Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Μηχανολόγων Μηχανικών

ΔΠΜΣ: Συστήματα Αυτοματισμού

Εξέταση δυνατοτήτων συνεργασίας ανθρώπου-βιομηχανικού ρομποτικού βραχίονα με προσομοίωση σε περιβάλλον ROS

Στάθας Ευθύμιος

Επιβλέπων: Γ.Χ Βοσνιάκος, Καθηγητής Ε.Μ.Π

Αθήνα, Φεβρουάριος 2021

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Μηχανολόγων Μηχανικών

ΔΠΜΣ: Συστήματα Αυτοματισμού

Εξέταση δυνατοτήτων συνεργασίας ανθρώπου-βιομηχανικού ρομποτικού βραχίονα με προσομοίωση σε περιβάλλον ROS

Στάθας Ευθύμιος

Επιβλέπων: Γ.Χ Βοσνιάκος, Καθηγητής Ε.Μ.Π

Αθήνα, Φεβρουάριος 2021

Ευχαριστίες

Στο πλαίσιο αυτής της εργασίας οφείλω να ευχαριστήσω θερμά κάποιους ανθρώπους οι οποίοι βοήθησαν στην ολοκλήρωση της εκπόνησης της διπλωματικής μου εργασίας καθώς και στην ολοκλήρωση των σπουδών μου.

Ευχαριστώ θερμά τον επιβλέποντα της διπλωματικής εργασίας μου, κύριο Γ.Χ Βοσνιάκο, Καθηγητή της Σχολής Μηχανολόγων Μηχανικών για την καθοδήγηση, την ενθάρρυνση και τη συνεργασία κατά τη συγγραφή της εργασίας. Η ανάθεση της παρούσας διπλωματικής εργασίας μου έδωσε τη δυνατότητα να ασχοληθώ με την ρομποτική σε βιομηχανικό επίπεδο. Με αυτή την ευκαιρία μπόρεσα να διευρύνω τις γνώσεις μου σε αυτόν τον τομέα και να ασχοληθώ με ανερχόμενες τεχνολογίες.

Στη συνέχεια, θα ήθελα να ευχαριστήσω όλους τους διδάσκοντες του ΔΠΜΣ Συστημάτων Αυτοματισμού. για τις γνώσεις που μου μετέδωσαν στη διάρκεια της φοίτησής μου.

Τέλος, ευχαριστώ από καρδίας την οικογένειά μου για την αμέριστη συμπαράσταση σε όλη τη διάρκεια των σπουδών μου.

Περίληψη

Στο πλαίσιο της παρούσας διπλωματικής εργασίας υλοποιείται ο προγραμματισμός του ρομποτικού βραχίονα Stäubli RX90L στο περιβάλλον ROS (Robot Operating System) προκειμένου να επιτευχθεί συνεργασία με τον άνθρωπο. Συγκεκριμένα, πραγματοποιείται η δημιουργία του πακέτου προγραμματισμού στο περιβάλλον του ROS και η ανάπτυξη αλγορίθμων σε γλώσσα Python για συνεργασία και ασφάλεια στον χώρο εργασίας (workspace) του ρομπότ. Πρακτικά, το ρομπότ θα πρέπει είτε να σταματά, είτε να ελαττώνει ταχύτητα, είτε να εκτελεί μια ορισμένη τροχιά όταν παρεμβάλλεται ο παράγοντας άνθρωπος στον χώρο εργασίας του.

Στο ρομπότ προσαρμόστηκε μια εμπορικά διαθέσιμη ηλεκτρική αρπάγη και χρησιμοποιήθηκε για την συνεργασία. Η εξαγωγή του απαραίτητου μοντέλου urdf όπου περιγράφεται με τυποποιημένο τρόπο η δομή του ρομπότ έγινε από το περιβάλλον του Solidworks χρησιμοποιώντας κατάλληλο plugin. Προκειμένου να δημιουργηθεί το πακέτο του ROS, χρησιμοποιήθηκαν δύο πλατφόρμες που λειτουργούν κάτω από τον πυρήνα του Ubuntu. Για την απεικόνιση του ρομπότ και των κινήσεων στον χώρο χρησιμοποιήθηκε η πλατφόρμα MoveIt,. Επίσης, τα δεδομένα που συλλέχθηκαν από τους αισθητήρες laser στο περιβάλλον του λογισμικού gazebo απεικονίζονται σε πραγματικό χρόνο στο rviz. Η πλήρης εικόνα του ανθρώπου δόθηκε στο λογισμικό gazebo καθώς ορίστηκε η τροχιά που αυτός ακολουθεί εντός του χώρου εργασίας.

Δημιουργήθηκαν ορισμένα σενάρια συνεργασίας ρομπότ-ανθρώπου καθώς και απλά σενάρια τύπου “pick and place” ενός αντικειμένου. Η ακριβής καταγραφή της θέσης και του προσανατολισμού του ανθρώπου στον χώρο έδωσε την δυνατότητα να δημιουργηθούν σενάρια που καλύπτουν αρκετές περιπτώσεις, από ένα απλό σενάριο διακοπής της κίνησης του ρομπότ όταν ο άνθρωπος εισέρχεται στον χώρο εργασίας του ρομπότ μέχρι και την συνεργασία ανθρώπου-ρομπότ για το βίδωμα κοχλιών στον τρισδιάστατο χώρο.

Λέξεις κλειδιά

Stäubli, βιομηχανικό ρομπότ, συνεργασία ανθρώπου-ρομπότ, ROS, Gazebo, Ασφάλεια, Workspace, Simulation, Αισθητήρας laser, Python, C++, V+

Πίνακας περιεχομένων

Πίνακας εικόνων

1 Εισαγωγή

1.1 Γενικά

Η ρομποτική είναι ένα διεπιστημονικό πεδίο που ενσωματώνει την επιστήμη των υπολογιστών και τη μηχανική. Γενικά, περιλαμβάνει σχεδιασμό, κατασκευή, λειτουργία και χρήση ρομπότ. Ο στόχος της ρομποτικής είναι ο σχεδιασμός μηχανών που μπορούν να βοηθήσουν τους ανθρώπους. Η ρομποτική ενσωματώνει μεταξύ άλλων τομείς της μηχανολογίας, της ηλεκτρολογίας, της πληροφορικής, της μηχανοτρονικής, της μηχανικής υπολογιστών. [1]

Τα ρομπότ μπορούν να χρησιμοποιηθούν σε πολλές καταστάσεις και για πολλούς σκοπούς, αλλά σήμερα πολλά χρησιμοποιούνται σε επικίνδυνα περιβάλλοντα (συμπεριλαμβανομένης της επιθεώρησης ραδιενεργών υλικών, ανίχνευσης και απενεργοποίησης βομβών), διαδικασιών κατασκευής ή όπου οι άνθρωποι δεν μπορούν να επιβιώσουν (π.χ. στο διάστημα, υποβρύχια, σε υψηλά επίπεδα θερμοκρασίας, σε περιβάλλον επικίνδυνων υλικών και ακτινοβολίας).

Η ιδέα της δημιουργίας ρομπότ που μπορούν να λειτουργούν αυτόνομα χρονολογείται από την κλασική εποχή, αλλά η έρευνα για τη λειτουργικότητα και τις πιθανές χρήσεις των ρομπότ δεν αναπτύχθηκε ουσιαστικά μέχρι τον 20ο αιώνα. Καθ' όλη τη διάρκεια της ιστορίας, έχει θεωρηθεί συχνά από διάφορους μελετητές, εφευρέτες, μηχανικούς και τεχνικούς ότι τα ρομπότ θα μπορούν μια μέρα να μιμούνται την ανθρώπινη συμπεριφορά και να διαχειρίζονται εργασίες με τρόπο που μοιάζει με τον αντίστοιχο των ανθρώπων. Σήμερα, η ρομποτική είναι ένα ταχέως αναπτυσσόμενο πεδίο, καθώς συνεχίζονται οι τεχνολογικές εξελίξεις. Η έρευνα, ο σχεδιασμός και η κατασκευή νέων ρομπότ εξυπηρετούν διάφορους πρακτικούς σκοπούς σε πολλούς τομείς.

Όλα τα ρομπότ αποτελούν ένα είδος μηχανικής κατασκευής. Ο τρόπος κατασκευής του ρομπότ το βοηθά να ολοκληρώσει εργασίες στο περιβάλλον για το οποίο έχει σχεδιαστεί. Για παράδειγμα, οι τροχοί του Mars 2020 Rover, είναι ατομικά μηχανοκίνητοι και κατασκευασμένοι από σωλήνες τιτανίου που τον βοηθούν να κινείται σταθερά στο σκληρό έδαφος του κόκκινου πλανήτη. Τα ρομπότ χρειάζονται ηλεκτρικά εξαρτήματα που ελέγχουν και τροφοδοτούν τα επιμέρους υποσυστήματα και εμπλέκουν κάποιο επίπεδο προγραμματισμού. Χωρίς ένα σύνολο προγραμματιστικού κώδικα που θα δίνει τις απαιτούμενες εντολές, ένα ρομπότ θα ήταν απλώς ένα ηλεκτρομηχανικό συναρμολόγημα. [2]

Πολλές πτυχές της ρομποτικής περιλαμβάνουν τεχνητή νοημοσύνη. Τα ρομπότ μπορεί να είναι εφοδιασμένα με το ισοδύναμο των ανθρώπινων αισθήσεων όπως η όραση, η αφή και η ικανότητα να ανιχνεύουν τη θερμοκρασία. Μερικά ρομπότ είναι πλέον ικανά για απλή λήψη αποφάσεων, ενώ η τρέχουσα έρευνα ρομποτικής στρέφεται στην επινόηση ρομπότ με βαθμό αυτάρκειας που θα επιτρέψει την λειτουργία του καθώς και τη λήψη αποφάσεων σε μη δομημένο περιβάλλον. [3]

1.2 Ιστορική αναδρομή

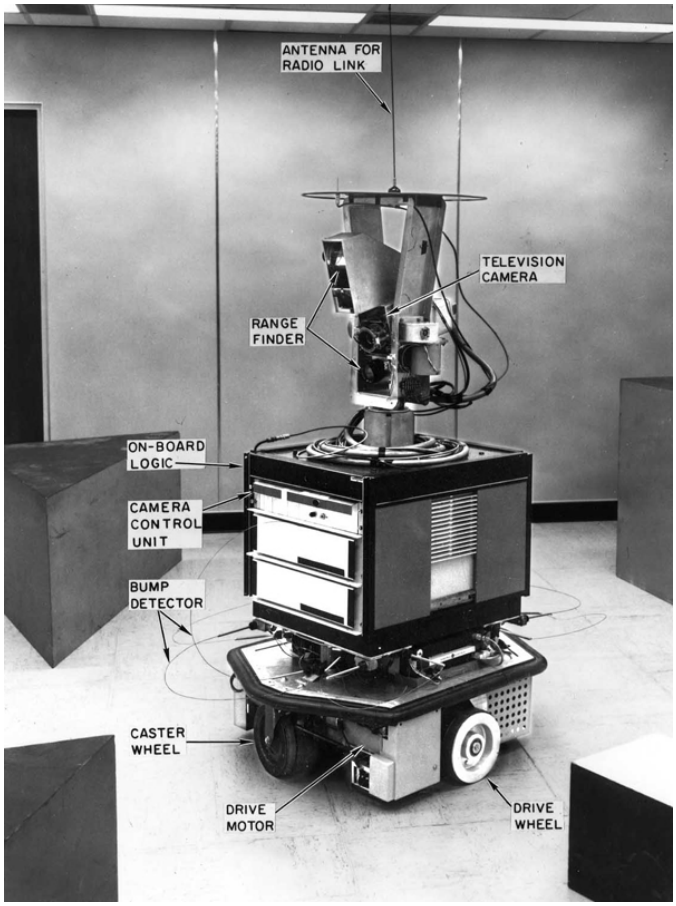
Παρόλο που η επιστήμη της ρομποτικής ήρθε μόλις τον 20ο αιώνα, η ιστορία των ρομπότ και του αυτοματισμού που εφευρέθηκε από τον άνθρωπο έχει πολύ μακρύτερο παρελθόν. Στην πραγματικότητα, ένας αρχαίος Έλληνας μηχανικός και εφευρέτης, ο Ήρων ο Αλεξανδρεύς, δημιούργησε δύο συγγραφικά έργα που σώθηκαν, το Πνευματικά Α,Β και το Αυτοματοποιητική, που μαρτυρούν την ύπαρξη εκατοντάδων διαφορετικών ειδών μηχανών που πραγματοποιούσαν αυτοματοποιημένες κινήσεις [4].

Η λέξη ρομποτική επινοήθηκε ακούσια από τον συγγραφέα επιστημονικής φαντασίας Isaac Asimov στο βιβλίο του 1941 «Liar!». Οι συγγραφείς της επιστημονικής φαντασίας σε όλη την ιστορία ενδιαφέρονται για την ικανότητα του ανθρώπου να παράγει αυτοκινούμενα μηχανήματα και μορφές ζωής, από τον αρχαίο ελληνικό μύθο του Πυγμαλίωνα έως τον Dr. Frankenstein της Mary Shelley και το HAL 9000 του Arthur C. Clarke. [4]

Το 1920, ο Karel Capek δημοσίευσε το έργο του R.U.R. (Rossum's Universal Robots), το οποία εισήγαγε τη λέξη «ρομπότ». Ο όρος προήλθε από μια παλιά σλαβική λέξη που σήμαινε κάτι παρόμοιο με «μονότονη ή καταναγκαστική εργασία». Ωστόσο, χρειάστηκαν περισσότερα από τριάντα χρόνια πριν ξεκινήσει να λειτουργεί το πρώτο βιομηχανικό ρομπότ. Στη δεκαετία του 1950, ο George Devol σχεδίασε το Unimate, μια ρομποτική συσκευή βραχίονα στο εργοστάσιο της General Motors στο Νιου Τζέρσεϋ, το οποίο άρχισε να λειτουργεί το 1961. Η εταιρεία Unimate που ιδρύθηκε από τον Devol με τον ανερχόμενο επιχειρηματία στον κλάδο των ρομπότ Joseph Engelberger, ήταν η πρώτη εταιρεία κατασκευής ρομπότ. [4]

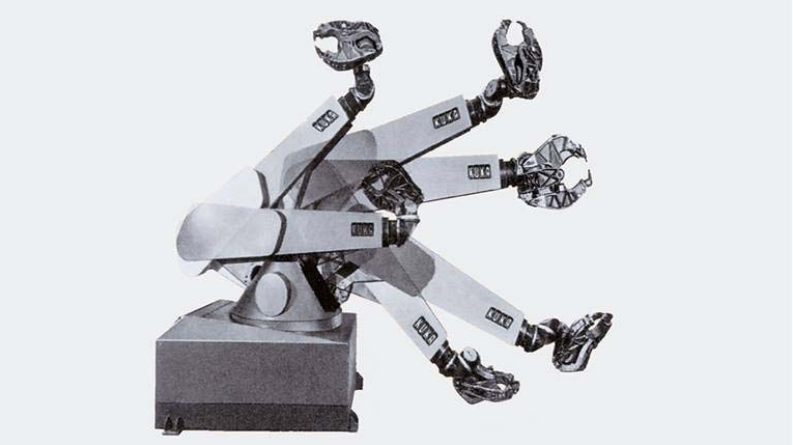
Εικόνα 1: Το πρώτο βιομηχανικό ρομπότ στην Ευρώπη.[5]

Επίσης, στα μέσα της δεκαετίας του 1950, η γερμανική εταιρεία Kuka ανέπτυξε μια αυτοματοποιημένη γραμμή συγκόλλησης για οικιακές συσκευές, καθώς και μια γραμμή συγκόλλησης πολλαπλών σημείων για τη Volkswagen. Μέχρι το 1968, η Kawasaki είχε παραδώσει τα σχέδια στην εταιρεία Unimate για την κατασκευή ενός υδραυλικού ρομπότ. Το 1969, η General Motors είχε επιτύχει το 90% των συγκολλήσεών της χρησιμοποιώντας ρομπότ Unimate σε ένα από τα εργοστάσιά της. Το 1970, το Πανεπιστήμιο του Στάνφορντ ανέπτυξε το λεγόμενο Stanford Arm, όπως είναι ακόμα γνωστό σήμερα, που χρησιμοποιείται για συναρμολόγηση μικρών ανταλλακτικών και ενσωματώνει ανατροφοδότηση αφής και πίεσης.[4]



Εικόνα 2: Κινούμενο ρομπότ με μηχανική όραση στο τμήμα έρευνας του Stanford το 1969.[5]

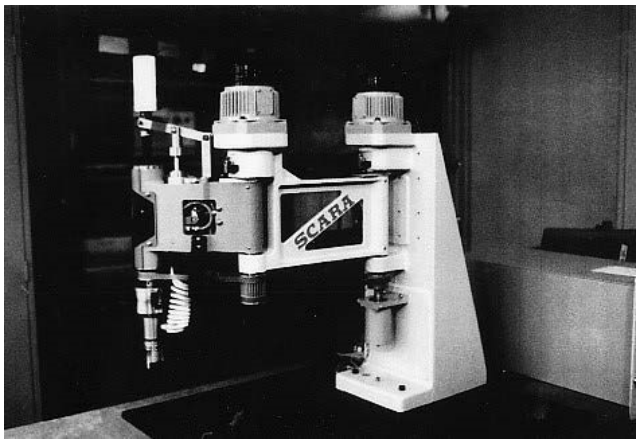
Η αυτοματοποιημένη συγκόλληση επρόκειτο να γίνει η σημαντική εφαρμογή βιομηχανικών ρομπότ, καθώς οι μηχανές μπορούσαν να παράγουν συγκολλήσεις υψηλής ποιότητας κάτω από αντίξοες συνθήκες. Η εταιρεία Kuka, το 1973, εισήγαγε τον βραχίονα έξι αξόνων, ο οποίος θα γινόταν το πρότυπο της βιομηχανίας. Ήταν περίπου την ίδια στιγμή που άρχισαν να εμφανίζονται τα πλήρως ηλεκτρικά ρομπότ. Η Cincinnati Milacron παρουσίασε ένα βιομηχανικό ρομπότ ελεγχόμενο από μικροϋπολογιστές για εμπορική χρήση τον ίδιο χρόνο.



Εικόνα 3: Το βιομηχανικό ρομπότ Famulus της Kuka το 1973.[6]

Το 1978, η Unimation κυκλοφόρησε, μαζί με τη General Motors, ένα νέο ανθρωπόμορφο ρομπότ με το όνομα PUMA (Programmable Universal Machine for Assembly). Το PUMA, θεωρήθηκε αρχέτυπο για τα ανθρωπόμορφα ρομπότ και η κινηματική του λήφθηκε ως παράδειγμα σε πολλά βιβλία ρομποτικής στην ακαδημία παγκοσμίως.

Τον ίδιο χρόνο (1978), ένα άλλο σημαντικό ορόσημο στην ιστορία των βιομηχανικών ρομπότ ήρθε, όταν ο Χιρόσι Μακίνο του Πανεπιστημίου Yamanashi (Ιαπωνία) εφεύρε το ρομπότ SCARA. Το ρομπότ αυτό, που πήρε το όνομά του από το ακρωνύμιο «Selective Compliance Assembly Robot Arm», είχε μια καινοτόμο κινηματική και ήταν τέλεια για τη συναρμολόγηση μικρών αντικειμένων. Η απλότητα της κινηματικής αλυσίδας έκανε τον έλεγχο εύκολο και πολύ γρήγορο. Επιπλέον, το κόστος ήταν σημαντικά χαμηλό, σε σύγκριση με άλλους τύπους χειριστών. Η διάδοση των ρομπότ SCARA έδωσε ώθηση στην παραγωγή ηλεκτρονικών καταναλωτικών αγαθών, τα οποία συναρμολογήθηκαν από αυτό τον τύπο ρομπότ.[7]



Εικόνα 4: Ρομπότ τύπου Scara.[8]

1.3 Βιομηχανικά ρομπότ

Ένα βιομηχανικό ρομπότ είναι ένα ρομποτικό σύστημα που χρησιμοποιείται στην παραγωγή. Τα βιομηχανικά ρομπότ είναι αυτοματοποιημένα, προγραμματιζόμενα και μπορούν να κινούνται σε τρεις ή περισσότερους άξονες. Οι τυπικές εφαρμογές ρομπότ περιλαμβάνουν συγκόλληση, βαφή, συναρμολόγηση, αποσυναρμολόγηση, λήψη και τοποθέτηση ("pick and place") και παλετοποίηση αντικειμένων. Όλα επιτυγχάνονται με υψηλή αντοχή, ταχύτητα και ακρίβεια. [9]

Τα βιομηχανικά ρομπότ χωρίζονται σε τύπους ανάλογα με την μηχανική τους διάταξη.[10] Αυτοί οι τύποι είναι:

- Καρτεσιανά ρομπότ: είναι τα ρομπότ που αποτελούνται από τρείς πρισματικές αρθρώσεις και οι άξονές τους συνδέονται με ένα καρτεσιανό σύστημα συντεταγμένων.
- Τύπου SCARA: έχουν δυο παράλληλες περιστροφικές αρθρώσεις
- Αρθρωτά ρομπότ: έχουν τουλάχιστον τρείς περιστροφικές αρθρώσεις
- Parallel/Delta robots: έχουν ταυτόχρονα πρισματικές ή περιστροφικές αρθρώσεις σε κλειστή κινηματική αλυσίδα
- Κυλινδρικά ρομπότ: οι άξονές του σχηματίζουν ένα κυλινδρικό σύστημα συντεταγμένων

Εικόνα 5: Delta ρομπότ της εταιρείας Abb.[11]

Οι βασικές παράμετροι που χαρακτηρίζουν ένα βιομηχανικό ρομπότ είναι οι εξής [9]:

- ***Αριθμός αξόνων***: Γενικά απαιτούνται δύο άξονες για να φτάσει σε οποιοδήποτε σημείο ενός επιπέδου ενώ απαιτούνται τρεις άξονες για να φτάσει σε οποιοδήποτε σημείο του χώρου.
- ***Βαθμοί ελευθερίας***: Σχετίζεται και πολλές φορές ταυτίζεται με τον αριθμό αξόνων του ρομπότ.
- ***Χώρος εργασίας (workspace)***: Είναι το υποσύνολο του τρισδιάστατου χώρου στο οποίο το ρομπότ έχει πρόσβαση και εκτελεί κάνει τις εργασίες του.
- ***Κινηματική***: Αποτελεί την διάταξη των μελών του ρομπότ, η οποία καθορίζει τις πιθανές κινήσεις του ρομπότ. Οι τύποι κινηματικής ρομπότ περιλαμβάνουν αρθρωτά, καρτεσιανά, παράλληλα και τύπου SCARA ρομπότ όπως αναφέρθηκε προηγουμένως.
- ***Μεταφορική ικανότητα ή ωφέλιμο φορτίο***: Είναι το βάρος που μπορεί να ανυψώσει ένα ρομπότ με το τελικό στοιχείο δράσης του (end-effector).
- ***Ταχύτητα:*** Είναι το μέγεθος που μετρά πόσο γρήγορα το ρομπότ μπορεί να μετακινηθεί από μια θέση σε μια άλλη για να εκτελέσει μια εργασία. Συνήθως αναφέρεται στο τελικό σημείο δράσης, αλλά προφανώς είναι συνάρτηση των επιμέρους ταχυτήτων των αρθρώσεων και της πόζας (εκάστοτε διαμόρφωσης στο χώρο) της κινηματικής αλυσίδας.
- ***Επιτάχυνση***: Αναφέρεται στους κινητήρες που οδηγούν τον βραχίονα και αποτελεί συνάρτηση του φορτίου που μεταφέρει το ρομπότ και συνδέεται με την κινηματική αλυσίδα.
- ***Ακρίβεια***: Περιγράφει πόσο καλά μπορεί ένα ρομπότ να προσεγγίσει μια επιθυμητή θέση. Συνήθως, οι encoders που χρησιμοποιούνται έχουν ανάλυση 0,01mm.
- ***Επαναληψιμότητα***: Περιγράφει πόσο κοντά μεταξύ τους είναι οι θέσεις στις οποίες βρίσκεται το ρομπότ κάθε φορά κατά την επαναληπτική εκτέλεση μιας προγραμματισμένης κίνησης.

Προκειμένου το ρομπότ να λειτουργήσει σε ένα βιομηχανικό περιβάλλον πρέπει πρώτα να προγραμματιστεί.[12] Υπάρχουν διάφοροι τρόποι με τους οποίους μπορεί να γίνει αυτό. Οι πιο βασικές μέθοδοι είναι οι εξής:

1) <u>*Διδασκαλία σε πραγματικό χρόνο με teach pendant*</u>: Πρόκειται για την πιο δημοφιλή μέθοδο προγραμματισμού ρομπότ. Σύμφωνα με την British Automation and Robot Association, πάνω από το 90% των ρομπότ προγραμματίζονται χρησιμοποιώντας αυτήν τη μέθοδο. Το teach pendant έχει αλλάξει πολύ καθ 'όλη τη διάρκεια της ζωής του, αλλά ουσιαστικά αποτελεί ένα φορητό υπολογιστή. Τα πρώτα teach pendant ήταν μεγάλα, γκρι κουτιά με σκληρούς δίσκους. Τα μοντέρνα μοιάζουν περισσότερο με ένα tablet αφής, καθώς η τεχνολογία έχει αναπτυχθεί για να ταιριάζει στους συνεχώς εξελισσόμενους χρήστες.

Εικόνα 6: Teach pendant της Comau.[13]

2) Προσομοίωση/προγραμματισμός εκτός γραμμής (*OLP)*: Αυτή η μέθοδος, χρησιμοποιείται συχνότερα στις ρομποτικές εφαρμογές για να διασφαλιστεί ότι οι αλγόριθμοι ελέγχου λειτουργούν σωστά προτού χρησιμοποιηθούν σε πραγματικές συνθήκες. Ωστόσο, χρησιμοποιείται επίσης για τη μείωση του χρόνου που το ρομπότ τίθεται εκτός λειτουργίας και τη βελτίωση της αποτελεσματικότητας. Η μέθοδος δεν επηρεάζει την παραγωγή και επιτρέπει προγραμματισμό χρησιμοποιώντας ένα εικονικό πρότυπο του ρομπότ και το περιβάλλον εργασίας του. Τα λογισμικά που χρησιμοποιούνται για αυτό τον σκοπό είναι εύκολα στην χρήση και μπορούν να προσομοιώσουν διαφορετικές ιδέες, τροχιές κλπ πριν την εφαρμογή στο πραγματικό περιβάλλον.

Εικόνα 7: Προγραμματισμός συγκόλλησης με το λογισμικό RoboDK.[14]

1.4 Ασφάλεια στον χώρο εργασίας

Η αλληλεπίδραση ανθρώπου-ρομπότ είναι κάτι σχετικά νέο που έχει προσελκύσει πολλή προσοχή τα τελευταία χρόνια λόγω της αυξανόμενης κατασκευής σύνθετων ρομπότ και της συνύπαρξης του ανθρώπου στην καθημερινή τους ζωή, π.χ. ως ρομποτικά παιχνίδια ή, ως ένα βαθμό, ως οικιακές συσκευές (ρομποτικές

ηλεκτρικές σκούπες). Τα ρομπότ αναπτύσσονται όλο και περισσότερο και χρησιμοποιούνται σε περιβάλλοντα που υπάρχει και ο ανθρώπινος παράγοντας.[15]

Τα συνεργατικά ρομπότ έχουν σχεδιαστεί για να συνυπάρχουν και να συνεργάζονται με ανθρώπους σε εφαρμογές όπως υποβοήθηση για χειρισμό βαρέων αντικειμένων, συνεργατική συναρμολόγηση, οικιακή εργασία, ψυχαγωγία, αποκατάσταση ή ιατρικές εφαρμογές. Είναι σαφές ότι τέτοια ρομπότ πρέπει να πληρούν διαφορετικές απαιτήσεις από αυτές που τυπικά πληρούνται σε συμβατικές βιομηχανικές εφαρμογές. Ενώ μπορεί παράγοντες όπως η ακρίβεια να μην λαμβάνονται σοβαρά υπόψη, πρέπει να αντιμετωπιστούν ζητήματα ασφάλειας και αξιοπιστίας καθώς εμπλέκονται ανθρώπινες ζωές . Αυτό συνεπάγεται πολλές διαφορές στο σχεδιασμό και τον έλεγχο και ανοίγει νέες προκλήσεις για έρευνα.[16]

Εικόνα 8: Συνεργασία cobot-ανθρώπου στο εργοστάσιο της Ford.

Από την πρώτη στιγμή που τα ρομπότ χρησιμοποιήθηκαν στην βιομηχανία, δόθηκε μεγάλη προσοχή στην ασφάλεια. Η πρώτη γραμμή άμυνας ήταν πάντα η λήψη όλων των μέτρων για την επιβολή του διαχωρισμού μεταξύ ρομπότ και ανθρώπων. Ωστόσο, το παράδειγμα διαχωρισμού αποτυγχάνει σε περιπτώσεις όπου ο άνθρωπος και το ρομπότ πρέπει να μοιράζονται το ίδιο περιβάλλον και σε εφαρμογές στις οποίες η επιτυχής ολοκλήρωση εργασιών απαιτεί συνεργασία.[16]

Τα δεδομένα τραυματισμών ή και θανάτων που σχετίζονται με βιομηχανικά ρομπότ δείχνουν ότι, ακόμη και σε παραδοσιακές εφαρμογές βιομηχανικών ρομπότ, η ασφάλεια δεν είναι ένα λυμένο πρόβλημα - ειδικά λόγω όλων των φάσεων λειτουργίας όπου ο ανθρώπινος χειριστής είναι αναγκαστικά κοντά στον βραχίονα. Μελέτες έχουν δείξει ότι πολλά ατυχήματα ρομπότ δεν συμβαίνουν υπό κανονικές συνθήκες λειτουργίας, αλλά κατά τη διάρκεια του προγραμματισμού, της προσαρμογής ή της βελτίωσης του προγράμματος, της συντήρησης, της επισκευής,

των δοκιμών, της εγκατάστασης ή της προσαρμογής. Κατά τη διάρκεια πολλών από αυτές τις λειτουργίες, ο χειριστής, ο προγραμματιστής ή συντηρητής μπορεί προσωρινά να βρίσκεται εντός του χώρου εργασίας (workspace) του ρομπότ όπου η ακούσια λειτουργία μπορεί να οδηγήσει σε ατυχήματα.[16]

1.5 Στόχος της διπλωματικής εργασίας

Η συγκεκριμένη εργασία έχει ως στόχο την ενσωμάτωση του ρομποτικού βραχίονα Stäubli RX90L στο περιβάλλον του ROS (Robot Operating System) προκειμένου να επιτευχθεί συνεργασία ανθρώπου-ρομπότ στο περιβάλλον της προσομοίωσης, Αυτό περιλαμβάνει την εισαγωγή του μοντέλου του ρομπότ στο περιβάλλον, την δημιουργία του απαραίτητου πακέτου καθώς και την ανάπτυξη σχετικών αλγορίθμων..

Η εργασία μπορεί να χωριστεί στις ακόλουθες θεματικές ενότητες :

- Στο *πρώτο μέρος,* γίνεται αρχικά μια εισαγωγή της εργασίας. Παρουσιάζονται ορισμένες βασικές έννοιες για τα βιομηχανικά ρομπότ, γίνεται μια ιστορική αναδρομή και αναφορά στην ασφάλεια στον χώρο εργασίας. Τέλος, γίνεται ανάλυση της συνεργασίας ανθρώπου-ρομπότ με έμφαση στην βιομηχανική ασφάλεια.
- Στο *δεύτερο μέρος,* γίνεται περιγραφή των χαρακτηριστικών και της γλώσσας προγραμματισμού V+ του ρομποτικού βραχίονα που χρησιμοποιήθηκε. Στην συνέχεια, αναπτύσσεται η αρχιτεκτονική του περιβάλλοντος του ROS καθώς και ο τρόπος σύνδεσης του ρομπότ σε αυτό.
- *Στο τρίτο μέρος,* γίνεται παρουσίαση των σεναρίων που υλοποιήθηκαν για την συνεργασία ανθρώπου-ρομπότ. Σε κάθε σενάριο, υπάρχει το αντίστοιχο διάγραμμα ροής και επεξηγείται ο κώδικας που αναπτύχθηκε και χρησιμοποιήθηκε.

2 Συνεργασία ανθρώπου-ρομπότ

2.1 Γενική περιγραφή

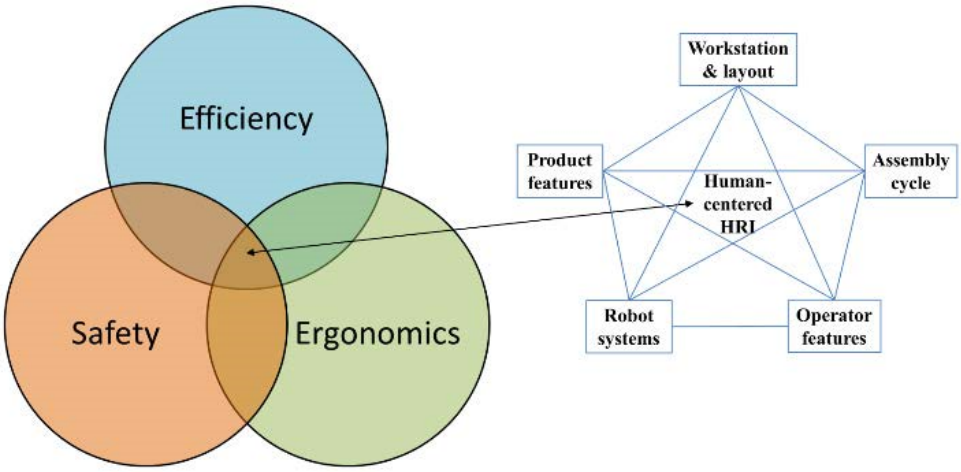
Η αρχή της 4ης βιομηχανικής επανάστασης (Industry 4.0) και ο αυτοματισμός στην παραγωγή και την συναρμολόγηση έχει επιφέρει ένα σημαντικό αριθμό αλλαγών. Ενώ ο αυτοματισμός στο παρελθόν προγραμματιζόταν ανεξάρτητα από τον ανθρώπινο παράγοντα, λόγω του διαχωρισμού των διαδικασιών αυτοματισμού από τις χειροκίνητες δραστηριότητες, αυτό έχει αλλάξει σημαντικά μες τα σημερινά δεδομένα. Ο χειριστής συνεργάζεται απευθείας με την μηχανή ή το ρομπότ. Συνεπώς, με την εισαγωγή των συνεργατικών ρομπότ (collaborative robots) πχ στην συναρμολόγηση, έγινε απαραίτητο να γίνει το περιβάλλον εργασίας πιο ασφαλές και εργονομικό. Τα συνεργατικά ρομπότ ενσωματώνουν μέτρα προστασίας που τους επιτρέπουν να εργάζονται με ασφάλεια στο ίδιο περιβάλλον με τον άνθρωπο, όμως αυτή η κατάσταση συνήθως αλλάζει όταν προστίθενται εργαλεία. όπως πχ μια αρπάγη. που μπορούν να προκαλέσουν τραυματισμό. [17]

Η αλληλεπίδραση ανθρώπου-ρομπότ είναι ένα από τα πιο συζητημένα θέματα στα σύγχρονα ρομποτικά συστήματα. Η αλληλεπίδραση ανθρώπου-ρομπότ (HRI) θα μπορούσε να παρουσιαστεί από διαφορετικές οπτικές γωνίες. Η πρώτη περίπτωση αφορά αλληλεπίδραση σε κοινόχρηστο χώρο εργασίας χωρίς άμεση επαφή ανθρώπου-ρομπότ και συγχρονισμό των καθηκόντων τους. Δεύτερη περίπτωση αφορά μια κοινή διαδικασία εκτέλεσης εργασιών, λαμβάνοντας υπόψη την προσαρμογή των κινήσεων του ρομπότ σύμφωνα με την ανθρώπινη κίνηση σε πραγματικό χρόνο. Αυτό είναι ένα από τα κεντρικά προβλήματα του HRI, το οποίο συναρτάται με την κατανομή των λειτουργιών μεταξύ ανθρώπου και ρομπότ. Είναι απαραίτητο να καθοριστεί ποιες λειτουργίες μπορούμε να μεταφέρουμε σε ένα ρομπότ που διαθέτει στοιχεία τεχνητής νοημοσύνης (AI) και ποιες σε ένα άτομο που διαθέτει γνωστικές ικανότητες προκειμένου να επιτευχθεί η μέγιστη αποτελεσματικότητα από την αλληλεπίδρασή τους. Ένα ενδιαφέρον γεγονός είναι η τήρηση των νόμων που το 1942 πρότεινε ο συγγραφέας επιστημονικής φαντασίας Asimov. Οι τρεις «Νόμοι της Ρομποτικής» που ισχύουν για την αλληλεπίδραση ανθρώπου-ρομπότ είναι οι εξής:[18]

1. *Πρώτος νόμος*: Ένα ρομπότ δεν μπορεί να τραυματίσει έναν άνθρωπο ή, μέσω αδράνειας, να επιτρέψει σε έναν άνθρωπο να τραυματιστεί.
2. *Δεύτερος νόμος*: Ένα ρομπότ πρέπει να υπακούει στις εντολές που του έδωσαν τα ανθρώπινα όντα, εκτός εάν οι αυτές έρχονται σε αντίθεση με τον πρώτο νόμο.
3. *Τρίτος νόμος*: Ένα ρομπότ πρέπει να προστατεύει τη δική του ύπαρξη, εφόσον αυτή η προστασία δεν έρχεται σε αντίθεση με τον πρώτο ή το δεύτερο νόμο.

Αυτοί οι νόμοι, μπορούν να εγγυηθούν την ασφάλεια της αλληλεπίδρασης ανθρώπου-ρομπότ. Στην πράξη, τα ζητήματα της αλληλεπίδρασης με την ασφάλεια είναι πολύ ευρύτερα και πιο περίπλοκα. Η ρομποτική εξελίσσεται συνεχώς. Έχει

περάσει πολύς καιρός από τότε που ο Isaac Asimov έγραψε για πρώτη φορά τους νόμους του για τα ρομπότ και ο συνεχώς διευρυνόμενος ρόλος του στη ζωή μας απαιτεί ένα ριζοσπαστικό νέο σύνολο κανόνων. Η διαδικασία συνεργασίας μεταξύ ανθρώπου και ρομπότ ονομάζεται Human-Robot-Collaboration (HRC). Σε αυτήν την κατηγορία ρομπότ ανήκουν οι βραχίονες που ονομάζονται συνεργατικά ρομπότ (cobots).[18]



Εικόνα 9: Βασικοί παράγοντες για συνεργατικά ρομπότ.[17]

Η εφαρμογή σχεδιασμού ασφάλειας για συμβατικά συστήματα ρομπότ συνεπαγόταν παραδοσιακά την απομόνωση ή εγκλωβισμό ενός ρομποτικού συστήματος μακριά από τον άνθρωπο-χειριστή για την αποτροπή επικίνδυνων αλληλεπιδράσεων. Η απομόνωση του βιομηχανικού ρομπότ είναι εφικτή για επαναλαμβανόμενα καθήκοντα παραγωγής που είναι κατάλληλα για ρομποτικό βραχίονα, αλλά είναι εξ ορισμού αδύνατη για περιπτώσεις που απαιτείται συνεργασία και αλληλεπίδραση με τον άνθρωπο στον ίδιο χώρο εργασίας. [19] Συνεπώς, για σωστή χρήση ενός συνεργατικού ρομπότ για την υποστήριξη του ανθρώπου, είναι καλό να μιλάμε για ανθρωποκεντρικό σχεδιασμό της Παραγωγής. Αυτός βασίζεται στην κατανόηση των συνθηκών εργασίας του ανθρώπου, βελτιώνοντας την ανθρώπινη ικανοποίηση και μειώνοντας τις αρνητικές επιδράσεις. Πρακτικά, γεννιούνται νέες προκλήσεις στον χώρο της υγιεινής και ασφάλειας των εργαζομένων και αφορούν ζητήματα όπως:[17]

- Πως γίνεται η διαχείριση των κινδύνων
- Πως ενσωματώνεται μια τέτοια εργονομική λύση
- Πως βελτιστοποιείται η χρήση των πόρων (άνθρωπος, ρομπότ)

2.2 Σύγκριση cobots με συμβατικά ρομπότ

Οι κύριες διαφορές μεταξύ των συνεργατικών ρομπότ (cobots) και των κοινών ρομπότ είναι:[18]

- *Δυνατότητα ασφαλούς αλληλεπίδρασης με τον άνθρωπο*. Τα cobots έχουν σχεδιαστεί ειδικά για να συνεργάζονται με ανθρώπους. Κατά τη χρήση τους, δεν απαιτούνται προστατευτικά εμπόδια. Επίσης, βοηθούν στην επίλυση πολύπλοκων εργασιών που δεν μπορούν να αυτοματοποιηθούν πλήρως.
- *Μείωση κινδύνου κατά την εκτέλεση επικίνδυνων εργασιών*. Τα cobots εκτελούν λειτουργίες που ενέχουν κίνδυνο για τον άνθρωπο. Αυτοί οι κίνδυνοι περιλαμβάνουν την ασφάλεια της μεταφοράς αιχμηρών και κοφτερών ή καυτών αντικειμένων, σύσφιξη μπουλονιών κ.λπ.
- *Ευελιξία και μάθηση*. Ο προγραμματισμός των συνεργατικών ρομπότ είναι ευκολότερος από τα βιομηχανικά ρομπότ. Μερικά cobots είναι ικανά για αυτομάθηση.
- *Δυνατότητα ευρείας χρήσης και γρήγορη προσαρμογή*. Τα συνεργατικά ρομπότ δεν είναι μόνο εύκολο να επαναπρογραμματιστούν, αλλά και σχετικά εύκολο να μετακινηθούν και να χρησιμοποιηθούν σε άλλα σημεία της αλυσίδας παραγωγής.

