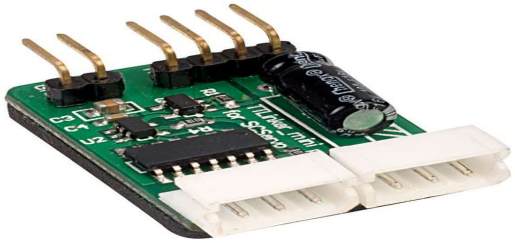
Εικόνα 4.1.3 *– DC-DC voltage regulator D36V50F9 PoLoLu*



Εικόνα 4.1.4 *– γραφική τάσης εισόδου και ρεύματος εξόδου για την οικογένεια των ολοκληρωμένων D36V50Fx με τυποποιημένες εξόδους από 3,3V έως 12V (εμείς έχουμε επιλέξει Vout 7.5V).*

3. Uart Interface-TTLinker

Για την επικοινωνία μεταξύ arduino και σερβοκινητήρων παρεμβάλλεται αυτή η μικρή μονάδα μετασχηματισμού σημάτων. Η συγκεκριμένη διάταξη δεν κάνει τίποτα περισσότερο απο το να μετατρέπει τα UART σήματα του arduino σε σήματα τύπου half duplex τα οποία χρησιμοποιούν τα servo SCS15.

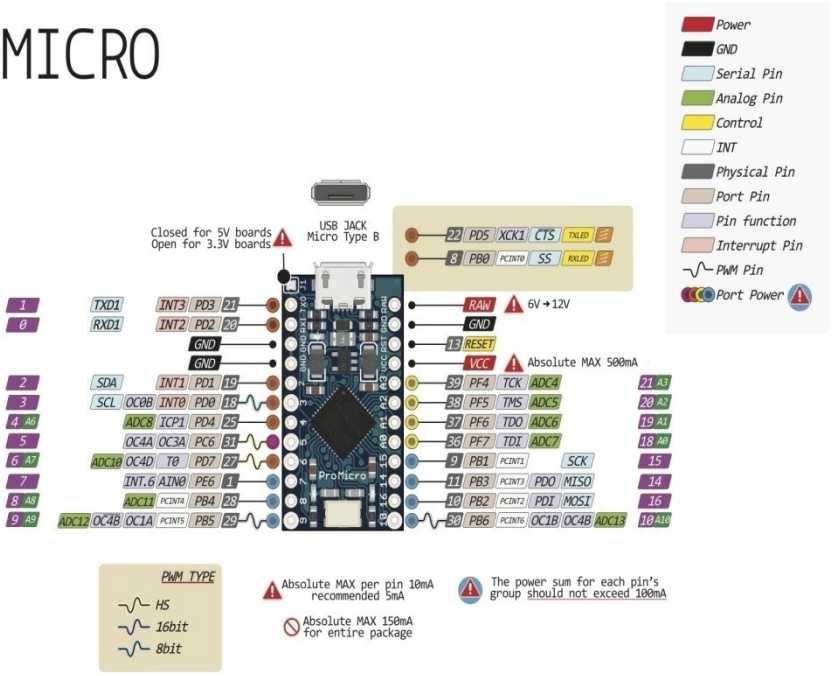


Εικόνα 4.1.5 *– UART interface, TTLINKER by Feetech*

4. Arduino pro micro

Η μονάδα ελέγχου είναι μια μικρών διαστάσεων πλακέτα ανάπτυξης Arduino βασισμένη στο μικροελεγκτή AtmelATmega32U4. Αν και δεν αποτελεί επίσημη υλοποίηση[2] της Arduino, είναι ιδιαίτερα διαδεδομένη από τρίτους κατασκευαστές. Ο μικροελεγκτής (εφεξής MCU) είναι μια αναβάθμιση του «κλασικού» ATmega328P (ArduinoUnoκ.α.), καθώς ενσωματώνει μονάδα USB (v. 2.0, fullspeed) πέραν της τυπικής σειριακής μονάδας. Η επικοινωνία με τον εξωτερικό Η/Υ μέσω θύρας USB γίνεται απ' ευθείας με τονMCU, αναιρώντας την ανάγκη χρήσης κάποιου επιπλέον ολοκληρωμένου στην πλακέτα.

Εν γένει, είναι ένας MCU που λειτουργεί σε λογική τάση 5V. Ενσωματώνει 32KB μνήμης Flash (μνήμη προγραμμάτων), 2KBSRAM (προσωρινή μνήμη) και 1KBEEPROMγια μόνιμη αποθήκευση δεδομένων. Διαθέτει μονάδα Analog-DigitalConverter (εφεξής ADC)ακρίβειας 10-bit (9 κανάλια), τέσσερις μονάδες Timer/Counter (δύο εκ των οποίων ακρίβειας 16-bit), με πολλαπλά κανάλια σήματος PWM, μονάδα αναλογικού συγκριτή κ.α.



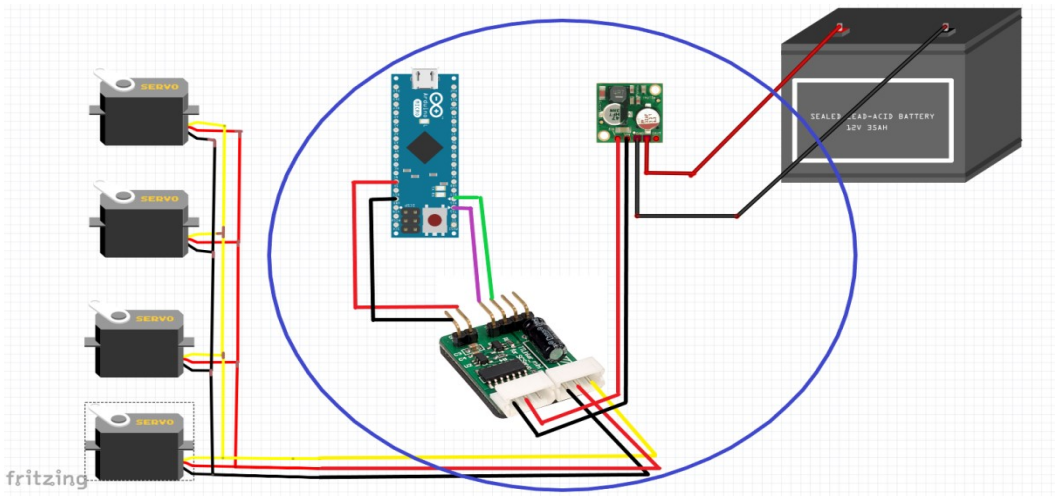
Εικόνα 4.1.6 *- Απεικόνιση της πλακέτας Pro Micro (ATmega32u4 MCU). Σημειώνονται οι βασικές και εναλλακτικές λειτουργίες των διαθέσιμων θυρών (επεξήγηση στο υπόμνημα στο δεξί μέρος). Επίσης, σημειώνονται οι μέγιστες επιτρεπτές τιμές ρεύματος των θυρών.*

Οι δυνατότητες της εν λόγω πλακέτας είναι πάνω-κάτω παρόμοιες με των υπολοίπων σε αυτή την κατηγορία και κρίνονται επαρκείς για την κάλυψη των αναγκών μας, οι οποίες συνοψίζονται ως εξής:

- TX(Serial1) κανάλι για την σύνδεση με το αντίστοιχο TX του TTLikner
- RX(Serial1) κανάλι για την σύνδεση με το αντίστοιχο RX του TTlinker
- 1 κανάλι τάσης 5V για την σύνδεση με το αντίστοιχο του TTlinker
- 1 κανάλι Ground για την σύνδεση με το αντίστοιχο του TTlinker
- Usb διασύνδεση (Serial) για την σειριακή επικοινωνία με τον H/Y
-

4.2 Διασυνδέσεις εσωτερικών στοιχείων

ι



Εικόνα 4.2.1 *Εσωτερικές διασυνδέσεις όπως είναι εγκατεστημένες μέσα στο αδιάβροχο Housing*

