ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

Διαδίκτυο των Πραγμάτων: Ευφυή Περιβάλλοντα σε δίκτυα Νέας Γενιάς

Μελέτη & υλοποίηση ρομποτικού συστήματος διαχωρισμού λευκών ειδών με υπολογιστική όραση και αντίστροφη κινηματική σε 5-DOF βραχίονες.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

των

Αρχοντή Χρήστου
&
Γιαννισλή Ιωάννη

Επιβλέπων : Καβαλλιεράτου Εργίνα

Μέλη εξεταστικής επιτροπής: Σταματάτος Ευστάθιος, Καπόρης Αλέξης

Σάμος, Φεβρουάριος 2023

Η σελίδα αυτή είναι σκόπιμα λευκή.

Πρόλογος και ευχαριστίες

Για την εκπόνηση της συγκεκριμένης εργασίας θέλουμε να ευχαριστήσουμε θερμά την επιβλέπουσα Καθηγήτρια του Τμήματος Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων Κα. Καβαλλιεράτου Εργίνα για την εμπιστοσύνη που μας έδειξε αναθέτοντάς μας αυτή την εργασία, την υπομονή και την καθοδήγησή της έως την εκπόνηση αυτής.

Επιπλέον, θέλουμε να ευχαριστήσουμε θερμά τις οικογένειές μας, οι οποίες μας στήριξαν στις σπουδές μας από την πρώτη στιγμή και μας ώθησαν στην ολοκλήρωση και αυτού του σταδίου εκπαίδευσης.

Τέλος, σημαντικότατη ήταν η γενικότερη προσφορά του τμήματος, το οποίο μας παραχώρησε υλικό και λογισμικό εξοπλισμό, αναγκαίο για την επίτευξη των πειραμάτων.

Η σελίδα αυτή είναι σκόπιμα λευκή.

Πίνακας περιεχομένων

Λίστα Σχημάτων

Λίστα Πινάκων

Ακρωνύμια

IoT	Internet of Things
DoF	Degrees of Freedom
HDMI	High-Definition Multimedia Interface
NOOBS	New Out of Box Software
kg	Kilogram
cm	Centimeter
V	Volt
OCR	Optical Character Recognition
HMM	Hidden Markov model
EM	Expectation – maximization
Cnn	Convolutional neural networks

Περίληψη

Στην εργασία αυτή μελετάται η κατασκευή ενός ρομποτικού συστήματος διαχείρισης λευκών ειδών χαμηλού κόστους σε ξενοδοχειακό περιβάλλον με στόχο την επεξεργασία των λερωμένων λευκών ειδών. Η παρούσα κατασκευή έχει πολύ μεγάλη χρηστικότητα στην εποχή του κορονοϊού, καθώς απομακρύνει το ανθρώπινο δυναμικό από πιθανών μολυσμένα είδη και παράλληλα μπορεί να χρησιμοποιηθεί, ώστε να βελτιωθεί η αειφορία και να μειωθεί το περιβαλλοντικό αντίκτυπο που προκαλεί πολλές φορές φορές ο άνθρωπος σε αυτούς τους χώρους, μέσα από την ελλιπή διαχείριση καθαριστικών προϊόντων.

Πρόκειται για ένα αυτόνομο σύστημα με δύο 5-DoF βραχίονες, το οποίο αρχικά θα τραβάει μια φωτογραφία του λευκού είδους που θα έχει μπροστά του, θα επεξεργάζεται την εικόνα ώστε να βρει τις γωνίες σε αυτό και στη συνέχεια θα το σηκώνει στον αέρα. Έπειτα ξανατραβώντας φωτογραφία θα αναγνωρίζει αν το λευκό είδος είναι πολύ ή όχι λερωμένο και θα το τοποθετεί στο κατάλληλο καλάθι. Μάλιστα για να επιτύχει όλα τα παραπάνω θα λειτουργεί χρησιμοποιώντας αντίστροφη κινηματική, θα είναι δηλαδή ικανότατο να προσδιορίσει μόνο του τη θέση του στο χώρο και σε σχέση με τα αντικείμενα.

Λέξεις Κλειδιά: Inserve Kinematic, Υπολογιστική όραση, 5 DoF, νευρωνικά δίκτυα

Abstract

In this work, the construction of a low-cost linen management robotic system in a hotel environment is studied with the aim of processing soiled linen. The present construction has great utility in the age of the Covid virus, as it removes human resources from possible infected linen and at the same time can be used to improve sustainability and reduce the environmental impact that man often causes in these spaces, through insufficient management of cleaning products.

It is a standalone system with two 5-DoF arms that firstly it will take a photo of the white object in front of it, process the image to find the angles on it, and then lift it into the air. Then by retaking a photo, it will recognize whether the white item is filthy or not and will place it in the appropriate basket. In fact, in order to achieve all of the above, it will work using inverse kinematics, that is, it will be able to determine its position in space and relation to objects.