Εικόνα 10: Cobot της Universal Robots σε εταιρία που ασχολείται με κατεργασίες μετάλλου.[20]

Υπάρχουν πολλοί παράγοντες κόστους που επηρεάζουν τη χρήση συνεργατικών ρομποτικών λύσεων, όπως:

- ✓ **Κόστος των ρομποτικών εξαρτημάτων**. Το κόστος των εξαρτημάτων του ρομπότ μπορεί να είναι πολύ μεγάλο(πχ αρπάγη). Η προσαρμογή των στοιχείων του ρομπότ εξαρτάται από την εργασία. Το κόστος της μηχανολογικής μελέτης είναι 30-50% του συνολικού κόστους του εγκατεστημένου συστήματος.

- ✓ **Κόστος εγκατάστασης**. Ακόμα κι αν μερικές φορές το κόστος εγκατάστασης προστίθεται άμεσα στο κόστος του ίδιου του ρομπότ, σε πολλές περιπτώσεις είναι απαραίτητο να αλλαχθεί η διάταξη του εργοστασίου με τον ένα ή τον άλλο τρόπο κατά την αλλαγή των παραγόμενων προϊόντων.

- ✓ **Κόστος της εκπαίδευσης**. Κατά τον επαναπροσδιορισμό του κόστους ενός ρομποτικού συστήματος, συχνά πολλοί εργαζόμενοι πρέπει να εκπαιδευτούν σε διάφορες πτυχές της εργασίας με ρομπότ. Η εγκατάσταση και η εκπαίδευση μπορεί συνήθως να κοστίζει 10-15% του συνολικού κόστους του συστήματος.

- ✓ **Κόστος συντήρησης**. Ανάλογα με το σχεδιασμό του ρομπότ και τις συνθήκες της εργασίας του θα απαιτηθούν διάφοροι βαθμοί προληπτικής συντήρησης, τακτική επισκευή και μικρές επισκευές. Το κόστος συντήρησης μπορεί να φτάσει ετησίως το 10% της αρχικής τιμής αγοράς.

- ✓ **Κόστος λειτουργίας**. Τα κύρια λειτουργικά κόστη συνήθως συνίστανται στο κόστος ενέργειας που απαιτείται για τη λειτουργία του ρομπότ και το κόστος του ελέγχου και του πρόσθετου εξοπλισμού.

- ✓ **Κόστος της εργασίας προγραμματισμού**. Πριν το ρομπότ μπορέσει να εκτελέσει την καθορισμένη εργασία, πρέπει να προγραμματιστεί και αυτό είναι ένα επιπλέον κόστος.

2.3 Ζητήματα ασφαλείας στην συνεργασία ανθρώπου-ρομπότ

Ο σκοπός των προτύπων ασφάλειας που έχουν δημιουργηθεί είναι η αύξηση της διαλειτουργικότητας των ρομπότ και των εξαρτημάτων τους. Επίσης, βοηθούν στη μείωση του κόστους ανάπτυξης και συντήρησής τους μέσω τυποποίησης και εναρμόνισης διαδικασιών, διεπαφών και παραμέτρων. Τα γενικά πρότυπα ασφαλείας στον τομέα Human-Robot-Interaction(HRI) κατατάσσονται σε τρεις κατηγορίες. Αυτές είναι:[18]

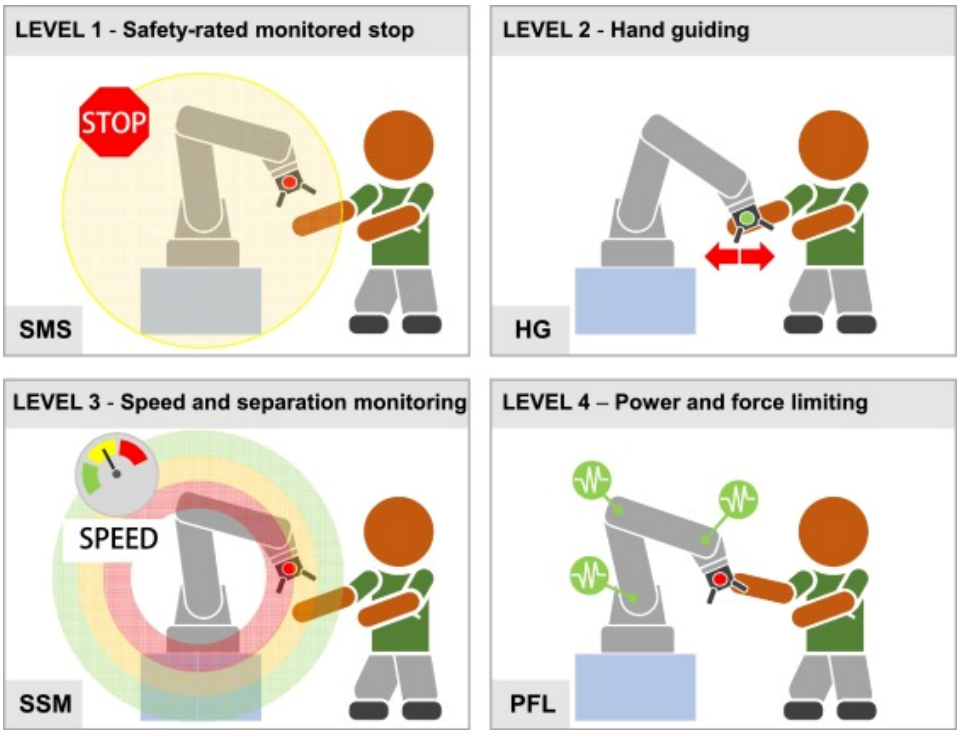
Α. Γενικές αρχές σχεδιασμού και εκτίμησης κινδύνου. Παρουσιάζονται γενικές απαιτήσεις για τη λειτουργική ασφάλεια ηλεκτρικών, ηλεκτρονικών, προγραμματιζόμενων ηλεκτρονικών συστημάτων που σχετίζονται με την ασφάλεια.

Β. Μέρη που σχετίζονται με την ασφάλεια των συστημάτων ελέγχου και συγκεκριμένες πτυχές ασφάλειας. Αρχές σχεδιασμού για λειτουργία έκτακτης ανάγκης.

Γ. Καθορίζει τις απαιτήσεις ασφάλειας για συνεργατικά βιομηχανικά ρομπότ συστήματα και το περιβάλλον εργασίας και συμπληρώνει τις απαιτήσεις και οδηγίες για τη συνεργατική βιομηχανική ρομποτική λειτουργία που παρέχονται στα ISO 10218-1 και ISO 10218-2.

Αυτή η ταξινόμηση δεν αποτελεί πλήρη περιγραφή όλων των προτύπων ασφαλείας. Υπάρχουν περισσότερα από 700 διαφορετικά πρότυπα και κανονισμοί. Οι κύριες προσεγγίσεις λειτουργίας στο πλαίσιο HRI που μπορούμε να εξετάσουμε μέσω της ταξινόμησης των γενικών προτύπων ασφαλείας εμπίπτουν σε τέσσερις συνεργατικές κατευθύνσεις:[18]

- "Παρακολούθηση λειτουργίας με βάση την ασφάλεια" - το ρομπότ σταματά όταν ο ανθρώπινος χειριστής εισέρχεται στο συνεργατικό χώρο εργασίας και συνεχίζει όταν ο χώρος εργασίας είναι ελεύθερος. Έτσι, επιτρέπεται η άμεση αλληλεπίδραση χειριστή με το σύστημα ρομπότ υπό συγκεκριμένες συνθήκες.
- «Χειροκίνητη καθοδήγηση» - οι κινήσεις ρομπότ ελέγχονται από τον ανθρώπινο χειριστή. Ο χειριστής χρησιμοποιεί μια χειροκίνητη συσκευή για τη μετάδοση εντολών κίνησης.
- «Παρακολούθηση ταχύτητας και διαχωρισμού» - ο χειριστής και το ρομπότ ενδέχεται να κινούνται ταυτόχρονα στον χώρο εργασίας.
- «Περιορισμός ισχύος και δύναμης» - οι δυνάμεις επαφής μεταξύ ανθρώπινου χειριστή και κινούμενου ρομπότ περιορίζονται τεχνικά σε ασφαλές επίπεδο.



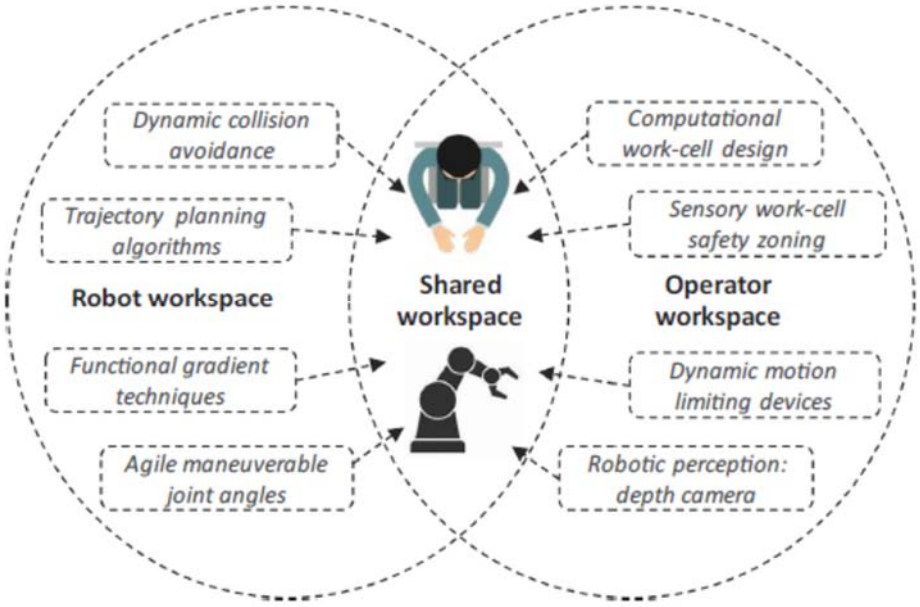
Εικόνα 11: Προσεγγίσεις λειτουργίας στο Human-Robot-Interaction.[21]

Αυτές οι προσεγγίσεις σχετίζονται με τις πιο δυσνόητες πτυχές της συνεργασίας ανθρώπου-ρομπότ. Σε κάθε περίπτωση, το ρομπότ και ο άνθρωπος μοιράζονται έναν κοινό χώρο εργασίας. Γενικά, θεωρούνται στοιχεία ενός σεναρίου και όχι μεμονωμένες πτυχές των τρόπων συνεργασίας ανθρώπου-ρομπότ. Σύμφωνα με τον Robert Nelson Shea, η ιδέα της χρήσης συνεργατικών ρομπότ στη βιομηχανία αντιμετωπίστηκε με σκεπτικισμό, καθώς οι διαθέσιμες λύσεις στόχευαν στο να αποφευχθεί η άμεση επαφή μεταξύ ανθρώπου και ρομπότ. Με την πάροδο του χρόνου, η τάση έχει αλλάξει: το ρομπότ και ο άνθρωπος μπορούν να επιτύχουν καλύτερη απόδοση μαζί, αλλά ταυτόχρονα τηρώντας τα πρότυπα ασφαλείας να αποτρέψουν τον κίνδυνο τραυματισμών.[18]

2.4 Σχεδιασμός σταθμών εργασίας ανθρώπου-ρομπότ

Για την Ευρωπαϊκή Ένωση υπάρχει ο κανονισμός που αφορά στην ασφάλεια των συνεργατικών ρομπότ στον χώρο ασφαλείας ISO 15066. Προσφέρει ορισμένους κανόνες σχετικά με τον σχεδιασμό των γραμμών παραγωγής προκειμένου να επιτευχθεί η ασφάλεια καθώς και η απόδοση ανθρώπου-ρομπότ στον βιομηχανικό χώρο.

Το πρότυπο ISO 31000 (διαχείριση κινδύνων, αρχές και οδηγίες) προτείνει μια δομημένη, γενική οδηγία για την αξιολόγηση κινδύνων σε κρίσιμες εγκαταστάσεις ασφαλείας και περιγράφει διάφορες πιθανές μεθόδους εκτίμησης κινδύνου. Το ISO 31000 περιγράφει τρεις κύριες φάσεις για τη δομή των διαδικασιών εκτίμησης κινδύνων, ευθυγραμμίζοντας έτσι τον κίνδυνο και το ρίσκο που υπάρχει σε κοινούς χώρους εργασίας, με προτάσεις σχεδιασμού όπως το ISO 15066.[17]



Εικόνα 12: Απεικόνιση διάφορων τεχνικών σχεδιασμού συνεργασίας A-P.[19]

Οδηγίες ως προς την ασφάλεια

- Ελαχιστοποίηση της πιθανότητας σύγκρουσης του βραχίονα με τον άνθρωπο
    a. Ορίζοντας τροχιές με τέτοιο τρόπο ώστε ο άνθρωπος να μην παγιδεύεται εύκολα ανάμεσα στο ρομπότ και στα αντικείμενα του περιβάλλοντος.
    b. Μειώνοντας την ταχύτητα των κινούμενων μερών
    c. Μειώνοντας τον χώρο στον οποίο μπορεί να δραστηριοποιηθεί το ρομπότ (workspace)
    d. Θέτοντας όρια στην ταχύτητα του ρομπότ εν ώρα συνεργασίας
    e. Ελαχιστοποιώντας την ροπή του βραχίονα από το λογισμικό
    f. Χρησιμοποιώντας αισθητήρες αφής για να ελέγχεται η επαφή
    g. Σχεδιάζοντας το τελικό στοιχείου δράσης ώστε να προσφέρει ασφάλεια

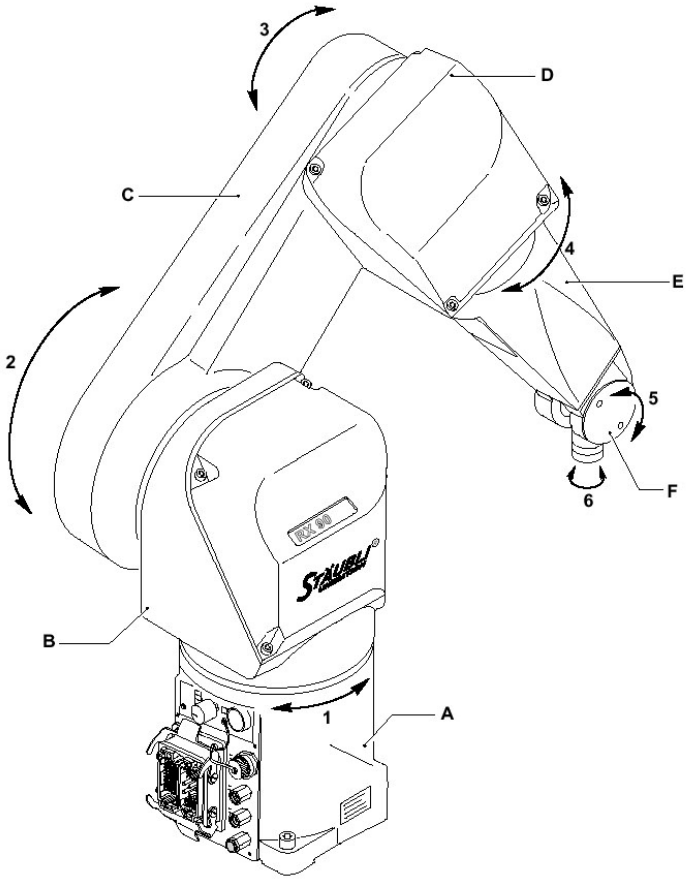
Οδηγίες ως προς την εργονομία

- Ελαχιστοποίηση κούρασης του ανθρώπου από επαναλαμβανόμενες εργασίες
    a. Αποφυγή εργασιών όπου απαιτείται ο άνθρωπος να έχει υψωμένα τα χέρια του για πολλή ώρα
    b. Αποφυγή εργασιών που απαιτείται εφαρμογή δύναμης από τον άνθρωπο
    c. Αποφυγή εργασιών που απαιτούν γρήγορες κινήσεις
    d. Αποφυγή περιβάλλοντος που απαιτείται επαφή με τα δάκτυλα για πολλή ώρα κατά την διάρκεια της συναρμολόγησης
- Ελαχιστοποίηση κούρασης του ανθρώπου από λήψη αντικειμένων
    a. Αποφυγή περιβάλλοντος όπου τα αντικείμενα είναι μακριά από τον άνθρωπο, π.χ. κατά την συναρμολόγηση
    b. Αποφυγή εργασιών στις οποίες απαιτούνται συχνές κινήσεις
    c. Μείωση του βάρος ή υποστήριξη των βαρέων αντικειμένων

3 Το ρομπότ Stäubli RX90L

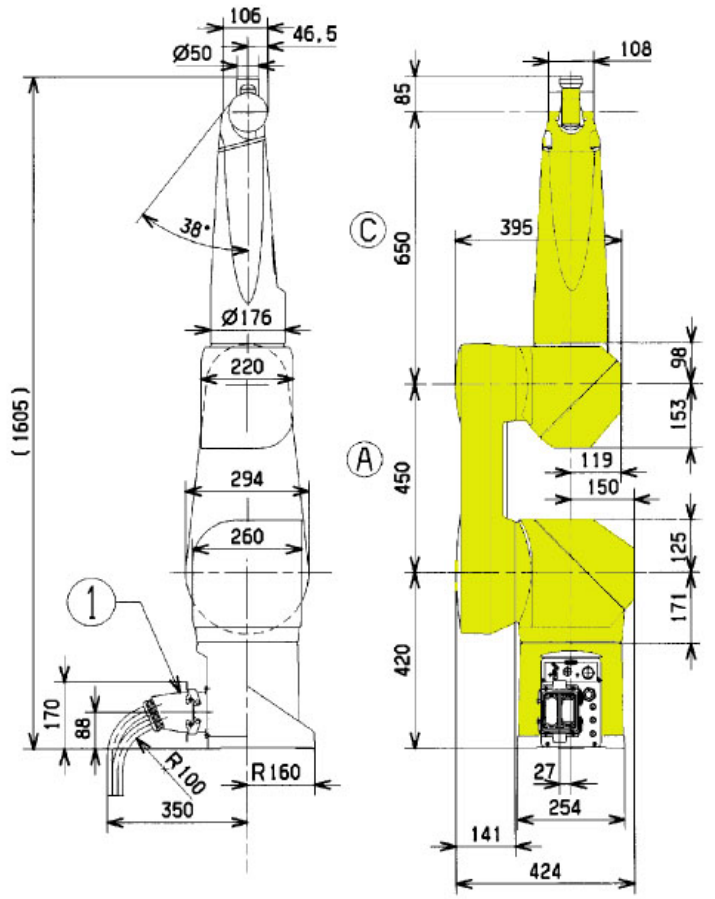
3.1 Περιγραφή

Το ρομπότ που χρησιμοποιήθηκε σε αυτή την εργασία είναι το Stäubli RX90L. Πρόκειται για ένα βιομηχανικού τύπου ρομπότ με σχετικά μικρή μεταφορική ικανότητα και μεγάλη ακρίβεια που χρησιμοποιείται σε εφαρμογές συναρμολόγησης καθώς και σε περιπτώσεις "pick and place". Το ρομπότ αποτελείται από έξι περιστροφικές αρθρώσεις όπως φαίνεται και στην παρακάτω εικόνα.

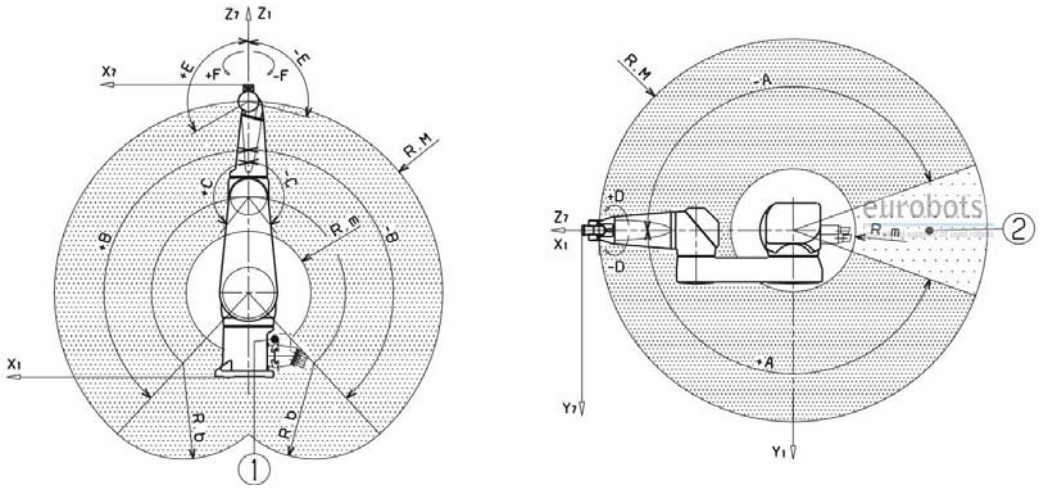


Εικόνα 13: Ρομπότ Stäubli RX90L

Το σχετικά μικρό του μέγεθος το καθιστά ιδανικό για πολλές εργασίες και του προσδίδει μεγάλη συμβατότητα με βιομηχανικά περιβάλλοντα. Στις παρακάτω εικόνες φαίνονται οι διαστάσεις του ρομπότ.



Εικόνα 14: Διαστάσεις Stäubli RX90L



Εικόνα 15: Χώρος εργασίας (workspace)

- R.M μέγιστη απόσταση μεταξύ 2ου και 5ου άξονα: 1100mm
- R.m ελάχιστη απόσταση μεταξύ 2ου και 5ου άξονα: 401mm
- R.b απόσταση μεταξύ 3ου και 5ου άξονα: 650mm

3.2 Χαρακτηριστικά

- Μέγιστο φορτίο: 3,5 kg (6 kg σε μειωμένη ταχύτητα)
- Μέγιστη ταχύτητα: 12,6 m/s
- Επαναληψιμότητα: 25 μm
- Βάρος: 113 kg

Αναλυτικά χαρακτηριστικά των αρθρώσεων του βραχίονα:

Άρθρωση	1	2	3	4	5	6
Εύρος γωνίας (°)	320	275	285	540	225	540
Εύρος εργασίας (ο)	± 160	±137.5	±142.5	±270	+120 -105	±270
Ονομαστική ταχύτητα (°/s)	236	200	286	401	320	580
Μέγιστη ταχύτητα (°/s)	356	356	296	409	800	1125
Γωνιακή ανάλυση ($^{o}.10^{-3}$)	0.87	0.87	0.72	1	1.95	2.75

3.3 Μονάδα ελέγχου CS7

3.3.1 Καμπίνα

Ο ελεγκτής του ρομπότ βρίσκεται μέσα σε μια κλειστή καμπίνα πλήρως αεριζόμενη. Στο εσωτερικό της συναντάμε όλα τα απαραίτητα ηλεκτρολογικά και ηλεκτρονικά στοιχεία προκειμένου το ρομπότ να είναι λειτουργικό. Στο εμπρός μέρος της καμπίνας, όπως φαίνεται και στην παρακάτω εικόνα, υπάρχει ένα πάνελ από το οποίο μπορεί να δει κανείς την κατάσταση του ρομπότ, να ενεργοποιήσει τους άξονές του, να κάνει αλλαγή από την λειτουργία local σε network καθώς και να διαπιστώσει αν υπάρχει κάποιο πρόβλημα στην λειτουργία του μέσω της λυχνίας fault. Επιπλέον, υπάρχει και μια μπουτονιέρα-μανιτάρι για περιπτώσεις έκτακτης ανάγκης η οποία σταματά κάθε κίνηση του βραχίονα.

Εικόνα 16: Panel του ρομπότ.

3.3.2 Χειριστήριο (teach pendant)

Στο μπροστινό πάνελ, συναντάμε μια θύρα για την σύνδεση του ρομπότ με το teach pendant. Το teach pendant ενσωματώνει τους ακόλουθους τρείς τρόπους (modes) διδασκαλίας:

- Λειτουργία διδασκαλίας εκφρασμένη στην βάση του ρομπότ (*world state*)

  Όταν έχει επιλεγεί το mode world τότε η κίνηση του ρομπότ στους άξονες x,y,z είναι παράλληλη σε ένα άξονα του χωρόδετου συστήματος συντεταγμένων. Αν για παράδειγμα, επιλεγεί το X1 στο teach pendant, πατώντας το + ή το – στην μπάρα ταχύτητας μετακινείται η φλάντζα του τελικού σημείου δράσης / εργαλείου (end effector) στον θετικό ή αρνητικό άξονα αντίστοιχα.

- Λειτουργία διδασκαλίας εκφρασμένη στο τελικό στοιχείο δράσης (*tool state*)

  Όταν έχει επιλεγεί το mode tool τότε η κίνηση στους άξονες είναι με βάση το τοπικό σωματόδετο σύστημα συντεταγμένων του εργαλείου. Αντίστοιχα με το world mode ακουμπώντας την μπάρα ταχύτητας μετακινούμαστε προς τα θετικά ή τα αρνητικά.

- Λειτουργία διδασκαλίας εκφρασμένη στον χώρο των αρθρώσεων (*joint state*)

  Όταν έχει επιλεγεί το mode joint τότε η κίνηση γίνεται γύρω από τον άξονα της άρθρωσης που έχει επιλεγεί προηγουμένως. Οι περιστροφικές αρθρώσεις

μπορούν να περιστραφούν η καθεμία μόνη της προς θετική ή αρνητική φορά, και η πρισματική άρθρωση μπορεί να μετακινηθεί αντίστοιχα. Επίσης, εάν έχουμε τοποθετήσει στο τελικό σημείο δράσης μια αρπάγη, αυτή μπορεί να ανοίξει ή να κλείσει αντίστοιχα.

- Λειτουργία διδασκαλίας με απελευθέρωση των αρθρώσεων (*free state*)

Όταν έχει επιλεγεί η συγκεκριμένη λειτουργία, οι επιλεγμένες αρθρώσεις από το teach pendant είναι ελεύθερες από τον έλεγχο των σερβοκινητήρων. Αυτό σημαίνει ότι ο χειριστής μπορεί να μετακινήσει με τα χέρια του τους συνδέσμους στις θέσεις που επιθυμεί προκειμένου να μάθει στο ρομπότ μια νέα θέση.

Τα functions keys που υπάρχουν πάνω στο χειριστήριο, επιτρέπουν στον χρήστη να ξεκινήσει ορισμένες λειτουργίες του ρομπότ. Το **EDIT**, επιτρέπει στον χρήστη να μεταβάλλει τις πραγματικές μεταβλητές ή τις μεταβλητές θέσης με το πληκτρολόγιο των αριθμών. Το **DISP**, αποτυπώνει στην οθόνη τις γωνίες των αρθρώσεων, την κατάσταση του συστήματος καθώς και σφάλματα που προέκυψαν από την χρήση. Το **CLR ERR** χρησιμοποιείται για να διαγραφεί ένα σφάλμα από την οθόνη και να επιστρέψει στην κανονική του λειτουργία. Το **CMD**, εμφανίζει στην οθόνη διάφορες λειτουργίες που σχετίζονται με τη βαθμονόμηση-calibration και την ενεργοποίηση των αποθηκευμένων προγραμμάτων. Το **PROG SET**, χρησιμοποιείται για την εκτέλεση προγραμμάτων που υπάρχουν στην μνήμη καθώς στην οθόνη εμφανίζονται παράμετροι όπως ο αριθμός των βημάτων, η ταχύτητα και η ενεργοποίηση του προγράμματος.

Στο teach pendant υπάρχουν επίσης πλήκτρα τα οποία επιτρέπουν την εναλλαγή των modes. Τα κουμπιά αυτά είναι τα ακόλουθα:

- Emergency stop

Σε περιπτώσεις έκτακτης ανάγκης ο χρήστης έχει την δυνατότητα να πιέσει και να ασφαλίσει το μπουτόν που υπάρχει πάνω στο χειριστήριο. Ενεργοποιώντας αυτή την λειτουργία κάθε κίνηση του ρομπότ σταματά. Για να συνεχιστεί η κίνηση πρέπει ο χρήστης να απασφαλίσει το κουμπί, να πατήσει το κουμπί COMP PWR και τέλος να πατήσει το κουμπί ARM POWER ON στο πάνελ του ελεγκτή.

- RUN/HOLD

Πιέζοντας αυτό το κουμπί διακόπτεται άμεσα η λειτουργία του ρομπότ. Πιέζοντάς το ξανά ξεκινά από εκεί που σταμάτησε.

- DIS PWR

Διακόπτει το ρεύμα στον βραχίονα

- COMP PWR

Προκειμένου να δοθεί ρεύμα στον βραχίονα πρέπει να πατηθεί το συγκεκριμένο κουμπί. Αμέσως μόλις γίνει αυτό και για 15 δευτερόλεπτα αναβοσβήνει το μπουτόν ARM POWER ON στο μπροστινό πάνελ το οποίο και πρέπει να πατηθεί προκειμένου να έχει ρεύμα το ρομπότ.

- MAN/HALT

Προκαλεί άμεσο σταμάτημα του ρομπότ και εμφανίζει στην οθόνη την ένδειξη E-STOP.

- Manual mode

Χρησιμοποιείται για να γίνει εναλλαγή στα modes του ρομπότ. Οι επιλογές για τον χρήστη είναι οι ακόλουθες και έχουν περιγραφεί αναλυτικά πιο πάνω: World, Tool, Joint, Free.

Εικόνα 17: Teach pendant

3.4 Γλώσσα προγραμματισμού V+

3.4.1 Γενική περιγραφή

Η V+ είναι μια γλώσσα προγραμματισμού που έχει αναπτυχθεί από την εταιρεία Adept και είναι ειδικά σχεδιασμένη για τον έλεγχο των ρομπότ της. Έχει όλα τα χαρακτηριστικά μιας σύγχρονης γλώσσας υψηλού επιπέδου όπως είναι οι κλάσεις και οι δομές. Η V+ «τρέχει» σε προγράμματα τύπου dumb terminals όπως είναι το Teraterm. Επιτρέπει σύνθετες κινήσεις για συνεχείς υπολογισμούς τροχιάς με μειωμένη χρήση υπολογιστικής ισχύος.

Το σύστημα που θα συνδεθεί (υπολογιστής) απαιτείται να έχει μια σειριακή θύρα RS232 προκειμένου να επιτευχθεί επικοινωνία με τον ελεγκτή του ρομπότ. Ο χρήστης έχει την δυνατότητα να επεξεργαστεί και να αποθηκεύει προγράμματα στην μνήμη του ελεγκτή.

Ο editor του controller μπορεί να εκτελέσει τα ακόλουθα:

- Τροποποίηση ενός αποθηκευμένου προγράμματος
- Εισαγωγή εντολών σε ένα πρόγραμμα
- Διαγραφή εντολών

Γενικά, μπορούμε να επεξεργαστούμε τα προγράμματά μας με δύο τρόπους. Ο line editor καλείται με την εντολή EDIT ενώ ο full-page editor με την εντολή SEE.

SEE prog.name: Με τον τρόπο αυτό, δημιουργείται ένα νέο πρόγραμμα την μνήμη του υπολογιστή με το όνομα που θα επιλέξουμε.

EDIT prog.name: Με αυτή την εντολή, γίνεται επεξεργασία του αλγορίθμου που είναι ήδη αποθηκευμένο στην μνήμη με το συγκεκριμένο όνομα.

3.4.2 Βασικές εντολές

- **EXECUTE prog.name**: εκτέλεση ενός αποθηκευμένου προγράμματος. Το όνομα που έχει οριστεί δεν πρέπει να ξεκινά ποτέ από αριθμό και δεν πρέπει να υπερβαίνει τους 15 χαρακτήρες.
- **ENABLE TRACE**: εμφανίζονται τα ίχνη-βήματα του προγράμματος που εκτελείται.
- **SET**: δήλωση σημείων συντεταγμένων στον χώρο
    - SET #a = #ppoint(j1_value, j2_value, j3_value, j4_value, j5_value, j6_value) – Σημείο με δήλωση περιστροφής των αρθρώσεων

- SET a = trans(X_value, Y_value, Z_value, y_value, p_value, r_value) – Καρτεσιανό σημείο

- **FDIRECTORY folder.name:** εμφάνιση όλων των αρχείων που εμπεριέχονται στον συγκεκριμένο φάκελο
- **DEF CD folder.name**: αλλαγή του φακέλου στον οποίο βρίσκεται ο χρήστης
- **STORE file.name = prog.name**: αποθηκεύει το πρόγραμμα που υπάρχει ήδη στην μνήμη RAM μέσα στον σκληρό δίσκο με το όνομα που θα δωθεί (file.name)
- **LOAD file.name:** φορτώνει στην μνήμη RAM το πρόγραμμα
- **FLIST file.name**: εμφανίζει τα περιεχόμενα του φακέλου
- **FCOPY file.name**: αντιγράφει ένα παλιό αρχείο σε ένα νέο
- **FRENAME file.name**: μετονομάζει το αρχείο
- **FDELETE file.name**: διαγράφει το αρχείο
- **MOVE setpoint.name**: το ρομπότ μετακινείται σε μια θέση που έχει προηγουμένως δηλωθεί και είναι γνωστή. Για παράδειγμα, το move a, δίνει την εντολή στο ρομπότ να πάει στο σημείο που έχει δηλωθεί με την εντολή set a.
- **DELAY time**: σταματά όλες τις κινήσεις του ρομπότ για ορισμένο χρονικό διάστημα
- **SPEED value**: θέτει το ποσοστό της ονομαστικής ταχύτητας των περιστροφικών αρθρώσεων. Αν προστεθεί η λέξη ALWAYS τότε η ταχύτητα παραμένει σταθερή καθ'όλη την διάρκεια εκτέλεσης του προγράμματος (πχ. SPEED 20 ALWAYS)

Για τα κεφάλαια 2.3 και 2.4 που αφορούν τον ελεγκτή του ρομπότ καθώς και την γλώσσα προγραμματισμού του χρησιμοποιήθηκαν περαιτέρω πληροφορίες υπάρχουν στα σχετικά manual [22]–[24]

3.5 Ηλεκτρική αρπάγη

Η αρπάγη της Robotiq έχει δυο δάκτυλα που το καθένα έχει δυο αρθρώσεις όπως φαίνεται στην παρακάτω εικόνα. Αυτός ο τύπος αρπάγης είναι ιδανικός για ένα απλό pick and place ενός αντικειμένου με χαμηλό βάρος. Τα δάκτυλα είναι υπο-ενεργούμενα που σημαίνει ότι υπάρχουν λιγότεροι κινητήρες από τον συνολικό

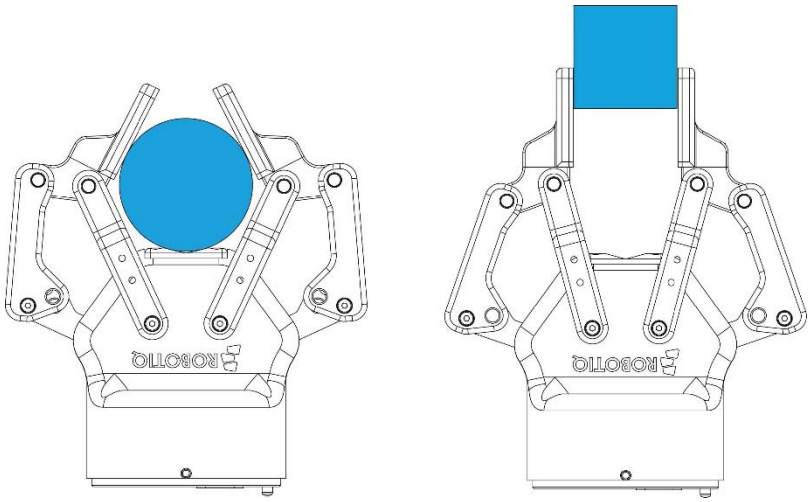
αριθμό των αρθρώσεων. Αυτή η διάταξη, επιτρέπει στην αρπάγη να προσαρμόζεται στο σχήμα του αντικειμένου και απλοποιεί τον έλεγχό της. Το άνοιγμα μεταξύ των δακτύλων της είναι 85mm. Τέλος, είναι πλήρως ηλεκτρική και δεν απαιτείται η σύνδεση με πεπιεσμένο αέρα. [25]

Εικόνα 18: Robotiq 2F-85

Στο κάτω μέρος υπάρχει μια ένδειξη κατάστασης LED. Το φως αλλάζει χρώμα ανάλογα με την κατάστασή της.

- Μπλε/κόκκινο κατά την εκκίνηση
- Μπλε σε κανονική λειτουργία χωρίς προβλήματα
- Κόκκινο αν υπάρχει πρόβλημα (επικοινωνία)
- Μπλε/κόκκινο (blink) αν υπάρχει σημαντικό θέμα

Η σύνδεση γίνεται με ένα απλό καλώδιο το οποίο περιέχει την τροφοδοσία του στα 24V DC καθώς και Modbus RTU επικοινωνίας μέσω θύρας RS-485.



Εικόνα 19: Προσαρμογή στο σχήμα του αντικειμένου

4 Το περιβάλλον ROS (Robot Operating System)

4.1 Περιγραφή

Το Robot Operating System (ROS) είναι μια σουίτα ρομποτικής ανοιχτού κώδικα. Παρόλο που το ROS δεν είναι λειτουργικό σύστημα, αλλά συλλογή framework για ανάπτυξη λογισμικού ρομπότ, παρέχει υπηρεσίες σχεδιασμένες για "heterogeneous computer cluster" όπως έλεγχος συσκευών χαμηλού επιπέδου, μετάδοση μηνυμάτων μεταξύ διεργασιών, και διαχείριση πακέτων. Τα τρέχοντα σύνολα διεργασιών που βασίζονται σε ROS αντιπροσωπεύονται από μια αρχιτεκτονική δικτύου όπου η επεξεργασία πραγματοποιείται σε κόμβους (nodes) που ενδέχεται να λαμβάνουν δεδομένα αισθητήρων, έλεγχο, κατάσταση και άλλα μηνύματα. Παρά τη σημασία της ανάδρασης και της χαμηλής καθυστέρησης μεταφοράς (latency) στον έλεγχο ρομπότ, το ίδιο το ROS δεν είναι λειτουργικό σύστημα πραγματικού χρόνου (RTOS). Ωστόσο, είναι δυνατό το ROS να ενσωματωθεί σε τέτοια συστήματα. Η έλλειψη υποστήριξης για συστήματα πραγματικού χρόνου αντιμετωπίστηκε με τη δημιουργία του ROS 2.0, μια σημαντική αναθεώρηση του ROS API που θα εκμεταλλευτεί τις σύγχρονες βιβλιοθήκες και τεχνολογίες για τη βασική λειτουργικότητα ROS και θα προσθέσει υποστήριξη για real-time coding.[26]

Το λογισμικό στο περιβάλλον του ROS μπορεί να χωριστεί σε τρείς κατηγορίες:

- Γλώσσες προγραμματισμού και εργαλεία για την υλοποίηση προγραμμάτων βασισμένων στο ROS
- Συνδέσεις με βιβλιοθήκες του ROS όπως roscpp (C++) και rospy (Python)
- Πακέτα που χρησιμοποιούν μια ή περισσότερες βιβλιοθήκες του ROS

Οι κύριες βιβλιοθήκες του ROS βασίζονται σε συστήματα τύπου Unix κυρίως λόγω της εξάρτησής τους από μια μεγάλη ποικιλία από ανοιχτού κώδικα λογισμικό. Για τον λόγο αυτό, το Ubuntu Linux θεωρούνται ως το κύριο λογισμικό για την ανάπτυξη του ROS. Παρ 'όλα αυτά υπάρχουν beta εκδόσεις και για λειτουργικά συστήματα Windows καθώς και MacOS.