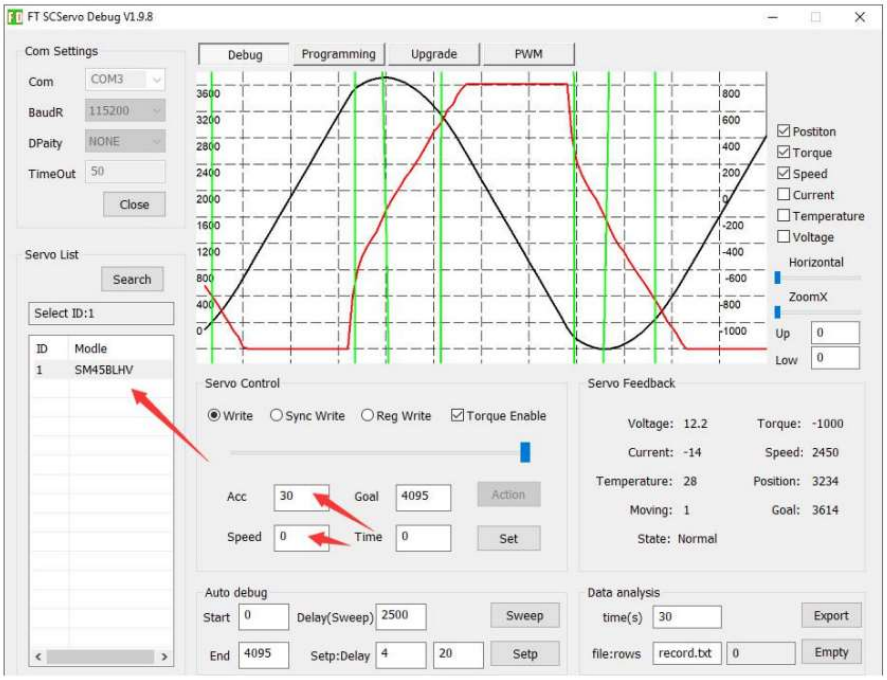
Στην παραπάνω Εικόνα 4.2.1 μπορεί κανείς να δει (εντός του κύκλου) όλες τις ηλεκτρονικές διασυνδέσεις που βρίσκονται εντός του αδιάβροχου housing. Μέσα στον αδιάβροχο χώρο καταφθάνουν (από δεξιά,μέσου του ενός στυπιοθλήπτη) 2 usb και τροφοδοσία 12 volt (καλώδιο 10 αγωγών). Το ένα εκ των δύο usb τροφοδοτεί το arduino που είναι υπεύθυνο για τα servo και το δεύτερο περνάει εντός του housing για να καταλήξει στο arduino του gripper. Ο λόγος για τον οποίο συνέβη αυτό, είναι εν μέρη αισθητικός αλλά και για την μείωση των καλωδίων (θέλαμε μόνο ένα καλώδιο να φθάνει στην διάταξη του Manipulator απο τον H/Y). Για το κομμάτι της τροφοδοσίας τα 12 volt φθάνουν στο ολοκληρωμένο κύκλωμα πτώσης τάσης ωστέ να επιτύχουμε τα 7.5 volt που είναι και η τάση καλής λειτουργίας των σέρβο. Με διακλάδωση εντός του housing η γραμμή των 12 volt συνεχίζει για να τροφοδοτήσει

τον κινητήρα του gripper, ο οποίος δουλεύει στα 12 v. Τελικά από την αριστερή πλευρά του housing εξέρχεται ένα καλώδιο 6 αγωγών (4 usb+2 τροφοδοσία) που καταλήγει στο τελικό στοιχείο δράσης. Όσον αφορά τον τρόπο με τον οποίο διασυνδέονται τα ηλεκτρονικά μέρη του Gripper καθώς και για τις εξωτερικές συνδέσεις της διάταξης, πρέπει κανείς να ανατρέξει στην διπλωματική εργασία που υπάρχει στο σχετικό usb και συγγραφέας της οποίας είναι ο κ.Μένεγας.

4.3 Προγραμματισμός των σερβοκινητήρων

Οι SCS15 σερβοκινητήρες προγραμματίζονται εύκολα μέσω του προγράμματος FD SCServo Debug V1.9.8 που παρέχει η εταιρεία . Μέσω αυτού του api μπορεί κανείς εύκολα να αλλάξει το ID, baud rate, minimum-maximum angle και γενικά οποιαδήποτε παράμετρο επιθυμεί . Το πρόγραμμα αυτό υποστηρίζεται μέσω των Windows. Για τον προγραμματισμό τους η συνδεσμολογία που ακολουθείται είναι πολύ συγκεκριμένη και φαίνεται στην ακόλουθη Εικόνα 4.3.1 . Απαραίτητη προυπόθεση η πλακέτα προγραμματισμού SCM-1 της εταιρείας Feetech. Επίσης παραθέτω μία απεικόνιση του πως μοιάζει το εν λόγω προγραμματιστικό περιβάλλον. Αναλυτικές οδηγίες για το πως μπορεί κανείς να χρησιμοποιήσει αυτό το interface δίνονται στο έντυπο που παρέχω στο usb που θα παραδωθεί μαζί με την παρούσα διπλωματική.

Εικόνα 4.3.1 *Διασύνδεση servo με SCM-1 και H/Y για διεργασίες debugging μέσω του προγράμματος FD V1.9.8*



Εικόνα 4.3.2 *Απεικόνιση προγραμματιστικού περιβάλλοντος FD 1.9.8*

Σε γενικές γραμμές η χρήση είναι σχετικά απλή. Ο χρήστης αφού πραγματοποιήσει την παραπάνω συνδεσμολογία ανοίγει το FD. Στην επιλογή που βρίσκεται πάνω αριστερά στο πρόγραμμα επιλέγουμε στο BaudR την τιμή 1000000, καθώς αυτό το baudrate κρίθηκε κατάλληλο για γρήγογρη μεταφορά δεδομένων, χωρίς να χάνονται καθόλου δεδομένα κατά την διεκπεραίωση διεργασιών ελέγχου. Ακολουθούμε τα παρακάτω βήματα για τον εντοπισμό του ζητούμενου σέρβο (κάθε φορά συνδεδεμένο πρέπει να έχουμε MONO ένα servo) και για να επέμβουμε σε αυτό.

a) baudR=1000000

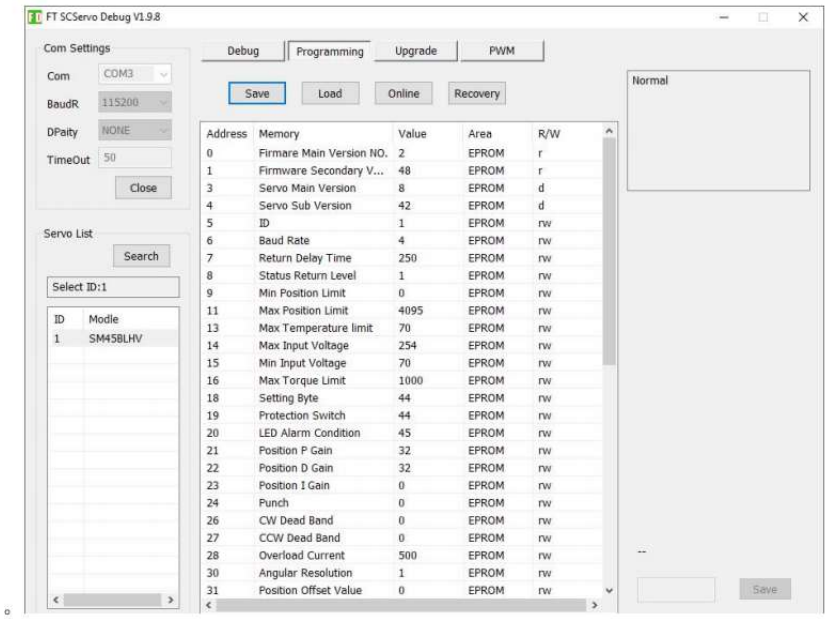
b) enter close

c) enter search (εδώ θα δούμε στην οθόνη μας να εμφανίζεται το ζητούμενο servo)

d) αφού το επιλέξουμε , ακολουθούμε τον σύνδεσμο programming που βρίσκεται στο κέντρο της οθόνης μας

e) στην νέα καρτέλα , όπως φαίνεται παρακάτω ,επιλέγουμε την μεταβολή που θέλουμε να πραγματοποιήσουμε και την αποθηκεύουμε

Το συγκεκριμένο πρόγραμμα για την παρούσα διπλωματική φάνηκε χρήσιμο μόνο για την αλλαγή των id και του baudrate. Καμία άλλη παρέμβαση δεν χρειάστηκε στον προγραμματισμό τους και αξιώνω πως ο μελλοντικός χρήστης του βραχίονα δεν θα χρειαστεί καθόλου να το χρησιμοποιήσει . Σε κάθε περίπτωση το αναλυτικό tutorial επισυνάπτεται στα έγγραφα για παν ενδεχόμενο.



Εικόνα 4.3.3 *Απεικόνιση προγραμματιστικού περιβάλλοντος FD 1.9.8*

4.4 Λογισμικό Λειτουργίας και ελέγχου

Ο κώδικας χαμηλού επιπέδου που τρέχει μέσα στην μονάδα του arduino για τον έλεγχο των κινητήρων φαίνεται παρακάτω. Στο υποκεφάλαιο αυτό θα προσπαθήσω να εξηγήσω αναλυτικά γραμμή –γραμμή την φιλοσοφία πίσω από κάθετί αποτελεί κομβικής σημασίας για την κατανόηση του αναγνώστη. Να αναφέρω ότι πολύ βασικό στοιχείο για να καταφέρω να πραγματοποιήσω αυτό τον προγραμματισμό είναι η ύπαρξη έτοιμων βιβλιοθηκών που παρέχει η εταιρεία για να υποστηρίξουν τη χρήση των servo στο περιβάλλον του arduino(SCServo). Αυτές οι βιβλιοθήκες παρέχουν έτοιμες συναρτήσεις για την ανάγνωση της θέσης,ταχύτητας,φορτίου και θερμοκρασίας , καθώς και συναρτήσεις για command position ,command velocity. Η βιβλιοθήκη παρέχεται στο usb που θα παραδώσω και κανείς θα πρέπει να την εγκαταστήσει στον υπολογιστή του και συγκεκριμένα μέσα στα libraries του arduino για να καταφέρει να τρέξει τον παρακάτω κώδικα. Επίσης σε αυτο το usb θα βρεί κανείς τόσο τον κώδικα για τον κόκκινο όσο και για τον κίτρινο manipulator. Οι κώδικες παρουσιάζουν μικρές αλλαγές στον τρόπο ανάγνωσης και καταχώρησης της ταχύτητας λόγω των διαφορετικών firmware που διαθέτουν κάποιοι εκ των σερβοκινητήρων (φυσικά αυτό δεν απασχολεί τον αναγνώστη διότι αποτελεί προγραμματισμό χαμηλού επιπέδου).

```
#define USE_USBCON  //we include this because we use arduino LEONARDO
#if (ARDUINO >= 100)
#include <Arduino.h>
#else
#include <WProgram.h>
#endif


//*******  HEADERS TO RUN SERVOS SCS15**********//



#include <SCServo.h>
#include <SCSProtocol.h>



//**********ROS DEMANDS AND SETTING UP COMMUNICATION WITH ROS NODES***********//


#include <ros.h>                                  // in every part of ros communica-
tion we must include the ros header file
#include <sensor_msgs/JointState.h>               // header files for the gia ton Pub-
lisher
#include<std_msgs/Float32MultiArray.h>       //header files for the ton Subscriber


ros::NodeHandle_<ArduinoHardware, 1, 1, 250, 250> nh; //THIS allows us to create
publishers and subscribers and also takes care for communications




//************VARIABLES AND INITIALIZATION***************//

SCServo SERVO;  //  we create the item SERVO
int NUM_MOTORS = 4;


int posJoint;
int velJoint1;
int velJoint2;
int velJoint3;
int velJoint4;
int effJoint;
int posJoint4;
int effJoint4;

float vel[] = {0, 0, 0 , 0};
float pos[] = {0, 0, 0 , 0};
float eff[] = {0, 0, 0 , 0};

char *robot_id = "a";
char *joint_name[4] = { "joint_01", "joint_02", "joint_03", "joint_04"};


float MODE = 0.0;
float command_Joint1 = 0.0;
float command_Joint2 = 0.0;
float command_Joint3 = 0.0;
float command_Joint4 = 0.0;

//we define our CallBack function that is used from our Subscriber

void ServoControlCallback(const std_msgs::Float32MultiArray& msg)
{
  MODE = msg.data[0];
  command_Joint1 = msg.data[1];
  command_Joint2 = msg.data[2];
  command_Joint3 = msg.data[3];
  command_Joint4 = msg.data[4];
}
```

```
// here we initialize our Publisher and Subscriber, which publishe messages of
type JointState on topic named "lbv150_uvms/jointstates", and subscribe messages
of type Float32MultiArray on topic with name "joints_cmds_topic"

sensor_msgs::JointState joint_state_msg;
ros::Publisher pub("lbv150_uvms/joint_states", &joint_state_msg);  //topic:
lbv150_uvms/joint_states , minimata: joint_state_msg
ros::Subscriber <std_msgs::Float32MultiArray> sub("joints_cmds_topic", ServoCon-
trolCallback);




//**************SETUP FUNCTION*******************//

void setup()  //put your setup code here to run once
{

  //***DEMANDS FOR ROS COMMUNICATION THAT MUST EXIST ON SETUP FUNCTION***//

  nh.getHardware()->setBaud(115200);
  nh.initNode();
  nh.advertise(pub);
  nh.subscribe(sub);
  delay(1000);


  Serial1.begin(1000000);  //inicialize the  communication betwenn arduino and
servos SCS15  at 1 Mbps
  SERVO.pSerial = &Serial1;  //defining that the communication between servos and
arduino takes place through Serial1 port
  delay(1000);

  //HOME POSITION initialization

  SERVO.EnableTorque(1, 1);
  SERVO.EnableTorque(2, 1);
  SERVO.EnableTorque(3, 1);
  SERVO.EnableTorque(4, 1);
  // We initialize the starting position of the arm as it is introduced in DH con-
figuration
  SERVO.WritePos(1, 512, 2000);    //512 steps are in 0 degrees configuration
  SERVO.WritePos(2, 512, 2000);
  SERVO.WritePos(3, 512, 2000);
  SERVO.WritePos(4, 512, 2000);




  joint_state_msg.name_length = NUM_MOTORS;
  joint_state_msg.position_length = NUM_MOTORS;
  joint_state_msg.velocity_length = NUM_MOTORS;
  joint_state_msg.effort_length = NUM_MOTORS;
  joint_state_msg.header.frame_id = robot_id;
  joint_state_msg.name = joint_name;


}


//**************VOID FUNCTION*******************//


void loop() //put your main code here to run repeatedly
{
  for (int id = 0; id < 3; id++) //Fulfill the arrays with motor readings
  {
    posJoint = posToDegs(SERVO.ReadPos(id + 1));
    nh.spinOnce();

    pos[id] = posJoint;

    effJoint = SERVO.ReadLoad(id + 1);
    eff[id] =   effJoint;
```

```
  }

  posJoint4 = 2 * posToDegs(SERVO.ReadPos(4));
  pos[3] = posJoint4;

  effJoint4 = SERVO.ReadLoad(4);
  eff[3] =   effJoint4;

  // deg/s measurements of joint rotation

  velJoint1 = 0.5 * 0.19 * SERVO.ReadSpe(1); //joint1 runs 85degs in 1 sec in
maximum speed (1:2 transmission and thats why the 0.5 exists in the equation)
  vel[0] = velJoint1;
  velJoint2 =  0.5 * 0.19 * SERVO.ReadSpe(2);
  vel[1] = velJoint2;
  velJoint3 = 0.5 * 0.19 * SERVO.ReadSpe(3);
  vel[2] = velJoint3;
  velJoint4 = 0.19 * SERVO.ReadSpe(4); //joint4 runs 170degs maximum speed in 1
sec cause of 1:1
  vel[3] = velJoint4;


  //fulfill the sensor_msg/JointState msg

  joint_state_msg.position = pos;
  joint_state_msg.velocity = vel;
  joint_state_msg.effort = eff;
  joint_state_msg.header.stamp = nh.now();
  pub.publish(&joint_state_msg);

  nh.spinOnce();
  delay(5);

  if (MODE < 0.5) //position control
  {
    SERVO.EnableTorque(1, 1);
    SERVO.EnableTorque(2, 1);
    SERVO.EnableTorque(3, 1);
    SERVO.EnableTorque(4, 1);


    SERVO.joinMode(1);
    SERVO.joinMode(2);
    SERVO.joinMode(3);
    SERVO.joinMode(4);

    SERVO.WritePos(1, DegsToSteps(command_Joint1), 2000);
    delay(5);

    SERVO.WritePos(2, DegsToSteps(command_Joint2), 2000);
    delay(5);
    SERVO.WritePos(3, DegsToSteps(command_Joint3), 2000);
    delay(5);
    SERVO.WritePos(4, DegsToSteps(command_Joint4), 2000);

  }

  if (MODE > 0.5) //velocity control
  {


    SERVO.EnableTorque(1, 1);
    SERVO.EnableTorque(2, 1);
    SERVO.EnableTorque(3, 1);
    SERVO.EnableTorque(4, 1);

    SERVO.wheelMode(1);
    SERVO.wheelMode(2);
    SERVO.wheelMode(3);
    SERVO.wheelMode(4);



    SERVO.WriteSpe(1, RadsToTicks(command_Joint1));
```

```
    SERVO.WriteSpe(2, RadsToTicks(command_Joint2));
    SERVO.WriteSpe(3, RadsToTicks(command_Joint3));
    SERVO.WriteSpe(4, RadsToTicks(command_Joint4));
  }


}

double DegsToSteps(int posInt)
{
  float steps = map(posInt, -50, 50, 0, 1023);

 //transmission 1:2 so I do not take full advantage of  range -100 to 100 degrees
in JOints 1 /2 /3 , but Joint 4 has full transmission, so when i send 50    de-
grees to this JOint, i read the real value which is 100 degrees(check how i use
posJOint4 above)

  return steps;
}

double posToDegs(int posInt)
{
  float position_degrees = map(posInt, 0, 1023, -50, 50);

  return position_degrees;
}

double RadsToTicks(int velInt)
{
  float ticks = map(velInt, -6.5 , 6.5 , -1000, 1000);  //max speed is 62rpm ,
62rpm= 6.5rad/s , max speed that servo can understand is 1000

  return ticks;
```

Ανάλυση Βασικών Σημείων του κώδικα

```
void ServoControlCallback(const std_msgs::Float32MultiArray& msg)
{
  MODE = msg.data[0];
  command_Joint1 = msg.data[1];
  command_Joint2 = msg.data[2];
  command_Joint3 = msg.data[3];
  command_Joint4 = msg.data[4];
}


// here we initialize our Publisher and Subscriber, which publishe messages of type JointState on topic named "lbv150_uvms/jointstates", and subscribe messages of type Float32MultiArray on topic with name "joints_cmds_topic"

sensor_msgs::JointState joint_state_msg;
ros::Publisher pub("lbv150_uvms/joint_states", &joint_state_msg);  //topic: lbv150_uvms/joint_states , minimata: joint_state_msg
ros::Subscriber <std_msgs::Float32MultiArray> sub("joints_cmds_topic", ServoControlCallback);
```