Το κύριο χαρακτηριστικό του ROS είναι ο τρόπος που το λογισμικό «εκτελείται» και «επικοινωνεί». Δίνει την δυνατότητα σχεδιασμού πολύπλοκου λογισμικού ανεξάρτητα από τον τρόπο λειτουργίας του hardware. Προσφέρει ένα τρόπο σύνδεσης μεταξύ των συσκευών με nodes που διαχειρίζεται το σύστημα. Τα nodes μπορούν να υπάρχουν σε πολλαπλές συσκευές και να μοιράζονται δεδομένα με το κεντρικό σύστημα. Ο τρόπος με τον οποίο πραγματοποιείται αυτό είναι η δημιουργία των publisher/subscriber μεταξύ των nodes. Ο publisher μοιράζεται δεδομένα σε συγκεκριμένη μορφή (message) και ο subscriber τα λαμβάνει έχοντας έτσι αμφίδρομη επικοινωνία. Παρέχει όχι μόνο τυπικές υπηρεσίες λειτουργικού συστήματος (όπως hardware abstraction, contention management, process

management), αλλά και high-level λειτουργίες (ασύγχρονη και σύγχρονη επικοινωνία, κεντρική βάση δεδομένων, σύστημα διαμόρφωσης ρομπότ κ.λπ.). [27]

Το ROS-Industrial (ROS-I) είναι μια επέκταση των προηγμένων δυνατοτήτων του ROS στην αυτοματοποίηση και τη ρομποτική σε βιομηχανικά περιβάλλοντα. Το ROS-Industrial περιλαμβάνει διεπαφές για κοινούς βιομηχανικούς χειριστές, αρπάγες και αισθητήρες. Παρέχει επίσης βιβλιοθήκες λογισμικού για αυτόματη βαθμονόμηση αισθητήρα 2D / 3D, path planning, εφαρμογές όπως το Scan-N-Plan, εργαλεία προγραμματιστών όπως το Qt Creator ROS Plugin και εκπαιδευτικά προγράμματα που είναι εξειδικευμένα για τις ανάγκες των κατασκευαστών.

Το ROS-I υποστηρίζεται από μια διεθνή κοινοπραξία βιομηχανικών και ερευνητικών μελών. Το έργο ξεκίνησε ως συνεργατική προσπάθεια μεταξύ της Yaskawa Motoman Robotics, του Southwest Research Institute και του Willow Garage για την υποστήριξη της χρήσης του ROS για την αυτοματοποίηση κατασκευής. Το πακέτο του GitHub δημοσιεύτηκε τον Ιανουάριο του 2012 από τον Shaun Edwards (SwRI).

Εταιρείες ρομποτικής έχουν αναπτύξει «πακέτα» προκειμένου να υπάρχει σύνδεση των προϊόντων τους με το περιβάλλον του ROS-Industrial. Ενδεικτικά, κάποιες από αυτές είναι: Abb, Fanuc, Kuka, Yaskawa, Stäubli, Universal Robots

4.2 Η ιστορία του ROS

Πριν δημιουργηθούν τα λειτουργικά συστήματα στα ρομπότ, κάθε σχεδιαστής και ερευνητής έπρεπε να ξοδέψει πολύ χρόνο για την σχεδίαση ενσωματωμένου λογισμικού στο ρομπότ. Συνήθως, τα προγράμματα που σχεδιάστηκαν με αυτόν τον τρόπο ήταν παρόμοια μεταξύ τους. Υπήρξε γενικά σημαντική επαναχρησιμοποίηση λογισμικού, καθώς αυτό ήταν στενά συνδεδεμένο με το hardware του ρομπότ.[28]

Πολλά frameworks για ρομπότ παράγονται για συγκεκριμένο λόγο όπως για προτυποποίηση. Το ROS προοριζόταν να είναι γενικότερου σκοπού, αν και οι σχεδιαστές του δεν πιστεύουν ότι είναι το απόλυτο λειτουργικό σύστημα που μπορεί να κάνει τα πάντα. Το ROS αναπτύχθηκε από την εταιρεία Willow Garage, που ιδρύθηκε το 2006.[28] Η Willow Garage διατηρεί στενούς δεσμούς με το Πανεπιστήμιο του Στάνφορντ, και περιγράφεται ως ερευνητικό εργαστήριο. Αναπτύσσει τόσο λογισμικό με ROS όσο και υλικό με τα ρομπότ PR2 και TurtleBot. Όλο το λογισμικό που παράγεται είναι ανοιχτού κώδικα. Η ιδέα είναι ότι εάν θέλουμε να δούμε τα ρομπότ να φτάνουν στα σπίτια μας, τότε η έρευνα πρέπει να επιταχυνθεί παρέχοντας σταθερές βάσεις υλικού και λογισμικού που είναι ανοιχτού κώδικα..[28]

4.3 Φιλοσοφία και χαρακτηριστικά

Η φιλοσοφία του ROS μπορεί να συνοψιστεί στις ακόλουθες πέντε βασικές αρχές:

- Peer-to-Peer αρχιτεκτονική
- Βασισμένο σε εργαλεία
- Προγραμματισμός σε πολλές γλώσσες
- Ελαφρύ
- Δωρεάν και ανοιχτού κώδικα

Peer to Peer: Ένα αρκετά σύνθετο ρομπότ περιλαμβάνει πολλούς μικροεπεξεργαστές που συνδέονται μέσω Ethernet, και εκτελούν εντατικές εργασίες υπολογισμού. Μια αρχιτεκτονική peer-to-peer σε συνδυασμό με ένα σύστημα ενδιάμεσης μνήμης (buffering) και ένα σύστημα αναζήτησης (μια υπηρεσία που ονομάζεται "master" στο ROS), επιτρέπει σε κάθε στοιχείο να επικοινωνεί απευθείας με οποιοδήποτε άλλο, συγχρονισμένα ή ασύγχρονα όπως απαιτείται.

Προγραμματισμός σε πολλές γλώσσες: Το ROS δεν χαρακτηρίζεται από κάποια γλώσσα προγραμματισμού και μπορούν να χρησιμοποιηθούν διάφορες γλώσσες. Η επικοινωνία λειτουργεί μέσω της ανταλλαγής μηνυμάτων. Οι συνδέσεις peer-to-peer λειτουργούν σε XML-RPC, το οποίο υπάρχει σε πολλές γλώσσες.

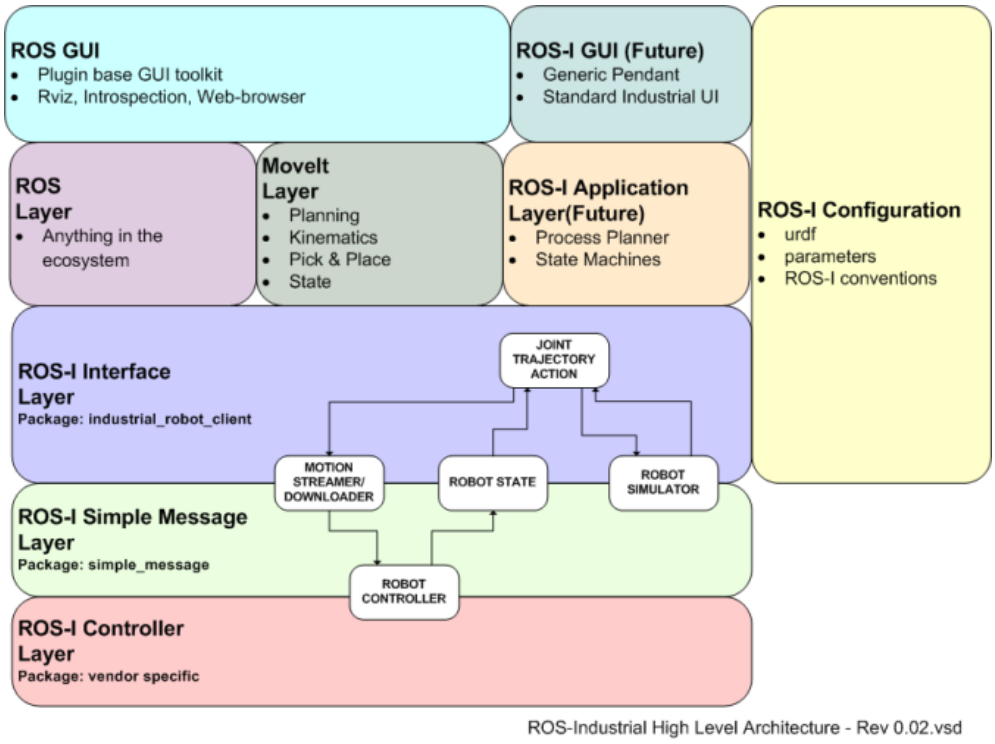
Βασισμένο σε εργαλεία (tool-based): Αντί για ένα παλαιό περιβάλλον runtime, το ROS υιοθέτησε μια σχεδίαση microkernel, η οποία χρησιμοποιεί μεγάλο αριθμό μικρών εργαλείων για την κατασκευή και τη λειτουργία των διαφόρων στοιχείων του ROS. Κάθε εντολή είναι στην πραγματικότητα εκτελέσιμη. Το πλεονέκτημα αυτού του συστήματος είναι ότι ένα πρόβλημα με ένα εκτελέσιμο δεν επηρεάζει τα άλλα, γεγονός που καθιστά το σύστημα πιο ανθεκτικό και ευέλικτο από ένα σύστημα που βασίζεται σε ένα κεντρικό περιβάλλον runtime.

Ελαφρύ: Για την καταπολέμηση της ανάπτυξης αλγορίθμων που έχουν εμπλακεί σε μικρότερο ή μεγαλύτερο βαθμό με το λειτουργικό σύστημα και επομένως είναι δύσκολο να επαναχρησιμοποιηθούν στη συνέχεια, οι developers του ROS περιλαμβάνουν προγράμματα οδήγησης και αλγορίθμους σε ξεχωριστά εκτελέσιμα αρχεία. Αυτό διασφαλίζει τη μέγιστη επαναχρησιμοποίηση και, πάνω απ 'όλα, διατηρεί το μέγεθός του μικρό. Αυτή η μέθοδος καθιστά το ROS εύκολο στη χρήση, καθώς η πολυπλοκότητα βρίσκεται στις βιβλιοθήκες

Δωρεάν και ανοιχτού κώδικα: Έχει την δυνατότητα να επικοινωνεί και να ανταλλάσσει δεδομένα με ένα μεγάλο πλήθος συσκευών. Αυτό το καθιστά λειτουργικό και ευέλικτο.

Η βασική αρχή ενός λειτουργικού συστήματος είναι η εκτέλεση παράλληλα ενός μεγάλου αριθμού εκτελέσιμων αρχείων, που πρέπει να είναι σε θέση να ανταλλάσσουν δεδομένα συγχρονισμένα ή ασύγχρονα. Για παράδειγμα, ένα λειτουργικό σύστημα ενός ρομπότ πρέπει να ρυθμίσει τους αισθητήρες του σε μια καθορισμένη συχνότητα (ultrasound ή infra-red distance sensor, pressure sensor,

temperature sensor, gyroscope, accelerometer), να ανακτήσει αυτά τα δεδομένα, να τα επεξεργαστεί, να τα μεταβιβάσει σε αλγόριθμους επεξεργασίας (επεξεργασία ομιλίας, τεχνητή όραση, SLAM - χαρτογράφηση) και ως αποτέλεσμα να δώσει την κατάλληλη εντολή στους κινητήρες. Αυτή η όλη διαδικασία πραγματοποιείται συνεχώς και παράλληλα. Τέλος, πρέπει να διαχειρίζεται τους πόρους που διαθέτει αποτελεσματικά.



Εικόνα 20: Αρχιτεκτονική του Ros-Industrial (wiki.ros.org/Industrial)

Παρακάτω αναλύεται η υποδομή του ROS προκειμένου να γίνει κατανοητή η αρχιτεκτονική του περιβάλλοντος.

Nodes Το ROS διαχειρίζεται όλες τις πληροφορίες χρησιμοποιώντας κόμβους. Ένα node μπορεί να αντιστοιχεί σε έναν αισθητήρα, ένα κινητήρα, ένα αλγόριθμο κλπ. Κάθε node που ξεκινά την λειτουργία του το γνωστοποιεί στον Master.

Master Ο Master είναι μια καταγραφή των nodes και ένα registration service που κάνει τα nodes να επικοινωνούν μεταξύ τους και να ανταλλάσσουν δεδομένα. Επίσης, περιλαμβάνει και τον Parameter Server.

Topics Τα δεδομένα ανταλλάσσονται με ένα topic ασύγχρονα ενώ με ένα service σύγχρονα. Ένα topic είναι ένα μέσο μετάδοσης δεδομένων που βασίζεται στο subscribe/publish σύστημα που αναφέρθηκε προηγουμένως. Ορισμένα nodes μπορούν να διαβάσουν δεδομένα από ένα topic ενώ κάποια άλλα να δημοσιεύσουν δεδομένα σε ένα topic.

Messages Ένα μήνυμα είναι μια συλλογή στοιχείων με συγκεκριμένη μορφή. Αποτελείται από ένα συνδυασμό διαφόρων τύπων πχ. ακέραιοι αριθμοί, χαρακτήρες και περιέχει πληροφορίες σχετικά με ένα topic. Για παράδειγμα, ένα node που αντιπροσωπεύει ένα σερβοκινητήρα ενός ρομπότ δημοσιεύει την κατάστασή του σε ένα topic με ένα message που περιέχει έναν ακέραιο αριθμό για θέση του κινητήρα, έναν δεκαδικό αριθμό για την θερμοκρασία του και έναν δεκαδικό αριθμό για την ταχύτητά του.

Services Ένα service είναι μια μέθοδος σύγχρονης επικοινωνίας μεταξύ δυο nodes.

Επιπλέον, θα πρέπει να αναφερθεί και μια άλλη ενδιαφέρουσα συνεισφορά στη ρομποτική από το ROS που είναι το **URDF (Unified Robot Description Format)**, μια μορφή XML που χρησιμοποιείται για να περιγράψει ένα ολόκληρο ρομπότ με τη μορφή ενός τυποποιημένου αρχείου. Τα ρομπότ που περιγράφονται με αυτόν τον τρόπο μπορούν να είναι στατικά ή δυναμικά και μπορούν να προστεθούν οι φυσικές τους ιδιότητες. Εκτός από το πρότυπο URDF, το ROS προσφέρει διάφορα εργαλεία που χρησιμοποιούνται για τη δημιουργία, ανάλυση ή έλεγχο αυτής της μορφής. Για παράδειγμα, το URDF χρησιμοποιείται από το λογισμικό προσομοίωσης Gazebo για την αναπαράσταση του ρομπότ.

4.4 Ένταξη του φυσικού ρομπότ στο ROS

Αρχικά, πραγματοποιήθηκε μια προσπάθεια ένταξης του ρομπότ Stäubli RX90L που υπάρχει στο Εργαστήριο Τεχνολογίας των Κατεργασιών στο περιβάλλον του ROS. Έγιναν όλες οι απαραίτητες εγκαταστάσεις για την επικοινωνία μεταξύ τους σε περιβάλλον Linux. Το πρόγραμμα Putty πήρε την θέση του προγράμματος Teraterm για την επικοινωνία μέσω Terminal με την σειριακή θύρα RS232.

Η δοκιμασία ήταν η αμφίδρομη επικοινωνία μεταξύ του ελεγκτή του ρομπότ και των nodes του ROS που μετέφεραν τις πληροφορίες της κατάστασης του ρομπότ. Ο σκοπός ήταν να αναπτυχθεί ένας αλγόριθμος που θα εκτελείται στον ελεγκτή του ρομπότ και θα είναι γραμμένος με την νοοτροπία ενός listener. Στην ουσία, στο περιβάλλον του ROS υπάρχουν ορισμένα nodes που πρέπει να γίνονται publish στον ελεγκτή όπου εκεί υπάρχει ένας subscriber που διαβάζει και κατανοεί τις αντίστοιχες τιμές.

Ο ελεγκτής που ρομπότ λειτουργεί με ένα είδος “dumb terminal” που είναι μια οθόνη προβολής που δεν έχει δυνατότητες επεξεργασίας, δηλαδή απλά μια συσκευή εξόδου που δέχεται δεδομένα από την CPU. Αντίθετα, ένα “smart terminal” είναι μια οθόνη που έχει τον δικό της επεξεργαστή για ειδικές λειτουργίες, όπως έντονοι χαρακτήρες και χαρακτήρες που αναβοσβήνουν.

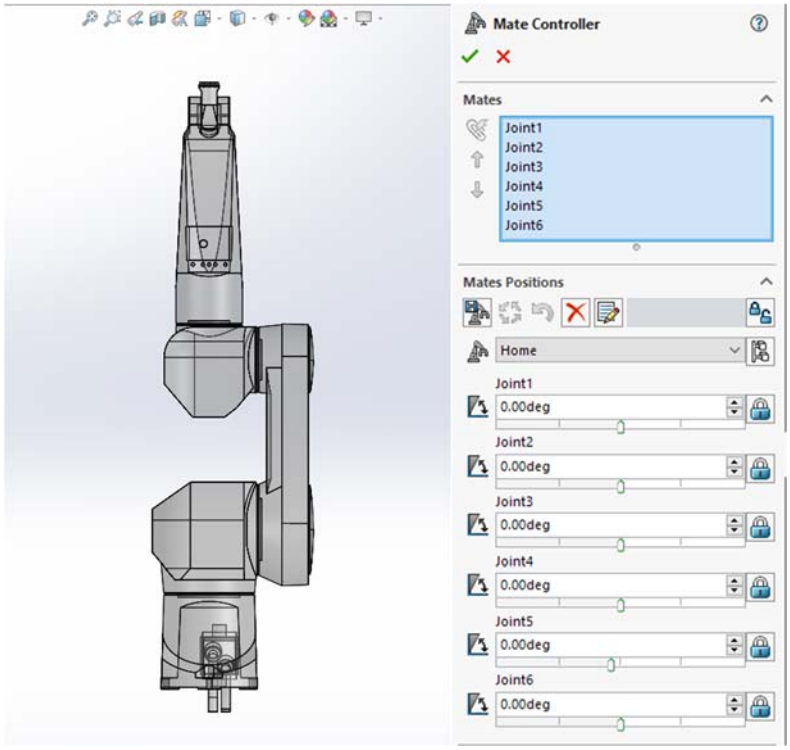
Επομένως, πρέπει στο τερματικό, να εκτυπώνονται οι αντίστοιχες εντολές σε γλώσσα V+ ανάλογα με τις κινήσεις που γίνονται στο περιβάλλον του ROS. Για τον λόγο αυτό, χρησιμοποιήθηκε ο κώδικας που παρουσιάζεται στο Παράρτημα 8.1, ο

οποίος κάνει την μετατροπή και την σύνταξη του μηνύματος προκειμένου να είναι κατανοητό από τον ελεγκτή του ρομπότ. Ο συγκεκριμένος κώδικας υπάρχει στο GitHub και έχει συνταχθεί από την εταιρεία Adept για έναν αριθμό ελεγκτών της εταιρείας με την ίδια ή παρόμοια τεχνολογία. [29]

Δυστυχώς, λόγω της επεξεργαστικής μονάδας του ελεγκτή 0030, δεν ήταν δυνατή η επικοινωνία μεταξύ των δυο συστημάτων. Η έκδοση του controller είναι η CS7 με ημερομηνία κατασκευής το 1995. Τα επόμενα χρόνια η εταιρία κυκλοφόρησε μια αναβαθμισμένη έκδοση του ελεγκτή την CS7B. Σε αυτή την έκδοση, έγινε αναβάθμιση του επεξεργαστή από 0030 σε 0040 και προστέθηκε θύρα Ethernet για χρήση του πρωτόκολλου TCP/IP.

4.5 Μοντέλο ρομπότ στο Solidworks

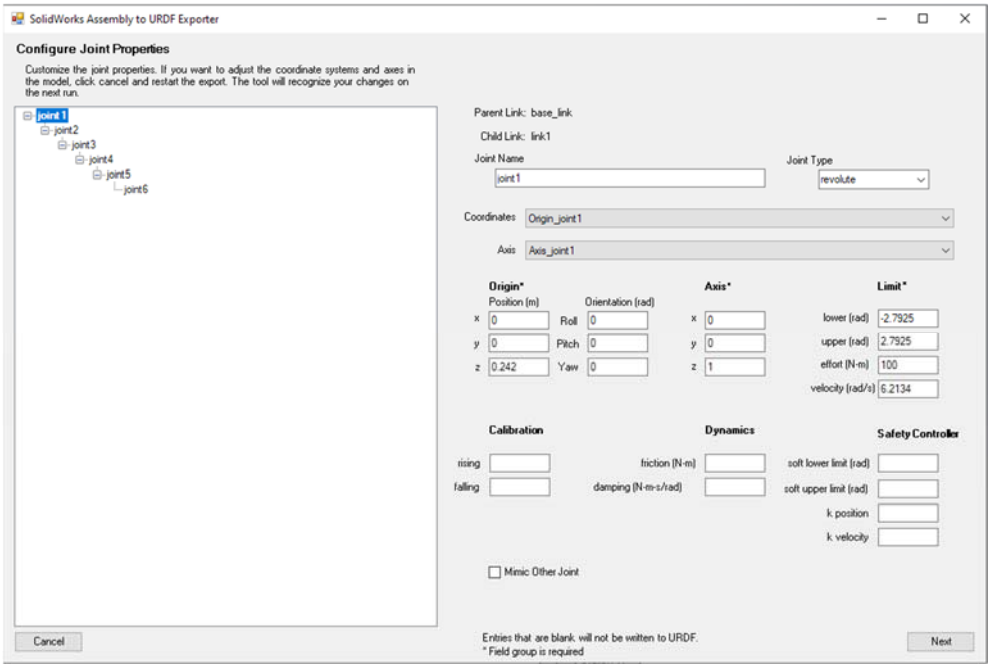
Τα μέλη (links) του ρομπότ υπήρχαν σε αρχεία συμβατά με το λογισμικό Solidworks. Αφού έγινε η εισαγωγή τους, δημιουργήθηκαν τα κατάλληλα mates και έγινε το assembly (συναρμολόγηση) με βάση τις διαστάσεις του κατασκευαστή καθώς και τα όρια γωνιών περιστροφής των αρθρώσεων. Ορίστηκαν τα κατάλληλα angle mates (περιορισμοί γωνιών) όπως φαίνεται στην παρακάτω εικόνα, και η διαμόρφωση με τις αρθρώσεις σε γωνία μηδέν, ονομάστηκε home.



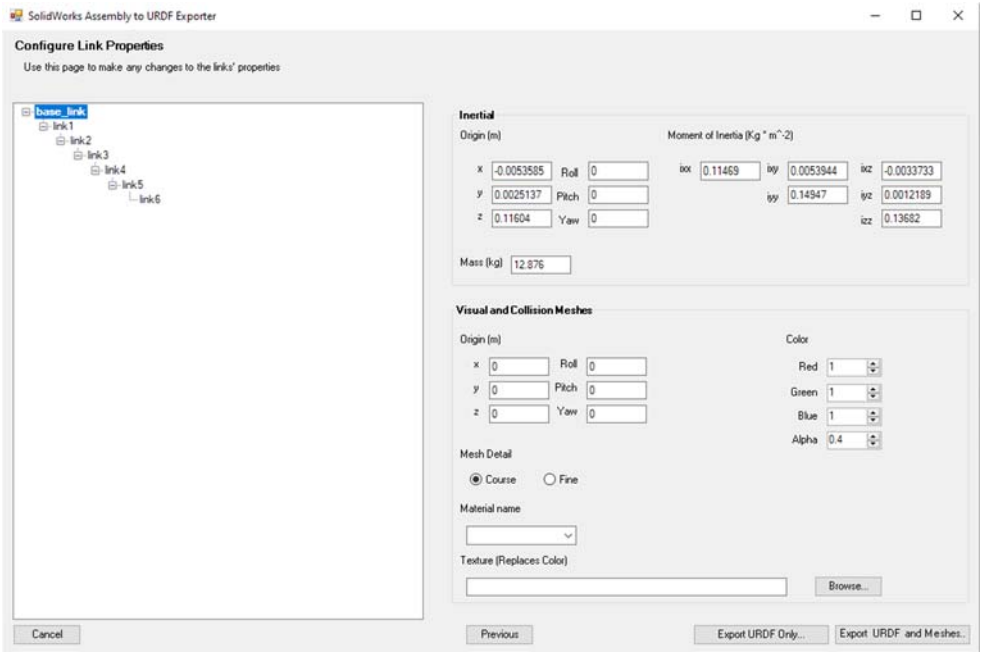
Εικόνα 21: Απεικόνιση στο Solidworks και Mate Controller

Προκειμένου να δημιουργηθεί το πακέτο του ROS, πρέπει το ρομπότ να μετασχηματιστεί στην μορφή URDF που αναφέρθηκε παραπάνω. Αυτή η μορφή, περιλαμβάνει όλες τις ιδιότητες των συνδέσμων και των αρθρώσεων του ρομπότ σε μορφή κώδικα προκειμένου να γίνουν κατανοητές από το περιβάλλον και τα

πρόσθετα του ROS. Αυτό έγινε με χρήση ενός plugin που αναπτύχθηκε από developers του ROS και ονομάζεται ‘Solidworks to URDF Exporter’.



Εικόνα 22: Ιδιότητες των αρθρώσεων στο plugin



Εικόνα 23: Ιδιότητες των συνδέσμων του ρομπότ(links)

Αυτό το πρόσθετο (plugin), μας δίνει την δυνατότητα να εξάγουμε μέρη(parts) ενός αντικειμένου που έχει σχεδιαστεί στο Solidworks αλλά και ολόκληρα συναρμολογημένα αντικείμενα (assemblies). Δημιουργεί ένα αρχικό πακέτο του ROS που περιέχει την δομή, τα χρώματα(textures), και το ρομπότ σε URDF μορφή. Διατηρεί την ιεραρχία του Solidworks και αποτυπώνει το ρομπότ σε μορφή δένδρου

(tree based) καθώς και αναγνωρίζει πλήρως το είδος της άρθρωσης (πρισματική ή περιστροφική) και τους άξονές του. [30]

4.6 MoveIt και Rviz

4.6.1 Περιγραφή

Το MoveIt! είναι εξελιγμένο λογισμικό για χειρισμό ρομποτικών συσκευών ενσωματώνοντας τις τελευταίες εξελίξεις στον σχεδιασμό τροχιάς (path planning), 3D perception (την τρισδιάστατη αντίληψη), την κινηματική, τον έλεγχο και την πλοήγηση. Παρέχει μια εύχρηστη πλατφόρμα για την ανάπτυξη προηγμένων εφαρμογών ρομποτικής, την υλοποίηση νέου σχεδιασμού σε ρομπότ και την κατασκευή ολοκληρωμένων προϊόντων ρομποτικής για βιομηχανικούς και εμπορικούς σκοπούς. Το MoveIt, είναι το πιο διαδεδομένο λογισμικό ανοιχτού κώδικα για χειρισμό βραχιόνων και έχει χρησιμοποιηθεί σε περισσότερα από 65 διαφορετικά ρομπότ. [31]

Η δημιουργία νέων λογισμικών ανοιχτού κώδικα όπως το ROS και το MoveIt! έχει καταστήσει τη ρομποτική πιο προσιτή σε νέους χρήστες, τόσο σε εφαρμογές έρευνας όσο και σε βιομηχανικές. Συγκεκριμένα, το ROS έχει φέρει επανάσταση στην κοινότητα των προγραμματιστών, παρέχοντας ένα σύνολο εργαλείων, υποδομών και βέλτιστων πρακτικών για τη δημιουργία νέων εφαρμογών (όπως το ερευνητικό ρομπότ Baxter). Το MoveIt, παρέχει τη βασική λειτουργικότητα για χειρισμό στο ROS και βασίζεται στους ακόλουθους πυλώνες:[31]

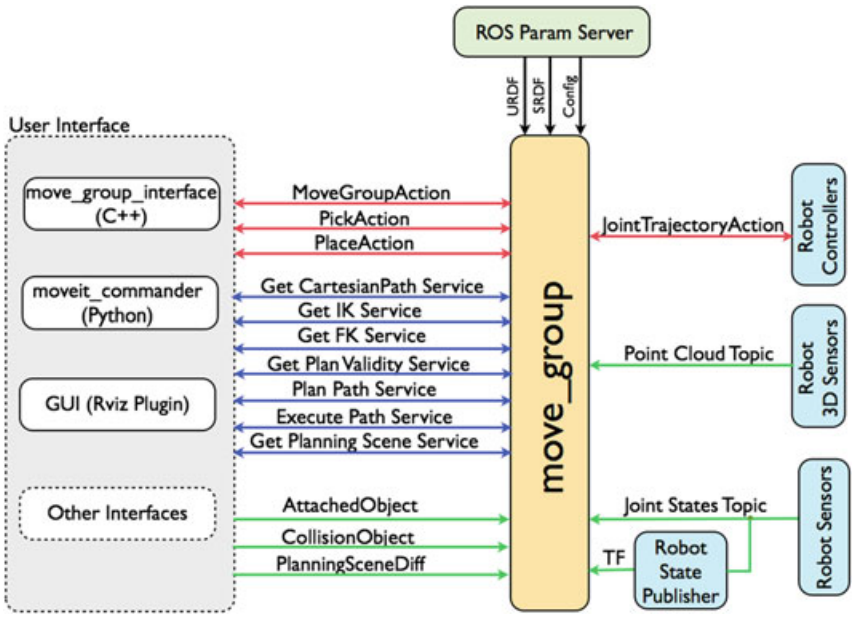
- *Βιβλιοθήκη δυνατοτήτων*: παρέχει ρομποτικές δυνατότητες για χειρισμό, σχεδιασμό κίνησης και έλεγχο των ρομπότ.
- *Ισχυρή κοινότητα*: χρήστες και προγραμματιστές βοηθούν στη διατήρηση και επέκταση του MoveIt! σε νέες εφαρμογές.
- *Εργαλεία*: αυτά επιτρέπουν στους νέους χρήστες να συνδέουν το MoveIt! με ρομπότ και στους προηγμένους χρήστες να αναπτύσσουν νέες εφαρμογές.

4.6.2 Αρχιτεκτονική

Ο κύριος κόμβος (node) του MoveIt ονομάζεται move_group. Έχει δημιουργηθεί με τρόπο ώστε να μην καταναλώνει πολλούς πόρους και να διαχειρίζεται πολλά νήματα (threads) την ίδια στιγμή όπως η κινηματική του ρομπότ, ο αλγόριθμος τροχιάς κλπ. Ο χρήστης μπορεί να έχει πρόσβαση στον πυρήνα του move_group με τρείς διαφορετικούς τρόπους:

- Με γλώσσα C++: Χρησιμοποιώντας το πακέτο move_group_interface μπορεί να τροποποιηθεί το περιβάλλον του move_group με χρήση αντικειμενοστραφούς γλώσσας προγραμματισμού C++.
- Με γλώσσα Python: Προγραμματισμός με γλώσσα Python, γίνεται χρήση του πακέτου moveit_commander.

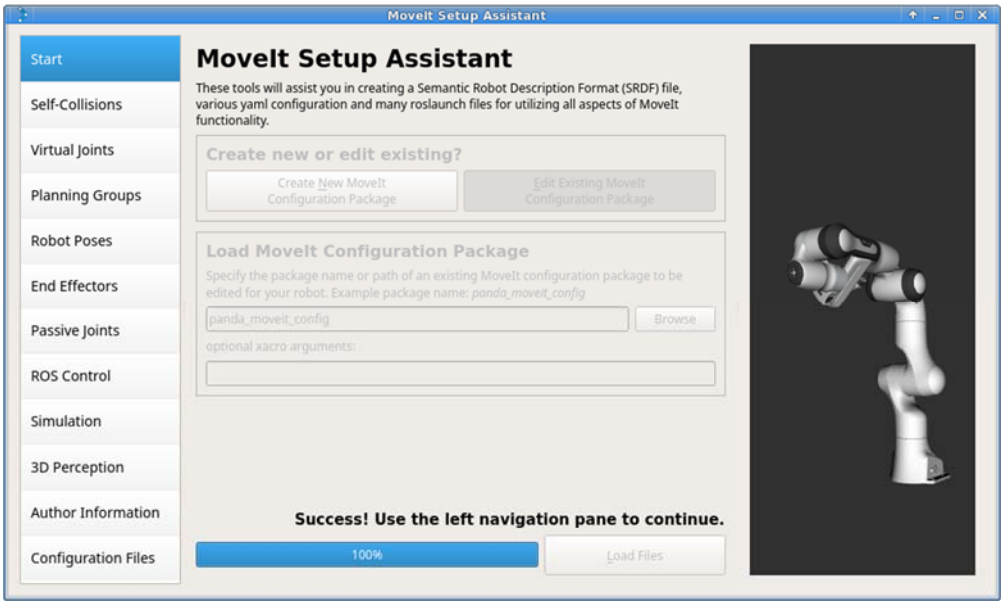
- Μέσα από το γραφικό περιβάλλον (GUI): Ο χρήστης μπορεί να κάνει χρήση του plugin Motion Planning που υπάρχει ενσωματωμένο στο Rviz προκειμένου να εκτελέσει μια κίνηση του βραχίονα.



Εικόνα 24: Αρχιτεκτονική του MoveIt.[31]

4.6.3 Οδηγός δημιουργίας πακέτου (MoveIt Setup Assistant)

Το πρώτο βήμα για να δημιουργηθεί το πακέτο του MoveIt είναι ο βοηθός εγκατάστασης (moveit setup assistant) που έχει σχεδιαστεί για να επιτρέπει στους χρήστες να εισάγουν νέα ρομπότ και να δημιουργούν ένα πακέτο MoveIt για την αλληλεπίδραση, την οπτικοποίηση και την προσομοίωση του ρομπότ (και του σχετικού χώρου εργασίας).



Εικόνα 25: Κεντρικό menu του moveit setup assistance

Η κύρια λειτουργία του βοηθού εγκατάστασης είναι να δημιουργήσει ένα αρχείο Semantic Robot Description Format (SRDF) για το ρομπότ. Δημιουργεί επίσης ένα σύνολο αρχείων που επιτρέπουν στον χρήστη να ξεκινήσει μια οπτική επίδειξη του ρομπότ αμέσως. Για την εκκίνησή του αρκεί να ανοίξουμε το terminal και να πληκτρολογήσουμε την εντολή: [32] *rosrun moveit_setup_assistant moveit_setup_assistant*

Ό χρήστης μπορεί να επιλέξει να επεξεργαστεί ένα υπάρχον πακέτο ή να δημιουργήσει ένα καινούργιο. Ο κύριος λόγος για να επεξεργαστεί ένα υπάρχον πακέτο είναι για να γίνει αυτόματη εξαγωγή του Allowed Collision Matrix (ACM) που αφορά τις συγκρούσεις μεταξύ των συνδέσμων. Ο συγκεκριμένος πίνακας πρέπει να ανανεώνεται κάθε φορά που συμβαίνουν τα ακόλουθα:

- ✓ Έχει αλλάξει η γεωμετρική μορφή του ρομπότ (URDF), επειδή για παράδειγμα έχει χρησιμοποιηθεί ένα διαφορετικό τελικό στοιχείο δράσης.
- ✓ Έχουν αλλάξει τα όρια γωνιών των αρθρώσεων.

Το κλειδί για την επιλογή του πίνακα **self-collision** είναι ο αριθμός των τυχαίων δειγμάτων που θα δημιουργηθούν. Χρησιμοποιώντας έναν μεγάλο αριθμό έχουμε σαν αποτέλεσμα την δημιουργία πολλών δειγμάτων(samples) που επιβραδύνει την λειτουργία του MoveIt. Αντιθέτως, αν χρησιμοποιηθεί ένας μικρός αριθμός, υπάρχει πιθανότητα να υπάρχουν συγκρούσεις μεταξύ του βραχίονα και ενός αντικειμένου στο περιβάλλον της προσομοίωσης.

Οι **εικονικές αρθρώσεις(virtual joints)**, χρησιμοποιούνται για να υποδείξουν που βρίσκεται το ρομπότ μέσα στο world (περιβάλλον εργασίας). Για ένα βιομηχανικό ρομπότ, μια εικονική άρθρωση μπορεί να σχετίζεται με την έδραση του ρομπότ στο έδαφος.

Planning groups, είναι ομάδες υποσυστημάτων ενός ρομπότ όπως ένα εργαλείο στο τελικό στοιχείο δράσης. Σε ένα βιομηχανικό βραχίονα, μια ομάδα μπορεί να αποτελεί ολόκληρο το ρομπότ πχ 6 αρθρώσεων και μια άλλη ομάδα να είναι το τελικό στοιχείο δράσης.

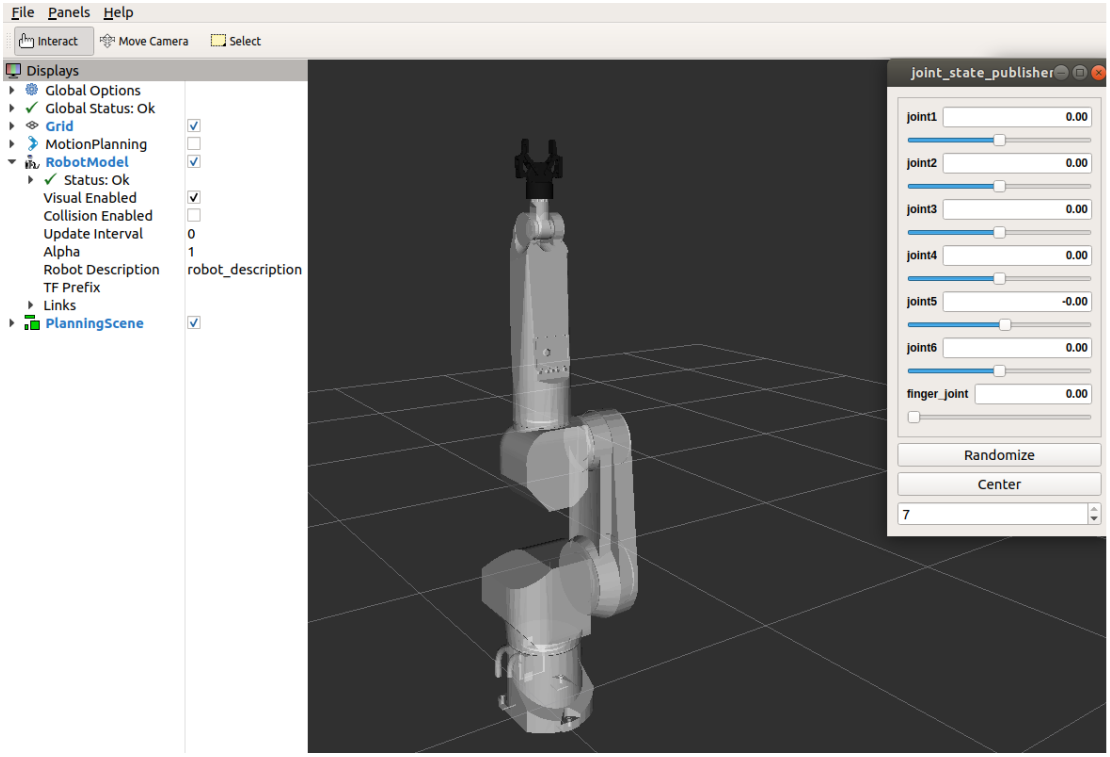
Robot poses (πόζες ρομπότ - διαμορφώσεις), είναι συγκεκριμμένες προκαθορισμένες θέσεις που παίρνει το ρομπότ. Στην περίπτωση της εργασίας, μια αποθηκευμένη θέση είναι η home.

Passive joints (ανενεργές-παθητικές αρθρώσεις), αποτελούν αρθρώσεις οι οποίες δεν λαμβάνονται υπόψη στον υπολογισμό τροχιάς και στον έλεγχο.[33]

4.6.4 Το περιβάλλον του Rviz

Το Rviz είναι το κύριο πρόσθετο (plugin) προκειμένου να δουλέψει κανείς με το MoveIt. Επιτρέπει στον χρήστη να δημιουργεί περιβάλλοντα εργασίας και να πραγματοποιεί κινήσεις χωρίς συγκρούσεις με αντικείμενα μέσα στον χώρο. Διαθέτει μια σειρά από λειτουργίες όπως είναι το Motion Planning, όπου γίνεται προγραμματισμός τροχιάς με δήλωση σημείων αρχής-τέλους (χωρίς συγκρούσεις-

collision free). Επιπλέον, υπάρχει η δυνατότητα ανάγνωσης δεδομένων από πραγματικούς αισθητήρες και αποτύπωσής τους στο περιβάλλον. Αυτό κάνει πολύ ρεαλιστικό το συγκεκριμένο λογισμικό και του δίνει την δυνατότητα να ενταχθεί εύκολα σε βιομηχανικά περιβάλλοντα λόγω της υψηλής αλληλεπίδρασης.[34]



Εικόνα 26: Stäubli RX90L στο περιβάλλον Rviz

4.6.5 Gazebo

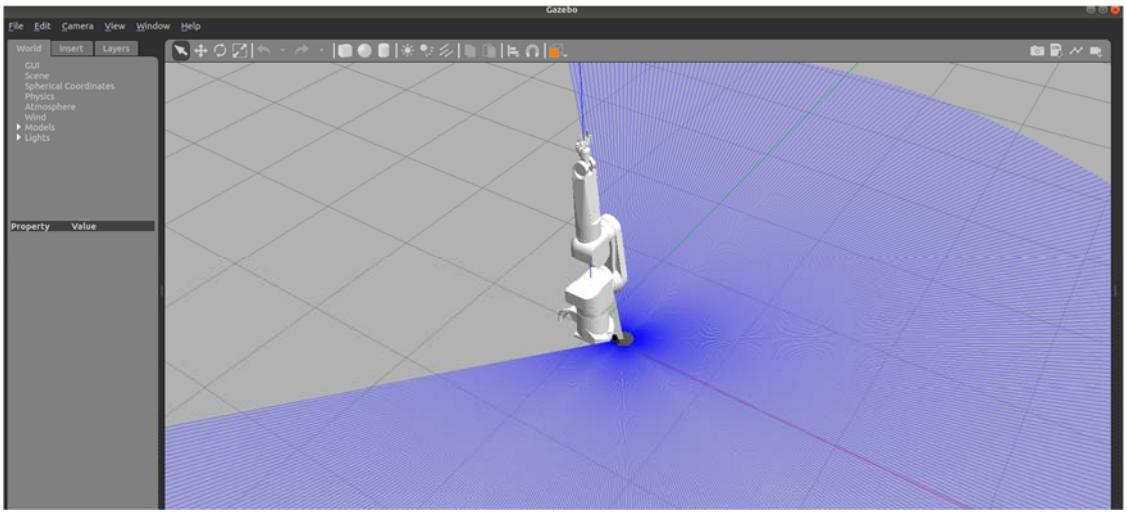
Το Gazebo είναι λογισμικό 3D προσομοίωσης ανοιχτού κώδικα. Ενσωματώνει την μηχανή φυσικών ιδιοτήτων ODE, φωτορεαλισμό (rendering) OpenGL καθώς και υποστήριξη για προσομοίωση αισθητήρων αλλά και έλεγχο κινητήρων. Το 2011, το Gazebo έγινε ένα ανεξάρτητο project που υποστηρίζονται από το Willow Garage. Το 2012, η Open Source Robotics Foundation έγινε ο υποστηρικτής του.[35]

Το Gazebo, προσφέρει ρεαλιστική απεικόνιση περιβάλλοντος με υψηλής ποιότητας φωτισμούς, σκιές και χρώματα. Μπορεί επίσης να προσομοιώσει αισθητήρες που μπορούν να «δουν» το περιβάλλον όπως αισθητήρες laser, κάμερες, Kinect style αισθητήρες και να μεταφέρει τα δεδομένα για απεικόνιση στο Rviz. [35]

Στην συγκεκριμένη εργασία, χρησιμοποιήθηκε ένας αισθητήρας laser της εταιρείας Hokuyo που υπάρχει στην βιβλιοθήκη του Gazebo. Αυτό επιτρέπει την αποτύπωση όλων των δεδομένων που απεικονίζονται στο Gazebo (πχ. άνθρωπος) στο περιβάλλον του rviz προκειμένου να ληφθούν υπόψη στον αλγόριθμο του προγραμματισμού τροχιάς.

Παρακάτω παρουσιάζεται ο κώδικας που χρησιμοποιείται για τον αισθητήρα laser. Όπως φαίνεται, ο χρήστης μπορεί να αλλάξει την θέση και τον προσανατολισμό του, την μέγιστη απόσταση καταγραφής καθώς και το εύρος της γωνίας του. Επίσης, είναι αισθητήρας με βάση την κάρτα γραφικών (gpu based) και αποτυπώνει τα δεδομένα του στο topic: rx90l/laser/scan.

```
<model name="hokuyo">
    <pose>0.2 0 0.025 0 0 0</pose>
    <link name="link">
      <gravity>false</gravity>
      <inertial>
        <mass>0.1</mass>
      </inertial>
      <visual name="visual">
        <geometry>
          <mesh>
            <uri>model://hokuyo/meshes/hokuyo.dae</uri>
          </mesh>
        </geometry>
      </visual>
      <sensor name="laser" type="gpu_ray">
        <pose>0.08 0 0.05 0 -0 0</pose>
        <ray>
          <scan>
            <horizontal>
              <samples>400</samples>
              <resolution>1</resolution>
              <min_angle>-2.3</min_angle>
              <max_angle>2.3</max_angle>
            </horizontal>
          </scan>
          <range>
            <min>0.08</min>
            <max>5</max>
            <resolution>0.01</resolution>
          </range>
          <noise>
            <type>gaussian</type>
            <mean>0.0</mean>
            <stddev>0.01</stddev>
          </noise>
        </ray>
        <plugin name="laser_back" filename="libgazebo_ros_gpu_laser.so">

            <robotNamespace>rx90l</robotNamespace>
            <topicName>laser/scan</topicName>
         </plugin>
        <always_on>1</always_on>
        <update_rate>30</update_rate>
        <visualize>true</visualize>
      </sensor>
    </link>


  </model>
```



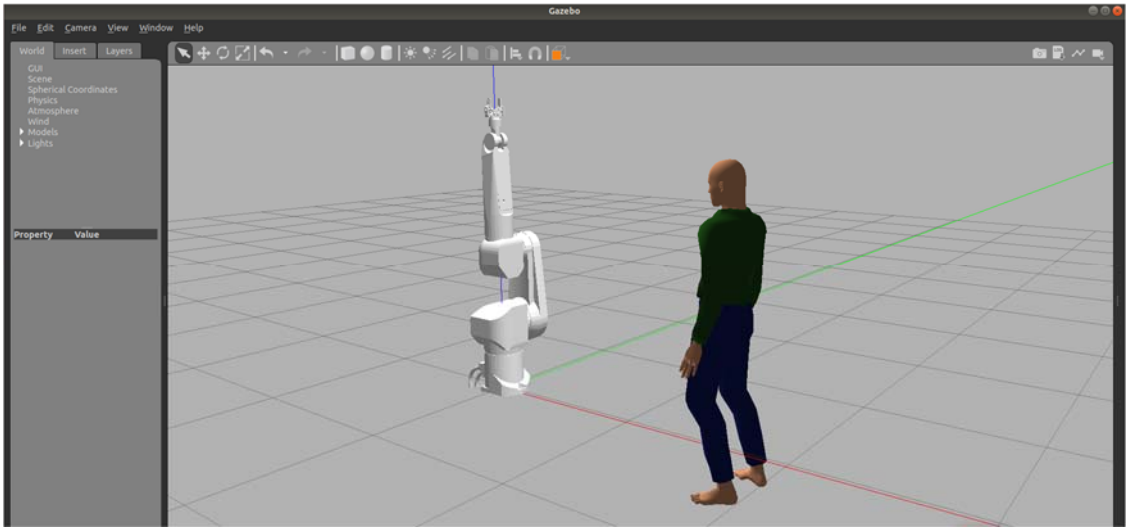
Εικόνα 27: Αισθητήρας laser στο gazebo

Για να μοντελοποιήσουμε τον άνθρωπο στο Gazebo χρησιμοποιήθηκε το animation "actor". Ο actor είναι ένα είδος μοντέλου που υπάρχει προ εγκατεστημένο στο Gazebo, το οποίο επιτρέπει την προσθήκη animations στο μοντέλο. Υπάρχουν δυο βασικά υποσυστήματα σε έναν actor: το skin και το animation. Το skin περιγράφει την εξωτερική εμφάνιση του ανθρώπου και είναι ένα αρχείο COLLADA (.dae). Το animation αφορά την κίνηση που θα εκτελέσει ο άνθρωπος (πχ. περπάτημα, τρέξιμο). Ο άνθρωπος αποτελείται από δέκα συνδέσμους και δέκα αρθρώσεις όπως δεξί-αριστερό χέρι και αποτελεί μια κινηματική αλυσίδα. Αυτό σημαίνει ότι η θέση και ο προσανατολισμός κάθε συνδέσμου μας είναι γνωστός αλλά δεν μπορούμε να μεταβάλουμε την θέση τους παρά μόνο μέσω του animation. [36]

Με τον παρακάτω κώδικα προγραμματίστηκαν οι κινήσεις του ανθρώπου στο gazebo. Όπως φαίνεται, ο χρήστης μπορεί να χρησιμοποιήσει διάφορα animation όπως stand, walking για την απεικόνιση διάφορων κινήσεων. Με την εντολή trajectory, δίνονται η θέση και ο χρόνος που ο άνθρωπος πρέπει να βρίσκεται εκεί και γίνεται ο προγραμματισμός της κίνησής του.

```
<actor name="actor">
      <skin>
        <filename>stand.dae</filename>
        <scale>1</scale>
      </skin>
      <animation name="standing">
        <filename>stand.dae</filename>
        <interpolate_x>false</interpolate_x>
      </animation>
         <animation name="walking">
        <filename>talk_b.dae</filename>

      </animation>
      <script>

         <trajectory id="0" type="standing">

         <waypoint>
            <time>0</time>
```

```
        <pose>3.5 0 -0.2 0 0 -3.14</pose>
      </waypoint>
     <waypoint>
        <time>4</time>
        <pose>1.8 0 -0.2 0 0 -3.14</pose>
      </waypoint>
     <waypoint>
        <time>7</time>
        <pose>1.8 0 -0.2 0 0 -3.14</pose>
      </waypoint>
      <waypoint>
        <time>9</time>
        <pose>1.8 0.8 -0.2 0 0 -3.14</pose>
      </waypoint>

    </trajectory>
  </script>
</actor>
```

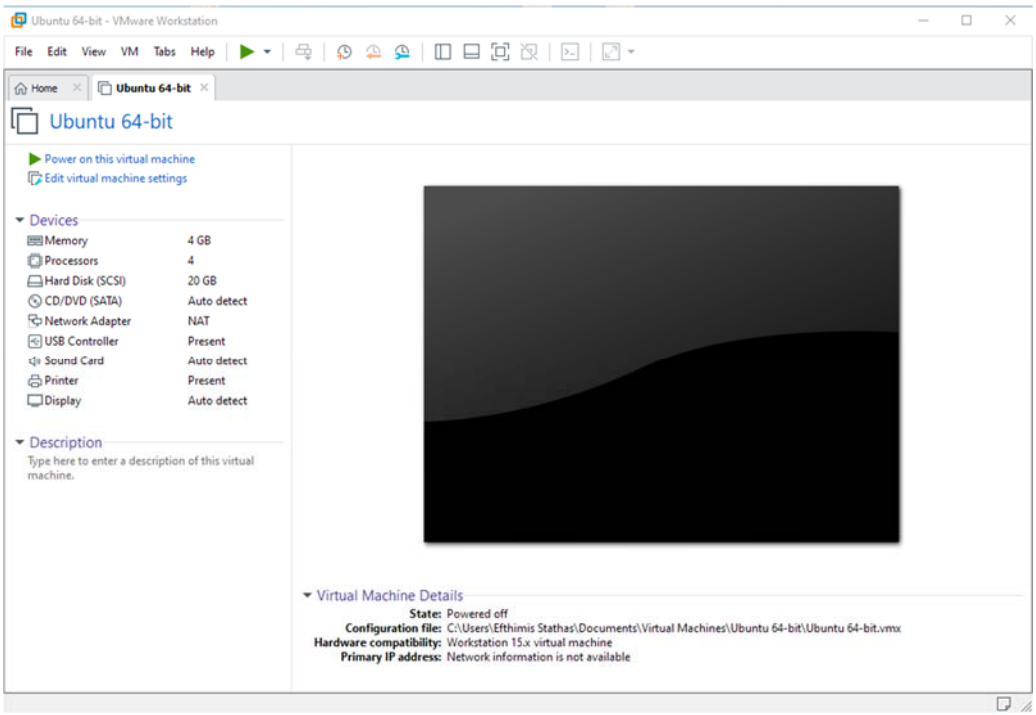


Εικόνα 28: Ο actor στο gazebo.

5 Σενάρια υλοποίησης στο περιβάλλον του ROS

5.1 Γενικά

Στο κύριο κομμάτι της εργασίας πραγματοποιήθηκε η υλοποίηση της προσομοίωσης στο περιβάλλον του ROS. Το λειτουργικό σύστημα που χρησιμοποιήθηκε είναι το Ubuntu 18.04LTS με την χρήση εικονικής μηχανής (Virtual Machine). Η έκδοση του ROS είναι η Melodic. Όταν ξεκίνησε η υλοποίηση της εργασίας, υπήρχαν και νεότερες εκδόσεις του ROS όμως σε δοκιμαστικές (beta) εκδόσεις. Οι περισσότεροι κατασκευαστές ρομπότ, είχαν αναπτύξει στο ROS-Industrial τα πακέτα για τα ρομπότ τους στην έκδοση ROS-Melodic που αποτελεί μια stable έκδοση.

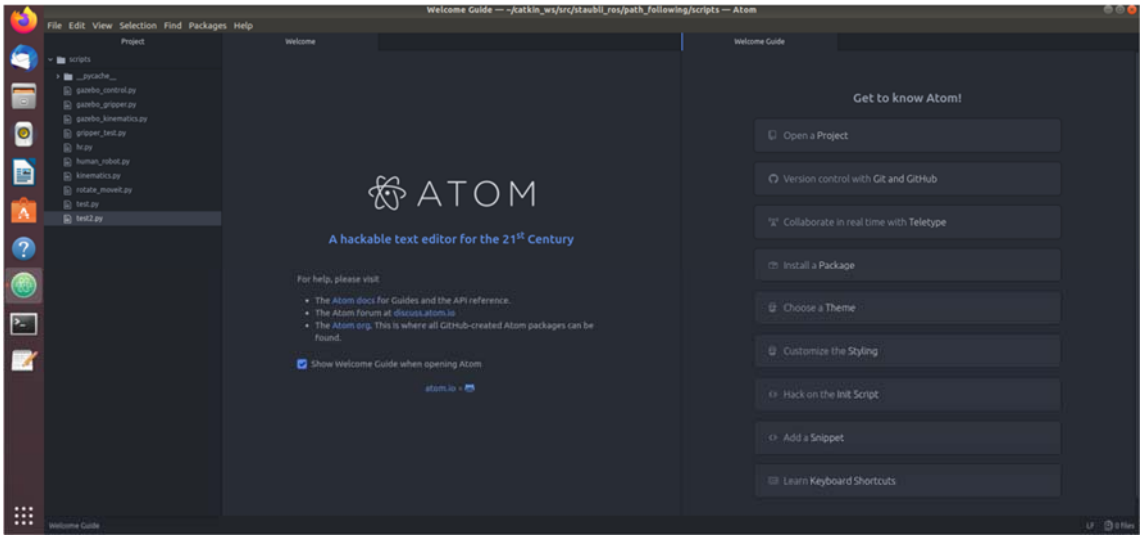


Εικόνα 29: Περιβάλλον εκκίνησης VMware.

Ακολουθήθηκε η διαδικασία εγκατάστασης του ROS όπως περιγράφεται και στην ιστοσελίδα του, και έγινε και η εγκατάσταση του πακέτου της γλώσσας Python [37]. Έπειτα, πραγματοποιήθηκε η εγκατάσταση του Gazebo και των απαραίτητων plugins προκειμένου να υπάρχει επικοινωνία με το MoveIt πακέτο (ROS Control).

Το γεγονός ότι η εγκατάσταση πραγματοποιήθηκε σε περιβάλλον εικονικής μηχανής δημιούργησε αρκετά προβλήματα κατά την υλοποίηση. Υπήρξαν ορισμένες ασυμβατότητες του Gazebo με την κάρτα γραφικών με αποτέλεσμα το πρόγραμμα να μην ανταποκρίνεται (3D Acceleration).

Για την συγγραφή του κώδικα, χρησιμοποιήθηκε το πρόγραμμα Atom. Πρόκειται για ένα λογισμικό ανοιχτού κώδικα που κυκλοφορεί για MacOS, Windows και Linux και υποστηρίζει ένα μεγάλο πλήθος γλωσσών προγραμματισμού όπως C, C++, Python, Javascript, XML κλπ. Έχει αναπτυχθεί από τους προγραμματιστές του GitHub οι οποίοι το χαρακτηρίζουν ως "hackable text editor for the 21st Century".



Εικόνα 30: Το παράθυρο εκκίνησης του Atom.

5.2 Κινήσεις του ρομπότ ανεξάρτητες της θέσης του ανθρώπου

5.2.1 Λήψη και τοποθέτηση / εναπόθεση (pick and place)

Ως πρώτο σενάριο υλοποίησης, επιλέχθηκε εργασία τύπου pick&place. Τέτοιου είδους εργασίες γίνονται στο μεγαλύτερο κομμάτι εφαρμογής του ρομποτικού βραχίονα σε βιομηχανικό περιβάλλον.

Το σχετικό διάγραμμα ροής δίνεται στην Εικόνα 3.1.

Η απεικόνιση του σεναρίου πραγματοποιήθηκε στο Rviz με τον κώδικα να αναπτύσσεται σε γλώσσα Python.

Κατά την αυτόματη εξαγωγή του πακέτου από τον οδηγό του MoveIt όπως έχει περιγραφεί πιο πάνω, γίνεται η δημιουργία ορισμένων αρχείων για την πρώτη διεπαφή του χρήστη με το ρομπότ κάτω από το περιβάλλον του ROS. Δημιουργούνται οι απαραίτητοι ελεγκτές προκειμένου να μπορεί να ελεγχθεί η κατάσταση των αρθρώσεων (joints) και να πραγματοποιηθεί η οποιαδήποτε κίνηση.

Εικόνα 31: Διάγραμμα ροής pick&place.

Το αρχείο demo.launch, είναι απαραίτητο να εκτελεστεί προκειμένου να ξεκινήσει η διαδικασία.

```
<?xml version="1.0"?>
<launch>

  <!-- specify the planning pipeline -->
  <arg name="pipeline" default="ompl" />

  <!-- By default, we do not start a database (it can be large) -->
  <arg name="db" default="false" />
  <!-- Allow user to specify database location -->
  <arg name="db_path" default="$(find rx90l)/default_warehouse_mongo_db" />

  <!-- By default, we are not in debug mode -->
  <arg name="debug" default="false" />

  <!--
  By default, hide joint_state_publisher's GUI

  MoveIt!'s "demo" mode replaces the real robot driver with the
joint_state_publisher.
  The latter one maintains and publishes the current joint configuration of
the simulated robot.
```

```
  It also provides a GUI to move the simulated robot around "manually".
  This corresponds to moving around the real robot without the use of
MoveIt.
  -->
  <arg name="use_gui" default="false" />

  <!-- Load the URDF, SRDF and other .yaml configuration files on the param
server -->
  <include file="$(find rx90l)/launch/planning_context.launch">
    <arg name="load_robot_description" value="true"/>
  </include>

  <!-- If needed, broadcast static tf for robot root -->


  <!-- We do not have a robot connected, so publish fake joint states -->
  <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher">
    <param name="use_gui" value="$(arg use_gui)"/>
    <rosparam
param="source_list">[move_group/fake_controller_joint_states]</rosparam>
  </node>

  <!-- Given the published joint states, publish tf for the robot links -->
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" respawn="true" output="screen" />

  <!-- Run the main MoveIt! executable without trajectory execution (we do
not have controllers configured by default) -->
  <include file="$(find rx90l)/launch/move_group.launch">
    <arg name="allow_trajectory_execution" value="true"/>
    <arg name="fake_execution" value="true"/>
    <arg name="info" value="true"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="pipeline" value="$(arg pipeline)"/>
  </include>

  <!-- Run Rviz and load the default config to see the state of the
move_group node -->
  <include file="$(find rx90l)/launch/moveit_rviz.launch">
    <arg name="rviz_config" value="$(find rx90l)/launch/moveit.rviz"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>

  <!-- If database loading was enabled, start mongodb as well -->
  <include file="$(find rx90l)/launch/default_warehouse_db.launch" if="$(arg
db)">
    <arg name="moveit_warehouse_database_path" value="$(arg db_path)"/>
  </include>

</launch>
```

Ο κώδικας είναι γραμμένος σε μορφή XML. Αρχικά, ορίζονται οι απαραίτητες παράμετροι για τα topics και τα nodes που πρέπει να «τρέχουν». Στην συνέχεια, γίνεται η φόρτωση του URDF του ρομπότ και της αρπάγης όπως αυτές έχουν εξαχθεί από το Solidworks (load robot description). Ξεκινά το node που θα δημοσιεύει τις τιμές των αρθρώσεων προκειμένου να γνωρίζουμε και να βλέπουμε πως κινείται το ρομπότ στον χώρο. Απαραίτητο είναι και το move_group αρχείο, καθώς στην συνέχεια θα γίνει υπολογισμός τροχιάς και εκτέλεσή της.

Έπειτα, εφόσον έχουν εκτελεστεί όλα τα απαραίτητα topics και nodes, μπορεί να γίνει η εκτέλεση του αρχείου python σε διαφορετικό παράθυρο του terminal.

Παρακάτω εξηγείται το περιεχόμενο του αρχείου για την υλοποίηση της διαδικασίας.

Αρχικά, συναντάμε την εισαγωγή ορισμένων βιβλιοθηκών που είναι απαραίτητες για την λειτουργία του αρχείο που έχει γραφεί (μαθηματικά σύμβολα, messages, nodes).

```
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from math import pi
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list
```

Έπειτα, ξεκινά η κλάση με το όνομα MoveGroupPythonIntefaceTutorial. Σε αυτήν γίνεται αρχικοποίηση των παραμέτρων του ρομπότ.

```
class MoveGroupPythonIntefaceTutorial(object):
  """MoveGroupPythonIntefaceTutorial"""
  def __init__(self):
    super(MoveGroupPythonIntefaceTutorial, self).__init__()

    moveit_commander.roscpp_initialize(sys.argv)
    rospy.init_node('moveit_python_rx90l', anonymous=True)

    robot = moveit_commander.RobotCommander()

    scene = moveit_commander.PlanningSceneInterface()

    group_name = "rx90l"
    move_group = moveit_commander.MoveGroupCommander(group_name)

    display_trajectory_publisher =
rospy.Publisher('/move_group/display_planned_path',

moveit_msgs.msg.DisplayTrajectory,
                                                    queue_size=20)


    self.box_name = ''
    self.robot = robot
    self.scene = scene
    self.move_group = move_group
    #self.gripper = gripper
    self.display_trajectory_publisher = display_trajectory_publisher
    self.planning_frame = planning_frame
    self.eef_link = eef_link
    self.group_names = group_names
```

Με την παρακάτω συνάρτηση add_box, γίνεται προσθήκη όλων των αντικειμένων στην σκηνή. Έχει γίνει παραμετροποίηση της θέσης των κουτιών προκειμένου να βρίσκονται μέσα στον χώρο εργασίας του ρομπότ. Όπως φαίνεται,

γίνεται εισαγωγή δύο τραπεζιών table1, table2, και ενός box που είναι και το αντικείμενο που θα μεταφερθεί από το ρομπότ.

```
def add_box(self, timeout=4):

    box_name = self.box_name
    scene = self.scene

    box_pose = geometry_msgs.msg.PoseStamped()
    box_pose.header.frame_id = "base_link"
    box_pose.pose.orientation.w = 1.0
    box_pose.pose.position.x = 1
    box_pose.pose.position.y = 0
    box_pose.pose.position.z = 0.5 # slightly above the end effector
    box_name = "box"
    scene.add_box(box_name, box_pose, size=(0.02, 0.02, 0.2))

    box_pose = geometry_msgs.msg.PoseStamped()
    box_pose.header.frame_id = "base_link"
    box_pose.pose.orientation.w = 1.0
    box_pose.pose.position.x = 1
    box_pose.pose.position.y = 0
    box_pose.pose.position.z = 0.2
    box_name = "table1"
    scene.add_box(box_name, box_pose, size=(0.2, 0.4, 0.4))

    box_pose = geometry_msgs.msg.PoseStamped()
    box_pose.header.frame_id = "base_link"
    box_pose.pose.orientation.w = 1.0
    box_pose.pose.position.x = 0
    box_pose.pose.position.y = 1
    box_pose.pose.position.z = 0.2
    box_name = "table2"
    scene.add_box(box_name, box_pose, size=(0.4, 0.2, 0.4))

    self.box_name="box"
    return self.wait_for_state_update(box_is_known=True, timeout=timeout)
```

Η συνάρτηση go_to_object με το όρισμα self, περιέχει τις δύο αρχικές κινήσεις του ρομπότ. Η κίνηση πραγματοποιείται με την δήλωση τιμών στις αρθρώσεις προκειμένου το ρομπότ να πάει στην επιθυμητή θέση. Όλα αυτά πραγματοποιούνται με το framework του move_group.

```
def go_to_object(self):
    move_group = self.move_group

    joint_goal = move_group.get_current_joint_values()
    joint_goal[0] = 0
    joint_goal[1] = 0.5
    joint_goal[2] = 1.61
    joint_goal[3] = 0
    joint_goal[4] = 0.54
    joint_goal[5] = 0
    joint_goal[6] = 0
    move_group.go(joint_goal, wait=True)
    move_group.stop()

    joint_goal = move_group.get_current_joint_values()
    joint_goal[0] = 0
    joint_goal[1] = 0.5
    joint_goal[2] = 1.61
    joint_goal[3] = 0
```

```
    joint_goal[4] = 0.54
    joint_goal[5] = 0
    joint_goal[6] = 0.55
    move_group.go(joint_goal, wait=True)
    move_group.stop()
```

Η συνάρτηση attach_box, θέτει το αντικείμενο στην ομάδα του gripper(αρπάγη). Με τον τρόπο αυτό, το αντικείμενο παραμένει στο εσωτερικό της αρπάγης κατά την διάρκεια της κίνησης.

```
def attach_box(self, timeout=4):

    box_name = self.box_name
    robot = self.robot
    scene = self.scene
    eef_link = self.eef_link
    group_names = self.group_names

    grasping_group = 'gripper'
    touch_links = robot.get_link_names(group=grasping_group)
    scene.attach_box(eef_link, box_name, touch_links=touch_links)

    return self.wait_for_state_update(box_is_attached=True,
box_is_known=False, timeout=timeout)
```

Με την συνάρτηση go_to_2ndtable, το ρομπότ μετακινεί το αντικείμενο από το τραπέζι 1 στο τραπέζι 2.

```
def go_to_2ndtable(self):

    move_group = self.move_group

    joint_goal = move_group.get_current_joint_values()
    joint_goal[0] = 1.55
    joint_goal[1] = 0.5
    joint_goal[2] = 1.61
    joint_goal[3] = 0
    joint_goal[4] = 0.54
    joint_goal[5] = 0
    joint_goal[6] = 0.55
    move_group.go(joint_goal, wait=True)
    move_group.stop()
```

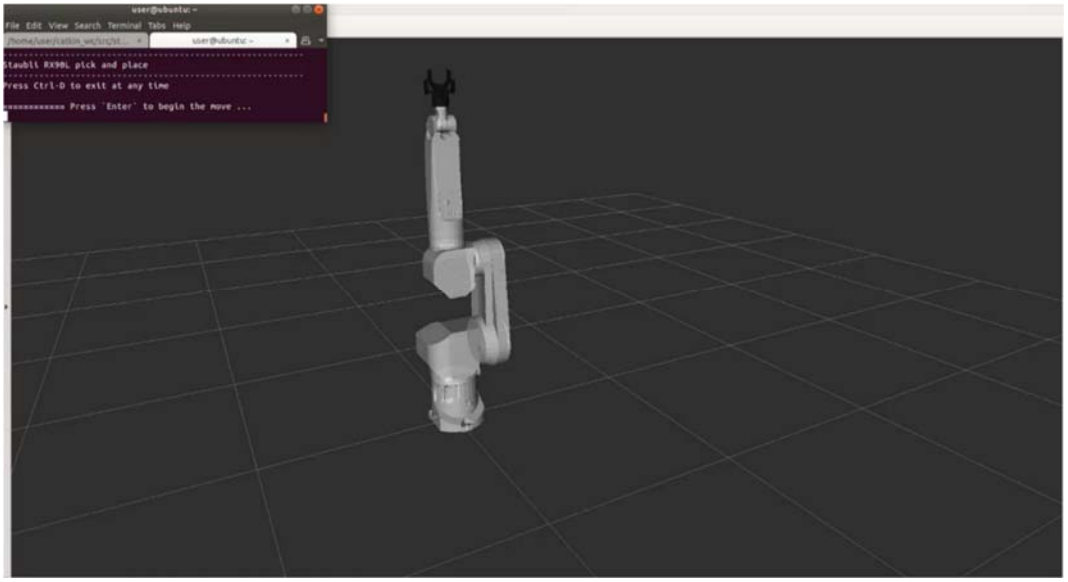
Η συνάρτηση remove_box, αφαιρεί όλα τα αντικείμενα από το περιβάλλον του rviz, προκειμένου να γίνει ο τερματισμός του προγράμματος.

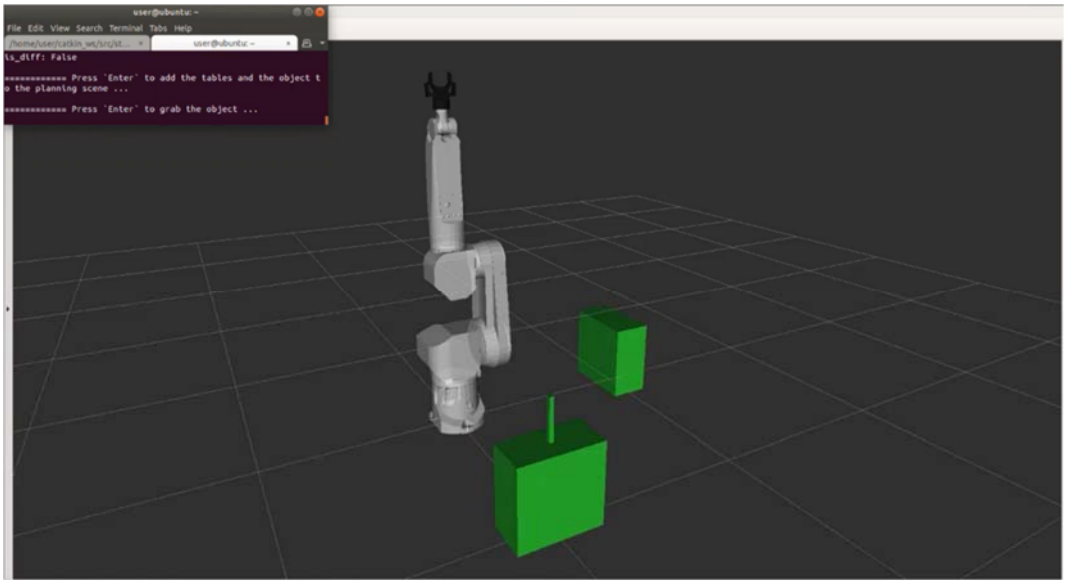
```
def remove_box(self, timeout=4):

    box_name = self.box_name
    scene = self.scene

    scene.remove_world_object(box_name)

    return self.wait_for_state_update(box_is_attached=False,
box_is_known=False, timeout=timeout)
```

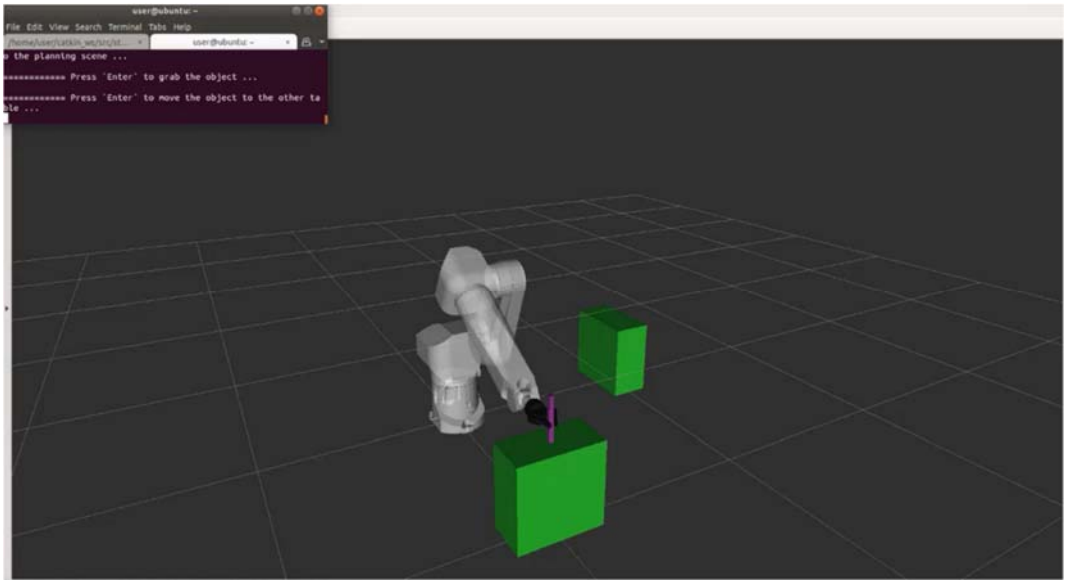
Τέλος, το κεντρικό κομμάτι κώδικα όπου καλούνται οι συναρτήσεις της κλάσης φαίνεται παρακάτω.

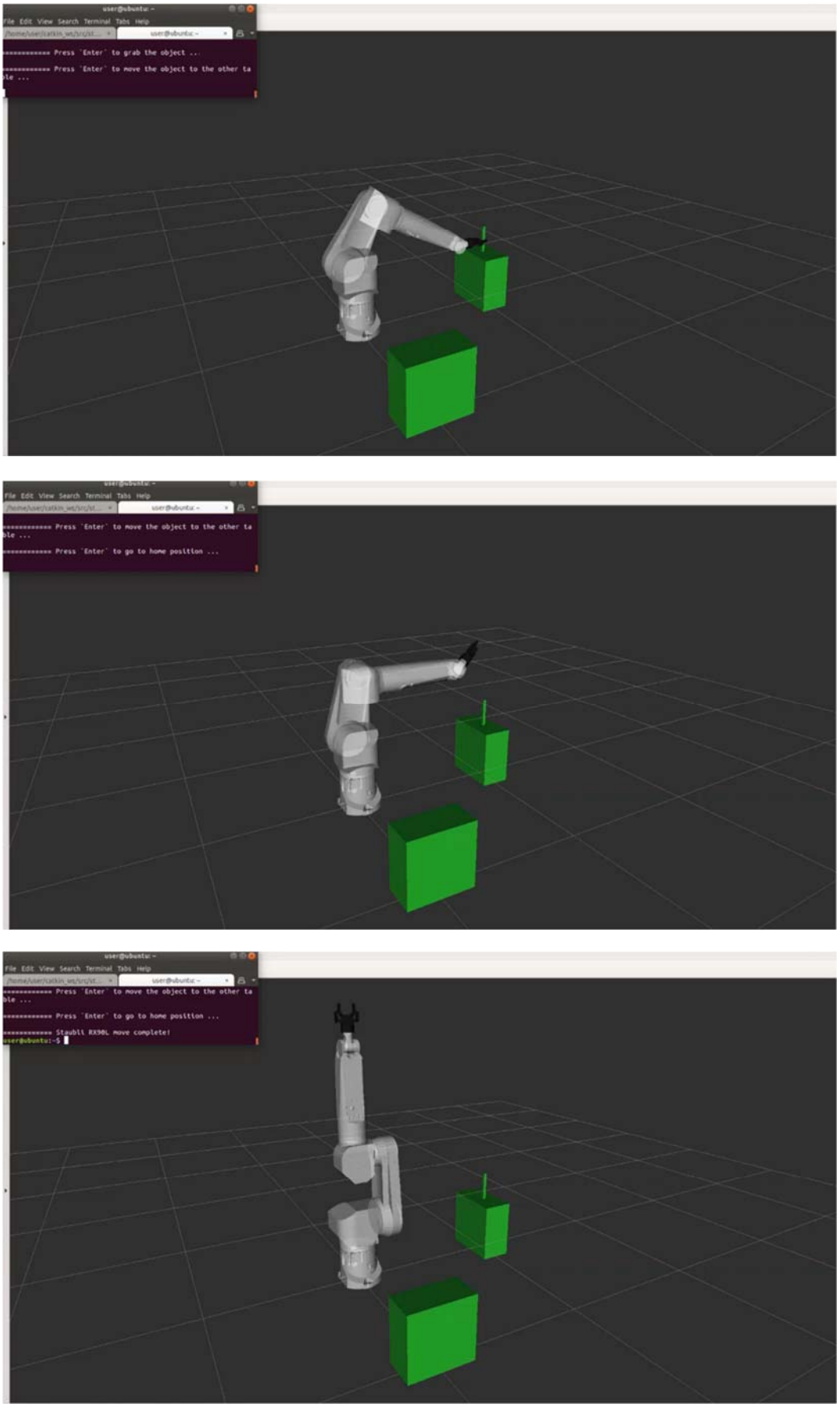
```
def main():
  try:
    rospy.sleep(10)
    print ""
    print "-----------------------------------------------------------"
    print "Stäubli RX90L pick and place"
    print "-----------------------------------------------------------"
    print "Press Ctrl-D to exit at any time"
    print ""
    print "============ Press `Enter` to begin the move ..."
    raw_input()
    tutorial = MoveGroupPythonIntefaceTutorial()

    print "============ Press `Enter` to add the tables and the object to
the planning scene ..."
    raw_input()
    tutorial.add_box()

    print "============ Press `Enter` to grab the object ..."
    raw_input()
    tutorial.go_to_object()
    tutorial.attach_box()

    print "============ Press `Enter` to move the object to the other table
..."
    raw_input()
    tutorial.go_to_2ndtable()
    tutorial.detach_box()
    tutorial.open_gripper()

    print "============ Press `Enter` to go to home position ..."
    raw_input()
    tutorial.go_to_home()

    print "============ Stäubli RX90L move complete!"
  except rospy.ROSInterruptException:
    return
  except KeyboardInterrupt:
    return

if __name__ == '__main__':
  main()
```

Στις παρακάτω εικόνες, παρουσιάζεται η κίνηση του βραχίονα που έχει προγραμματιστεί, στο περιβάλλον του Rviz. Στο παράθυρο του terminal, φαίνονται οι εκτυπώσεις που πραγματοποιούνται και περιμένουν την ανάδραση από τον χρήση με το πάτημα του πλήκτρου ENTER για να προχωρήσουν.


user@ubuntu: ~
File Edit View Search Terminal Tabs Help
Staubli RX90L pick and place
Press Ctrl-D to exit at any time
Press `Enter` to begin the move ...


user@ubuntu: ~
File Edit View Search Terminal Tabs Help
is_diff: False
Press `Enter` to add the tables and the object t
o the planning scene ...
Press `Enter` to grab the object ...


user@ubuntu: ~
File Edit View Search Terminal Tabs Help
o the planning scene ...
Press `Enter` to grab the object ...
Press `Enter` to move the object to the other ta
ble ...



Εικόνα 32: Υλοποίηση σεναρίου στο λογισμικό Rviz

5.2.2 Εκτέλεση τροχιάς με κινηματικό έλεγχο 3 αρθρώσεων

Σε αυτό το σενάριο, πραγματοποιήθηκε εκτέλεση τροχιάς με κινηματικό έλεγχο τριών αρθρώσεων q1,q2,q3. Οι αρθρώσεις 4,5,6 θεωρήθηκαν μηδενικές δηλαδή q4=q5=q6=0.