Σε αυτό το σημείο ορίζουμε τον publisher και subscriber που θα πραγματοποιούν την επικοινωνία μεταξύ υπολογιστή (ROS) και arduino. Ο publisher εκδίδει στο topic lbv/150_uvms/jointstates, messages τύπου joint_state_msg προεγκατεστημένα στο ROS. Αυτά τα μηνύματα στέλνουν αναφορά σχετικά με τη θέση,ταχύτητα και load απο κάθε Joint. Αντίστοιχα ο subscriber λαμβάνει απο τον Η/Υ μηνύματα τύπου Float32MultiArray, που σχετίζονται με τα command θέσης και ταχύτητας στους επενεργητές, που εκδίδονται στο topic joints_cmds_topic. Συγκεκριμένα αυτό το μήνυμα έχει δομηθεί με τέτοιο τρόπο έτσι ώστε αν το MODE=0 να αντιλαμβάνονται οι servo command θέσης , ενώ αν MODE=1 να αντιλαμβάνονται

command ταχύτητας(βλέπε δομή της ServoControlCallback). Η δομή αυτών των μηνυμάτων στο ROS φαίνεται παρακάτω.

sensor_msgs/JointState Message

File: sensor_msgs/JointState.msg

Raw Message Definition

```
# This is a message that holds data to describe the state of a set of torque controlled joints.
#
# The state of each joint (revolute or prismatic) is defined by:
#  * the position of the joint (rad or m),
#  * the velocity of the joint (rad/s or m/s) and
#  * the effort that is applied in the joint (Nm or N).
#
# Each joint is uniquely identified by its name
# The header specifies the time at which the joint states were recorded. All the joint states
# in one message have to be recorded at the same time.
#
# This message consists of a multiple arrays, one for each part of the joint state.
# The goal is to make each of the fields optional. When e.g. your joints have no
# effort associated with them, you can leave the effort array empty.
#
# All arrays in this message should have the same size, or be empty.
# This is the only way to uniquely associate the joint name with the correct
# states.


Header header

string[] name
float64[] position
float64[] velocity
float64[] effort
```

std_msgs/Float32MultiArray Message

File: std_msgs/Float32MultiArray.msg

Raw Message Definition

```
# Please look at the MultiArrayLayout message definition for
# documentation on all multiarrays.

MultiArrayLayout  layout        # specification of data layout
float32[]         data          # array of data
```

```
//HOME POSITION initialization

SERVO.EnableTorque(1, 1);
SERVO.EnableTorque(2, 1);
SERVO.EnableTorque(3, 1);
SERVO.EnableTorque(4, 1);
// We initialize the starting position of the arm as it is introduced in DH configuration
SERVO.WritePos(1, 512, 2000);    //512 steps are in 0 degrees configuration
SERVO.WritePos(2, 512, 2000);
SERVO.WritePos(3, 512, 2000);
SERVO.WritePos(4, 512, 2000);
```

Τα συγκεκριμένα σέρβο για να τεθούν σε λειτουργία πρέπει να τρέξουμε την συνάρτηση EnableTorque(id,1). Σε αυτό το σημείο κάνουμε την αρχικοποίηση του βραχίονα, δηλαδή όπως και αν έχουμε μετατοπίσει “με το χέρι” τη διάταξη, όταν τρέξει ο αλγόριθμος ο βραχίονας θα επανέλθει και θα βρεί την “μηδενική θέση ” όπως την έχω ορίσει στο κινηματικό μοντέλο.

```
for (int id = 0; id < 3; id++) //Fulfill the arrays with motor readings
{
  posJoint = posToDegs(SERVO.ReadPos(id + 1));
  nh.spinOnce();

  pos[id] = posJoint;

  effJoint = SERVO.ReadLoad(id + 1);
  eff[id] =   effJoint;



}

posJoint4 = 2 * posToDegs(SERVO.ReadPos(4));
pos[3] = posJoint4;

effJoint4 = SERVO.ReadLoad(4);
eff[3] =   effJoint4;

// deg/s measurements of joint rotation

velJoint1 = 0.5 * 0.17 * SERVO.ReadSpe(1); //joint1 runs 85degs in 1 sec in maximum speed (1:2 transmission and thats why the 0.5 exists in the equation)
vel[0] = velJoint1;
velJoint2 =  0.5 * 0.17 * SERVO.ReadSpe(2);
vel[1] = velJoint2;
velJoint3 = 0.5 * 0.17 * SERVO.ReadSpe(3);
vel[2] = velJoint3;
velJoint4 = 0.17 * SERVO.ReadSpe(4); //joint4 runs 170degs maximum speed in 1 sec cause of 1:1
vel[3] = velJoint4;
```

Ο τρόπος με τον οποίο αντλούμε την πληροφορία σχετικά με τη θέση,ταχύτητα και φορτίο φαίνεται σε αυτό το σημείο. Η συναρτήσεις που χρησιμοποιούνται για αυτή την εργασία είναι οι ReadPos,ReadSpe και ReadLoad. Ο αναγνώστης σε αυτό το σημείο ίσως αναρωτηθεί τι συμβαίνει με το τέταρτο joint και γιατί διαχωρίζεται απο τα υπόλοιπα. Συμβαίνει η εξής ιδιομορφία, ενώ όλα τα joint του βραχίονα έχουν εκ κατασκευής μετάδοση 1:2 (χάνουν δηλαδή το μισό της απόλυτης τιμής τους), στο τέταρτο Joint έχουμε πλήρη μετάδοση. Γι αυτό το λόγο βλέπουμε ότι μπροστά απο το ReadSpe(4) δεν υπάρχει ο συντελεστής μείωσης 0.5 και κατ επέκταση ο τρόπος με τον οποίο καταχωρείται η θέση (SERVO.ReadPos(4)) είναι κλιμακωμένος με την διπλάσια τιμή. Εδώ ίσως κανείς αναρωτηθεί σχετικά με το γιατί να χρειαστεί να διπλασιάσουμε την πραγματική μετρούμενη τιμή, αλλά αυτό γίνεται ξεκάθαρο όταν αναλυθεί στη συνέχεια η δομή της συνάρτησης μετατροπής της θέσης (postoDegs) όπως την ανατιλαμβάνονται τα σέρβο (0-1024 steps), σε μοίρες. Τέλος ο συντελεστής 0,19 μπροστά απο την καταχώρηση της τιμής της ταχύτητας σχετίζεται με την μετατροπή των step/s που καταγράφουν τα σέρβο, σε deg/s.

```
double DegsToSteps(int posInt)
{
  float steps = map(posInt, -50, 50, 0, 1023); //transmission 1:2 so I do not take full advantage of  range -100 to 100 degrees in JOints 1 /2 /3 , but Joint 4 has full transmission, so when i send 50    degrees to this JOint, i read the real value which is 100 degrees(

  return steps;
}
double posToDegs(int posInt)
{
  float position_degrees = map(posInt, 0, 1023, -50, 50);

  return position_degrees;
}

double RadsToTicks(int velInt)
{
  float ticks = map(velInt, -6.5 , 6.5 , -1000, 1000);  //max speed is 62rpm , 62rpm= 6.5rad/s , max speed that servo can understand is 1000

  return ticks;
}
```

Οι συγκεκριμένοι σερβοκινητήρες αντιλαμβάνονται με συγκεκριμένο τρόπο την θέση και την ταχύτητα. Συγκεκριμένα για την θέση, έχουν ένα εύρος μέτρησης απο 0 έως 1024 steps, που σε μοίρες μεταφράζεται σε 0-200 μοίρες. Βλέπουμε ότι έχουμε ένα εύρος εκμετάλλευσης 200 μοιρών που πρέπει να μοιράσουμε ομοιόμορφα. Επιλέγουμε την κεντρική τιμή (512) να αντιστοιχεί στις μηδέν μοίρες, ώστε να έχουμε ισόποσο μοίρασμα της κίνησης σε 100 μοίρες

αριστερά και 100 δεξιά. Δυστυχώς όπως προείπαμε, εκ κατασκευής ο βραχίονας σε όλα τα joint εκτός του τέταρτου αξιοποιεί το μισό απο αυτό που προσφέρουν οι κινητήρες λόγω μετάδοσης. Γι αυτό το λόγο κατασκευάζουμε τις συναρτήσεις PostoDegs και DegsToSteps με τον τρόπο που φαίνεται παραπάνω. Σε αυτό το σημείο ο αναγνώστης μπορεί να καταλάβει γιατί παραπάνω πολλαπλασιάσαμε την τιμή της θέσης για το τέταρτο Joint με δύο(διότι το mapping γίνεται απο 50 έως -50 μοίρες ενώ στο Joint4 εκμεταλλευόμαστε το πλήρες εύρος 100 μοιρών λόγω μετάδοσης 1:1)

Σχετικά με την ταχύτητα ο τρόπος που μετατρέπουμε τα μετρούμενα deg/s σε rad/s (διότι έτσι μετράται η ταχύτητα στο Ros) φαίνεται μέσω της συνάρτησης RadsToTicks. Εδώ το mapping γίνεται βάση του μέγιστου ορίου που μπορούν να φτάσουν αυτά τα σέρβο και το οποίο είναι 1000 step/s ή 62rpm. Η μετατροπή των 62 rpm σε rad/s δίνει 6.5 rad/s.