Υπολογίστηκε αναλυτικά η αντίστροφη κινηματική με ενεργές τις τρείς πρώτες αρθρώσεις:

$$\boldsymbol{q_3} = cos^{-1}\frac{{p_x}^2 + {p_y}^2 + (p_z - l1)^2 - (l3 + l4 + lE)^2 - l2^2}{2l2(l3 + l4 + lE)}$$

$$\boldsymbol{q_1} = tan^{-1}\frac{p_y}{p_x}$$

$$\boldsymbol{q_2} = tan^{-1}\frac{p_y c3(l3 + l4 + lE) + p_y l2 - (p_z - l1)(l3 + l4 + lE)s3s1}{s1(p_z - l1)(c3(l3 + l4 + lE) + l2) + p_y s3(l3 + l4 + lE)}$$

Για την υλοποίηση του σεναρίου, δημιουργήθηκαν δυο αρχεία γλώσσας python. Το πρώτο αρχείο που ονομάστηκε kinematics.py, περιέχει τους υπολογισμούς των γωνιών και της ευθείας κινηματικής του ρομπότ. Η κεντρική κλάση ονομάζεται rx90l_kinematics() και περιέχει τις συναρτήσεις compute_angles καθώς και τους πίνακες μετασχηματισμού.

```
class rx90l_kinematics():
    def __init__(self):

        self.l1 = 0.420
        self.l2 = 0.450
        self.l3 = 0.650
        self.l4 = 0.085
        self.le = 0.100


        pass

    def compute_angles(self, ee_position):

        joint_1 = math.atan2(ee_position[1],ee_position[0])
        joint_3 =
math.acos((ee_position[0]**2+ee_position[1]**2+(ee_position[2]-self.l1)**2-
(self.l3+self.l4+self.le)**2-
self.l2**2)/(2*self.l2*(self.l3+self.l4+self.le)))
        joint_2 =
math.atan2((ee_position[1]*math.cos(joint_3)*(self.l3+self.l4+self.le)+ee_po
sition[1]*self.l2-(ee_position[1]-
self.l1)*(self.l3+self.l4+self.le)*math.sin(joint_3)*math.sin(joint_1)),(mat
h.sin(joint_1)*(ee_position[1]-
self.l1)*(math.cos(joint_3)*(self.l3+self.l4+self.le)+self.l2)+ee_position[1
]*math.sin(joint_3)*(self.l3+self.l4+self.le)))
        joint_4 = 0
        joint_6 = 0
        joint_5 = 0
```

```
        joint_angles = np.matrix([ joint_1, joint_2, joint_3, joint_4,
joint_5, joint_6 ])

        return joint_angles


    def tf_A01(self, r_joints_array):
        tf = np.matrix([[math.cos(r_joints_array[0]) , 0 , -
math.sin(r_joints_array[0]) , 0],\
                        [math.sin(r_joints_array[0]) , 0 ,
math.cos(r_joints_array[0]) , 0],\
                        [0 , -1 , 0 , self.l1],\
                        [0 , 0 , 0 , 1]])
        return tf

    def tf_A02(self, r_joints_array):
        tf_A12 = np.matrix([[math.sin(r_joints_array[1]) ,
math.cos(r_joints_array[1]) , 0 , self.l2*math.sin(r_joints_array[1])],\
                            [-math.cos(r_joints_array[1]) ,
math.sin(r_joints_array[1]) , 0 , -self.l2*math.sin(r_joints_array[1])],\
                            [0 , 0 , 1 , 0],\
                            [0 , 0 , 0 , 1]])
        tf = np.dot( self.tf_A01(r_joints_array), tf_A12 )
        return tf

    def tf_A03(self, r_joints_array):
        tf_A23 = np.matrix([[-math.sin(r_joints_array[2]) , 0 ,
math.cos(r_joints_array[2]) , 0],\
                            [math.cos(r_joints_array[2]) , 0 ,
math.sin(r_joints_array[2]) , 0],\
                            [0 , 1 , 0 , 0],\
                            [0 , 0 , 0 , 1]])
        tf = np.dot( self.tf_A02(r_joints_array), tf_A23 )
        return tf

    def tf_A04(self, r_joints_array):
        tf_A34 = np.matrix([[math.cos(r_joints_array[3]) , 0 , -
math.sin(r_joints_array[3]) , 0],\
                            [math.sin(r_joints_array[3]) , 0 ,
math.cos(r_joints_array[3]) , 0],\
                            [0 , -1 , 0 , self.l3],\
                            [0 , 0 , 0 , 1]])
        tf = np.dot( self.tf_A03(r_joints_array), tf_A34 )
        return tf

    def tf_A05(self, r_joints_array):
        tf_A45 = np.matrix([[math.cos(r_joints_array[4]) , 0 ,
math.sin(r_joints_array[4]) , 0],\
                            [math.sin(r_joints_array[4]) , 0 , -
math.cos(r_joints_array[4]) , 0],\
                            [0 , 1 , 0 , 0],\
                            [0 , 0 , 0 , 1]])
        tf = np.dot( self.tf_A04(r_joints_array), tf_A45 )
        return tf

    def tf_A06(self, r_joints_array):
        tf_A56 = np.matrix([[math.cos(r_joints_array[5]) , -
math.sin(r_joints_array[5]) , 0 , 0],\
                            [math.sin(r_joints_array[5]) ,
math.cos(r_joints_array[5]) , 0 , 0],\
                            [0 , 0 , 1 , self.l4+self.le],\
                            [0 , 0 , 0 , 1]])
        tf = np.dot( self.tf_A05(r_joints_array), tf_A56 )
        return tf
```

Το δεύτερο αρχείο, περιέχει όλα τα απαραίτητα nodes-topics για την έναρξη της προσομοίωσης καθώς και τον αλγόριθμο της πολυωνυμικής τροχιάς (3ου βαθμού). Αρχικά, γίνεται εισαγωγή των απαραίτητων στοιχείων. Όπως φαίνεται, περιλαμβάνει και η κλάση rx90l_kinematics().

```
import sys
import copy
import rospy, roslib
from sensor_msgs.msg import LaserScan
#import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from std_msgs.msg import String
from std_msgs.msg import Float64
from sensor_msgs.msg import JointState
#Math imports
from math import sin, cos, atan2, pi, sqrt
from numpy.linalg import inv, det, norm, pinv
import numpy as np
import time as t
from numpy import array, inf

from kinematics import rx90l_kinematics
```

Το επαναληπτικό κομμάτι του κώδικα φαίνεται παρακάτω.

```
while not rospy.is_shutdown():
            laserlist = array(self.laser.ranges)  #ekxorisi timwn tou laser
ston pinaka laserlist
            laserlist[laserlist == inf] = 200       #antikatastasi twn inf
me enan arithmo gia na ginei i sigkrisi
            if(np.all(laserlist == 200)) :      #oso den vlepei tipota to
laser
```

Ορισμός των σημείων του y με τα x,z να παραμένουν σταθερά.

```
yd0 = 0.2
yd1 = 0.9
yd2 = 0.2
xd0 = 0.8
xd1 = 0.8
zd0 = 0.2
zd1 = 0.2
Tf = 12

dt = 0.01
kmax = Tf/dt +1

xd = np.empty(int(kmax))
yd = np.empty(int(kmax))
zd = np.empty(int(kmax))

yd[0] = yd0
xd[0] = xd0
zd[0] = zd0
lambda_x = (xd1-xd0)/Tf;
lambda_z = (zd1-zd0)/Tf;
a1 = (yd1 - yd0) / 8
a2 = (yd2 - yd1) / 8
```

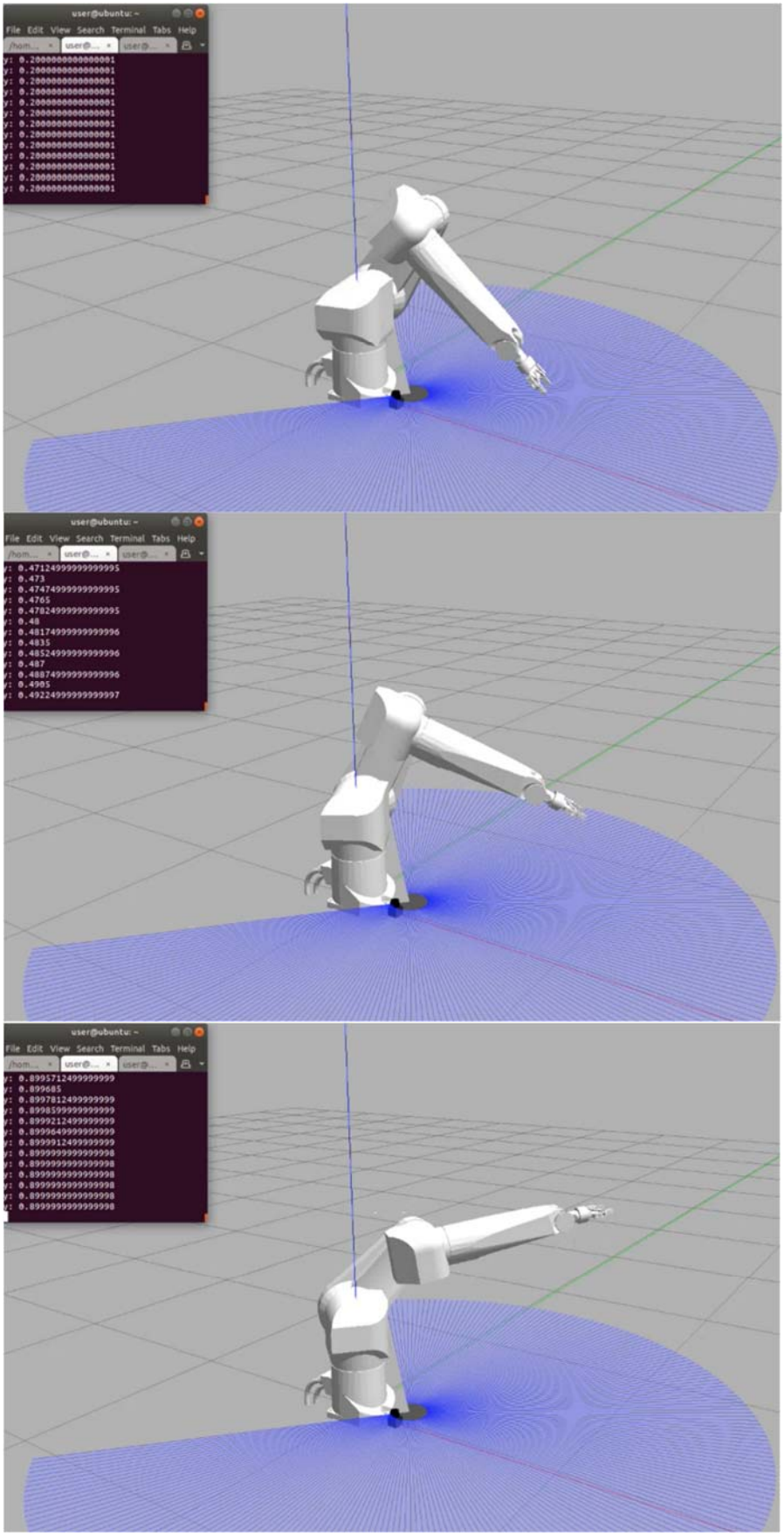
Εκκίνηση αλγορίθμου για την εκτέλεση ομαλής πολυωνυμικής τροχιάς

```
            for i in range(1,int(kmax),1):

                tt = dt*i
                if (tt <= 1):
                    yd[i] = (a1*tt**2 + yd0)
                elif (tt > 1) and (tt <= 4):
                    yd[i] = (yd[int(1/dt)] + a1*2*(tt - 1))
                elif (tt > 4) and (tt <= 5):
                    yd[i] = (yd[int(4/dt)] + a1*2*(tt - 4) - a1*(tt -
4)**2)
                elif (tt > 5) and (tt <=6):
                    yd[i] = (yd[int(i-1)])
                elif (tt > 6) and (tt <= 7):
                    yd[i] = (a2*(tt-6)**2 + yd1)
                elif (tt > 7) and (tt <= 10):
                    yd[i] = (yd[int(7/dt)] + a2*2*(tt - 7))
                elif (tt > 10) and (tt <= 11):
                    yd[i] = (yd[int(10/dt)] + a2*2*(tt - 10) - a2*(tt -
10)**2)
                else:
                    yd[i] = (yd[int(i-1)])


                xd[i] = xd[i-1] + lambda_x*dt;
                zd[i] = zd[i-1] + lambda_z*dt;
                print("y:",yd[i])

                ee_p3 = [xd[i], yd[i], zd[i]]
```

Κλήση της συνάρτησης joints_angles προκειμένου να βρεθούν οι γωνίες των τριών αρθρώσεων με βάση το σημείο x,y,z που έχει δοθεί.

```
                joint_angles = self.kinematics.compute_angles(ee_p3)
                self.joint_angpos[0] = joint_angles[0,0]
                self.joint_angpos[1] = joint_angles[0,1]
                self.joint_angpos[2] = joint_angles[0,2]

                # Publish the new joint's angular positions
                self.joint1_pos_pub.publish(self.joint_angpos[0])
                self.joint2_pos_pub.publish(self.joint_angpos[1])
                self.joint3_pos_pub.publish(self.joint_angpos[2])
                self.joint4_pos_pub.publish(self.joint_angpos[3])
                self.joint5_pos_pub.publish(self.joint_angpos[4])
                self.joint6_pos_pub.publish(self.joint_angpos[5])
                self.joint7_pos_pub.publish(self.joint_angpos[6])

                self.pub_rate.sleep()


        else:
            print("empodio")
```

Στις επόμενες εικόνες παρουσιάζονται το ρομπότ στην αρχή της κίνησης (y=0.2), η μεταβολή των τιμών της θέσης y κατά την διάρκεια της κίνησης και το ρομπότ στην τελική του θέση y=0.9



Εικόνα 33: Το ρομπότ εκτελεί τροχιά με κινηματικό έλεγχο κατά τον άξονα y.

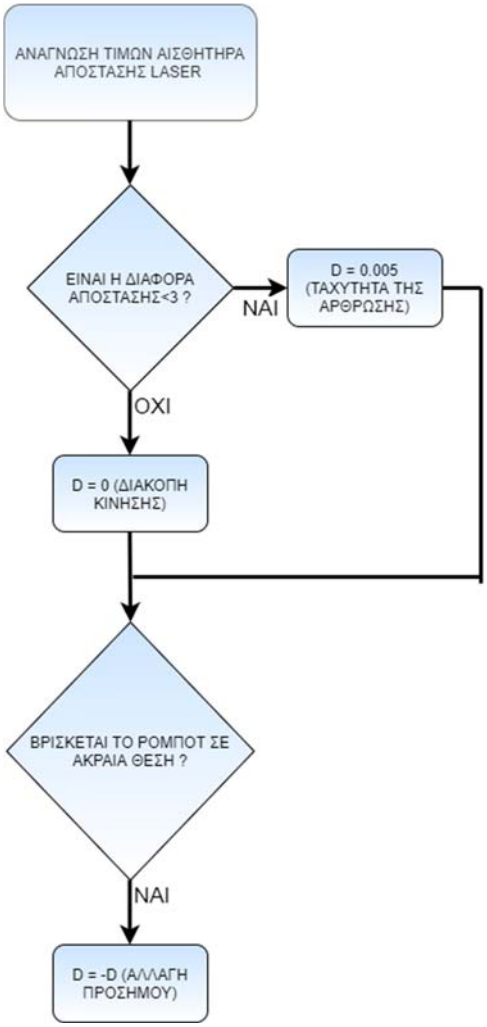
5.3 Σταμάτημα ρομπότ με την είσοδο του ανθρώπου στο χώρο εργασίας

Στο συγκεκριμένο σενάριο, όταν ο άνθρωπος εισέρχεται στον χώρο εργασίας του ρομπότ τότε αυτό σταματά.

Το ρομπότ εκτελεί μια επαναλαμβανόμενη κίνηση περιστροφής της 1ης άρθρωσης.

Έχει χρησιμοποιηθεί ένας αισθητήρας laser ο οποίος επιστρέφει σε ένα πίνακα την απόσταση της βάσης του ρομπότ από το σύνδεσμο του ανθρώπου που βρίσκεται πιο κοντά σε αυτό (πχ. δεξί χέρι ή αριστερό πόδι). Αν η απόσταση είναι κάτω από ένα ορισμένο όριο τότε η κίνηση σταματά.

Ο αισθητήρας υπάρχει στο gazebo και μεταφέρει τα δεδομένα του μέσω ενός topic στο rviz. Όπως παρουσιάζεται και στις σχετικές εικόνες, είναι ορατά τα βήματα (footprints) του ανθρώπου στον χώρο.



Εικόνα 34: Διάγραμμα ροής σεναρίου σταματήματος του ρομπότ όταν ο άνθρωπος .

Αρχικά γίνεται εισαγωγή των απαραίτητων βιβλιοθηκών.

```
import sys
import copy
import rospy, roslib
from sensor_msgs.msg import LaserScan
import moveit_msgs.msg
import geometry_msgs.msg
from std_msgs.msg import String
from std_msgs.msg import Float64
from sensor_msgs.msg import JointState
from math import sin, cos, atan2, pi, sqrt
from numpy.linalg import inv, det, norm, pinv
import numpy as np
import time as t
from numpy import array, inf
```

Η κλάση rx90l_controller() περιέχει όλες τις συναρτήσεις που απαιτούνται για να υλοποιηθεί αυτό το σενάριο.

```
class rx90l_controller():
    """Class to compute and publish joints positions"""
    def __init__(self,rate):
        self.joint_angpos = [0, 0, 0, 0, 0, 0, 0]
        self.joint_states = JointState()

         self.joint_states_sub = rospy.Subscriber('/rx90l/joint_states',
JointState, self.joint_states_callback, queue_size=1)
        self.joint1_pos_pub =
rospy.Publisher('/rx90l/joint1_position_controller/command', Float64,
queue_size=1)
        self.joint2_pos_pub =
rospy.Publisher('/rx90l/joint2_position_controller/command', Float64,
queue_size=1)
        self.joint3_pos_pub =
rospy.Publisher('/rx90l/joint3_position_controller/command', Float64,
queue_size=1)
        self.joint4_pos_pub =
rospy.Publisher('/rx90l/joint4_position_controller/command', Float64,
queue_size=1)
        self.joint5_pos_pub =
rospy.Publisher('/rx90l/joint5_position_controller/command', Float64,
queue_size=1)
        self.joint6_pos_pub =
rospy.Publisher('/rx90l/joint6_position_controller/command', Float64,
queue_size=1)
        self.joint7_pos_pub =
rospy.Publisher('/rx90l/gripper_controller/command', Float64, queue_size=1)
        self.laserscan_sub = rospy.Subscriber('/rx90l/laser/scan',
LaserScan, self.callback, queue_size=1)

        #Publishing rate
        self.period = 1.0/rate
        self.pub_rate = rospy.Rate(rate)

        self.publish()
```

Παραπάνω έχουν οριστεί οι controllers(ελεγκτές) που ελέγχουν τις τιμές των αρθρώσεων. Οι τιμές αυτές δημοσιεύονται (γίνονται published) από τον χρήστη κατά την διάρκεια του προγράμματος προκειμένου να επιτευχθεί η κίνηση. Τα δεδομένα από τον αισθητήρα laser, διαβάζονται από τον κώδικα προκειμένου να γίνουν οι απαραίτητοι υπολογισμοί. Καλώντας την συνάρτηση callback, η διάταξη του

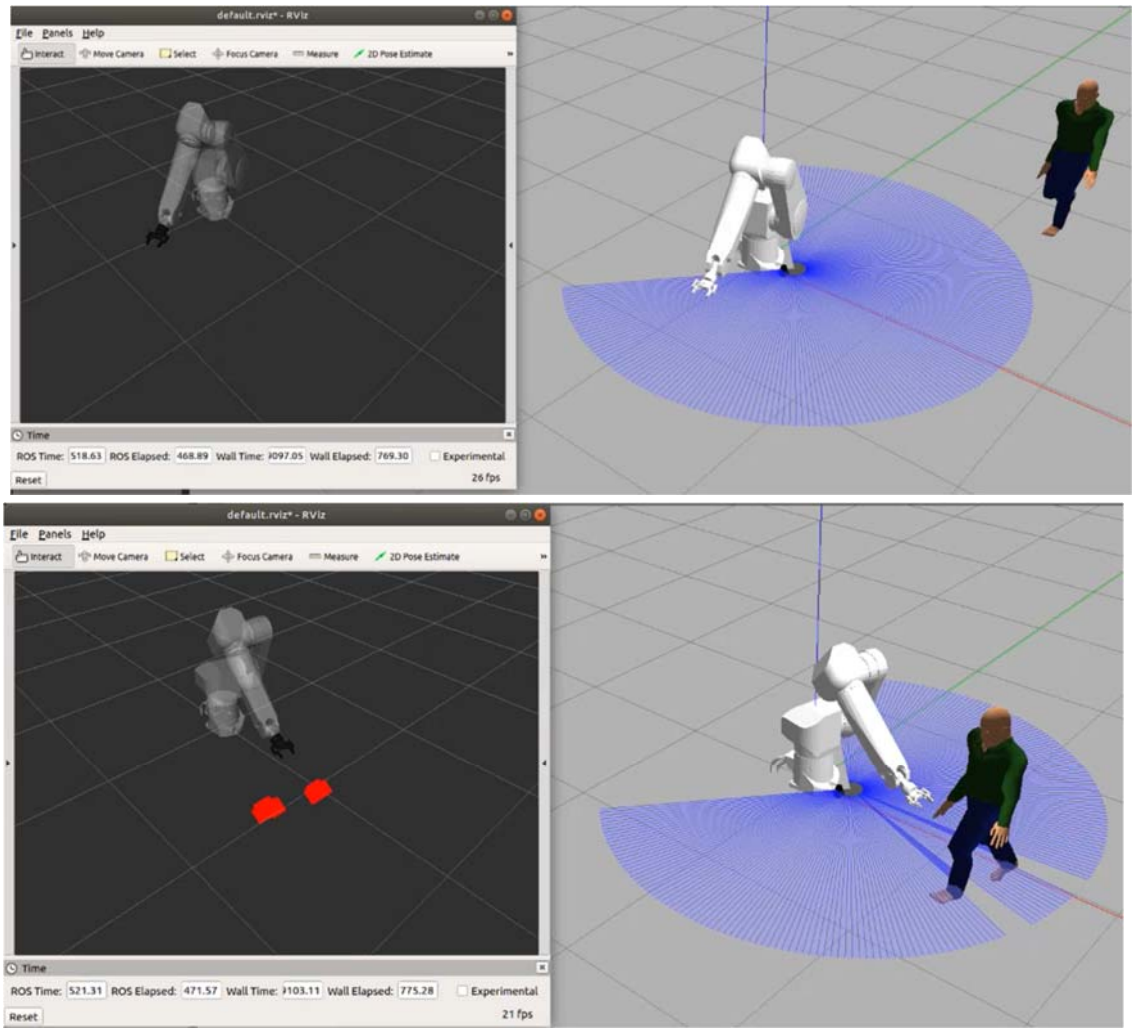
μηνύματος περνά στην μεταβλητή self.laser προκειμένου να αξιοποιηθεί η πληροφορία.

```
    #SENSING CALLBACKS
    def joint_states_callback(self, msg):
        # ROS callback to get the joint_states
        self.joint_states = msg
        # (e.g. the angular position of joint 1 is stored in ::
self.joint_states.position[0])

    def callback(self, msg):
        self.laser = msg

    def publish(self):

        # set configuration
        self.joint_angpos = [0, 0.5, 1.8, 0, 0.5, 0, 0]
        tmp_rate = rospy.Rate(1)
        tmp_rate.sleep()
        self.joint1_pos_pub.publish(self.joint_angpos[0])
        tmp_rate.sleep()
        self.joint2_pos_pub.publish(self.joint_angpos[1])
        self.joint3_pos_pub.publish(self.joint_angpos[2])
        self.joint4_pos_pub.publish(self.joint_angpos[3])
        self.joint5_pos_pub.publish(self.joint_angpos[4])
        self.joint6_pos_pub.publish(self.joint_angpos[5])
        self.joint7_pos_pub.publish(self.joint_angpos[6])
        tmp_rate.sleep()
        print("The system is ready to execute your algorithm...")
        i = 0
        t = 0.01
        d = t
        akraia_thesi = False
        rostime_now = rospy.get_rostime()
        time_now = rostime_now.to_nsec()

        while not rospy.is_shutdown():
            laserlist = array(self.laser.ranges)
            laserlist[laserlist == inf] = 200

            if (np.any(laserlist < 3)) :
                 d = 0.005
            else:
                 d = 0

            if ((i >= 1.4) or (i <= -1.4)) :
                d = -d

            i = i + d
            self.joint_angpos[0] = i
            rospy.sleep(0.01)
```

Για όσο χρονικό διάστημα δεν έχει σταματήσει η λειτουργία του κώδικα (while not rospy.is shutdown), θα εκτελείται το συγκεκριμένο τμήμα κώδικα. Στον πίνακα laserlist αποθηκεύονται οι τιμές του αισθητήρα (self.laser.ranges) προκειμένου να συγκριθούν. Για τον λόγο ότι όταν ο αισθητήρας δεν ανιχνεύει κάτι βγάζει σαν έξοδο την τιμή inf(infinity), έχει γίνει η μετατροπή των τιμών αυτών σε μια μέγιστη τιμή που είναι το 200.

```
            # Publish the new joint's angular positions
            self.joint1_pos_pub.publish(self.joint_angpos[0])
```

```
            self.joint2_pos_pub.publish(self.joint_angpos[1])
            self.joint3_pos_pub.publish(self.joint_angpos[2])
            self.joint4_pos_pub.publish(self.joint_angpos[3])
            self.joint5_pos_pub.publish(self.joint_angpos[4])
            self.joint6_pos_pub.publish(self.joint_angpos[5])
            self.joint7_pos_pub.publish(self.joint_angpos[6])

            self.pub_rate.sleep()

    def turn_off(self):
        pass

def controller_py():
    # Starts a new node
    rospy.init_node('controller_node', anonymous=True)
    controller = rx90l_controller(100)
    rospy.on_shutdown(controller.turn_off)
    rospy.spin()

if __name__ == '__main__':
    try:
        controller_py()
    except rospy.ROSInterruptException:
        pass
```

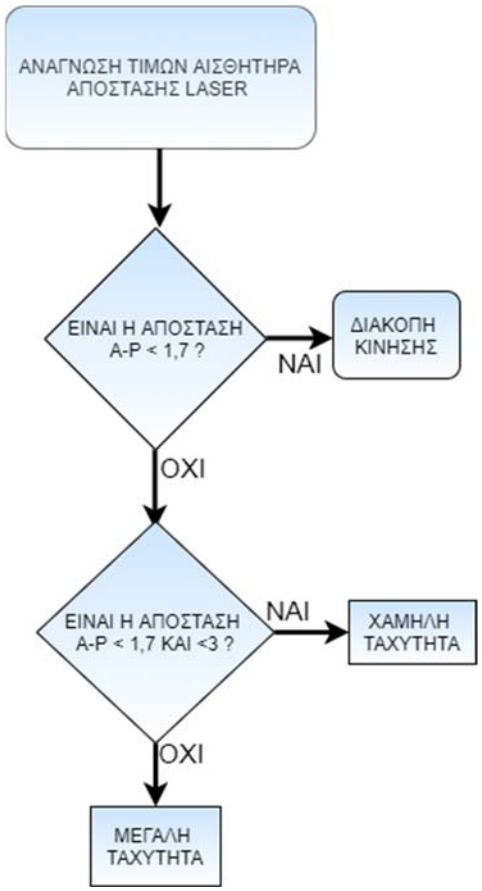


Εικόνα 35: Ο άνθρωπος εισέρχεται στον χώρο εργασίας και το ρομπότ σταματά.

Στην πρώτη εικόνα, ο άνθρωπος βρίσκεται εκτός του χώρου εργασίας του ρομπότ συνεπώς αυτό εκτελεί κανονικά την κίνησή του. Όταν ο άνθρωπος εισέρχεται (κάτω εικόνα), το ρομπότ ακινητοποιείται και τα βήματα του ανθρώπου απεικονίζονται στο λογισμικό rviz σε πραγματικό χρόνο.

5.4 Μείωση ταχύτητας ρομπότ καθώς πλησιάζει ο άνθρωπος

Το συγκεκριμένο σενάριο αφορά την είσοδο του ανθρώπου στον χώρο εργασίας και την μείωση της ταχύτητας περιστροφής του ρομπότ. Δημιουργήθηκαν τρείς ταχύτητες για τον ρομποτικό βραχίονα. Καθώς ο άνθρωπος πλησιάζει προς το ρομπότ, η περιστροφική άρθρωση χαμηλώνει την ταχύτητά της μέχρι που σταματά όταν πλέον η απόσταση από τα πόδια του ανθρώπου μέχρι την βάση του ρομπότ είναι αρκετά μικρή. Ακολουθεί το διάγραμμα ροής.



Εικόνα 36: Διάγραμμα ροής του σεναρίου μείωσης ταχύτητας του ρομπότ με την απόσταση από τον άνθρωπο

Το κύριο τμήμα του κώδικα που πραγματοποιεί αυτή την κίνηση παρουσιάζεται παρακάτω.

```
while not rospy.is_shutdown():
            laserlist = array(self.laser.ranges)  #ekxorisi timwn tou laser
ston pinaka laserlist
            laserlist[laserlist == inf] = 200
```

```
                                        #antikatastasi twn inf me enan
arithmo gia na ginei i sigkrisi
```

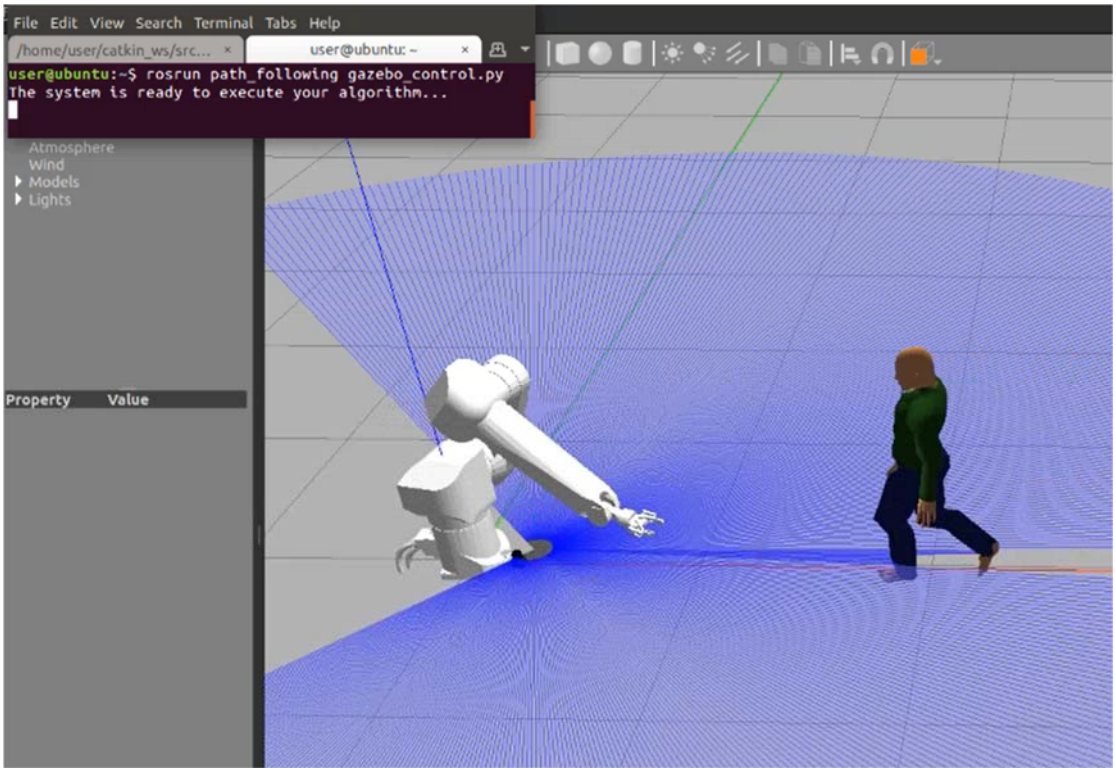
Αρχικά γίνεται έλεγχος για την κατεύθυνση του ρομπότ (δεξιόστροφη ή αριστερόστροφη). Στην συνέχεια, εξετάζεται η απόσταση της βάσης του ρομπότ από τον άνθρωπο.

Αν η απόσταση είναι >=3 τότε ο ρυθμός αύξησης είναι 0.02 (μεγάλη ταχύτητα)
Αν η απόσταση είναι >=1.7 και <3 τότε ο ρυθμός αύξησης είναι 0.005 (μικρή ταχύτητα)
Αν η απόσταση είναι <1.7 τότε ο ρυθμός αύξησης είναι 0 (σταμάτημα)

```
            if (i >= 1.4):
                #d = -d
                #temp = d
                akraia_thesi = 1
                if (np.any(laserlist < 1.7)) :
                    d = 0
                elif (np.any(laserlist >= 1.7) and np.any(laserlist < 3)):
                    d = -0.005
                else:
                    d = -0.02

            elif (i <= -1.4):
                #d = d
                #temp = d
                akraia_thesi = 2
                if (np.any(laserlist < 1.7)) :
                    d = 0
                elif (np.any(laserlist >= 1.7) and np.any(laserlist < 3)):
                    d = 0.005
                else:
                    d = 0.02

            else:
                #d = temp

                if (np.any(laserlist < 1.7)) :      #oso den vlepei tipota
to laser
                    if (akraia_thesi == 1):
                        d = 0
                    elif (akraia_thesi == 2):
                        d = 0

                elif (np.any(laserlist >= 1.7) and np.any(laserlist < 3)):
                    if (akraia_thesi == 1):
                        d = -0.005
                    elif (akraia_thesi == 2):
                        d = 0.005

                else:
                    if (akraia_thesi == 1):
                        d = -0.02
                    elif (akraia_thesi == 2):
                        d = 0.02



            i = i + d
            self.joint_angpos[0] = i
            rospy.sleep(0.01)

            # Publish the new joint's angular positions
            self.joint1_pos_pub.publish(self.joint_angpos[0])
```

```
            self.joint2_pos_pub.publish(self.joint_angpos[1])
            self.joint3_pos_pub.publish(self.joint_angpos[2])
            self.joint4_pos_pub.publish(self.joint_angpos[3])
            self.joint5_pos_pub.publish(self.joint_angpos[4])
            self.joint6_pos_pub.publish(self.joint_angpos[5])
            self.joint7_pos_pub.publish(self.joint_angpos[6])

            self.pub_rate.sleep()
```

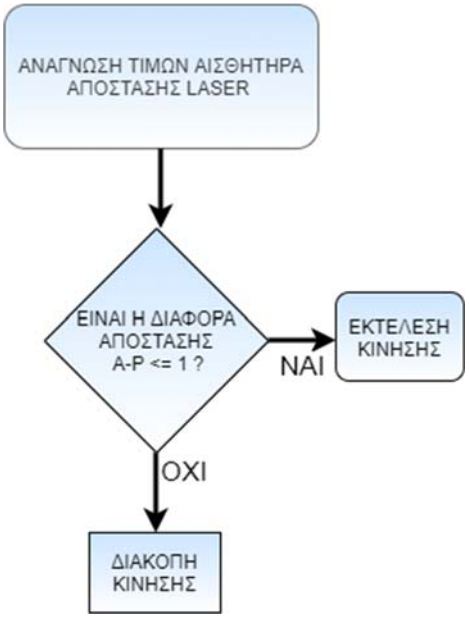
Στην παρακάτω εικόνα, φαίνεται το σενάριο να εκτελείται στο περιβάλλον του gazebo.



Εικόνα 37: Ο άνθρωπος εισέρχεται στον χώρο εργασίας και η ταχύτητα του βραχίονα μειώνεται.

5.5 Κίνηση του ρομπότ όταν ο άνθρωπος βρίσκεται σε συγκεκριμένη απόσταση

Στο συγκεκριμένο σενάριο, γίνεται εκτέλεση τροχιάς μόνο όταν ο άνθρωπος βρίσκεται σε μία συγκεκριμένη θέση. Το ρομπότ για το χρονικό διάστημα που δεν υπάρχει κάποιος μέσα στον χώρο εργασίας εμφανίζει το μήνυμα “No human detected in workspace". Η μέτρηση της απόστασης ανθρώπου-ρομπότ, γίνεται από το άκρο του βραχίονα (τελικό στοιχείο δράσης) μέχρι και το κέντρο μάζας του ανθρώπου



Εικόνα 38: Διάγραμμα ροής σεναρίου.

.

Παρακάτω ακολουθεί το βασικό κομμάτι κώδικα που εκτελεί αυτή την επαναληπτική διαδικασία.

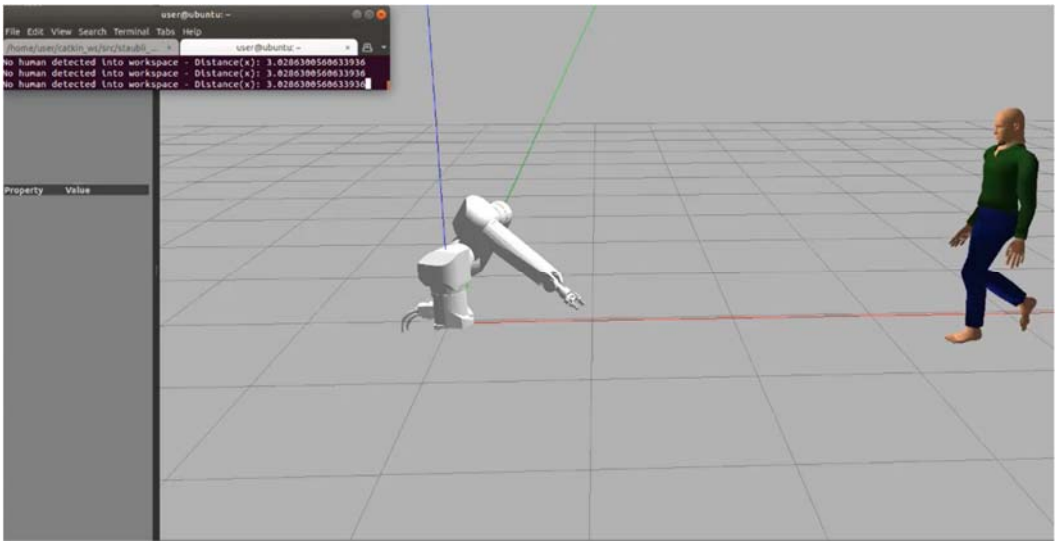
```
def joint_states_callback(self, msg):
        # ROS callback to get the joint_states

        self.joint_states = msg
        # (e.g. the angular position of joint 1 is stored in ::
self.joint_states.position[0])

    def callback(self, msg):

        self.laser = msg


    def link_callback(self, msg):

        self.link = msg
```

Η συνάρτηση *joint_states_callback* επιστρέφει την δομή του μηνύματος που αποθηκεύεται για κάθε άρθρωση. Για παράδειγμα, για την 1η άρθρωση, η γωνιακή θέση της αποθηκεύεται στο *self.joint_states.position[0]*.

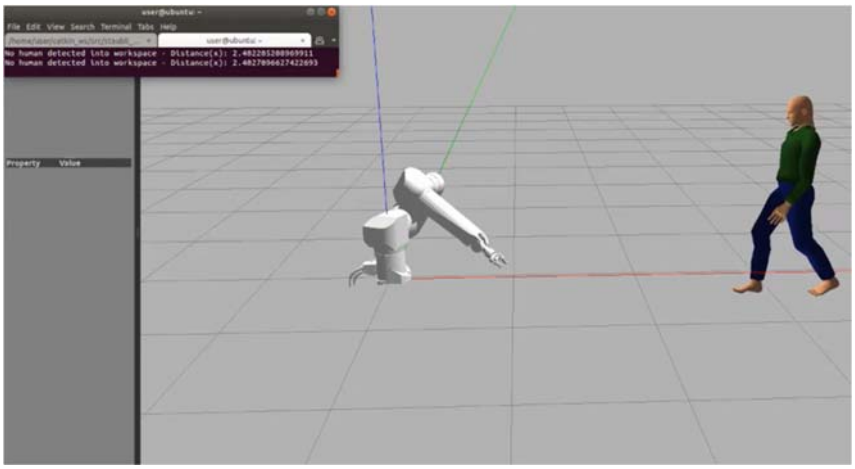
Η συνάρτηση *callback*, επιστρέφει το μήνυμα που μεταδίδει το laser στο περιβάλλον του gazebo με τον ίδιο τρόπο όπως και στα προηγούμενα σενάρια. Η συνάρτηση *link_callback*, επιστρέφει τις τιμές που αφορούν τις αρθρώσεις. Η ζητούμενες τιμές είναι οι θέσεις του ρομπότ και του ανθρώπου. Όπως φαίνεται και στον κώδικα παρακάτω, η τιμή του άξονα x για τον ρομπότ αποθηκεύεται στην θέση: *self.link.pose[39].position.x* και για τον άνθρωπο στην θέση: *self.link.pose[1].position.x* . Αν η διαφορά μεταξύ τους είναι μικρότερη του 1, τότε το ρομπότ ξεκίνα την κίνηση του με τις περιστροφές της 1ης και 2ης άρθρωσης.

```
while not rospy.is_shutdown():

            robot_x = self.link.pose[39].position.x
            robot_y = self.link.pose[39].position.y
            robot_z = self.link.pose[39].position.z

            actor_x = self.link.pose[1].position.x
            actor_y = self.link.pose[1].position.y
            actor_z = self.link.pose[1].position.z

            diff_x = actor_x - robot_x
            diff_y = robot_y - actor_y


            if (diff_x <= 1) :      #oso den vlepei tipota to laser

                i = i + d
                p = p + r
                self.joint_angpos[0] = i
                self.joint_angpos[2] = p
                rospy.sleep(0.01)


                if (i >= 0.6):
                    d = -t

                if (i <= -0.6):
                    d = t

                if (p >= 1.6):
                    r = -s

                if (p <= 0.5):
                    r = s


                    # Publish the new joint's angular positions
                self.joint1_pos_pub.publish(self.joint_angpos[0])
                self.joint2_pos_pub.publish(self.joint_angpos[1])
                self.joint3_pos_pub.publish(self.joint_angpos[2])
                self.joint4_pos_pub.publish(self.joint_angpos[3])
                self.joint5_pos_pub.publish(self.joint_angpos[4])
                self.joint6_pos_pub.publish(self.joint_angpos[5])
                self.joint7_pos_pub.publish(self.joint_angpos[6])

                print("Collaboration started - Distance(x):", diff_x)

                self.pub_rate.sleep()


            else:
                print("No human detected into workspace - Distance(x):",
diff_x)
```

Στην πρώτη εικόνα, ο άνθρωπος δεν έχει εισέλθει ακόμα στον χώρο εργασίας. Στο terminal εμφανίζεται το μήνυμα: No human detected into workspace με την απόσταση μεταξύ του άκρου του ρομπότ και του ανθρώπου κατά τον x άξονα σε πραγματικό χρόνο. Στην δεύτερη και τρίτη εικόνα, η συνθήκη της απόστασης έχει ικανοποιηθεί και η κίνηση του ρομπότ έχει ξεκινήσει. Στην οθόνη εμφανίζεται το μήνυμα Collaboration-started και η απόσταση ανθρώπου-ρομπότ.

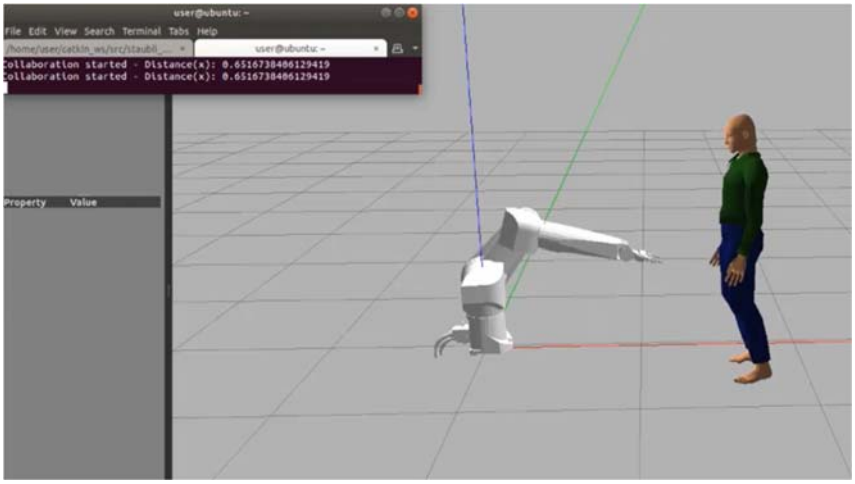


Εικόνα 39: Ο άνθρωπος ξεκινά να κατευθύνεται προς το ρομπότ. Μέγιστη απόσταση μεταξύ τους περίπου 3 μέτρα.

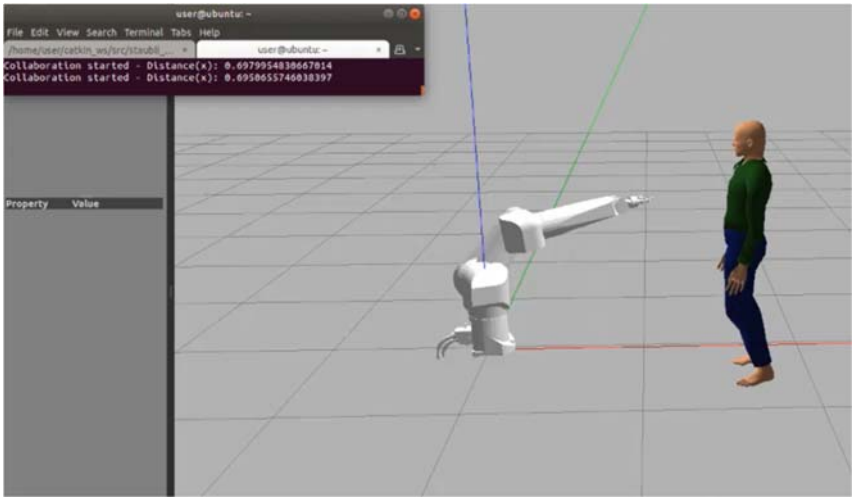


Εικόνα 40: Ο άνθρωπος συνεχίζει το περπάτημα προς το ρομπότ και βρίσκεται ακόμα εκτός χώρου εργασίας. Απόσταση μεταξύ τους περίπου 2,5 μέτρα. μηνύματος.

Εικόνα 41: Το τελευταίο βήμα του ανθρώπου πριν εισέλθει στον χώρο εργασίας και ξεκινήσει η περιστροφική κίνηση του ρομπότ. Απόσταση περίπου 1,2 μέτρα.



Εικόνα 42: Εκκίνηση λειτουργίας του ρομπότ με τον άνθρωπο να βρίσκεται στον χώρο εργασίας. Εμφάνιση μηνύματος συνεργασίας (collaboration started).



Εικόνα 43: Ενδιάμεση θέση του ρομπότ καθώς εκτελεί την περιστροφική κίνηση με τον άνθρωπο να βρίσκεται εντός του χωρίου εργασίας.

5.6 Απομάκρυνση του ρομπότ από τον άνθρωπο

Στο παρακάτω σενάριο, καθώς ο άνθρωπος εισέρχεται στον χώρο εργασίας, το ρομπότ εκτελεί κίνηση αποφυγής σε καρτεσιανό σύστημα συντεταγμένων προς τα αρνητικά. Η απομάκρυνση του ρομπότ γίνεται κατά τον άξονα x και βάσει της απόστασης του άκρου του ρομπότ από το κέντρο μάζας του ανθρώπου.

Παρακάτω δίνεται το κεντρικό κομμάτι κώδικα υλοποίησης του σεναρίου.

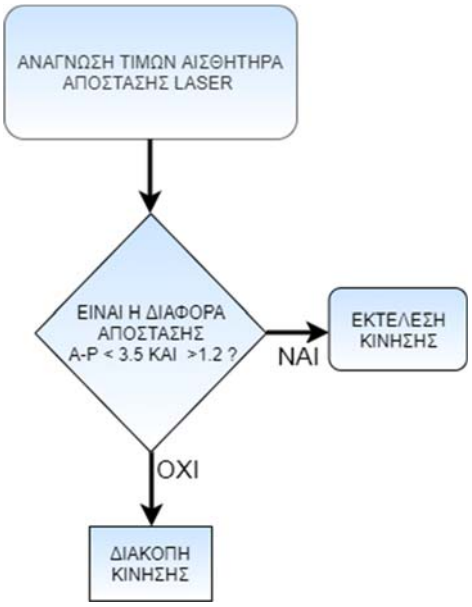
```
while not rospy.is_shutdown():
```

- *Εκχώρηση της θέσης του ρομπότ στην μεταβλητή thesi. Για την εύρεση της θέσης στον άξονα x χρησιμοποιείται το robot_x όπως φαίνεται παρακάτω.*

```
    thesi = move_group.get_current_pose().pose

    robot_x = thesi.position.x
    robot_y = thesi.position.y
```

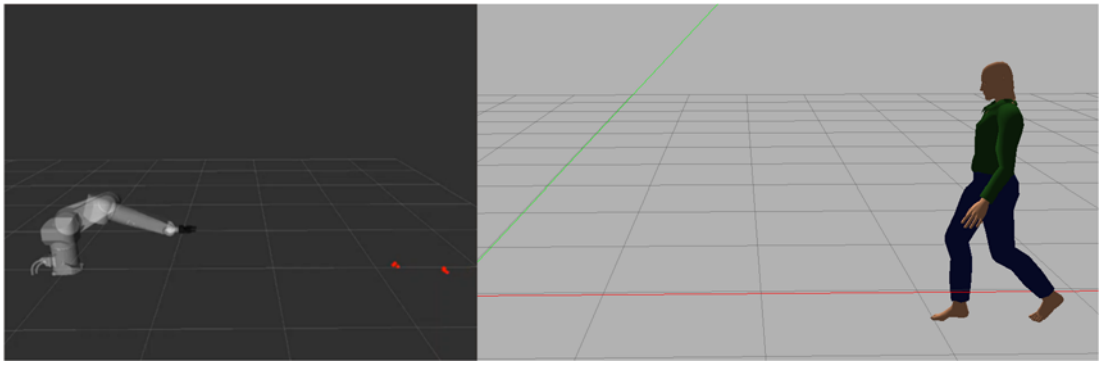
```
            robot_z = thesi.position.z


            diff_x = actor_x - robot_x
            diff_y = robot_y - actor_y

            if (diff_x < 3.5 and diff_x >1.2):

                 waypoints = []
                thesi.position.x -= scale * 0.06  # Second move
forward/backwards in (x)
                waypoints.append(copy.deepcopy(thesi))

                (plan, fraction) = move_group.compute_cartesian_path(
                                             waypoints,   # waypoints to
follow
                                             0.01,        # eef_step
                                             0.0)         # jump_threshold

                move_group.execute(plan, wait=True)
```

Η κίνηση του ρομπότ γίνεται εφόσον η απόσταση (στον άξονα x) μεταξύ του ανθρώπου-ρομπότ βρίσκεται στο εύρος 1,2-3,5. Γίνεται αρχικοποίηση του πίνακα που περιέχει τα σημεία της μετατόπισης waypoint[]. Η θέση του ρομπότ μετατοπίζεται κατά 6% προς τα αρνητικά. Ο υπολογισμός του σημείου γίνεται με την εντολή: *thesi.position.x -= scale*0.06*. Έπειτα, η θέση αυτή αποθηκεύεται στο τέλος του πίνακα waypoints. Προκειμένου να γίνει η κίνηση πρέπει πρώτα να πραγματοποιηθεί ο υπολογισμός της στο framework του move_group. Αυτό γίνεται με την εντολή *move_group.compute_cartesian_path*. Η εκτέλεση της κίνησης πραγματοποιείται με την εντολή *move_group.execute*.

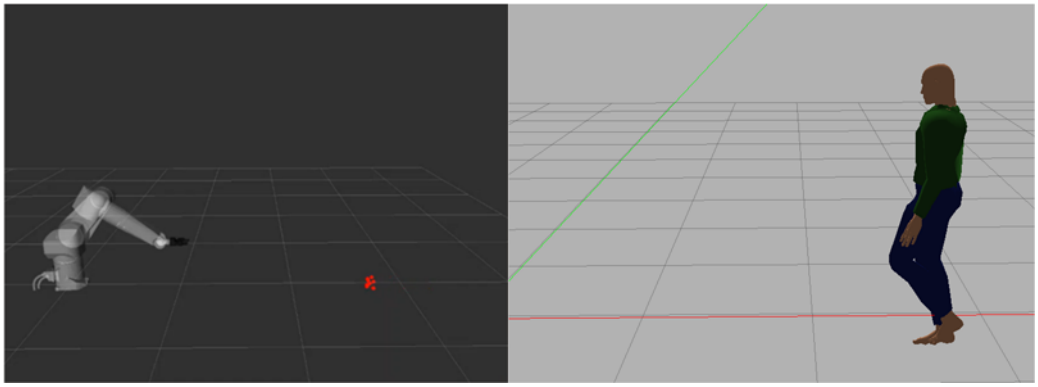


Εικόνα 44: Σενάριο ροής απομάκρυνσης του ρομπότ.

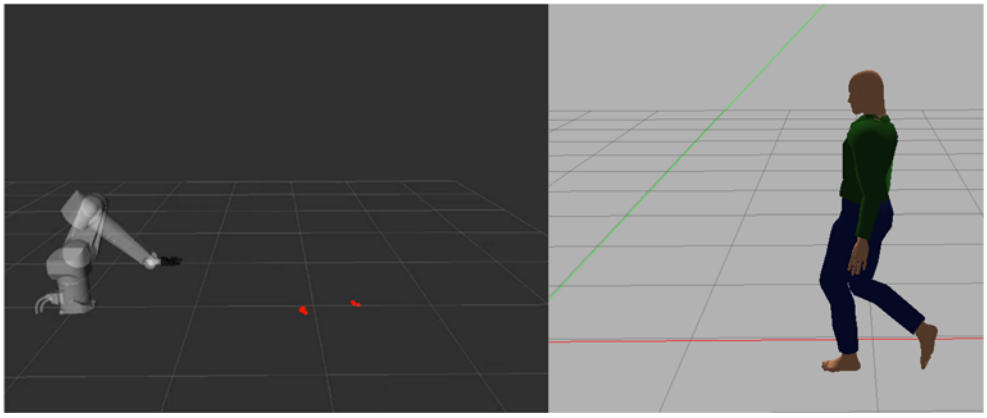
Στις παρακάτω εικόνες, απεικονίζεται η σταδιακή απομάκρυνση του ρομπότ από τον άνθρωπο.



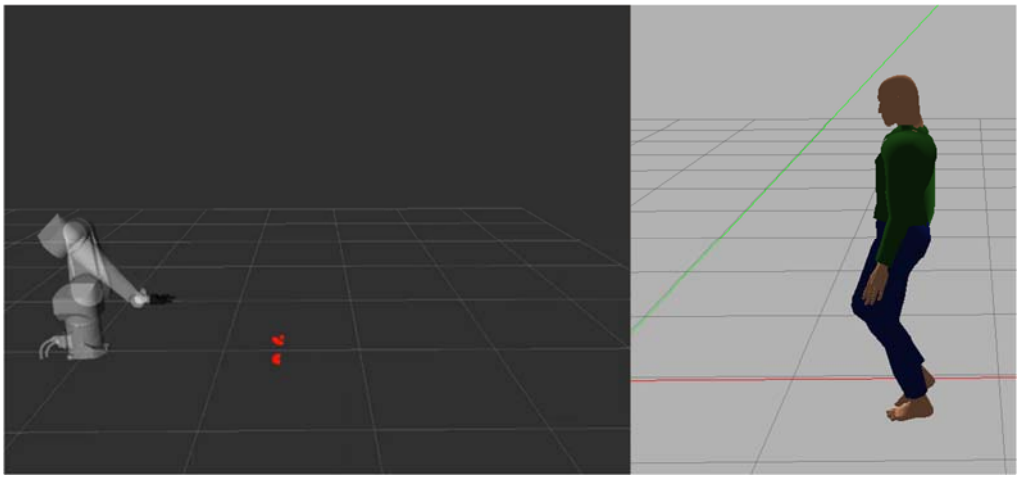
Εικόνα 45: Μέγιστη απόσταση A-P. Αρχή κίνησης ανθρώπου.



Εικόνα 46: Ο άνθρωπος πλησιάζει. Το ρομπότ εκτελεί την πρώτη κίνηση προς τα αρνητικά στον άξονα x.



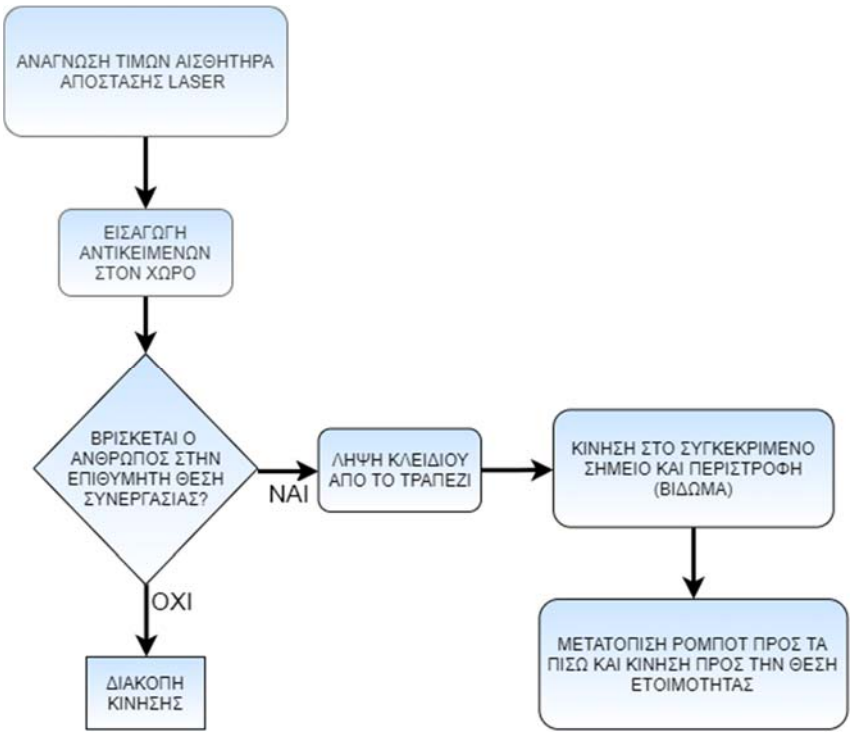
Εικόνα 47: Ενδιάμεση θέση της κίνησης του ανθρώπου. Το ρομπότ συνεχίζει να πραγματοποιεί οπισθοχώρηση.



Εικόνα 48: Ελάχιστη απόσταση A-P. Μέγιστη υποχώρηση του ρομπότ για την διατήρηση ασφαλούς απόστασης.

5.7 Εκτέλεση συνεργατικού καθήκοντος ανθρώπου-ρομπότ

Στο τελευταίο σενάριο, υλοποιήθηκε η συνεργασία ανθρώπου ρομπότ για την επίτευξη μιας εργασίας στον ίδιο χώρο. Η εργασία που αναπτύσσεται είναι το βίδωμα ενός περικόχλιου σε ένα κοχλία με την βοήθεια ενός κλειδιού. Στην ουσία, το κλειδί είναι τοποθετημένο πάνω σε ένα τραπέζι στον χώρο. Καθώς ο άνθρωπος πλησιάζει και φτάνει στην επιθυμητή θέση, όταν φέρει τα χέρια του στο σημείο που βρίσκεται ο κοχλίας το ρομπότ ξεκινά την συνεργασία. Η πρώτη κίνησή του είναι να πάρει το κλειδί που βρίσκεται πάνω στο τραπέζι με την βοήθεια της αρπάγης. Έπειτα, πηγαίνει με αυτό στην θέση που βρίσκεται το περικόχλιο και το βιδώνει περιστρέφοντας την έκτη άρθρωση. Στην συνέχεια, η αρπάγη ανοίγει, το ρομπότ απομακρύνεται προς τα πίσω χωρίς να προξενήσει κάποιο ατύχημα και επιστρέφει στην θέση ετοιμότητας (home).



Εικόνα 49: Διάγραμμα ροής συνεργασίας A-P.

Παρακάτω ακολουθεί το κύριο κομμάτι κώδικα που υλοποιεί το σενάριο.

```
def publish(self):
        global scale
        move_group = self.move_group
        scale = 1
        "Go to home"
```

```
joint_goal = move_group.get_current_joint_values()
joint_goal[0] = 0
joint_goal[1] = 0
joint_goal[2] = 0
joint_goal[3] = 0
joint_goal[4] = 0
joint_goal[5] = 0
joint_goal[6] = 0
move_group.go(joint_goal, wait=True)
move_group.stop()
```

Προσθήκη του τραπεζιού και του κλειδιού μέσα στον χώρο του rviz – καθορισμός θέσης, προσανατολισμού και μεγέθους.

```
"Insert boxes on scene"

box_name = self.box_name
scene = self.scene
eef_link = self.eef_link

scene.remove_attached_object(eef_link, name='box')
scene.remove_world_object('box')

box_pose = geometry_msgs.msg.PoseStamped()
box_pose.header.frame_id = "base_link"
box_pose.pose.orientation.w = 1.0
box_pose.pose.position.x = 0
box_pose.pose.position.y = 1
box_pose.pose.position.z = 0.5 # slightly above the end effector
box_name = "box"
scene.add_box(box_name, box_pose, size=(0.02, 0.02, 0.2))

 box_pose = geometry_msgs.msg.PoseStamped()
box_pose.header.frame_id = "base_link"
box_pose.pose.orientation.w = 1.0
box_pose.pose.position.x = 0
box_pose.pose.position.y = 1
box_pose.pose.position.z = 0.2
box_name = "table2"
scene.add_box(box_name, box_pose, size=(0.4, 0.2, 0.4))

while not rospy.is_shutdown():
```

Όταν ο άνθρωπος βρίσκεται στην θέση που πρέπει τότε η συνεργατική κίνηση ξεκινά και το ρομπότ παίρνει το κλειδί.

```
    if (actor_x < 1.7 and actor_y > 0.75 ):

        "Go to box, grab and attach to gripper"
        print("Grabbing object")

        joint_goal = move_group.get_current_joint_values()
        joint_goal[0] = 1.57
        joint_goal[1] = 0.5
        joint_goal[2] = 1.61
        joint_goal[3] = 0
        joint_goal[4] = 0.54
        joint_goal[5] = 0
        joint_goal[6] = 0
        move_group.go(joint_goal, wait=True)
        move_group.stop()
```

```
joint_goal = move_group.get_current_joint_values()
joint_goal[0] = 1.57
joint_goal[1] = 0.5
joint_goal[2] = 1.61
joint_goal[3] = 0
joint_goal[4] = 0.54
joint_goal[5] = 0
joint_goal[6] = 0.55
move_group.go(joint_goal, wait=True)
move_group.stop()

robot = self.robot
scene = self.scene
eef_link = self.eef_link
group_names = self.group_names
grasping_group = 'gripper'
touch_links = robot.get_link_names(group=grasping_group)
scene.attach_box(eef_link, 'box', touch_links=touch_links)
```

Καθορισμός των γωνιών των αρθρώσεων προκειμένου το ρομπότ να πάει στο επιθυμητό σημείο όπου υπάρχει το περικόχλιο και να περιστραφεί.

```
"Collaboration work"
print("Collaboration work")

joint_goal = move_group.get_current_joint_values()
joint_goal[0] = 0.43
joint_goal[1] = 0.95
joint_goal[2] = 0.96
joint_goal[3] = 0
joint_goal[4] = 0.34
joint_goal[5] = 0
joint_goal[6] = 0.55
move_group.go(joint_goal, wait=True)
move_group.stop()


"Gripper rotation"
joint_goal = move_group.get_current_joint_values()
joint_goal[0] = 0.43
joint_goal[1] = 0.95
joint_goal[2] = 0.96
joint_goal[3] = 0
joint_goal[4] = 0.34
joint_goal[5] = 4.71
joint_goal[6] = 0.55
move_group.go(joint_goal, wait=True)
move_group.stop()
```

Εφόσον η δουλειά έχει ολοκληρωθεί, το αντικείμενο(κλειδί) αφαιρείται από το περιβάλλον του rviz με τις παρακάτω εντολές του τμήματος: Unmount object.

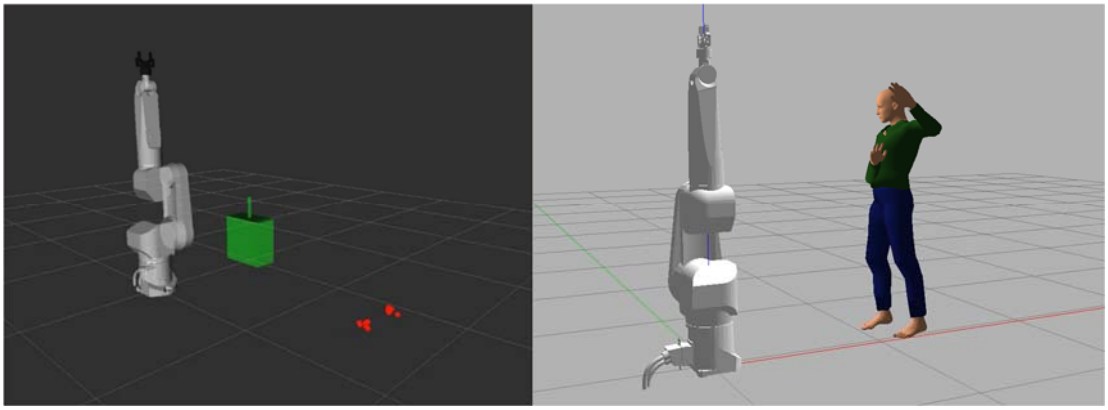
```
"Unmount object"
print("End of work")

scene.remove_attached_object(eef_link, name='box')
scene.remove_world_object('box')
rospy.sleep(0.5)
#print("flag")
joint_goal = move_group.get_current_joint_values()
joint_goal[0] = 0.43
joint_goal[1] = 0.95
```

```
joint_goal[2] = 0.96
joint_goal[3] = 0
joint_goal[4] = 0.34
joint_goal[5] = 4.71
joint_goal[6] = 0
move_group.go(joint_goal, wait=True)
move_group.stop()
rospy.sleep(0.5)
```

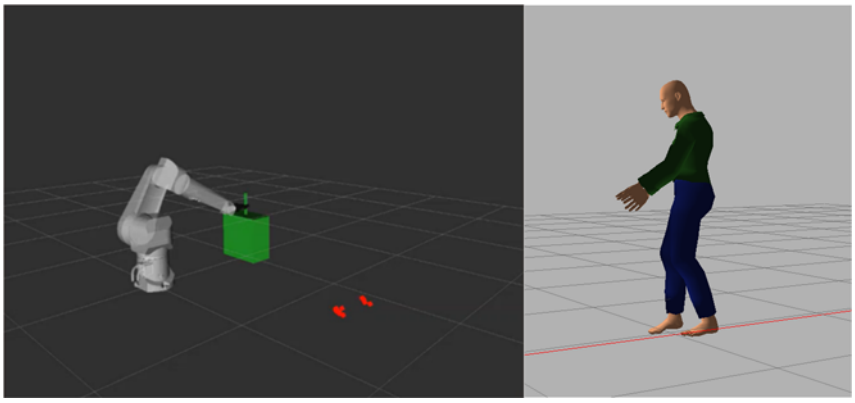
Κίνηση του ρομπότ προς τα αρνητικά στο καρτεσιανό σύστημα συντεταγμένων. Αυτό γίνεται για να μην υπάρξει σύγκρουση ή τραυματισμός του ανθρώπου που βρίσκεται στο περιβάλλον εργασίας. Τέλος, το ρομπότ μεταβαίνει σε θέση ετοιμότητας(θέση home).

```
"Move backwards cartesian"
print("Going to ready position")

thesi = move_group.get_current_pose().pose
waypoints = []
thesi.position.x -= 1 * 0.4  # Second move forward/backwards
in (x)
waypoints.append(copy.deepcopy(thesi))
thesi.position.y -= 1 * 0.4
waypoints.append(copy.deepcopy(thesi))

(plan, fraction) = move_group.compute_cartesian_path(
                                waypoints,   # waypoints to
follow
                                0.01,        # eef_step
                                0.0)         # jump_threshold

move_group.execute(plan, wait=True)

"Go home"

joint_goal = move_group.get_current_joint_values()
joint_goal[0] = 0
joint_goal[1] = 0
joint_goal[2] = 0
joint_goal[3] = 0
joint_goal[4] = 0
joint_goal[5] = 0
joint_goal[6] = 0
move_group.go(joint_goal, wait=True)
move_group.stop()

rospy.sleep(2)
```

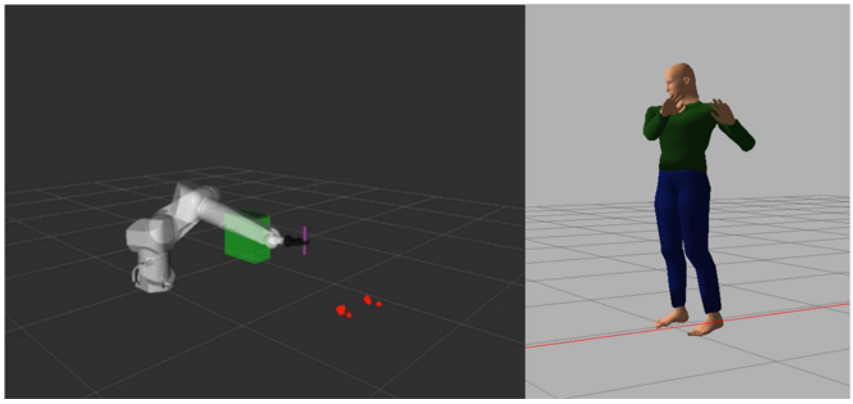
Τα στάδια της συνεργασίας φαίνονται στις παρακάτω εικόνες.



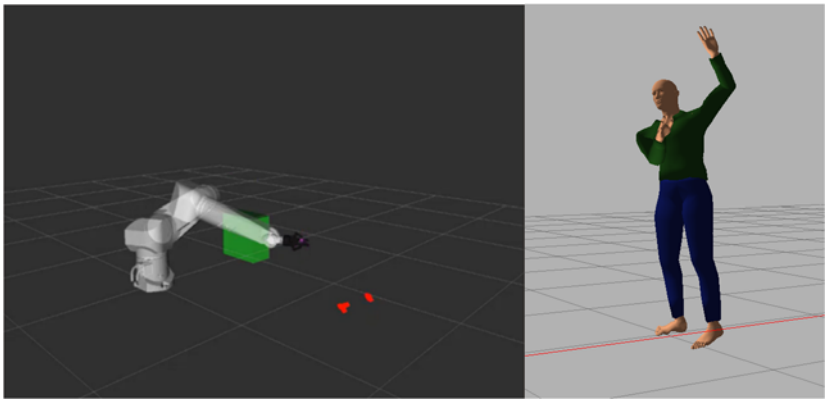
Εικόνα 50: Ο άνθρωπος βρίσκεται στην συγκεκριμένη περιοχή και με νεύμα του ξεκινά η κίνηση του ρομπότ.



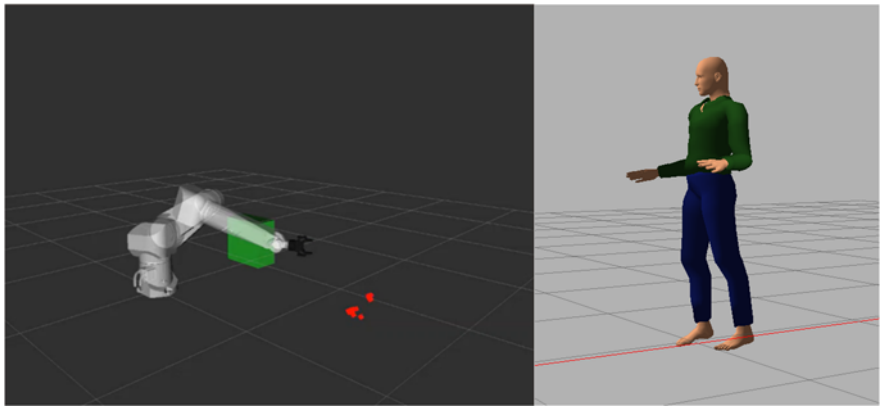
Εικόνα 51: Το ρομπότ παραλαμβάνει το κλειδί από το τραπέζι.



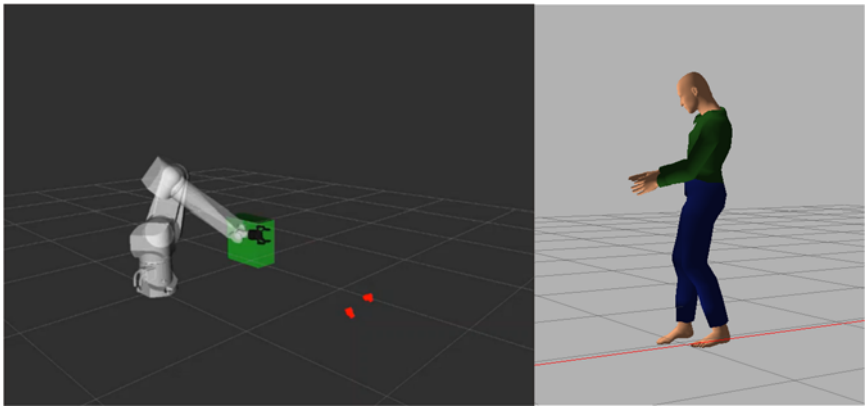
Εικόνα 52: Το ρομπότ παίρνει την σωστή θέση για την εκτέλεση της εργασίας.



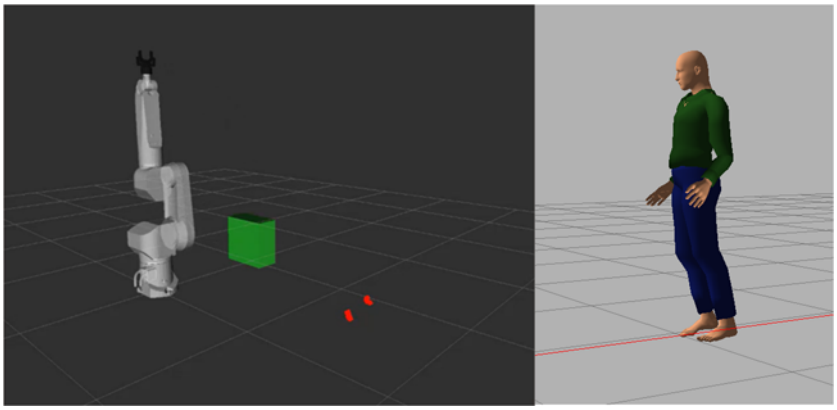
Εικόνα 53: Περιστροφή της έκτης άρθρωσης του ρομπότ για το βίδωμα του περικοχλίου.



Εικόνα 54: Αφαίρεση του κλειδιού από την σκηνή και άνοιγμα της αρπάγης.



Εικόνα 55: Κίνηση σε καρτεσιανό σύστημα συντεταγμένων προς τα αρνητικά για την αποφυγή σύγκρουσης με τον άνθρωπο.



Εικόνα 56: Μετάβαση του ρομπότ σε θέση ετοιμότητας (θέση home)..

5.8 Σχολιασμός των αποτελεσμάτων

5.8.1 Συνδυασμός ανεπτυγμένης λειτουργικότητας σε νέα σενάρια

Η ρομποτική πλατφόρμα του ROS προσφέρει μεγάλες δυνατότητες και ευελιξία στην ανάπτυξη νέων σεναρίων. Τα σενάρια που έχουν υλοποιηθεί στα πλαίσια αυτής της εργασίας, υποδεικνύουν το μεγάλο εύρος συσκευών που μπορούν να χρησιμοποιηθούν (πχ. αισθητήρες laser, κάμερες) καθώς και την αλληλεπίδρασή τους με στοιχεία του περιβάλλοντος όπως ο άνθρωπος στην περίπτωση της

συνεργασίας. Ο προγραμματισμός του ROS σε γλώσσες υψηλού επιπέδου όπως Python και C++, δίνει την δυνατότητα στον χρήστη να επέμβει μέχρι και στον πυρήνα του λογισμικού παραμετροποιώντας τις μεταβλητές που χρειάζεται με βάση τις ανάγκες του σεναρίου.

Οι παραλλαγές των σεναρίων που μπορούν να δημιουργηθούν είναι θεωρητικά άπειρες και βασίζονται στο ποια εργασία ζητά ο χρήστης να πραγματοποιείται. Για παράδειγμα, σε μια γραμμή παραγωγής αυτοκινήτων, τα εξαρτήματα που ο εργαζόμενος καλείται να συναρμολογήσει πολλές φορές έχουν μεγάλο βάρος αλλά και απαιτούν κινήσεις που κουράζουν τον εργαζόμενο (όπως ύψωση χεριών με βάρος για αρκετή ώρα). Σε μια τέτοια παραλλαγή, θα μπορούσε να γίνει συνδυασμός των σεναρίων που έχουν περιγραφεί σε αυτή την εργασία. Αρχικά, το ρομπότ κάνει λήψη ενός αντικειμένου που βρίσκεται πάνω σε ένα τραπέζι με υλοποίηση του σεναρίου λήψης και εναπόθεσης (pick and place). Το αντικείμενο προς συναρμολόγηση τοποθετείται στο κατάλληλο σημείο της γραμμής παραγωγής προκειμένου να συναρμολογηθεί από τον εργαζόμενο. Η αναγνώριση της θέσης που πρέπει να τοποθετηθεί, μπορεί να γίνεται με την ανάγνωση barcode που υπάρχει πάνω στο αντικείμενο ή με την βοήθεια της κάμερας και του machine learning. Οι γλώσσες προγραμματισμού που χρησιμοποιούνται επιτρέπουν στον χρήστη να αναπτύξει κατάλληλους αλγορίθμους προκειμένου να γίνουν κατανοητά από το ρομπότ διάφορες χειρονομίες του ανθρώπου ή, γενικότερα, κινήσεις του.

Ένα διαφορετικό σενάριο υλοποίησης συνεργασίας ανθρώπου-ρομπότ μπορεί να είναι η τοποθέτηση ανθρακοϋφασμάτων σε ένα καλούπι.[38]–[40] Η διαδικασία μπορεί να είναι η εξής. Η συνεργασία ξεκινά με την έλευση του ανθρώπου στον χώρο εργασίας και την κατάλληλη χειρονομία από τον χειριστή για την εκκίνηση της εργασίας. Το ρομπότ, μέσω κάμερας και με την επεξεργασία εικόνας μέσα από το περιβάλλον του ROS κατανοεί το συγκεκριμένο σήμα. Ο κύκλος εργασίας ξεκινά και η αρπάγη κάνει λήψη του πρώτου υφάσματος. Το τελικό στοιχείο δράσης παίρνει την κατάλληλη θέση-προσανατολισμό για την αφαίρεση του προστατευτικού film που υπάρχει στο ύφασμα. Με την βοήθεια αισθητήρων πίεσης και αφής αλλά και με επεξεργασία εικόνας, το ρομπότ καταλαβαίνει ότι το film έχει αφαιρεθεί. Ο τρόπος με τον οποίο μπορεί να γίνει αυτό, είναι η εκμάθηση του διαχωρισμού του υφάσματος με film ή χωρίς μέσω machine learning. Τέλος, ο βραχίονας τοποθετεί το ύφασμα στο καλούπι και η εργασία ολοκληρώνεται.

5.8.2 Μεταφορά από την προσομοίωση στο πραγματικό περιβάλλον

Όπως έχει αναφερθεί και στο 4ο κεφάλαιο, η προσπάθεια σύνδεσης του πραγματικού ρομπότ με το περιβάλλον του ROS δεν μπόρεσε να πραγματοποιηθεί. Σε έναν πιο σύγχρονο βραχίονα με διασύνδεση Ethernet, μπορεί να επιτευχθεί αμφίδρομη επικοινωνία και να εκτελεστούν τα παραπάνω σενάρια. Με το ROS-Industrial, οι περισσότερες εταιρίες κατασκευής ρομποτικών βραχιόνων όπως Kuka, Yaskawa, ABB αλλά και η Stäubli έχουν δημοσιεύσει στο Github πακέτα για την σύνδεση των ρομπότ τους. Όλα αυτά τα πακέτα, περιέχουν τα απαραίτητα αρχεία για

την διασύνδεση του βραχίονα με λογισμικά όπως το rviz και το gazebo για την εκτέλεση κινηματικού ελέγχου και προσομοίωσης.

Σε ένα πραγματικό περιβάλλον, τα σήματα που λαμβάνονται από αισθητήρες laser και κάμερες μεταδίδονται μέσω δικτύου στο περιβάλλον του ROS. Πολλοί επώνυμοι κατασκευαστές έχουν υλοποιήσει έτοιμα πακέτα για την διασύνδεση. Στην περίπτωση του σεναρίου σταματήματος του ρομπότ με την είσοδο του ανθρώπου στον χώρο εργασίας, ο αισθητήρας laser μεταδίδει πραγματικά δεδομένα από το περιβάλλον και το λογισμικό rviz και τα αποτυπώνει όπως και στην περίπτωση της προσομοίωσης. Ο άνθρωπος που χρησιμοποιήθηκε στο λογισμικό gazebo αντικαθίσταται από τον ‘συνεργάτη’ του βραχίονα ο οποίος εκτελεί είσοδο στον χώρο εργασίας. Πρακτικά, η συμπεριφορά του πραγματικού ρομπότ θα είναι ίδια με αυτή της προσομοίωσης, χωρίς να χρειαστεί επέμβαση στον αλγόριθμο που εκτελείται.

Στην περίπτωση της συνεργασίας ανθρώπου-ρομπότ, είναι σημαντικό οι μεταβλητές όπως η θέση του ανθρώπου καθώς και του εργαλείου να είναι ίδιες με αυτές στην προσομοίωση. Σε διαφορετική περίπτωση, μπορεί να γίνει αλλαγή της θέσης-προσανατολισμού αντικειμένου και ανθρώπου προκειμένου να προσαρμοστεί το σενάριο στο πραγματικό περιβάλλον. Η συμπεριφορά του βραχίονα θα ήταν και σε αυτή την περίπτωση παρόμοια καθώς ο τρόπος εκτέλεσης της κίνησης παραμένει ο ίδιος.