5 Πειραματική Διαδικασία IBVS

5.1 Imaged Based Visual Servoing Θεωρητική Ανάλυση

Το visual servo control αναφέρεται στη χρήση δεδομένων όρασης υπολογιστή, για τον έλεγχο κίνησης ενός ρομπότ. Τα δεδομένα όρασης αντλούνται από μια κάμερα που είναι τοποθετημένη απευθείας πάνω στο τελικό στοιχείο δράσης κάποιου βραχίονα, είτε πάνω σε κάποιο κινούμενο ρομπότ, όπου σε αυτή τη περίπτωση η κίνηση του ρομπότ προκαλεί κίνηση της κάμερας. Ο σκοπός όλων των διεργασιών control με χρήση vision είναι η ελαχιστοποίηση κάποιου σφάλματος της μορφής:

$$\mathbf{e}(t) = \mathbf{s}(\mathbf{m}(t), \mathbf{a}) - \mathbf{s}^*$$

Το διάνυσμα s δεν είναι τίποτα άλλο από τις πραγματικές μετρούμενες ποσότητες, (Image features) μεταφρασμένες σε αποστάσεις του στόχου, συνήθως κάποιος ARUCO marker, από την κάμερα. Πιο συγκεκριμένα για ένα σημείο στον τρισδιάστατο χώρο με συντεταγμένες (X,Y,Z) ως προς το frame της κάμερας, ο μετασχηματισμός στον δισδιάστατο χώρο που αντιλαμβάνεται η κάμερα, είναι το διάνυσμα s με συντεταγμένες (χ,γ).

Όπου το $\chi = \frac{X}{Z} = \frac{απόσταση\ στον\ άξονα\ X,του\ 3D\ στόχου\ απο\ την\ κάμερα}{απόσταση\ στον\ άξονα\ Z,του\ 3D\ στόχου\ απο\ την\ κάμερα}$

Και $\gamma = \frac{Y}{Z} = \frac{απόσταση\ στον\ άξονα\ y,του\ 3D\ στόχου\ απο\ την\ κάμερα}{απόσταση\ στον\ άξονα\ Z,του\ 3D\ στόχου\ απο\ την\ κάμερα}$

Το s* είναι ίδιας λογικής διάνυσμα, απλά αυτή τη φορά με καταχωρημένες τιμές τις επιθυμητές αποστάσεις που θέλουμε να έχει η κάμερα μας από τον στόχο. Όπως γίνεται αντιληπτό, στην διεργασία αυτή στόχος μας είναι να μηδενίσουμε το σφάλμα και άρα εκκινώντας απο οποιοδήποτε σημείο να καταλήξουμε στην desired απόσταση που έχουμε ορίσει στο διάνυσμα s*.

Ο ζητούμενος νόμος ελέγχου ταχύτητας, ο οποίος είναι υπεύθυνος να παράγει τις τιμές της ταχύτητας που θα στέλνονται στο εκάστοτε servo, για τον μηδενισμό του σφάλματος είναι ο παρακάτω:

$$\dot{\boldsymbol{q}} = -\boldsymbol{\lambda} * [(\boldsymbol{Le} * \boldsymbol{Vc}) * \boldsymbol{Jac}]^{\boldsymbol{T}} * \boldsymbol{e}$$

Όπου λ= κάποιο κέρδος

e= το προαναφερθέν σφάλμα

Le= Interaction matrix by default

$$\mathrm{Le} = \begin{bmatrix} -1/\mathrm{Z} & 0 & \mathrm{x}/\mathrm{Z} & \mathrm{x}\gamma & -(1+\chi^2) & \gamma \\ 0 & -1/\mathrm{Z} & \gamma/\mathrm{Z} & 1+\gamma^2 & -\chi\gamma & -\chi \end{bmatrix}$$

Vc= Spatial motion transform Matrix from the vision frame(camera to the end effector frame).

Στην ουσία πρόκειται για έναν πίνακα μετασχηματισμού ταχυτήτων απο το frame της κάμερας στο frame του end effector.

$$V_C^E = \begin{bmatrix} R_c^E & [t_c^E] * R_c^E \\ 0 & R_c^E \end{bmatrix}$$

$$t_c^E = [distance\ from\ x\ axis \quad distance\ from\ y\ axis \quad distance\ from\ z\ axis]^T$$

Στην περίπτωση μας το frame της κάμερας απέχει από το frame του end effector 5cm κατά τον χ και -8 cm από τον z, οπότε έχουμε

$t_c^E = [0.05 \quad 0 \quad -0.08]^T$ και

$$[t_c^E] = \begin{bmatrix} 0 & 0{,}08 & 0 \\ -0{,}08 & 0 & 0{,}05 \\ 0 & 0{,}05 & 0 \end{bmatrix}$$

$$R_c^E = [Rotation\ on\ x\ axis * Rotation\ on\ y\ axis * Rotation\ on\ z\ axis]$$

$$R_c^E = [Rotx(30)] * Roty(-90) * Rotz(0)$$

$$R_c^E = \begin{bmatrix} 0 & 1 & 0 \\ -\sqrt{3}/2 & 0 & -0.5 \\ -0.5 & 0 & \sqrt{3}/2 \end{bmatrix}$$

Κάνοντας της πράξεις καταλήγουμε στον ζητούμενο πίνακα μετασχηματισμού ταχυτήτων όπως καταγράφεται στο ακόλουθο υποκεφάλαιο, εντός του κώδικα της Pyhton.

Τέλος το μητρώο Jac δεν είναι τίποτα περισσότερο απο την Ιακωβιανή που διέπει την διάταξη του βραχίονα 4 βαθμών ελευθερίας και η οποία υπολογίζεται απο το ακόλουθο πρόγραμμα της python.

Έχοντας λοιπόν εξηγήσει πως ακριβώς υλοποιείται ο νόμος ελέγχου ταχυτήτων βασιζόμενος στην τεχνική του IBVS, είμαστε σε θέση να παρουσιάσουμε την υλοποίηση του σε κώδικα Pyhton.

5.2 Imaged Based Visual Servoing Python Script

Σε αυτό το κεφάλαιο παραθέτω τον κώδικα που τρέχει την υλοποίηση IBVS. Στο πρώτο του μέρος υπολογίζει την κινηματική αλυσίδα και την Ιακωβίανη του βραχίονα, δίνοντας όλη την πληροφορία που χρειάζεται η κλάση IBVS ARM CLASS για να ολοκληρώσει το ζητούμενο task. Το πείραμα έχει ως στόχο, κάμερα προσαρτημένη πάνω στο τελικό στοιχείο δράσης, να παρέχει όλη την πληροφορία της απόστασης του βραχίονα από τους 4 aruco markers και ο αλγόριθμος να παράγει μέσω του νόμου ελέγχου, τις ταχύτητες διόρθωσης θέσης στους σερβοκινητήρες. Τελικά ο βραχίονας θα πρέπει να ακολουθεί την διάταξη των 4 aruco markers, καθώς τους κινούμε. Βέβαια να σημειώσουμε ότι λόγω των 4 βαθμών ελευθερίας του βραχίονα, οι θέσεις στις οποίες υποχρεώνουμε τον Manipulator να κινηθεί, πρέπει να είναι εφικτές. Θυμίζουμε ότι 6 βαθμοί ελευθερίας απαιτούνται από έναν βραχίονα για να είναι σε θέση να περιγράψει κάθε σημείο στον τρισδιάστατο χώρο. Παρακάτω φαίνεται ο κώδικας όπως αναπτύχθηκε και υλοποιήθηκε.

```
#!/usr/bin/env python

#ROS msgs
from sensor_msgs.msg import JointState
from geometry_msgs.msg import Vector3Stamped,Quaternion,Vector3,TransformStamped
from std_msgs.msg import Float32MultiArray
from fiducial_msgs.msg import FiducialTransform, FiducialTransformArray


# Python imports
from PyKDL import *

# Python Libs
import numpy as np
import roslib

# Ros Libs
import rospy
import math
from math import *
import time
import traceback

####### BUILDING THE KINEMATIC CHAIN WITH PyKdl only For Hand ########

arm_chain=Chain()
```

```
noDOFS=4
gain=50.0

#FROM BASE FRAME TO EE FINAL FRAME

joint0 = Joint(Joint.None)
frame0 = Frame(Vector(0.0, 0.0, 32.8e-3))
segment0 = Segment(joint0, frame0)
arm_chain.addSegment(segment0)

joint1 = Joint(Joint.RotZ)
frame1 = Frame(Vector(0.0, 0.0, 45e-3))
segment1 = Segment(joint1, frame1)
arm_chain.addSegment(segment1)

joint2 = Joint(Joint.RotY)
frame2 = Frame(Vector(0.0, 0.0, 147.7e-3))
segment2 = Segment(joint2, frame2)
arm_chain.addSegment(segment2)

joint3 = Joint(Joint.RotY)
frame3 = Frame(Vector(-28e-3, 0.0, 75.4e-3))
segment3 = Segment(joint3, frame3)
arm_chain.addSegment(segment3)

joint4 = Joint(Joint.RotZ)
frame4 = Frame( Vector(0.0, 0.0, 176e-3))
segment4 = Segment(joint4, frame4)
arm_chain.addSegment(segment4)

######## LBV KINEMATICS CLASS ##############

''' ARM Kinematics Class '''
class arm_uvms_kinematics:

    def __init__(self):
        print "ARM UVMS CLASS INITIALIZED"
                #Subscribers

rospy.Subscriber("/lbv150_uvms/joint_states",JointState,self.updateArmStates,queue
_size=1)
        #Publishers
                self.pub_ee_pos =
rospy.Publisher("/lbv150_uvms/ee/position",Vector3Stamped,queue_size=1)
        self.pub_ee_orient =
rospy.Publisher("/lbv150_uvms/ee/orientation",Vector3Stamped,queue_size=1)

                self.arm_states = np.array([0.0, 0.0, 0.0, 0.0])


    ''' Calculate Forward Kinematics '''
    def fk(self, jointAngles):
            if isinstance (jointAngles, JntArray):
                    joints = jointAngles
            elif isinstance (jointAngles, np.ndarray):
                        joints = JntArray(4)
                        for i in range (0,4):
                            joints[i] = jointAngles[i]
            fk=ChainFkSolverPos_recursive(arm_chain)
            finalFrame=Frame()
            fk.JntToCart(joints,finalFrame)
            return finalFrame


    ''' Calculate Jacobian Matrix'''
    def jac(self, jointAngles):
            if isinstance (jointAngles, JntArray):
                    joints = jointAngles
            elif isinstance (jointAngles, np.ndarray):
                    joints = JntArray(4)
                    for i in range (0,4):
                        joints[i] = jointAngles[i]
            jacobian = Jacobian(4)
            solver = ChainJntToJacSolver(arm_chain)
            solver.JntToJac(joints,jacobian)
            return jacobian
```