6 Συμπεράσματα

6.1 Κριτική επισκόπηση

Η παρούσα εργασία, ασχολείται με την συνεργασία ανθρώπου-ρομπότ στο περιβάλλον του ROS. Αυτό περιλαμβάνει την διεπαφή του χρήστη με το ρομπότ, την εκτέλεση εργασιών λήψης και τοποθέτησης καθώς και την συνεργατική κίνηση για την επίτευξη ενός σεναρίου.

Η ενσωμάτωση του ρομπότ Stäubli RX90L στο περιβάλλον του ROS, αποτέλεσε μεγάλη πρόκληση, καθώς χρειάστηκε να υλοποιηθούν μια σειρά από απαιτητικά βήματα για την σύνδεσή του και την δημιουργία του πακέτου.

Τα πρώτα σενάρια, όπως αυτό της λήψης και τοποθέτησης / εναπόθεσης (pick and place) καθώς και της εκτέλεσης τροχιάς με κινηματικό έλεγχο τριών αρθρώσεων, είναι περιπτώσεις όπου δεν υπεισέρχεται ο ανθρώπινος παράγοντας. Πραγματοποιήθηκαν κυρίως για να αναδείξουν τις δυνατότητες της πλατφόρμας του ROS σε επίπεδο προσομοίωσης. Όμως, ο χρήστης μπορεί να συνδέσει και ένα πραγματικό ρομπότ με αυτή την πλατφόρμα μειώνοντας τον χρόνο εκμάθησης και ελαχιστοποιώντας τα λάθη υλοποίησης. Στο σενάριο pick&place, το ρομπότ μεταφέρει ένα αντικείμενο από ένα τραπέζι σε ένα άλλο χρησιμοποιώντας την αρπάγη στο περιβάλλον rviz. Στο σενάριο του κινηματικού ελέγχου, στο οποίο έχουν θεωρηθεί ενεργές μόνο οι τρείς πρώτες αρθρώσεις, το ρομπότ εκτελεί ομαλή τροχιά με βάση τα σημεία που του έχουν δοθεί.

Στα υπόλοιπα σενάρια, υλοποιήθηκαν αλγόριθμοι, οι οποίοι είχαν επίκεντρο τον άνθρωπο. Κάθε χρόνο στις βιομηχανίες συμβαίνουν χιλιάδες ατυχήματα με ρομποτικούς βραχίονες. Η συνεργασία ανθρώπου-ρομπότ αποτελεί κρίσιμο αντικείμενο πλέον. Έγινε εκτέλεση διάφορων τεχνικών αποφυγής όπως η ενσωμάτωση αισθητήρα laser για την αναγνώριση του ανθρώπου στον χώρο εργασίας καθώς και η άντληση δεδομένων για την θέση και τον προσανατολισμό απευθείας από το λογισμικό Gazebo.

Η διακοπή κίνησης και η μείωση ταχύτητας όταν υπάρχει άνθρωπος στο περιβάλλον γίνεται με την βοήθεια του αισθητήρα laser για την αναγνώριση. Η εκτέλεση τροχιάς σε συγκεκριμένη θέση, γίνεται με τα δεδομένα θέσης του παρέχει το Gazebo στα topics που δημοσιεύει. Η απομάκρυνση του ρομπότ από τον άνθρωπο, πραγματοποιείται στο καρτεσιανό σύστημα συντεταγμένων προκειμένου να αποφευχθεί οποιαδήποτε σύγκρουση. Τέλος, η εκτέλεση συνεργασίας A-P, αποτελεί τον βασικό άξονα της παρούσας εργασίας και επιδεικνύει τις δυνατότητες της πλατφόρμας σε βιομηχανικό επίπεδο.

Η ροή εργασιών για την υλοποίηση των παραπάνω είχε ως εξής και θεωρείται ότι με τον ίδιο τρόπο μπορεί να αντιμετωπισθεί οποιοδήποτε σχετικό έργο. Πραγματοποιήθηκε η δημιουργία όλου του απαραίτητου πακέτου προκειμένου να

υπάρχει σύνδεση με το ROS. Με την βοήθεια του Solidworks, έγινε η συναρμολόγηση του βραχίονα με βάση τα διαγράμματα που έχει δώσει ο κατασκευαστής. Επίσης, ορίστηκαν οι γωνίες μεταξύ των αξόνων για τις περιστροφικές αρθρώσεις με βάση το εγχειρίδιο. Έπειτα, έγινε η εξαγωγή του απαραίτητου URDF μοντέλου και των αρχείων κώδικα για το ROS, με την βοήθεια ενός πρόσθετου (plugin) απευθείας μέσα από το Solidworks. Ακολούθησε η εγκατάσταση του περιβάλλοντος και όλου του απαραίτητου λογισμικού για την λειτουργία του. Με την βοήθεια του MoveIt Setup Assistant, έγινε η δημιουργία του πακέτου για την απεικόνιση στο rviz. Τέλος, έγινε ανάπτυξη των αλγορίθμων σε γλώσσα Python.

6.2 Μελλοντικές επεκτάσεις

Τα παραπάνω σενάρια που διερευνήθηκαν αναφέρονται σε τυπικά κομμάτια λειτουργικής συνεργασίας ανθρώπου-ρομπότ στο περιβάλλον του ROS. Γενικά, όπως είδαμε και στα προηγούμενα κεφάλαια, το ROS έχει την ικανότητα να επεξεργάζεται δεδομένα από διάφορους αισθητήρες ακόμη δε και στον τομέα της μηχανικής όρασης. Ακόμη, η μηχανική μάθηση (machine learning) μπορεί να έχει πολλαπλές εφαρμογές στο περιβάλλον του ROS. Η ανάπτυξη αλγορίθμων με γλώσσες προγραμματισμού (όπως Python, C++), δίνει την δυνατότητα ενσωμάτωσης εργαλείων που μπορούν να προσθέσουν πολυπλοκότητα αλλά και ευφυία σε ένα αρχικά απλό σενάριο.

Σε ένα βιομηχανικό περιβάλλον όπου ο άνθρωπος μπορεί να βρίσκεται σε διάφορα σημεία μέσα στον χώρο και να κάνει πολλές κινήσεις με τα χέρια του ή κρατώντας άλλα αντικείμενα, η αναγνώριση τους με τον αισθητήρα απόστασης laser δεν επαρκεί. Την θέση αυτού του αισθητήρα μπορεί να πάρει μια κάμερα. Χρησιμοποιώντας το Tensorflow για την δημιουργία ενός νευρωνικού δικτύου στο οποίο μπορεί να γίνει εκπαίδευση (train) για την αναγνώριση αντικειμένων και χειρονομιών του ανθρώπου η κάμερα «αρχίζει να μιλά» στην γλώσσα του ρομπότ. Με την πλατφόρμα OpenCV, είναι δυνατή η επεξεργασία εικόνας σε πραγματικό χρόνο (computer vision), και επιτρέπει στο ρομπότ να γνωρίζει τι ακριβώς συμβαίνει μπροστά του. Χρησιμοποιώντας αυτή την τεχνολογία, ο βραχίονας μπορεί να προσαρμοστεί θεωρητικά σε οποιοδήποτε περιβάλλον και να εκτελέσει διάφορα είδη συνεργασίας. Αυτό βέβαια απαιτεί την ανάπτυξη κατάλληλων νευρωνικών δικτύων για την επεξεργασία της πληροφορίες που έχει ως αποτέλεσμα την αυξημένη ανάγκη επεξεργαστικής ισχύος.

7 Βιβλιογραφία

[1] Wikipedia, “Robotics.” https://en.wikipedia.org/wiki/Robotics.

[2] Wikipedia, “What is robotics?” https://builtin.com/robotics.

[3] Britannica, “Robotics-Tecnhology.” https://www.britannica.com/technology/robotics.

[4] Thomasnet, “History of Robots and Robotics.” https://www.thomasnet.com/articles/automation-electronics/history-of-robotics/.

[5] I. F. of Robotics, “Timeline of robotics history.” https://ifr.org/robot-history.

[6] Kuka, “The history of KUKA.” https://www.kuka.com/en-us/about-kuka/history.

[7] A. Gasparetto and L. Scalera, “From the unimate to the delta robot: The early decades of industrial robotics,” in *History of Mechanism and Machine Science*, vol. 37, 2019.

[8] U. Bartlett School of Architecture, “Scara robot.” http://www.interactivearchitecture.org/performing-scara-robots.html.

[9] Wikipedia, “Industrial robots.” https://en.wikipedia.org/wiki/Industrial_robot.

[10] I. F. of Robotics, “Robots in industry.” https://ifr.org/industrial-robots.

[11] Abb, “IRB 360 FlexPicker®.” https://new.abb.com/products/robotics/industrial-robots/irb-360.

[12] Alex Owen-Hill, “What Are the Different Programming Methods for Robots?” https://blog.robotiq.com/what-are-the-different-programming-methods-for-robots.

[13] Comau, “Comau teach pendant.” comau.com.

[14] RoboDK, “RoboDK Offline Programming Software.” robodk.com.

[15] Interaction-design.org, “Human robot interaction.” https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/human-robot-interaction.

[16] A. Bicchi, M. A. Peshkin, and J. E. Colgate, “Safety for Physical Human–Robot Interaction,” *Springer Handb. Robot.*, pp. 1335–1348, 2008, doi: 10.1007/978-3-540-30301-5_58.

[17] L. Gualtieri, E. Rauch, R. Vidoni, and D. T. Matt, “Safety, Ergonomics and Efficiency in Human-Robot Collaborative Assembly: Design Guidelines and Requirements,” *Procedia CIRP*, vol. 91, pp. 367–372, 2020, doi: 10.1016/j.procir.2020.02.188.

[18] R. R. Galin and R. V. Meshcheryakov, “Human-Robot Interaction Efficiency and Human-Robot Collaboration,” *Stud. Syst. Decis. Control*, vol. 272, no.

January, pp. 55–63, 2020, doi: 10.1007/978-3-030-37841-7_5.

[19] P. Chemweno, L. Pintelon, and W. Decre, “Orienting safety assurance with outcomes of hazard analysis and risk assessment: A review of the ISO 15066 standard for collaborative robot systems,” *Saf. Sci.*, vol. 129, no. May, p. 104832, 2020, doi: 10.1016/j.ssci.2020.104832.

[20] U. Robots, “Universal Robots Lets BWIndustrie Improve Productivity and Keep Manufacturing in France.” https://www.universal-robots.com/case-stories/bw-industrie-ur16e/.

[21] V. Villani, F. Pini, F. Leali, and C. Secchi, “Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications,” *Mechatronics*, vol. 55, pp. 248–266, 2018, doi: https://doi.org/10.1016/j.mechatronics.2018.02.009.

[22] Adept Technology Inc, “V + Language,” no. September, pp. 432–0888, 1997.

[23] Adept Technology Inc, “V + Operating System User ’ s Guide,” 1997.

[24] Staubli, “Arm - RX series 90B family Instruction manual,” p. 104, 2008.

[25] Robotiq, “Robotiq 2F-85 & 2F-140 for e-Series Universal Robots InstructionManual Original,” pp. 1–145, 2018, [Online]. Available: https://assets.robotiq.com/website-assets/support_documents/document/2F-85_2F-140_Instruction_Manual_e-Series_PDF_20190206.pdf.

[26] Wikipedia, “Robot Operating System.” https://en.wikipedia.org/wiki/Robot_Operating_System.

[27] A. Ademovic, “An Introduction to Robot Operating System: The Ultimate Robot Application Framework.” https://www.toptal.com/robotics/introduction-to-robot-operating-system.

[28] V. Mazzari, “History of ROS.” https://blog.generationrobots.com/en/ros-robot-operating-system-2/.

[29] Adept Technology Inc, “Adept V+ ROS listener.” https://github.com/ros-industrial/swri-ros-pkg/blob/master/adept/adept_common/V%2B/ros.v2.

[30] ROS, “Solidworks to URDF Exporter.” http://wiki.ros.org/sw_urdf_exporter.

[31] S. Chitta, “MoveIt !: An Introduction,” pp. 3–27, doi: 10.1007/978-3-319-26054-9.

[32] The-construct-sim, “Ros-moveit,” [Online]. Available: https://www.theconstructsim.com/ros-movelt/.

[33] J. Sumon, “What is MoveIt! ROS? A Jump-Start guide to MoveIt!” https://medium.com/@jonathansumon/what-is-moveit-ros-a-jump-start-guide-to-moveit-873e0102d7e4.

[34] “Moveit - Official webpage.” https://moveit.ros.org/.

[35] Wikipedia, “Gazebo Simulator.” https://en.wikipedia.org/wiki/Gazebo_simulator.

[36] L. He, P. Glogowski, K. Lemmerz, B. Kuhlenkötter, and W. Zhang, “Method

to Integrate Human Simulation into Gazebo for Human-robot Collaboration," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 825, no. 1, 2020, doi: 10.1088/1757-899X/825/1/012006.

[37] ROS, "Ubuntu install of ROS Melodic." http://wiki.ros.org/melodic/Installation/Ubuntu.

[38] E. Matsas, G. C. Vosniakos, and D. Batras, "Modelling simple human-robot collaborative manufacturing tasks in interactive virtual environments," 2016, doi: 10.1145/2927929.2927948.

[39] E. Matsas, G. C. Vosniakos, and D. Batras, "Prototyping proactive and adaptive techniques for human-robot collaboration in manufacturing using virtual reality," *Robot. Comput. Integr. Manuf.*, vol. 50, 2018, doi: 10.1016/j.rcim.2017.09.005.

[40] E. Matsas, G. C. Vosniakos, and D. Batras, "Effectiveness and acceptability of a virtual environment for assessing human–robot collaboration in manufacturing," *Int. J. Adv. Manuf. Technol.*, vol. 92, no. 9–12, 2017, doi: 10.1007/s00170-017-0428-5.

8 Παράρτημα: Κώδικες προγραμματισμού

8.1 Κώδικας μετατροπής μηνυμάτων ROS- Adept

```
.PROGRAM a.ros()
;
; ABSTRACT:  Main entry point for the ROS - Stäubli/Adept serial interface.
;
; INPUTS:       None.
;
; OUTPUTS:      None.
;
; SIDE EFFECTS: Starts the serial command server and the serial feedback
server.
;
; DATA STRUCT:  None.
;
;
; Define globals

        delay.cmd = 1/10    ;Loop delay for joint commands
        delay.fdb = 1/10    ;Loop delay for joint feedback

; Start running the Serial protocols

        run = TRUE
        EXECUTE 5 ros.srv.cmd()
        EXECUTE 6 ros.srv.fdb()

; Return

        RETURN
.END
.PROGRAM ros.msg.create(msg.len, msg.type, cmd.type, msg.reply, msg.unused,
$msg)
;
; ABSTRACT:  Format a ROS message.
;
; INPUTS:       msg.len     The length of the message in bytes
;               msg.type    The message type
;               cmd.type    The command type
;               msg.reply   The message reply
;               msg.unused  The unused portion of the message (0)
;
; OUTPUTS:      $msg        The formatted message
;
; SIDE EFFECTS: None.
;
; DATA STRUCT:  None.
;
;
        AUTO $msg.len, $msg.type, $cmd.type, $msg.reply, $msg.unused

        ; Convert inputs to bytes

        $msg.len = $LNGB(msg.len)
        $msg.type = $LNGB(msg.type)
        $cmd.type = $LNGB(cmd.type)
        $msg.reply = $LNGB(msg.reply)
        $msg.unused = $LNGB(msg.unused)

        ; Reverse the bytes to match ROS

        CALL ros.rev.bytes($msg.len, $msg.len.rev)
```

```
        CALL ros.rev.bytes($msg.type, $msg.type.rev)
        CALL ros.rev.bytes($cmd.type, $cmd.type.rev)
        CALL ros.rev.bytes($msg.reply, $msg.reply.rev)
        CALL ros.rev.bytes($msg.unused, $msg.unused.rev)

        ; Format the message

        $msg = ""
        $msg =
$ENCODE($msg.len.rev,$msg.type.rev,$cmd.type.rev,$msg.reply.rev,$msg.unused.
rev)

        RETURN

.END
.PROGRAM ros.read.4bytes(slun, long)
;
; ABSTRACT:  Reads 4 bytes from the ROS Serial port and converts it to a V+
string.
;
; INPUTS:       slun     The Serial slun to read
;
; OUTPUTS:      $msg    The V+ string
;
; SIDE EFFECTS: Reverses the incoming byte order.
;
; DATA STRUCT:  None.
;

        AUTO $tmp, i, c[4]

        $tmp = ""
        FOR i = 1 TO 4
             c[i] = GETC(slun,2)
            $tmp = $tmp+$CHR(c[i])
        END
        CALL ros.rev.bytes($tmp, $msg)
        long = LNGB($msg)

.END
.PROGRAM ros.read.joints(slun, jts[])
;
; ABSTRACT: Reads 10 joint angles from ROS.
;
; INPUTS:       slun     The Serial slun to read
;
; OUTPUTS:      jts[10] The array of joint angles in degrees
;
; SIDE EFFECTS: Reverses the byte order
;
; DATA STRUCT:  None


        AUTO c[4], idx, $tmp, $msg.jts[10]

        FOR idx = 1 TO 10
            $tmp = ""
            FOR i = 1 TO 4
                c[i] = GETC(lun,2)
                $tmp = $tmp+$CHR(c[i])
            END
            CALL ros.rev.bytes($tmp, $msg.jts[idx])
            jts[idx] = FLTB($msg.jts[idx])
            jts[idx] = (jts[idx]*180)/PI
        END

.END
.PROGRAM ros.rev.bytes($in, $out)
```

```
;
; ABSTRACT: Reverse the bytes
;
; INPUTS:       $in     The input string
;
; OUTPUTS:      $out    The output string
;
; SIDE EFFECTS: Reverses the bytes
;
; DATA STRUCT:  None.


        AUTO str.len, i, $chr

        str.len = LEN($in)
        $chr = ""
        $out = ""

        FOR i = str.len TO 1 STEP -1
            $chr = $MID($in,i,1)
            $out = $out+$chr
        END

        RETURN
.END
.PROGRAM ros.srv.cmd()
;
; ABSTRACT:  Serial server for listening to joint commands from ROS.
;
; INPUTS:       None.
;
; OUTPUTS:      None.
;
; SIDE EFFECTS: Causes robot motion.
;
; DATA STRUCT:  None.

        AUTO slun, status, idx
        AUTO msg.len, msg.type, cmd.type, msg.reply, msg.unused,
$msg.jts[10], $tmp.str
        AUTO REAL jts[10]
        AUTO $msg

        ; Initialize autos

        msg.len = 0
        msg.type = 0
        cmd.type = 0
        msg.reply = 0

        ; Initialize joints

        FOR idx = 1 TO 10
            jts[idx] = 0
        END

        ; Open a serial port

        ATTACH (slun, 11) "SERIAL:1"
        status = IOSTAT(slun)
        IF (status < 0) THEN
            TYPE "Error opening cmd serial port: "+$ERROR(status)
            GOTO 100
        END

        ; Read 4 bytes from connection

        CALL ros.read.4bytes(slun, msg.unused)
```

```
        TYPE "ROS connected to V+ command server"

        ; Attach the robot and set speeds and accels for move

        SELECT ROBOT = 1
        ATTACH ()
        SPEED 20 ALWAYS
        ACCEL (1) 100, 100

        ; Server loop

        WHILE run DO

            ; Read data from ROS

            CALL ros.read.4bytes(slun, msg.len)     ;Message length
            CALL ros.read.4bytes(slun, msg.type)    ;Message type
            CALL ros.read.4bytes(slun, cmd.type)    ;Command type
            CALL ros.read.4bytes(slun, msg.reply)   ;Message reply
            CALL ros.read.4bytes(slun, msg.unused)  ;Discard 4 bytes
            CALL ros.read.joints(slun, jts[])       ;Joint values

            ; Check for STOP from ROS

            IF (msg.unused == -2) THEN
                GOTO 100
            END

            ; Type the joint values for debug

            ;TYPE
$ENCODE(jts[1],",",jts[2],",",jts[3],",",jts[4],",",jts[5],",",jts[6])

            ; Format a reply message to send to ROS

            CALL ros.msg.create(56, 10, 3, 1, 0, $msg)

            ; Append the joint position to the message

            SET #loc = #PPOINT(jts[1],jts[2],jts[3],jts[4],jts[5],jts[6])
            FOR idx = 1 TO 10
                jts[idx] = 0
                $msg = $msg+$msg.jts[idx]
            END

            ; Send the message to ROS

            WRITE (slun) $msg, /S

            ; Start the move

            MOVE #loc

            ; Loop delay

            ;WAIT.EVENT , delay.cmd

        END

        ; Close the Serialport

        DETACH (slun)

        ; Return

        CALL ros.stop()

    100 RETURN
```

```
.END
.PROGRAM ros.srv.fdb()
;
; ABSTRACT:  Serial server for sending joint feedback to ROS.
;
; INPUTS:       None.
;
; OUTPUTS:      None.
;
; SIDE EFFECTS: None.
;
; DATA STRUCT:  Server streams  <LENGTH(bytes), <10>, <1(COMM_TYPE -
TOPIC)>, <0 (REPLY_TYPE - N/A)>,
;                               <UNUSED <JOINT_DATA[10] (in rads (floats))>
;


        AUTO idx, slun, status
        AUTO REAL jts[10]
        AUTO $tmp.str, $msg.len, $msg.type, $cmd.type, $msg.reply,
$msg.jts[10], $msg.unused
        AUTO LOC #cur.loc
        AUTO $msg

        $tmp.str = ""

        ; Initialize joints

        FOR idx = 1 TO 10
            jts[idx] = 0
        END

        ; Open a Serial port

        ATTACH (slun, 4) "SERIAL:1"
        status = IOSTAT(slun)
        IF (status < 0) THEN
            TYPE "Error opening Serial port: "+$ERROR(status)
            GOTO 100
        END

        ; Server loop

        WHILE run DO

            ; Format a message to send to ROS

            CALL ros.msg.create(56, 10, 1, 0, 0, $msg)

            ; Get the current joint position

            HERE #cur.loc
            DECOMPOSE jts[1] = #cur.loc

            ; Format the joint data and append it to the message

            FOR idx = 1 TO 10
                jts[idx] = ((jts[idx]*PI)/180)
                $tmp.str = $FLTB(jts[idx])
                CALL ros.rev.bytes($tmp.str, $msg.jts[idx])
                $msg = $msg+$msg.jts[idx]
            END

            ; Send the message to ROS

            WRITE (slun) $msg, /S
```

```
            ; Loop delay
            WAIT.EVENT , delay.fdb
        END
        ; Close the Serial port
        DETACH (slun)
        ; Return
   100  RETURN
.END
.PROGRAM ros.stop()
;
; ABSTRACT:  Stops the V+ servers.
;
; INPUTS:       None.
;
; OUTPUTS:      None.
;
; SIDE EFFECTS: Stops ros.srv.cmd() and ros.srv.fdb()
;
; DATA STRUCT:  None.
;
        run = FALSE
.END
```

8.2 ROS – Gazebo

8.2.1 Stäubli_empty.world (διαμόρφωση περιβάλλοντος – Gazebo)

```
<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <include>
      <uri>model://sun</uri>
    </include>

    <physics type="ode" update_rate="100.0">
      <max_step_size>0.001</max_step_size>
      <real_time_factor>1</real_time_factor>
      <real_time_update_rate>1000</real_time_update_rate>
      <gravity>0 0 -9.81</gravity>
    </physics>


    <model name="hokuyo">
    <pose>0.2 0 0.025 0 0 0</pose>
    <link name="link">
      <gravity>false</gravity>
      <inertial>
        <mass>0.1</mass>
      </inertial>
      <visual name="visual">
        <geometry>
          <mesh>
```

```
          <uri>model://hokuyo/meshes/hokuyo.dae</uri>
        </mesh>
      </geometry>
    </visual>
    <sensor name="laser" type="gpu_ray">
      <pose>0.08 0 0.05 0 -0 0</pose>
      <ray>
        <scan>
          <horizontal>
            <samples>400</samples>
            <resolution>1</resolution>
            <min_angle>-2.3</min_angle>
            <max_angle>2.3</max_angle>
          </horizontal>
        </scan>
        <range>
          <min>0.08</min>
          <max>5</max>
          <resolution>0.01</resolution>
        </range>
        <noise>
          <type>gaussian</type>
          <mean>0.0</mean>
          <stddev>0.01</stddev>
        </noise>
      </ray>
      <plugin name="laser_back" filename="libgazebo_ros_gpu_laser.so">

          <robotNamespace>rx90l</robotNamespace>
          <topicName>laser/scan</topicName>
       </plugin>
      <always_on>1</always_on>
      <update_rate>30</update_rate>
      <visualize>true</visualize>
    </sensor>
  </link>


</model>

    <actor name="actor">
    <skin>
      <filename>stand.dae</filename>
      <scale>1</scale>
    </skin>
    <animation name="standing">
      <filename>stand.dae</filename>
      <interpolate_x>false</interpolate_x>
    </animation>
       <animation name="walking">
      <filename>talk_b.dae</filename>

    </animation>
    <script>

       <trajectory id="0" type="standing">

       <waypoint>
          <time>0</time>
          <pose>3.5 0 -0.2 0 0 -3.14</pose>
        </waypoint>
       <waypoint>
          <time>4</time>
          <pose>1.8 0 -0.2 0 0 -3.14</pose>
        </waypoint>
       <waypoint>
          <time>7</time>
          <pose>1.8 0 -0.2 0 0 -3.14</pose>
```

```
          </waypoint>
          <waypoint>
            <time>9</time>
            <pose>1.8 0.8 -0.2 0 0 -3.14</pose>
          </waypoint>

        </trajectory>

        <trajectory id="1" type="walking">

        <waypoint>
            <time>9</time>
            <pose>0.8 0.8 -0.2 0 0 -1.57</pose>
          </waypoint>
          <waypoint>
            <time>60</time>
            <pose>0.8 0.8 -0.2 0 0 -1.57</pose>
          </waypoint>

        </trajectory>
      </script>
    </actor>


  </world>
</sdf>
```

8.2.2 demo.launch (εκκίνηση rviz)

```
<?xml version="1.0"?>
<launch>

  <!-- specify the planning pipeline -->
  <arg name="pipeline" default="ompl" />

  <!-- By default, we do not start a database (it can be large) -->
  <arg name="db" default="false" />
  <!-- Allow user to specify database location -->
  <arg name="db_path" default="$(find rx90l)/default_warehouse_mongo_db" />

  <!-- By default, we are not in debug mode -->
  <arg name="debug" default="false" />

  <!--
  By default, hide joint_state_publisher's GUI

  MoveIt!'s "demo" mode replaces the real robot driver with the
joint_state_publisher.
  The latter one maintains and publishes the current joint configuration of
the simulated robot.
  It also provides a GUI to move the simulated robot around "manually".
  This corresponds to moving around the real robot without the use of
MoveIt.
  -->
  <arg name="use_gui" default="false" />

  <!-- Load the URDF, SRDF and other .yaml configuration files on the param
server -->
  <include file="$(find rx90l)/launch/planning_context.launch">
    <arg name="load_robot_description" value="true"/>
  </include>

  <!-- If needed, broadcast static tf for robot root -->


  <!-- We do not have a robot connected, so publish fake joint states -->
```

```
  <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher">
    <param name="use_gui" value="$(arg use_gui)"/>
    <rosparam
param="source_list">[move_group/fake_controller_joint_states]</rosparam>
  </node>

  <!-- Given the published joint states, publish tf for the robot links -->
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" respawn="true" output="screen" />

  <!-- Run the main MoveIt! executable without trajectory execution (we do
not have controllers configured by default) -->
  <include file="$(find rx90l)/launch/move_group.launch">
    <arg name="allow_trajectory_execution" value="true"/>
    <arg name="fake_execution" value="true"/>
    <arg name="info" value="true"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="pipeline" value="$(arg pipeline)"/>
  </include>

  <!-- Run Rviz and load the default config to see the state of the
move_group node -->
  <include file="$(find rx90l)/launch/moveit_rviz.launch">
    <arg name="rviz_config" value="$(find rx90l)/launch/moveit.rviz"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>

  <!-- If database loading was enabled, start mongodb as well -->
  <include file="$(find rx90l)/launch/default_warehouse_db.launch" if="$(arg
db)">
    <arg name="moveit_warehouse_database_path" value="$(arg db_path)"/>
  </include>

</launch>
```

8.2.3 control_gazebo.launch (εκκίνηση Gazebo)

```
<?xml version="1.0"?>
<launch>
  <arg name="paused" default="false"/>
  <arg name="gazebo_gui" default="true"/>
  <arg name="urdf_path"
default="/home/user/catkin_ws/src/Stäubli_ros/robot.urdf"/>

  <!-- startup simulated world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find
rx90l)/worlds/Stäubli_empty.world"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="gui" value="$(arg gazebo_gui)"/>
  </include>

  <!-- send robot urdf to param server -->
  <param name="robot_description" textfile="$(arg urdf_path)" />

  <!-- push robot_description to factory and spawn robot in gazebo at the
origin, change x,y,z arguments to spawn in a different position -->
  <node name="spawn_gazebo_model" pkg="gazebo_ros" type="spawn_model"
args="-urdf -param robot_description -model rx90l -x 0 -y 0 -z 0"
    respawn="false" output="screen" />

  <!-- load the corresponding controllers -->
  <include file="$(find
rx90l_controller)/launch/rx90l_position_control.launch">
  </include>
```

```
</launch>
```

8.2.4 rx90l_position_controllers.yaml (παράμετροι ελεγκτών)

```
rx90l:
  # Publish all joint states -----------------------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

  # Position Controllers ---------------------------------------
  joint1_position_controller:
    type: position_controllers/JointPositionController
    joint: joint1
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint2_position_controller:
    type: position_controllers/JointPositionController
    joint: joint2
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint3_position_controller:
    type: position_controllers/JointPositionController
    joint: joint3
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint4_position_controller:
    type: position_controllers/JointPositionController
    joint: joint4
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint5_position_controller:
    type: position_controllers/JointPositionController
    joint: joint5
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint6_position_controller:
    type: position_controllers/JointPositionController
    joint: joint6
    pid: {p: 100.0, i: 0.01, d: 10.0}
  gripper_controller:
    type: position_controllers/JointPositionController
    joint: finger_joint
    pid: {p: 20.0, i: 0.1, d: 0.0}
```

8.2.5 rx90l_position_control.launch (εκκίνηση ελεγκτών για PID έλεγχο αρθρώσεων)

```
<launch>

  <arg name="namespace" default="rx90l"/>

  <!-- Load joint controller configurations from YAML file to parameter
server -->
  <rosparam file="$(find
rx90l_controller)/config/rx90l_position_controllers.yaml" command="load"/>


  <!-- run_demo: load the position controllers -->
  <node name="position_controller_spawner"
    pkg="controller_manager"
    type="spawner"
    respawn="false"
    output="screen"
    ns="/$(arg namespace)"
    args="joint1_position_controller joint2_position_controller
```

```
          joint3_position_controller joint4_position_controller
          joint5_position_controller joint6_position_controller
gripper_controller joint_state_controller"/>



  <!-- convert joint states to TF transforms for rviz, etc -->
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher"
    respawn="false" output="screen">
    <remap from="/joint_states" to="/$(arg namespace)/joint_states" />
  </node>

</launch>
```

8.3 Σενάρια

8.3.1 Λήψη και εναπόθεση (pick and place)

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

import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from math import pi
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list


def all_close(goal, actual, tolerance):
  """
  Convenience method for testing if a list of values are within a tolerance
of their counterparts in another list
  @param: goal       A list of floats, a Pose or a PoseStamped
  @param: actual     A list of floats, a Pose or a PoseStamped
  @param: tolerance  A float
  @returns: bool
  """
  all_equal = True
  if type(goal) is list:
    for index in range(len(goal)):
      if abs(actual[index] - goal[index]) > tolerance:
        return False

  elif type(goal) is geometry_msgs.msg.PoseStamped:
    return all_close(goal.pose, actual.pose, tolerance)

  elif type(goal) is geometry_msgs.msg.Pose:
    return all_close(pose_to_list(goal), pose_to_list(actual), tolerance)

  return True


class MoveGroupPythonIntefaceTutorial(object):
  """MoveGroupPythonIntefaceTutorial"""
  def __init__(self):
    super(MoveGroupPythonIntefaceTutorial, self).__init__()

    moveit_commander.roscpp_initialize(sys.argv)
    rospy.init_node('moveit_python_rx90l', anonymous=True)
```

```
    robot = moveit_commander.RobotCommander()

    scene = moveit_commander.PlanningSceneInterface()

    group_name = "rx90l"
    move_group = moveit_commander.MoveGroupCommander(group_name)

    display_trajectory_publisher =
rospy.Publisher('/move_group/display_planned_path',

moveit_msgs.msg.DisplayTrajectory,
                                                    queue_size=20)

    planning_frame = move_group.get_planning_frame()
    print "============ Planning frame: %s" % planning_frame

    eef_link = move_group.get_end_effector_link()
    print "============ End effector link: %s" % eef_link

    group_names = robot.get_group_names()
    print "============ Available Planning Groups:", robot.get_group_names()

    print "============ Printing robot state"
    print robot.get_current_state()
    print ""

    # Misc variables
    self.box_name = ''
    self.robot = robot
    self.scene = scene
    self.move_group = move_group
    self.display_trajectory_publisher = display_trajectory_publisher
    self.planning_frame = planning_frame
    self.eef_link = eef_link
    self.group_names = group_names


  def go_to_home(self):
    move_group = self.move_group

    joint_goal = move_group.get_current_joint_values()
    joint_goal[0] = 0
    joint_goal[1] = 0
    joint_goal[2] = 0
    joint_goal[3] = 0
    joint_goal[4] = 0
    joint_goal[5] = 0
    joint_goal[6] = 0
    move_group.go(joint_goal, wait=True)
    move_group.stop()


    current_joints = move_group.get_current_joint_values()
    return all_close(joint_goal, current_joints, 0.01)



  def go_to_object(self):
    move_group = self.move_group

    joint_goal = move_group.get_current_joint_values()
    joint_goal[0] = 0
    joint_goal[1] = 0.5
    joint_goal[2] = 1.61
    joint_goal[3] = 0
    joint_goal[4] = 0.54
    joint_goal[5] = 0
    joint_goal[6] = 0
```

```
    move_group.go(joint_goal, wait=True)
    move_group.stop()

    joint_goal = move_group.get_current_joint_values()
    joint_goal[0] = 0
    joint_goal[1] = 0.5
    joint_goal[2] = 1.61
    joint_goal[3] = 0
    joint_goal[4] = 0.54
    joint_goal[5] = 0
    joint_goal[6] = 0.55
    move_group.go(joint_goal, wait=True)
    move_group.stop()

     current_joints = move_group.get_current_joint_values()
    return all_close(joint_goal, current_joints, 0.01)

  def go_to_2ndtable(self):
    move_group = self.move_group

    joint_goal = move_group.get_current_joint_values()
    joint_goal[0] = 1.55
    joint_goal[1] = 0.5
    joint_goal[2] = 1.61
    joint_goal[3] = 0
    joint_goal[4] = 0.54
    joint_goal[5] = 0
    joint_goal[6] = 0.55
    move_group.go(joint_goal, wait=True)
    move_group.stop()

     current_joints = move_group.get_current_joint_values()
    return all_close(joint_goal, current_joints, 0.01)

  def open_gripper(self):

    move_group = self.move_group
    joint_goal = move_group.get_current_joint_values()
    joint_goal[0] = 1.55
    joint_goal[1] = 0.5
    joint_goal[2] = 1.61
    joint_goal[3] = 0
    joint_goal[4] = 0.54
    joint_goal[5] = 0
    joint_goal[6] = 0
    move_group.go(joint_goal, wait=True)
    move_group.stop()


    current_joints = move_group.get_current_joint_values()
    return all_close(joint_goal, current_joints, 0.01)

  def go_to_pose_goal(self):
    move_group = self.move_group

    pose_goal = geometry_msgs.msg.Pose()
    pose_goal.orientation.w = 1.0
    pose_goal.position.x = 0.4
    pose_goal.position.y = 0.1
    pose_goal.position.z = 0.4

    move_group.set_pose_target(pose_goal)

    plan = move_group.go(wait=True)
    move_group.stop()
    move_group.clear_pose_targets()

    current_pose = self.move_group.get_current_pose().pose
```

```
    return all_close(pose_goal, current_pose, 0.01)


  def plan_cartesian_path(self, scale=1):
    move_group = self.move_group

    waypoints = []

    wpose = move_group.get_current_pose().pose
    wpose.position.z -= scale * 0.1  # First move up (z)
    wpose.position.y += scale * 0.2  # and sideways (y)
    waypoints.append(copy.deepcopy(wpose))

    wpose.position.x += scale * 0.1  # Second move forward/backwards in (x)
    waypoints.append(copy.deepcopy(wpose))

    wpose.position.y -= scale * 0.1  # Third move sideways (y)
    waypoints.append(copy.deepcopy(wpose))

    (plan, fraction) = move_group.compute_cartesian_path(
                                       waypoints,   # waypoints to follow
                                       0.01,        # eef_step
                                       0.0)         # jump_threshold

    return plan, fraction



  def display_trajectory(self, plan):
    robot = self.robot
    display_trajectory_publisher = self.display_trajectory_publisher

    display_trajectory = moveit_msgs.msg.DisplayTrajectory()
    display_trajectory.trajectory_start = robot.get_current_state()
    display_trajectory.trajectory.append(plan)
    # Publish
    display_trajectory_publisher.publish(display_trajectory);



  def execute_plan(self, plan):
    move_group = self.move_group

    move_group.execute(plan, wait=True)


  def wait_for_state_update(self, box_is_known=False, box_is_attached=False,
timeout=4):
    box_name = self.box_name
    scene = self.scene

    start = rospy.get_time()
    seconds = rospy.get_time()
    while (seconds - start < timeout) and not rospy.is_shutdown():
      attached_objects = scene.get_attached_objects([box_name])
      is_attached = len(attached_objects.keys()) > 0

      is_known = box_name in scene.get_known_object_names()

      if (box_is_attached == is_attached) and (box_is_known == is_known):
        return True

      rospy.sleep(0.1)
      seconds = rospy.get_time()

    return False

  def add_box(self, timeout=4):
```

```
    box_name = self.box_name
    scene = self.scene

    box_pose = geometry_msgs.msg.PoseStamped()
    box_pose.header.frame_id = "base_link"
    box_pose.pose.orientation.w = 1.0
    box_pose.pose.position.x = 1
    box_pose.pose.position.y = 0
    box_pose.pose.position.z = 0.5 # slightly above the end effector
    box_name = "box"
    scene.add_box(box_name, box_pose, size=(0.02, 0.02, 0.2))

    box_pose = geometry_msgs.msg.PoseStamped()
    box_pose.header.frame_id = "base_link"
    box_pose.pose.orientation.w = 1.0
    box_pose.pose.position.x = 1
    box_pose.pose.position.y = 0
    box_pose.pose.position.z = 0.2
    box_name = "table1"
    scene.add_box(box_name, box_pose, size=(0.2, 0.4, 0.4))

    box_pose = geometry_msgs.msg.PoseStamped()
    box_pose.header.frame_id = "base_link"
    box_pose.pose.orientation.w = 1.0
    box_pose.pose.position.x = 0
    box_pose.pose.position.y = 1
    box_pose.pose.position.z = 0.2
    box_name = "table2"
    scene.add_box(box_name, box_pose, size=(0.4, 0.2, 0.4))


    self.box_name="box"
    return self.wait_for_state_update(box_is_known=True, timeout=timeout)



  def attach_box(self, timeout=4):
    box_name = self.box_name
    robot = self.robot
    scene = self.scene
    eef_link = self.eef_link
    group_names = self.group_names

    grasping_group = 'gripper'
    touch_links = robot.get_link_names(group=grasping_group)
    scene.attach_box(eef_link, box_name, touch_links=touch_links)
    ## END_SUB_TUTORIAL

    # We wait for the planning scene to update.
    return self.wait_for_state_update(box_is_attached=True,
box_is_known=False, timeout=timeout)


  def detach_box(self, timeout=4):
    box_name = self.box_name
    scene = self.scene
    eef_link = self.eef_link

    scene.remove_attached_object(eef_link, name=box_name)

    return self.wait_for_state_update(box_is_known=True,
box_is_attached=False, timeout=timeout)


  def remove_box(self, timeout=4):
    box_name = self.box_name
    scene = self.scene
```