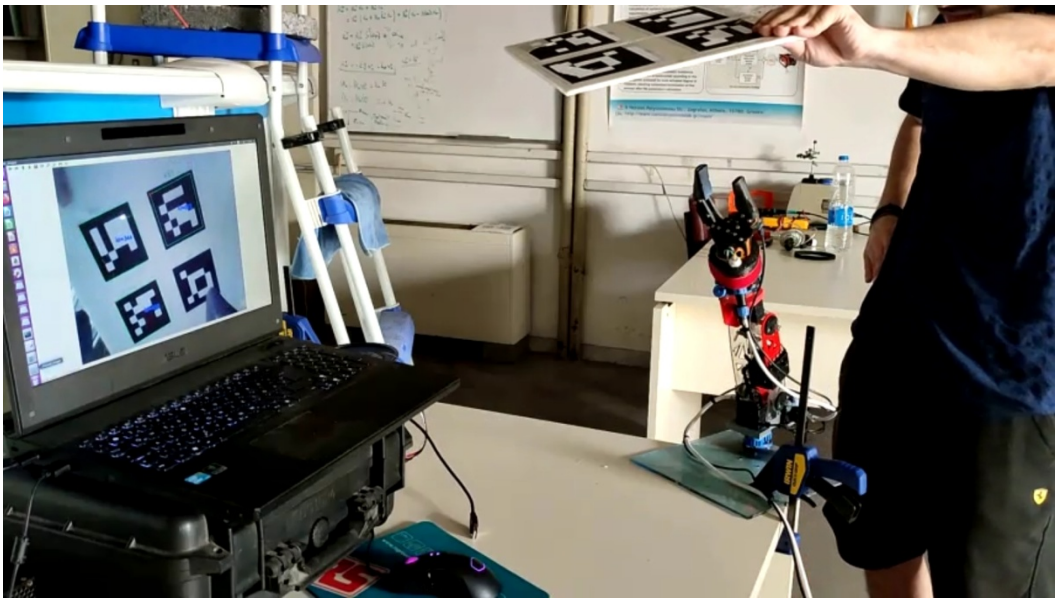
```
    def updateArmStates(self, joint_state_msg):
            self.arm_states[0] = joint_state_msg.position[0]*math.pi/180
            self.arm_states[1] = joint_state_msg.position[1]*math.pi/180
            self.arm_states[2] = joint_state_msg.position[2]*math.pi/180
            self.arm_states[3] = joint_state_msg.position[3]*math.pi/180


################### IBVS ARM CLASS ################

#       Le:Returns the Image Based Visual Servoing Interaction Matrix
#       ed:Returns the error vector for IBVS
#       Vc: matrix transforming coordinates from camera frame to EE frame


''' ImageBased VisualServoing Class '''
class ImageBasedVS_arm:

       def __init__(self):
        print "ImageBasedVisualServoing ARM CLASS INITIALIZED"
                #Subscribers
        rospy.Subscriber("/fiducial_transforms", FiducialTransformArray,
self.updatefiducialtransforms, queue_size=1)  #a subscriber to incoming transforms
        #Publishers
                self.cmdPub = rospy.Publisher("/joints_cmds_topic",
Float32MultiArray, queue_size=1)  # A publisher for manipulator motion commands



                # The name of the coordinate frame of the fiducial we are inter-
ested in
                self.target_fiducial1 = rospy.get_param("~target_fiducial",
"fid344")
                self.target_fiducial2 = rospy.get_param("~target_fiducial",
"fid822")
                self.target_fiducial3 = rospy.get_param("~target_fiducial",
"fid456")
                self.target_fiducial4 = rospy.get_param("~target_fiducial",
"fid233")

                # Here we declare the values X,Y,Z(CAMERA READINGS) of initializa-
tion, with the hypothesis that we are in the desired target.
                # this values lead to x0=x0d anf g0=g0d and so error=0, no veloc-
ity command will be sent,until a fiducial is recognised

                self.targetA=np.array([0.0295633363599, -0.0454764621775,
0.185576697632])    #822
                self.targetB=np.array([-0.041023299567, 0.0233090248379,
0.191104968899])    #456
                self.targetC=np.array([0.0300123010723, 0.0232874475034,
0.194020598406])     #233
                self.targetG=np.array([-0.0401000713759,  -0.0460535908577,
0.184299802391])   #344


                self.Vc=np.array([[0.0, 1.0, 0.0, 0.0693, 0.0, -0.04],
                           [-0.866, 0.0, -0.5, -0.025, -0.08, 0.0433],
                                  [-0.5, 0.0, 0.866, -0.0433, 0.0, -0.025],
                                                [0.0, 0.0, 0.0, 0.0, 1.0, 0.0],
                                                   [0.0, 0.0, 0.0, -0.866, 0.0, -
0.5],
                                                                 [0.0, 0.0, 0.0, -0.5,
0.0, 0.866]])
                self.ed=np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0]).reshape(8,1)

                self.hysteresis_count = rospy.get_param("~hysteresis_count", 20)

                self.got_fid822 = False
                self.got_fid456 = False
                self.got_fid233 = False
                self.got_fid344 = False

                # How many loop iterations to keep  velocity after fiducial
                # disappears
```

```
            self.hysteresis_count = rospy.get_param("~hysteresis_count", 10)



'''Called when a fiducial marker has been received'''
def updatefiducialtransforms(self,msg):
     imageTime = msg.header.stamp

     for m in msg.transforms:
         id=m.fiducial_id
         trans = m.transform.translation
         rot = m.transform.rotation

         t = TransformStamped()
         t.child_frame_id = "fid%d" % id
         t.header.frame_id = msg.header.frame_id
         t.header.stamp = imageTime

         t.transform.translation.x = trans.x
         t.transform.translation.y = trans.y
         t.transform.translation.z = trans.z
         t.transform.rotation.x = rot.x
         t.transform.rotation.y = rot.y
         t.transform.rotation.z = rot.z
         t.transform.rotation.w = rot.w




         if t.child_frame_id == self.target_fiducial2:
          self.targetA=np.array([trans.x,trans.y,trans.z])
          self.got_fid822 = True

         if t.child_frame_id == self.target_fiducial3:
          self.targetB=np.array([trans.x,trans.y,trans.z])
          self.got_fid456 = True
         if t.child_frame_id == self.target_fiducial4:
          self.targetC=np.array([trans.x,trans.y,trans.z])
          self.got_fid233 = True
         if t.child_frame_id == self.target_fiducial1:
          self.targetG=np.array([trans.x,trans.y,trans.z])
          self.got_fid344 = True

def ImageBasedVS(self):
  Z0 = self.targetA[2]
  Z1 = self.targetB[2]
  Z2 = self.targetC[2]
  Z3 = self.targetG[2]

  if Z0 == 0.0:
   Z0 = 0.001
  if Z1 == 0.0:
   Z1 = 0.001
  if Z2 == 0.0:
   Z2 = 0.001
  if Z3 == 0.0:
   Z3 = 0.001


  x0 = self.targetA[0]/Z0
  x1 = self.targetB[0]/Z1
  x2 = self.targetC[0]/Z2
  x3 = self.targetG[0]/Z3

  g0 = self.targetA[1]/Z0
  g1 = self.targetB[1]/Z1
  g2 = self.targetC[1]/Z2
  g3 = self.targetG[1]/Z3


  L0 = np.array([[-1.0/Z0, 0.0, x0/Z0, x0*g0, -(1.0+x0*x0), g0],
                 [0.0, -1.0/Z0, g0/Z0, 1.0+g0*g0, -x0*g0, -x0]])

  L1 = np.array([[-1.0/Z1, 0.0, x1/Z1, x1*g1, -(1.0+x1*x1), g1],
                  [0.0, -1.0/Z1, g1/Z1, 1.0+g1*g1, -x1*g1, -x1]])
```

```
        L2 = np.array([[-1.0/Z2, 0.0, x2/Z2, x2*g2, -(1.0+x2*x2), g2],
                       [0.0, -1.0/Z2, g2/Z2, 1.0+g2*g2, -x2*g2, -x2]])

        L3 = np.array([[-1.0/Z3, 0.0, x3/Z3, x3*g3, -(1.0+x3*x3), g3],
                        [0.0, -1.0/Z3, g3/Z3, 1.0+g3*g3, -x3*g3, -x3]])

        Le = np.vstack([L0, L1, L2, L3])


         #Setting the desired Goal Position that the arm must reach


        x0d =   0.159305218   #marker id= 822
        g0d =  -0.245054809

        x1d =   -0.214663699  #marker id= 456
        g1d =    0.121969748

        x2d = 0.154686159  #marker id= 233
        g2d = 0.120025645

        x3d = -0.217580653  #marker id= 344
        g3d = -0.249884103

        s = np.array([x0, g0, x1, g1, x2, g2, x3, g3]).reshape(8,1)

        self.ed = np.array([x0-x0d, g0-g0d, x1-x1d, g1-g1d, x2-x2d, g2-g2d, x3-
x3d, g3-g3d]).reshape(8,1)


        return Le


###################### MAIN #########################

if __name__ == '__main__':
    try:
        rospy.init_node('arm_kinematics_IBVS_node')

        rate_it = rospy.Rate(10)

        arm_kin = arm_uvms_kinematics()  #code name of the constructed class that
we will use in main
        ibvs=ImageBasedVS_arm()          # ----//-----

        qvels=np.array([0.0, 0.0, 0.0, 0.0])

        times_since_last_fid= 0
        velID1=0
        velID2=0
        velID3=0
        velID4=0

        while not rospy.is_shutdown():
                t = rospy.Time.now()

                curr_states = np.array([ arm_kin.arm_states[0],
arm_kin.arm_states[1], -arm_kin.arm_states[2], arm_kin.arm_states[3]])
                curr_frame = arm_kin.fk(curr_states)

                #1st part calculates and publish the end effectors pose and orien-
tation
                x_ee_i = curr_frame.p[0]
                y_ee_i = curr_frame.p[1]
                z_ee_i = curr_frame.p[2]
        euler_ee_z, euler_ee_y, euler_ee_x = curr_frame.M.GetEulerZYX()

                pos_ee_msg = Vector3Stamped()
                orient_ee_msg = Vector3Stamped()

                pos_ee_msg.header.stamp = t
                pos_ee_msg.vector.x = x_ee_i
                pos_ee_msg.vector.y = y_ee_i
        pos_ee_msg.vector.z = z_ee_i
```

```
                orient_ee_msg.header.stamp = t
        orient_ee_msg.vector.x = euler_ee_x
        orient_ee_msg.vector.y = euler_ee_y
        orient_ee_msg.vector.z = euler_ee_z

                arm_kin.pub_ee_pos.publish(pos_ee_msg)
        arm_kin.pub_ee_orient.publish(orient_ee_msg)



              #2nd part calculates  the qvels commands that need to be send to
the arm according to IBVS methothology

                #Get Current Jacobian
                Jac = arm_kin.jac(curr_states)


                #Transform KDL Jacobian to np.array for mathematical manipulation
                useJacobian = np.zeros([6,noDOFS], float)

                for i in range (0,6):
                  for j in range (0,noDOFS):
                     useJacobian[i,j] = Jac[i,j]

                #Calculate the Jacobian which connects the command velocities with
the errors
                Js = np.dot(np.dot(ibvs.ImageBasedVS(),ibvs.Vc), useJacobian)

                #Calculate the pseudoinverse of Js for the command law, as Js is
of 8x4 dimensions
                JsInversed=np.linalg.pinv(Js)

                #control Law
                qvels=-gain*np.dot(JsInversed,ibvs.ed)  #gain has been set  in the
beggining of code



                if ibvs.got_fid822 and ibvs.got_fid456 and ibvs.got_fid233 and
ibvs.got_fid344  :
                  times_since_last_fid = 0
                else:
                  times_since_last_fid += 1

                #publishing the qvels that have been calculated to the servo mo-
tors

                if ibvs.got_fid822 and ibvs.got_fid456 and ibvs.got_fid233 and
ibvs.got_fid344  :
                  # Make sure that the  speed is within limits  , speed 1 rad/s is
my desired highest speed to illustrate visual servoing
                  velID1=qvels[0]
                  if velID1>1.0:
                     velID1=1.0
                  if velID1<-1.0:
                     velID1=-1.0

                  velID2=qvels[1]
                  if velID2>1.0:
                     velID2=1.0
                  if velID2<-1.0:
                     velID2=-1.0

                  velID3=qvels[2]
                  if velID3>1.0:
                     velID3=1.0
                  if velID3<-1.0:
                     velID3=-1.0

                  velID4=qvels[3]
                  if velID4>1.0:
                     velID4=1
                  if velID4<-1.0:
                    velID4=-1
```

```
                array=[1,velID1,velID2,velID3,velID4]
                print("fiducials are all in target")

            # Hysteresis, don't immediately stop if a fiducial is lost
            elif not ibvs.got_fid822 or ibvs.got_fid456 or ibvs.got_fid233 or
ibvs.got_fid344 and times_since_last_fid <ibvs.hysteresis_count :
                array=[1,0.0,0.0,0.0,0.0]
                print("HYSTERISIS STOP ON")



            vel_cmd_msg=Float32MultiArray(data=array)

            ibvs.cmdPub.publish(vel_cmd_msg)
        # We already acted on the current fiducial
            ibvs.got_fid822 = False
            ibvs.got_fid456 = False
            ibvs.got_fid233 = False
            ibvs.got_fid344 = False
        rate_it.sleep()
        rospy.spin()
    except rospy.ROSInterruptException: pass
```