```
    scene.remove_world_object(box_name)


    return self.wait_for_state_update(box_is_attached=False,
box_is_known=False, timeout=timeout)


def main():
  try:
    rospy.sleep(10)
    print ""
    print "-----------------------------------------------------------"
    print "Stäubli RX90L pick and place"
    print "-----------------------------------------------------------"
    print "Press Ctrl-D to exit at any time"
    print ""
    print "============ Press `Enter` to begin the move ..."
    raw_input()
    tutorial = MoveGroupPythonIntefaceTutorial()

    print "============ Press `Enter` to add the tables and the object to
the planning scene ..."
    raw_input()
    tutorial.add_box()

    print "============ Press `Enter` to grab the object ..."
    raw_input()
    tutorial.go_to_object()
    tutorial.attach_box()

    print "============ Press `Enter` to move the object to the other table
..."
    raw_input()
    tutorial.go_to_2ndtable()
    tutorial.detach_box()
    tutorial.open_gripper()



    print "============ Press `Enter` to go to home position ..."
    raw_input()
    tutorial.go_to_home()

    print "============ Stäubli RX90L move complete!"
  except rospy.ROSInterruptException:
    return
  except KeyboardInterrupt:
    return

if __name__ == '__main__':
  main()
```

8.3.2 Διακοπή κίνησης όταν εισέρχεται στον χώρο εργασίας

```
#!/usr/bin/env python3

"""
Start ROS node to publish angles for the position control of the RX90L.
"""

# Ros handlers services and messages
import sys
import copy
import rospy, roslib
from sensor_msgs.msg import LaserScan
#import moveit_commander
```

```
import moveit_msgs.msg
import geometry_msgs.msg
from std_msgs.msg import String

from std_msgs.msg import Float64
from sensor_msgs.msg import JointState
#Math imports
from math import sin, cos, atan2, pi, sqrt
from numpy.linalg import inv, det, norm, pinv
import numpy as np
import time as t
from numpy import array, inf


class rx90l_controller():
    """Class to compute and publish joints positions"""
    def __init__(self,rate):

        # joints' angular positions
        self.joint_angpos = [0, 0, 0, 0, 0, 0, 0]
        # joints' states
        self.joint_states = JointState()
        # joints' transformation matrix wrt the robot's base frame
        #self.A01 = self.kinematics.tf_A01(self.joint_angpos)
        #self.A02 = self.kinematics.tf_A02(self.joint_angpos)
        #self.A03 = self.kinematics.tf_A03(self.joint_angpos)
        #self.A04 = self.kinematics.tf_A04(self.joint_angpos)
        #self.A05 = self.kinematics.tf_A05(self.joint_angpos)
        #self.A06 = self.kinematics.tf_A06(self.joint_angpos)
        #self.A07 = self.kinematics.tf_A07(self.joint_angpos)


        # ROS SETUP
        # initialize subscribers for reading encoders and publishers for
performing position control in the joint-space
        # Robot
        self.joint_states_sub = rospy.Subscriber('/rx90l/joint_states',
JointState, self.joint_states_callback, queue_size=1)
        self.joint1_pos_pub =
rospy.Publisher('/rx90l/joint1_position_controller/command', Float64,
queue_size=1)
        self.joint2_pos_pub =
rospy.Publisher('/rx90l/joint2_position_controller/command', Float64,
queue_size=1)
        self.joint3_pos_pub =
rospy.Publisher('/rx90l/joint3_position_controller/command', Float64,
queue_size=1)
        self.joint4_pos_pub =
rospy.Publisher('/rx90l/joint4_position_controller/command', Float64,
queue_size=1)
        self.joint5_pos_pub =
rospy.Publisher('/rx90l/joint5_position_controller/command', Float64,
queue_size=1)
        self.joint6_pos_pub =
rospy.Publisher('/rx90l/joint6_position_controller/command', Float64,
queue_size=1)
        self.joint7_pos_pub =
rospy.Publisher('/rx90l/gripper_controller/command', Float64, queue_size=1)
        self.laserscan_sub = rospy.Subscriber('/rx90l/laser/scan',
LaserScan, self.callback, queue_size=1)

        #Publishing rate
        self.period = 1.0/rate
        self.pub_rate = rospy.Rate(rate)

        self.publish()
```

```
    #SENSING CALLBACKS
    def joint_states_callback(self, msg):
        # ROS callback to get the joint_states

        self.joint_states = msg
        # (e.g. the angular position of joint 1 is stored in ::
self.joint_states.position[0])

    def callback(self, msg):

        self.laser = msg

    def publish(self):

        # set configuration
        self.joint_angpos = [0, 0.5, 1.8, 0, 0.5, 0, 0]
        tmp_rate = rospy.Rate(1)
        tmp_rate.sleep()
        self.joint1_pos_pub.publish(self.joint_angpos[0])
        tmp_rate.sleep()
        self.joint2_pos_pub.publish(self.joint_angpos[1])
        self.joint3_pos_pub.publish(self.joint_angpos[2])
        self.joint4_pos_pub.publish(self.joint_angpos[3])
        self.joint5_pos_pub.publish(self.joint_angpos[4])
        self.joint6_pos_pub.publish(self.joint_angpos[5])
        self.joint7_pos_pub.publish(self.joint_angpos[6])
        tmp_rate.sleep()
        print("The system is ready to execute your algorithm...")
        i = 0
        t = 0.01
        d = t
        akraia_thesi = False
        rostime_now = rospy.get_rostime()
        time_now = rostime_now.to_nsec()



        while not rospy.is_shutdown():
            laserlist = array(self.laser.ranges)  #ekxorisi timwn tou laser
ston pinaka laserlist
            laserlist[laserlist == inf] = 200
                                          #antikatastasi twn inf me enan
arithmo gia na ginei i sigkrisi
            if (np.any(laserlist < 3)) :      #oso den vlepei tipota to
laser

                d = 0.005


            else:

                d = 0.01

            #print(akraia_thesi)
            i = i + d
            self.joint_angpos[0] = i
            rospy.sleep(0.01)

            # Publish the new joint's angular positions
            self.joint1_pos_pub.publish(self.joint_angpos[0])
            self.joint2_pos_pub.publish(self.joint_angpos[1])
            self.joint3_pos_pub.publish(self.joint_angpos[2])
            self.joint4_pos_pub.publish(self.joint_angpos[3])
            self.joint5_pos_pub.publish(self.joint_angpos[4])
            self.joint6_pos_pub.publish(self.joint_angpos[5])
            self.joint7_pos_pub.publish(self.joint_angpos[6])
```

```
            self.pub_rate.sleep()


    def turn_off(self):
        pass

def controller_py():
    # Starts a new node
    rospy.init_node('controller_node', anonymous=True)
    controller = rx90l_controller(100)
    rospy.on_shutdown(controller.turn_off)
    rospy.spin()

if __name__ == '__main__':
    try:
        controller_py()
    except rospy.ROSInterruptException:
        pass
```

8.3.3 Εκτέλεση τροχιάς με κινηματικό έλεγχο

8.3.3.1 gazebo_kinematics.py

```
#!/usr/bin/env python3

# Ros handlers services and messages
import sys
import copy
import rospy, roslib
from sensor_msgs.msg import LaserScan
import moveit_msgs.msg
import geometry_msgs.msg
from std_msgs.msg import String

from std_msgs.msg import Float64
from sensor_msgs.msg import JointState
#Math imports
from math import sin, cos, atan2, pi, sqrt
from numpy.linalg import inv, det, norm, pinv
import numpy as np
import time as t
from numpy import array, inf

from kinematics import rx90l_kinematics


class rx90l_controller():
    """Class to compute and publish joints positions"""
    def __init__(self,rate):

        self.kinematics = rx90l_kinematics()

        # joints' angular positions
        self.joint_angpos = [0, 0, 0, 0, 0, 0, 0]
        # joints' states
        self.joint_states = JointState()
        # joints' transformation matrix wrt the robot's base frame
        #self.A01 = self.kinematics.tf_A01(self.joint_angpos)
        #self.A02 = self.kinematics.tf_A02(self.joint_angpos)
        #self.A03 = self.kinematics.tf_A03(self.joint_angpos)
        #self.A04 = self.kinematics.tf_A04(self.joint_angpos)
        #self.A05 = self.kinematics.tf_A05(self.joint_angpos)
        #self.A06 = self.kinematics.tf_A06(self.joint_angpos)
        #self.A07 = self.kinematics.tf_A07(self.joint_angpos)
```

```
        # ROS SETUP
        # initialize subscribers for reading encoders and publishers for
performing position control in the joint-space
        # Robot
        self.joint_states_sub = rospy.Subscriber('/rx90l/joint_states',
JointState, self.joint_states_callback, queue_size=1)
        self.joint1_pos_pub =
rospy.Publisher('/rx90l/joint1_position_controller/command', Float64,
queue_size=1)
        self.joint2_pos_pub =
rospy.Publisher('/rx90l/joint2_position_controller/command', Float64,
queue_size=1)
        self.joint3_pos_pub =
rospy.Publisher('/rx90l/joint3_position_controller/command', Float64,
queue_size=1)
        self.joint4_pos_pub =
rospy.Publisher('/rx90l/joint4_position_controller/command', Float64,
queue_size=1)
        self.joint5_pos_pub =
rospy.Publisher('/rx90l/joint5_position_controller/command', Float64,
queue_size=1)
        self.joint6_pos_pub =
rospy.Publisher('/rx90l/joint6_position_controller/command', Float64,
queue_size=1)
        self.joint7_pos_pub =
rospy.Publisher('/rx90l/gripper_controller/command', Float64, queue_size=1)
        self.laserscan_sub = rospy.Subscriber('/rx90l/laser/scan',
LaserScan, self.callback, queue_size=1)

        #Publishing rate
        self.period = 1.0/rate
        self.pub_rate = rospy.Rate(rate)

        self.publish()



    #SENSING CALLBACKS
    def joint_states_callback(self, msg):
        # ROS callback to get the joint_states

        self.joint_states = msg
        # (e.g. the angular position of joint 1 is stored in ::
self.joint_states.position[0])

    def callback(self, msg):

        self.laser = msg

    def publish(self):

        # set configuration
        self.joint_angpos = [0, 0, 0, 0, 0, 0, 0]
        tmp_rate = rospy.Rate(1)
        tmp_rate.sleep()
        self.joint1_pos_pub.publish(self.joint_angpos[0])
        tmp_rate.sleep()
        self.joint2_pos_pub.publish(self.joint_angpos[1])
        self.joint3_pos_pub.publish(self.joint_angpos[2])
        self.joint4_pos_pub.publish(self.joint_angpos[3])
        self.joint5_pos_pub.publish(self.joint_angpos[4])
        self.joint6_pos_pub.publish(self.joint_angpos[5])
        self.joint7_pos_pub.publish(self.joint_angpos[6])
        tmp_rate.sleep()
        print("The system is ready to execute your algorithm...")
        rostime_now = rospy.get_rostime()
        time_now = rostime_now.to_nsec()
```

```
        while not rospy.is_shutdown():
            laserlist = array(self.laser.ranges)  #ekxorisi timwn tou laser
ston pinaka laserlist
            laserlist[laserlist == inf] = 200       #antikatastasi twn inf
me enan arithmo gia na ginei i sigkrisi
            if(np.all(laserlist == 200)) :      #oso den vlepei tipota to
laser

                yd0 = 0.2
                yd1 = 0.9
                yd2 = 0.2
                xd0 = 0.8
                xd1 = 0.8
                zd0 = 0.2
                zd1 = 0.2
                Tf = 12

                dt = 0.01
                kmax = Tf/dt +1

                xd = np.empty(int(kmax))
                yd = np.empty(int(kmax))
                zd = np.empty(int(kmax))


                yd[0] = yd0
                xd[0] = xd0
                zd[0] = zd0
                lambda_x = (xd1-xd0)/Tf;
                lambda_z = (zd1-zd0)/Tf;

                a1 = (yd1 - yd0) / 8
                a2 = (yd2 - yd1) / 8

                for i in range(1,int(kmax),1):

                    tt = dt*i
                    if (tt <= 1):
                        yd[i] = (a1*tt**2 + yd0)
                    elif (tt > 1) and (tt <= 4):
                        yd[i] = (yd[int(1/dt)] + a1*2*(tt - 1))
                    elif (tt > 4) and (tt <= 5):
                        yd[i] = (yd[int(4/dt)] + a1*2*(tt - 4) - a1*(tt -
4)**2)
                    elif (tt > 5) and (tt <=6):
                        yd[i] = (yd[int(i-1)])
                    elif (tt > 6) and (tt <= 7):
                        yd[i] = (a2*(tt-6)**2 + yd1)
                    elif (tt > 7) and (tt <= 10):
                        yd[i] = (yd[int(7/dt)] + a2*2*(tt - 7))
                    elif (tt > 10) and (tt <= 11):
                        yd[i] = (yd[int(10/dt)] + a2*2*(tt - 10) - a2*(tt -
10)**2)
                    else:
                        yd[i] = (yd[int(i-1)])

                    xd[i] = xd[i-1] + lambda_x*dt;
                    zd[i] = zd[i-1] + lambda_z*dt;
                    print("y:",yd[i])



                    ee_p3 = [xd[i], yd[i], zd[i]]
```

```
                    joint_angles = self.kinematics.compute_angles(ee_p3)
                    self.joint_angpos[0] = joint_angles[0,0]
                    self.joint_angpos[1] = joint_angles[0,1]
                    self.joint_angpos[2] = joint_angles[0,2]

                    # Publish the new joint's angular positions
                    self.joint1_pos_pub.publish(self.joint_angpos[0])
                    self.joint2_pos_pub.publish(self.joint_angpos[1])
                    self.joint3_pos_pub.publish(self.joint_angpos[2])
                    self.joint4_pos_pub.publish(self.joint_angpos[3])
                    self.joint5_pos_pub.publish(self.joint_angpos[4])
                    self.joint6_pos_pub.publish(self.joint_angpos[5])
                    self.joint7_pos_pub.publish(self.joint_angpos[6])

                    self.pub_rate.sleep()


            else:
                print("empodio")


    def turn_off(self):
        pass

def controller_py():
    # Starts a new node
    rospy.init_node('controller_node', anonymous=True)
    controller = rx90l_controller(100)
    rospy.on_shutdown(controller.turn_off)
    rospy.spin()

if __name__ == '__main__':
    try:
        controller_py()
    except rospy.ROSInterruptException:
        pass
```

8.3.3.2 *kinematics.py*

```
#!/usr/bin/env python3

import numpy as np
import math

# Checks if a matrix is a valid rotation matrix.
def isRotationMatrix(R) :
    Rt = np.transpose(R)
    shouldBeIdentity = np.dot(Rt, R)
    I = np.identity(3, dtype = R.dtype)
    n = np.linalg.norm(I - shouldBeIdentity)
    return n < 1e-6

class rx90l_kinematics():
    def __init__(self):

        self.l1 = 0.420
        self.l2 = 0.450
        self.l3 = 0.650
        self.l4 = 0.085
        self.le = 0.100


        pass
```

```
    def compute_angles(self, ee_position):

        joint_1 = math.atan2(ee_position[1],ee_position[0])
        joint_3 =
math.acos((ee_position[0]**2+ee_position[1]**2+(ee_position[2]-self.l1)**2-
(self.l3+self.l4+self.le)**2-
self.l2**2)/(2*self.l2*(self.l3+self.l4+self.le)))
        joint_2 =
math.atan2((ee_position[1]*math.cos(joint_3)*(self.l3+self.l4+self.le)+ee_po
sition[1]*self.l2-(ee_position[1]-
self.l1)*(self.l3+self.l4+self.le)*math.sin(joint_3)*math.sin(joint_1)),(mat
h.sin(joint_1)*(ee_position[1]-
self.l1)*(math.cos(joint_3)*(self.l3+self.l4+self.le)+self.l2)+ee_position[1
]*math.sin(joint_3)*(self.l3+self.l4+self.le)))
        joint_4 = 0
        joint_6 = 0
        joint_5 = 0


        joint_angles = np.matrix([ joint_1, joint_2, joint_3, joint_4,
joint_5, joint_6 ])

        return joint_angles


    def tf_A01(self, r_joints_array):
        tf = np.matrix([[math.cos(r_joints_array[0]) , 0 , -
math.sin(r_joints_array[0]) , 0],\
                        [math.sin(r_joints_array[0]) , 0 ,
math.cos(r_joints_array[0]) , 0],\
                        [0 , -1 , 0 , self.l1],\
                        [0 , 0 , 0 , 1]])
        return tf

    def tf_A02(self, r_joints_array):
        tf_A12 = np.matrix([[math.sin(r_joints_array[1]) ,
math.cos(r_joints_array[1]) , 0 , self.l2*math.sin(r_joints_array[1])],\
                            [-math.cos(r_joints_array[1]) ,
math.sin(r_joints_array[1]) , 0 , -self.l2*math.sin(r_joints_array[1])],\
                            [0 , 0 , 1 , 0],\
                            [0 , 0 , 0 , 1]])
        tf = np.dot( self.tf_A01(r_joints_array), tf_A12 )
        return tf

    def tf_A03(self, r_joints_array):
        tf_A23 = np.matrix([[-math.sin(r_joints_array[2]) , 0 ,
math.cos(r_joints_array[2]) , 0],\
                            [math.cos(r_joints_array[2]) , 0 ,
math.sin(r_joints_array[2]) , 0],\
                            [0 , 1 , 0 , 0],\
                            [0 , 0 , 0 , 1]])
        tf = np.dot( self.tf_A02(r_joints_array), tf_A23 )
        return tf

    def tf_A04(self, r_joints_array):
        tf_A34 = np.matrix([[math.cos(r_joints_array[3]) , 0 , -
math.sin(r_joints_array[3]) , 0],\
                            [math.sin(r_joints_array[3]) , 0 ,
math.cos(r_joints_array[3]) , 0],\
                            [0 , -1 , 0 , self.l3],\
                            [0 , 0 , 0 , 1]])
        tf = np.dot( self.tf_A03(r_joints_array), tf_A34 )
        return tf

    def tf_A05(self, r_joints_array):
        tf_A45 = np.matrix([[math.cos(r_joints_array[4]) , 0 ,
math.sin(r_joints_array[4]) , 0],\
```

```
                            [math.sin(r_joints_array[4]) , 0 , -
math.cos(r_joints_array[4]) , 0],\
                            [0 , 1 , 0 , 0],\
                            [0 , 0 , 0 , 1]])
        tf = np.dot( self.tf_A04(r_joints_array), tf_A45 )
        return tf

    def tf_A06(self, r_joints_array):
        tf_A56 = np.matrix([[math.cos(r_joints_array[5]) , -
math.sin(r_joints_array[5]) , 0 , 0],\
                            [math.sin(r_joints_array[5]) ,
math.cos(r_joints_array[5]) , 0 , 0],\
                            [0 , 0 , 1 , self.l4+self.le],\
                            [0 , 0 , 0 , 1]])
        tf = np.dot( self.tf_A05(r_joints_array), tf_A56 )
        return tf



    # Calculates rotation matrix to euler angles
    # The result is the same as MATLAB except the order
    # of the euler angles ( x and z are swapped ).
    def rotationMatrixToEulerAngles(self, R) :

        assert(isRotationMatrix(R))

        sy = math.sqrt(R[0,0] * R[0,0] +  R[1,0] * R[1,0])

        singular = sy < 1e-6

        if  not singular :
            x = math.atan2(R[2,1] , R[2,2])
            y = math.atan2(-R[2,0], sy)
            z = math.atan2(R[1,0], R[0,0])
        else :
            x = math.atan2(-R[1,2], R[1,1])
            y = math.atan2(-R[2,0], sy)
            z = 0

        return np.array([x, y, z])
```

8.3.4 Μείωση ταχύτητας ρομπότ

```
#!/usr/bin/env python3

# Ros handlers services and messages
import sys
import copy
import rospy, roslib
from sensor_msgs.msg import LaserScan
import moveit_msgs.msg
import geometry_msgs.msg
from std_msgs.msg import String

from std_msgs.msg import Float64
from sensor_msgs.msg import JointState
#Math imports
from math import sin, cos, atan2, pi, sqrt
from numpy.linalg import inv, det, norm, pinv
import numpy as np
import time as t
from numpy import array, inf


class rx90l_controller():
    """Class to compute and publish joints positions"""
    def __init__(self,rate):
```

```
        # joints' angular positions
        self.joint_angpos = [0, 0, 0, 0, 0, 0, 0]
        self.joint_states = JointState()
        self.joint_states_sub = rospy.Subscriber('/rx90l/joint_states',
JointState, self.joint_states_callback, queue_size=1)
        self.joint1_pos_pub =
rospy.Publisher('/rx90l/joint1_position_controller/command', Float64,
queue_size=1)
        self.joint2_pos_pub =
rospy.Publisher('/rx90l/joint2_position_controller/command', Float64,
queue_size=1)
        self.joint3_pos_pub =
rospy.Publisher('/rx90l/joint3_position_controller/command', Float64,
queue_size=1)
        self.joint4_pos_pub =
rospy.Publisher('/rx90l/joint4_position_controller/command', Float64,
queue_size=1)
        self.joint5_pos_pub =
rospy.Publisher('/rx90l/joint5_position_controller/command', Float64,
queue_size=1)
        self.joint6_pos_pub =
rospy.Publisher('/rx90l/joint6_position_controller/command', Float64,
queue_size=1)
        self.joint7_pos_pub =
rospy.Publisher('/rx90l/gripper_controller/command', Float64, queue_size=1)
        self.laserscan_sub = rospy.Subscriber('/rx90l/laser/scan',
LaserScan, self.callback, queue_size=1)

        #Publishing rate
        self.period = 1.0/rate
        self.pub_rate = rospy.Rate(rate)

        self.publish()



    #SENSING CALLBACKS
    def joint_states_callback(self, msg):
        # ROS callback to get the joint_states

        self.joint_states = msg
        # (e.g. the angular position of joint 1 is stored in ::
self.joint_states.position[0])

    def callback(self, msg):

        self.laser = msg

    def publish(self):

        # set configuration
        self.joint_angpos = [0, 0.5, 1.8, 0, 0.5, 0, 0]
        tmp_rate = rospy.Rate(1)
        tmp_rate.sleep()
        self.joint1_pos_pub.publish(self.joint_angpos[0])
        tmp_rate.sleep()
        self.joint2_pos_pub.publish(self.joint_angpos[1])
        self.joint3_pos_pub.publish(self.joint_angpos[2])
        self.joint4_pos_pub.publish(self.joint_angpos[3])
        self.joint5_pos_pub.publish(self.joint_angpos[4])
        self.joint6_pos_pub.publish(self.joint_angpos[5])
        self.joint7_pos_pub.publish(self.joint_angpos[6])
        tmp_rate.sleep()
        print("The system is ready to execute your algorithm...")
        i = 0
        t = 0.01
```

```
        d = t
        akraia_thesi = False
        rostime_now = rospy.get_rostime()
        time_now = rostime_now.to_nsec()



        while not rospy.is_shutdown():
            laserlist = array(self.laser.ranges)  #ekxorisi timwn tou laser
ston pinaka laserlist
            laserlist[laserlist == inf] = 200
                                          #antikatastasi twn inf me enan
arithmo gia na ginei i sigkrisi
            if (i >= 1.4):
                akraia_thesi = 1
                if (np.any(laserlist < 1.7)) :
                    d = 0
                elif (np.any(laserlist >= 1.7) and np.any(laserlist < 3)):
                    d = -0.005
                else:
                    d = -0.02

            elif (i <= -1.4):
                akraia_thesi = 2
                if (np.any(laserlist < 1.7)) :
                    d = 0
                elif (np.any(laserlist >= 1.7) and np.any(laserlist < 3)):
                    d = 0.005
                else:
                    d = 0.02

            else:
                #d = temp

                if (np.any(laserlist < 1.7)) :      #oso den vlepei tipota
to laser
                    if (akraia_thesi == 1):
                        d = 0
                    elif (akraia_thesi == 2):
                        d = 0

                elif (np.any(laserlist >= 1.7) and np.any(laserlist < 3)):
                    if (akraia_thesi == 1):
                        d = -0.005
                    elif (akraia_thesi == 2):
                        d = 0.005

                else:
                    if (akraia_thesi == 1):
                        d = -0.02
                    elif (akraia_thesi == 2):
                        d = 0.02



            i = i + d
            self.joint_angpos[0] = i
            rospy.sleep(0.01)

            # Publish the new joint's angular positions
            self.joint1_pos_pub.publish(self.joint_angpos[0])
            self.joint2_pos_pub.publish(self.joint_angpos[1])
            self.joint3_pos_pub.publish(self.joint_angpos[2])
            self.joint4_pos_pub.publish(self.joint_angpos[3])
            self.joint5_pos_pub.publish(self.joint_angpos[4])
            self.joint6_pos_pub.publish(self.joint_angpos[5])
            self.joint7_pos_pub.publish(self.joint_angpos[6])
```

```
            self.pub_rate.sleep()



    def turn_off(self):
        pass

def controller_py():
    # Starts a new node
    rospy.init_node('controller_node', anonymous=True)
    controller = rx90l_controller(100)
    rospy.on_shutdown(controller.turn_off)
    rospy.spin()

if __name__ == '__main__':
    try:
        controller_py()
    except rospy.ROSInterruptException:
        pass
```

8.3.5 Εκτέλεση τροχιάς όταν βρίσκεται σε συγκεκριμένη θέση

```
#!/usr/bin/env python3

# Ros handlers services and messages
import sys
import copy
import rospy, roslib
from sensor_msgs.msg import LaserScan
from gazebo_msgs.msg import LinkStates
import moveit_msgs.msg
import geometry_msgs.msg
from std_msgs.msg import String

from std_msgs.msg import Float64
from sensor_msgs.msg import JointState
#Math imports
from math import sin, cos, atan2, pi, sqrt
from numpy.linalg import inv, det, norm, pinv
import numpy as np
import time as t
from numpy import array, inf

from kinematics import rx90l_kinematics

class rx90l_controller():
    """Class to compute and publish joints positions"""
    def __init__(self,rate):

        # Init xArm7 kinematics handler
        self.kinematics = rx90l_kinematics()

        # joints' angular positions
        self.joint_angpos = [0, 0, 0, 0, 0, 0, 0]
        # joints' states
        self.joint_states = JointState()
        self.joint_states_sub = rospy.Subscriber('/rx90l/joint_states',
JointState, self.joint_states_callback, queue_size=1)
        self.joint1_pos_pub =
rospy.Publisher('/rx90l/joint1_position_controller/command', Float64,
queue_size=1)
        self.joint2_pos_pub =
rospy.Publisher('/rx90l/joint2_position_controller/command', Float64,
queue_size=1)
```

```
        self.joint3_pos_pub =
rospy.Publisher('/rx90l/joint3_position_controller/command', Float64,
queue_size=1)
        self.joint4_pos_pub =
rospy.Publisher('/rx90l/joint4_position_controller/command', Float64,
queue_size=1)
        self.joint5_pos_pub =
rospy.Publisher('/rx90l/joint5_position_controller/command', Float64,
queue_size=1)
        self.joint6_pos_pub =
rospy.Publisher('/rx90l/joint6_position_controller/command', Float64,
queue_size=1)
        self.joint7_pos_pub =
rospy.Publisher('/rx90l/gripper_controller/command', Float64, queue_size=1)
        self.laserscan_sub = rospy.Subscriber('/rx90l/laser/scan',
LaserScan, self.callback, queue_size=1)
        self.link_states_sub = rospy.Subscriber('/gazebo/link_states',
LinkStates, self.link_callback, queue_size=1)

        #Publishing rate
        self.period = 1.0/rate
        self.pub_rate = rospy.Rate(rate)

        self.publish()



    #SENSING CALLBACKS
    def joint_states_callback(self, msg):
        # ROS callback to get the joint_states

        self.joint_states = msg
        # (e.g. the angular position of joint 1 is stored in ::
self.joint_states.position[0])

    def callback(self, msg):

        self.laser = msg


    def link_callback(self, msg):

        self.link = msg


    def publish(self):

        # set configuration
        self.joint_angpos = [0, 0.7, 0.7, 0, 0, 0, 0]
        tmp_rate = rospy.Rate(1)
        tmp_rate.sleep()
        self.joint1_pos_pub.publish(self.joint_angpos[0])
        tmp_rate.sleep()
        self.joint2_pos_pub.publish(self.joint_angpos[1])
        self.joint3_pos_pub.publish(self.joint_angpos[2])
        self.joint4_pos_pub.publish(self.joint_angpos[3])
        self.joint5_pos_pub.publish(self.joint_angpos[4])
        self.joint6_pos_pub.publish(self.joint_angpos[5])
        self.joint7_pos_pub.publish(self.joint_angpos[6])
        tmp_rate.sleep()
        print("The system is ready to execute your algorithm...")
        i = 0
        p = 0
        t = 0.003
        d = t
        s = 0.002
        r = s
```

```
        rostime_now = rospy.get_rostime()
        time_now = rostime_now.to_nsec()



        while not rospy.is_shutdown():
            robot_x = self.link.pose[39].position.x
            robot_y = self.link.pose[39].position.y
            robot_z = self.link.pose[39].position.z

            actor_x = self.link.pose[1].position.x
            actor_y = self.link.pose[1].position.y
            actor_z = self.link.pose[1].position.z

            diff_x = actor_x - robot_x
            diff_y = robot_y - actor_y



            if (diff_x <= 1) :      #oso den vlepei tipota to laser

                i = i + d
                p = p + r
                self.joint_angpos[0] = i
                #self.joint_angpos[1] = i/2 + 0.2
                self.joint_angpos[2] = p
                rospy.sleep(0.01)


                if (i >= 0.6):
                    d = -t

                if (i <= -0.6):
                    d = t

                if (p >= 1.6):
                    r = -s

                if (p <= 0.5):
                    r = s

                    # Publish the new joint's angular positions
                self.joint1_pos_pub.publish(self.joint_angpos[0])
                self.joint2_pos_pub.publish(self.joint_angpos[1])
                self.joint3_pos_pub.publish(self.joint_angpos[2])
                self.joint4_pos_pub.publish(self.joint_angpos[3])
                self.joint5_pos_pub.publish(self.joint_angpos[4])
                self.joint6_pos_pub.publish(self.joint_angpos[5])
                self.joint7_pos_pub.publish(self.joint_angpos[6])

                print("Collaboration started - Distance(x):", diff_x)

                self.pub_rate.sleep()


            else:
                print("No human detected into workspace - Distance(x):",
diff_x)



    def turn_off(self):
        pass

def controller_py():
    # Starts a new node
    rospy.init_node('controller_node', anonymous=True)
    controller = rx90l_controller(100)
```

```
    rospy.on_shutdown(controller.turn_off)
    rospy.spin()

if __name__ == '__main__':
    try:
        controller_py()
    except rospy.ROSInterruptException:
        pass
```

8.3.6 Απομάκρυνση του ρομπότ από τον άνθρωπο

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

import sys
import copy
import rospy, roslib
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from sensor_msgs.msg import LaserScan
from gazebo_msgs.msg import LinkStates
from math import pi
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list
from std_msgs.msg import Float64
import numpy as np
from numpy import array, inf
import time as t

class rx90l_controller():

    def __init__(self):

        moveit_commander.roscpp_initialize(sys.argv)

        robot = moveit_commander.RobotCommander()

        scene = moveit_commander.PlanningSceneInterface()

        self.laserscan_sub = rospy.Subscriber('/rx90l/laser/scan',
LaserScan, self.callback, queue_size=1)
        self.link_states_sub = rospy.Subscriber('/gazebo/link_states',
LinkStates, self.link_callback, queue_size=1)

        group_name = "rx90l"
        move_group = moveit_commander.MoveGroupCommander(group_name)

        display_trajectory_publisher =
rospy.Publisher('/move_group/display_planned_path',

moveit_msgs.msg.DisplayTrajectory,
                                                    queue_size=20)


        # Misc variables
        self.box_name = ''
        self.robot = robot
        self.scene = scene
        self.move_group = move_group
        self.display_trajectory_publisher = display_trajectory_publisher
        self.publish()


    def callback(self, msg):
```

```
        global laserlist
        self.laser = msg
        laserlist = array(self.laser.ranges)  #ekxorisi timwn tou laser ston
pinaka laserlist
        laserlist[laserlist == inf] = 200

    def link_callback(self, msg):

        global actor_x,actor_y,actor_z

        self.link = msg

        actor_x = self.link.pose[1].position.x
        actor_y = self.link.pose[1].position.y
        actor_z = self.link.pose[1].position.z


    def publish(self):

        global scale

        move_group = self.move_group

        scale = 1



        "Go to home"

        joint_goal = move_group.get_current_joint_values()
        joint_goal[0] = 0
        joint_goal[1] = 1.34
        joint_goal[2] = 0.41
        joint_goal[3] = 0
        joint_goal[4] = 0.17
        joint_goal[5] = 0
        joint_goal[6] = 0
        move_group.go(joint_goal, wait=True)
        move_group.stop()

        "Avoidance"

        while not rospy.is_shutdown():

            thesi = move_group.get_current_pose().pose

            robot_x = thesi.position.x
            robot_y = thesi.position.y
            robot_z = thesi.position.z


            diff_x = actor_x - robot_x
            diff_y = robot_y - actor_y

            if (diff_x < 3.5 and diff_x >1.2):


                waypoints = []
                thesi.position.x -= scale * 0.06  # Second move
forward/backwards in (x)
                waypoints.append(copy.deepcopy(thesi))

                (plan, fraction) = move_group.compute_cartesian_path(
                                        waypoints,   # waypoints to
follow
                                        0.01,        # eef_step
                                        0.0)         # jump_threshold
```

```
                move_group.execute(plan, wait=True)


    def turn_off(self):
        pass


def controller_py():
    # Starts a new node
    rospy.init_node('controller_node', anonymous=True)
    controller = rx90l_controller()
    rospy.on_shutdown(controller.turn_off)
    rospy.spin()

if __name__ == '__main__':
    try:
        controller_py()
    except rospy.ROSInterruptException:
        pass
```

8.3.7 Εκτέλεση συνεργατικού καθήκοντος ανθρώπου-ρομπότ

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

import sys
import copy
import rospy, roslib
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from sensor_msgs.msg import LaserScan
from gazebo_msgs.msg import LinkStates
from math import pi
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list
from std_msgs.msg import Float64
import numpy as np
from numpy import array, inf
import time as t

class rx90l_controller():

    def __init__(self):
        moveit_commander.roscpp_initialize(sys.argv)
        robot = moveit_commander.RobotCommander()
        scene = moveit_commander.PlanningSceneInterface()

        self.laserscan_sub = rospy.Subscriber('/rx90l/laser/scan',
LaserScan, self.callback, queue_size=1)
        self.link_states_sub = rospy.Subscriber('/gazebo/link_states',
LinkStates, self.link_callback, queue_size=1)

        group_name = "rx90l"
        move_group = moveit_commander.MoveGroupCommander(group_name)

        display_trajectory_publisher =
rospy.Publisher('/move_group/display_planned_path',

moveit_msgs.msg.DisplayTrajectory,
                                                    queue_size=20)

        eef_link = move_group.get_end_effector_link()
        group_names = robot.get_group_names()

        # Misc variables
        self.box_name = ''
        self.robot = robot
```

```
        self.scene = scene
        self.move_group = move_group
        #self.gripper = gripper
        self.display_trajectory_publisher = display_trajectory_publisher
        self.eef_link = eef_link
        self.group_names = group_names

        self.publish()


    def callback(self, msg):

        global laserlist
        self.laser = msg
        laserlist = array(self.laser.ranges)  #ekxorisi timwn tou laser ston
pinaka laserlist
        laserlist[laserlist == inf] = 200

    def link_callback(self, msg):

        global actor_x,actor_y,actor_z

        self.link = msg

        actor_x = self.link.pose[1].position.x
        actor_y = self.link.pose[1].position.y
        actor_z = self.link.pose[1].position.z

    def publish(self):

        global scale

        move_group = self.move_group

        scale = 1

        "Go to home"

        joint_goal = move_group.get_current_joint_values()
        joint_goal[0] = 0
        joint_goal[1] = 0
        joint_goal[2] = 0
        joint_goal[3] = 0
        joint_goal[4] = 0
        joint_goal[5] = 0
        joint_goal[6] = 0
        move_group.go(joint_goal, wait=True)
        move_group.stop()

        "Insert boxes on scene"

        box_name = self.box_name
        scene = self.scene
        eef_link = self.eef_link

        scene.remove_attached_object(eef_link, name='box')
        scene.remove_world_object('box')

        box_pose = geometry_msgs.msg.PoseStamped()
        box_pose.header.frame_id = "base_link"
        box_pose.pose.orientation.w = 1.0
        box_pose.pose.position.x = 0
        box_pose.pose.position.y = 1
        box_pose.pose.position.z = 0.5 # slightly above the end effector
        box_name = "box"
        scene.add_box(box_name, box_pose, size=(0.02, 0.02, 0.2))
```

```
box_pose = geometry_msgs.msg.PoseStamped()
box_pose.header.frame_id = "base_link"
box_pose.pose.orientation.w = 1.0
box_pose.pose.position.x = 0
box_pose.pose.position.y = 1
box_pose.pose.position.z = 0.2
box_name = "table2"
scene.add_box(box_name, box_pose, size=(0.4, 0.2, 0.4))

while not rospy.is_shutdown():
    if (actor_x < 1.7 and actor_y > 0.75 ):

        "Go to box, grab and attach to gripper"
        print("Grabbing object")

        joint_goal = move_group.get_current_joint_values()
        joint_goal[0] = 1.57
        joint_goal[1] = 0.5
        joint_goal[2] = 1.61
        joint_goal[3] = 0
        joint_goal[4] = 0.54
        joint_goal[5] = 0
        joint_goal[6] = 0
        move_group.go(joint_goal, wait=True)
        move_group.stop()

        joint_goal = move_group.get_current_joint_values()
        joint_goal[0] = 1.57
        joint_goal[1] = 0.5
        joint_goal[2] = 1.61
        joint_goal[3] = 0
        joint_goal[4] = 0.54
        joint_goal[5] = 0
        joint_goal[6] = 0.55
        move_group.go(joint_goal, wait=True)
        move_group.stop()



        robot = self.robot
        scene = self.scene
        eef_link = self.eef_link
        group_names = self.group_names
        grasping_group = 'gripper'
        touch_links = robot.get_link_names(group=grasping_group)
        scene.attach_box(eef_link, 'box', touch_links=touch_links)


        "Collaboration work"
        print("Collaboration work")

        joint_goal = move_group.get_current_joint_values()
        joint_goal[0] = 0.43
        joint_goal[1] = 0.95
        joint_goal[2] = 0.96
        joint_goal[3] = 0
        joint_goal[4] = 0.34
        joint_goal[5] = 0
        joint_goal[6] = 0.55
        move_group.go(joint_goal, wait=True)
        move_group.stop()


        "Gripper rotation"
        joint_goal = move_group.get_current_joint_values()
        joint_goal[0] = 0.43
        joint_goal[1] = 0.95
        joint_goal[2] = 0.96
```

```
                joint_goal[3] = 0
                joint_goal[4] = 0.34
                joint_goal[5] = 4.71
                joint_goal[6] = 0.55
                move_group.go(joint_goal, wait=True)
                move_group.stop()


                "Unmount object"
                print("End of work")

                scene.remove_attached_object(eef_link, name='box')
                scene.remove_world_object('box')
                rospy.sleep(0.5)
                joint_goal = move_group.get_current_joint_values()
                joint_goal[0] = 0.43
                joint_goal[1] = 0.95
                joint_goal[2] = 0.96
                joint_goal[3] = 0
                joint_goal[4] = 0.34
                joint_goal[5] = 4.71
                joint_goal[6] = 0
                move_group.go(joint_goal, wait=True)
                move_group.stop()
                rospy.sleep(0.5)


                "Move backwards cartesian"
                print("Going to ready position")

                thesi = move_group.get_current_pose().pose
                waypoints = []
                thesi.position.x -= 1 * 0.4  # Second move forward/backwards
in (x)
                waypoints.append(copy.deepcopy(thesi))
                thesi.position.y -= 1 * 0.4
                waypoints.append(copy.deepcopy(thesi))

                (plan, fraction) = move_group.compute_cartesian_path(
                                            waypoints,   # waypoints to
follow
                                            0.01,        # eef_step
                                            0.0)         # jump_threshold

                move_group.execute(plan, wait=True)

                "Go home"

                joint_goal = move_group.get_current_joint_values()
                joint_goal[0] = 0
                joint_goal[1] = 0
                joint_goal[2] = 0
                joint_goal[3] = 0
                joint_goal[4] = 0
                joint_goal[5] = 0
                joint_goal[6] = 0
                move_group.go(joint_goal, wait=True)
                move_group.stop()

                rospy.sleep(2)

    def turn_off(self):
        pass


def controller_py():
    # Starts a new node
    rospy.init_node('controller_node', anonymous=True)
```

```
    controller = rx90l_controller()
    rospy.on_shutdown(controller.turn_off)
    rospy.spin()

if __name__ == '__main__':
    try:
        controller_py()
    except rospy.ROSInterruptException:
        pass
```