Εικόνα 5.2.1 *Υλοποίηση IBVS με τον βραχίονα στην τελική του μορφή*

Ο βραχίονας κατάφερε να ακολουθήσει με επιτυχία τον κινούμενο στόχο με κάποιες μικρές παρατηρήσεις. Δεδομένου ότι επιλέξαμε να ακολουθούμε 4 στόχους για μεγαλύτερη ευρωστία του συστήματος, κάθε φορά που ένας στόχος δεν ανιχνευόταν, η διάταξη σταματούσε την κίνηση της (αυτή η παύση τέθηκε ως δικλείδα ασφαλείας για την κίνηση του βραχίονα, κατά την εκτέλεση του πειράματος). Αυτό όπως θα φανεί και στο σχετικό βίντεο το οποίο επισυνάπτεται στο usb, κάνει την απόκριση του συστήματος στην κίνηση του στόχου να φαίνεται σχετικά αργή. Όπως προείπα το γεγονός αυτό δεν σχετίζεται με τον νόμο ελέγχου της διεργασίας IBVS , αλλά απο την ικανότητα του χειριστή του στόχου να έχει κάθε φορά και τα 4 aruco markers εντός του οπτικού πεδίου της κάμερας.

Μέσω της εκτέλεσης αυτής της πειραματικής διαδικασίας επαληθεύουμε 2 πολύ σημαντικά πράγματα για τον μελλοντικό χρήστη της διάταξης. Πρώτον, ότι όσον αφορά τον προγραμματισμό του χαμηλού επιπέδου που έχει γίνει στο περιβάλλον του arduino όλα έχουν καλώς οριστεί(feedback και velocity commands). Δεύτερον, ότι η κινηματική ανάλυση που έχει γίνει είναι σωστή. Σε κάθε περίπτωση λαθεμένης κινηματικής ανάλυσης, ο βραχίονας δεν θα μπορούσε να ακολουθήσει το στόχο. Συνοψίζοντας, ο φοιτητής που θα καταπιαστεί με τη χρήση της διάταξης αυτού του βραχίονα πρέπει να είναι βέβαιος ότι το κινηματικό μοντέλο είναι ορθό και μπορεί να το χρησιμοποιήσει με ασφάλεια και κατ επέκταση, υπάρχουν έτοιμες αναμονές για εντολές ταχύτητας και θέσης καθώς και σωστή πληροφορία για το feedback θέσης και ταχύτητας.

6 Οδηγός χρήσης και καλής λειτουργίας

Σε αυτό το τελευταίο κεφάλαιο σκοπεύω να δώσω όλες εκείνες τις σημαντικές πληροφορίες που θα χρειαστεί κανείς για να θέσει τον βραχίονα σε λειτουργία με ασφάλεια και μικρό κόπο. Ίσως είναι το σημαντικότερο κεφάλαιο που πρέπει να διαβάσει ο επίδοξος χειριστής, διότι θα εξασφαλίσει και θα αποτρέψει την καταστροφή μηχανικών μέρων της διάταξης και θα δώσει όλη την γνώση που αποκτήθηκε μέσα απο πειραματισμό και επιπλοκές που συνάντησα. Θα ακολουθηθεί δομή που δεν θα έχει ακριβώς την μορφή κειμένου, ωστέ να είναι αρκετά ξεκάθαρα όλα τα βήματα. Σκοπός μου είναι, κάποιος που θα κάνει την ανάγνωση αυτών των οδηγιών να φτάσει γρήγορα και με ασφάλεια στη λειτουργία της διάταξης.

Οι οδηγίες ξεκινούν παρακάτω:

1) Λογισμικό : το λογισμικό που πρέπει κάποιος να εγκαταστήσει στο μηχάνημα του για την λειτουργία της διάταξης είναι:

a) Arduino IDE
b) Ubuntu 16.04 LTS
c) ROS Kinetic

2) Μετά την εγκατάσταση του Arduino IDE, η βιβλιοθήκη SCServo που παρέχω στο usb πρέπει να αποθηκευτεί στα Libraries του Arduino IDE για να μπορεί να γίνει χρήση και προγραμματισμός των αλγορίθμων χαμηλού επιπέδου που τρέχουν μέσα στο arduino, εφόσον για κάποιο λόγο χρειαστεί.

3) Η εγκατάσταση της έκδοσης των Ubuntu 16.04 δούλεψε με όλα τα πακέτα που χρησιμοποιήθηκαν στο ROS και προτείνεται ανεπιφύλακτα

4) Εγκατάσταση του ROS kinetic. Μόλις η διαδικασία ολοκληρωθεί τα πακέτα που χρειάστηκα για την υλοποίηση αυτής της διπλωματικής ήταν τα :

a)**rosserial** , για επικοινωνία μεταξύ Η/Υ και arduino, binary installation

b)**cv_camera** , για την ανάγνωση της κάμερας, binary installation ή άντληση απο το usb , και catkinize μέσα στο src του catkin_ws

c)**fiducials**, για την αναγνώριση των aruco markers, binary installation ή άντληση απο το usb , και catkinize μέσα στο src του catkin_ws

d)ibvs_arm, για την εκτέλεση της διεργασίας IBVS, είναι custom πακέτο που έφτιαξα, οπότε άντληση απο το usb , και catkinize μέσα στο src του catkin_ws

5) Άνοιγμα τροφοδοσίας 12 V

6) Σύνδεση του καλωδίου που αναγράφει SCS15 στον Η/Υ, με άνοιγμα του arduino IDE πρέπει κανείς να βλέπει στα boards, συνδεδεμένο το arduino LEONARDO σε κάποιο COM (συνήθως COM0). Αυτή είναι και η θείρα που θα χρησιμοποιηθεί για την σειριακή επικοινωνία μεταξύ ROS και Arduino

7) Κάνοντας χρήση του USB cable που αναγράφει Cam συνδέουμε την κάμερα στον Η/Υ. Για να ελέγξουμε το Port της σύνδεσης, σε ένα terminal γράφουμε :

```
lsusb
ls-lrth /dev/video*
```

Στην δική μου περίπτωση, η κάμερα του βραχίονα που θα χρησιμοποιηθεί για visual servoing αναγνωρίζεται ως video1, διότι εργάζομαι απο λάπτοπ και ήδη υπάρχει εγκατεστημένη η κάμερα του μηχανήματος μου. Αυτή η παρατήρηση είναι πολύ σημαντική, μιας και σε κάθε shell σχετικό με διεργασίες κάμερας, θα πρέπει να δηλώνουμε σε ποιο port είναι η κάμερα που θα χρησιμοποιηθεί.

8) Για την σύνδεση του gripper, κάνουμε χρήση του καλωδίου πάνω στο οποίο αναγράφεται UNDEE. Εδώ αν κανείς θέλει να ανοίξει ή να κλείσει τον gripper αρκεί να ανοίξει την serial monitor μέσα στο arduino IDE και να γράψει την εντολή open ή close αντίστοιχα. Αν κάποιος επιθυμεί να συγκρατήσει κάποιο αντικείμενο τότε παραπέμπεται στην διπλωματική του κ.Μένεγα για την πληροφόρηση του για την αντίστοιχη εντολή.

9)Σε ένα terminal ξεκινάμε καταχωρώντας την εντολή

roscore

9) Σε νέο terminal ξεκινάμε την επικοινωνία μεταξύ ROS και arduino γράφοντας την παρακάτω εντολή. Αν έχεις διαφορετικό port στο οποίο ανιχνεύεται το arduino Leonardo πχ ACM1,διόρθωσε την εντολή ανάλογα.

rosrun rosserial_python serial_node.py /dev/ttyACM0 _baud:=115200

10) Ήδη είμαστε σε θέση να ελέγξουμε κάποια πρώτα μηνύματα σχετικά με την θέση, την ταχύτητα και το φορτίο των servo. Αρκεί να διαβάσουμε το αντίστοιχο topic :

rostopic echo /lbv150_uvms/joint_states

11) Αν κάποιος θέλει να στείλει εντολές θέσης ή ταχύτητας στον βραχίονα μέσω του terminal αρκεί να κάνει Publish την παρακάτω δομή στο αντίστοιχο topic. Μπορούμε να στείλουμε εντολές θέσης ή ταχύτητας επιλέγοντας Mode=0 για θέση (εύρος εντολών 50 έως -50 μοίρες) είτε Mode=1 για εντολές ταχύτητας (έυρος εντολών -6.5 έως 6.5 για τον κόκκινο βραχίονα και -3.25 έως 3.25 για τον κίτρινο βραχίονα). Σημειώνεται ότι για εντολές ταχύτητας μικρότερες της μονάδας δεν παρατηρείται απόκριση κίνησης στα servo.

ΠΡΟΣΟΧΗ! Αν στείλουμε command ταχύτητας (Mode=1), τότε για να επανέλθουμε σε εντολές θέσης (Mode=0) πρέπει να επανεκινήσουμε την τροφοδοσία των σερβοκινητήρων. Αν αυτό δεν γίνει τότε τα servo θα συνεχίσουν να δέχονται command ταχύτητας και θα χάσουμε τον έλεγχο της διάταξης. Η μορφή του μηνύματος για command θέσης ή ταχύτητας φαίνεται παράκατω:

rostopic pub /joints_cmds_topic std_msgs/Float32MultiArray "layout:
dim:
- label: ''
size: 5
stride: 0
data_offset: 0
data:
- 0.0 ←
- 0.0
- 0.0
- 0.0
- 0.0
"

Εδώ έχουμε ένα παράδειγμα εντολής θέσης όπου υποχρεώνει όλα τα servo να πάνε στις 0 μοίρες. Στο μηδενικό όπου έχω σημειώσει με βελάκι καταχωρούμε την τιμή 0 ή 1 ανάλογα αν θέλουμε να κάνουμε έλεγχο θέσης ή ταχύτητας. Τα υπόλοιπα με την σειρά αφορούν command θέσης στο Joint1,joint2,joint3 και joint4. Υπενθυμίζω ότι το εύρος της εντολής θέσης είναι 50 έως -50 μοίρες.

Ένα αντίστοιχο παράδειγμα για εντολή ταχύτητας δίνεται στη συνέχεια. Δεν προτείνεται πειραματισμός με εντολές ταχύτητας στη διάταξη, καθώς εκκινώντας μία εντολή ταχύτητας, το servo θα φτάσει γρήγορα σε μηχανικό τέρμα του βραχίονα. Παρόλα αυτά αν κανείς επιθυμεί να δοκιμάσει τέτοιες εντολές, δίνοντας μία μικρή τιμή του 1 rad/s σε κάποιο joint και όντας έτοιμος να διακόψει την τροφοδοσία μπορεί να ελέγξει τη λειτουργικότητα των εντολών ταχύτητας.

rostopic pub /joints_cmds_topic std_msgs/Float32MultiArray "layout:
dim:
- label: ''
size: 5
stride: 0
data_offset: 0
data:
- 1.0
- 0.0
- 0.0
- 1.0 ←καταχώρηση ταχύτητας 1 rad/s στο joint 3

- 0.0

"

12) Όσον αφορά την κάμερα, η επικινωνία της με το ROS ξεκινά γράφοντας:

source ~/catkin_ws/devel/setup.bash

rosparam set cv_camera/device_id 1 (αν η κάμερα σου είναι στο video0 άλλαξε την εντολή αντίστοιχα)

rosrun cv_camera cv_camera_node

13) Για την αναγνώριση των aruco markers σηκώνουμε το launch αρχείο που περιέχεται στο πακέτο που έχουμε προεγκαταστήσει. Πριν το κάνουμε όμως, πρέπει να βρισκόμαστε ήδη μέσα στο αρχείο αυτό. Οπότε :

cd ~/catkin_ws/src/fiducials-kinetic-devel/aruco_detect/launch
rosparam set cv_camera/device_id 1
source ~/catkin_ws/devel/setup.bash
roslaunch aruco_detect.launch

14) Για να δούμε το αποτέλεσμα της αναγνώρισης των markers στην οθόνη μας, γράφουμε:

source ~/catkin_ws/devel/setup.bash
rosparam set cv_camera/device_id 1
rosrun image_view image_view image:=/fiducial_images

15) Αν επιθυμούμε να δούμε τις συντεταγμένες των markers, γράφουμε :

source ~/catkin_ws/devel/setup.bash
rosparam set cv_camera/device_id 1
rostopic echo /fiducial_transforms

16) Οι επόμενες εντολές αφορούν την εκτέλεση της διαδικασίας IBVS, κάποιος που δεν επιθυμεί να τρέξει το εν λόγω πείραμα μέχρι εδώ έχει όλη την πληροφορία που χρειάζεται για την χρήση του βραχίονα και της κάμερας, καθώς και την αναίχνευση των aruco markers. Για να ξεκινήσουμε την διεργασία IBVS σηκώνουμε το launch αρχείο που βρίσκεται στο αντίστοιχο πακέτο :

cd ~/catkin_ws/src/ibvs_arm/launch
source ~/catkin_ws/devel/setup.bash
roslaunch ibvs_arm.launch

17) Για να ελέγξουμε τις τιμές που στέλνει ο ελεγκτής του IBVS αρκεί να ακούσουμε το αντίστοιχο topic όπου καταχωρούνται οι εντολές θέσης και ταχύτητας:

rostopic echo /joints_cmds_topic