Keywords: *Inserve Kinematic, Computer Vision, 5 DoF, neural networks*

1 Εισαγωγή

1.1 Ρομποτικοί βραχίονες και ταξινόμηση λευκών ειδών

Στην παρούσα διπλωματική εργασία θα μελετηθεί ο σχεδιασμός και η ανάπτυξη ενός ρομποτικού συστήματος, ικανό να αναγνωρίσει και να ξεχωρίσει με επιτυχία λερωμένα από καθαρά λευκά είδη. Ορισμένα από τα χαρακτηριστικά του χώρου εφαρμογής της διπλωματικής, είναι η διαφορετικότητα και η μοναδικότητα της κάθε περίπτωσης, η αυτόνομη κίνηση βραχιόνων σε τρεις άξονες, και η δυνατότητα αναγνώρισης βάθους και ύψους, μέσω υπολογιστικής όρασης.

1.2 Αντικείμενο διπλωματικής

Η διπλωματική μελέτη στοχεύει όπως αναφέρθηκε στη μελέτη και στην υλοποίηση ενός ρομποτικού συστήματος το οποίο καλείται να τοποθετηθεί σε ξενοδοχειακά περιβάλλοντα ώστε να μπορέσει να αντιμετωπίσει σύγχρονα προβλήματα. Στη σημερινή εποχή το μεγαλύτερο ίσως πρόβλημα που αντιμετωπίζει ο πλανήτης μας είναι η κλιματική αλλαγή και η διαχείριση της ενέργειας. Η κλιματική αλλαγή επιβαρύνεται από τη σπάταλη χρήση διαφόρων ουσιών, οι οποίες παρόλο που αρχικά στοχεύανε στη βελτίωση της ανθρώπινης ζωής, πλέον τη θέτουν σε κίνδυνο χρησιμοποιώντας καθαριστικές ουσίες χωρίς μέτρο και χωρίς να ακολουθούνται οι οδηγίες που αναγράφονται στα καθαριστικά προϊόντα. Όσον αφορά τη διαχείριση ενέργειας ένα ρομποτικό σύστημα που θα αναγνωρίζει το επίπεδο λεκέδων στα υφάσματα, θα είναι καταλληλότερο από έναν άνθρωπο στον ορθό διαχωρισμό των πλύσεων ανάμεσα στα πολύ λερωμένα υφάσματα και στα ελαφρώς λερωμένα. Μάλιστα η σημαντικότητά της ορθότερης διαχείρισης των απορρυπαντικών προκύπτει από το γεγονός ότι περίπου το 19% του συνόλου των καθαριστικών που χρησιμοποιούνται στην Αυστραλία είναι για το πλύσιμο των ρούχων και αντίστοιχα στη Γερμανία είναι 13% [11], [12], [13], [14].

Το παρόν σύστημα είναι σε θέση να κατανοήσει την ύπαρξή του σε έναν περιορισμένο χώρο και να αναγνωρίσει διαφορές που αντιλαμβάνεται το ανθρώπινο μάτι. Χρησιμοποιώντας υπολογιστική όραση καταφέρνει να έχει την επιθυμητή ικανότητα αντίληψης των αντικειμένων και μέσω κατάλληλα εκπαιδευμένου νευρωνικού δικτύου μπορεί να διαφοροποιήσει τα υφάσματα που βλέπει και να λάβει αποφάσεις. Αναπόσπαστο κομμάτι του συστήματος είναι η αντίστροφη κινηματική που χρησιμοποιείται για την ορθή και πλήρως αυτόνομη πλοήγηση των δύο βραχιόνων του συστήματος στο χώρο του.

1.3 Ξενοδοχειακά περιβάλλοντα

Ένα ξενοδοχειακό περιβάλλον είναι ο χώρος εκείνος, όπου πλήθος ανθρώπων αναζητά κατάλυμα κάτω από την ίδια στέγη. Τις περισσότερες φορές οι άνθρωποι αυτοί είναι άγνωστοι μεταξύ τους και ο καθένας βρίσκεται στο χώρο για διαφορετικούς σκοπούς, άλλοτε εργασιακούς και άλλοτε προσωπικούς (π.χ. αναψυχή).

Ο χώρος του ξενοδοχειακού περιβάλλοντος, τις περισσότερες φορές είναι αρκετά μεγάλος, παρουσιάζοντας συγκεκριμένες ανέσεις. Αυτές συνήθως είναι από ένα απλό δωμάτιο, μέχρι σουίτες και από ένα απλό χώρο φαγητού, μέχρι πολλές διαφορετικές αίθουσες με διαφορετική θεματολογία φαγητού και ποτού. Παράλληλα ένα ξενοδοχειακό περιβάλλον μπορεί να περιλαμβάνει από πισίνα, σπα, τζακούζι και αύλειο χώρο με παιδική χαρά, μέχρι εκτάσεις με γήπεδα (τένις, γκολφ, μπάσκετ) και πολύ πιθανό γυμναστήρια και κέντρα αισθητικής. Ορισμένα μάλιστα ξενοδοχειακά περιβάλλοντα φέρουν πρόσβαση σε παραλία, όπου εκεί παρέχουν πρόσθετες ανέσεις. Οι χώροι αυτοί όπως είναι λογικό χρειάζονται ένα μεγάλο πλήθος λευκών ειδών τις περισσότερες φορές διαφορετικό ανάλογα τη δραστηριότητα.

1.4 Παρόμοια περιβάλλοντα

Φυσικά όπως εύκολα προκύπτει, υπάρχουν αρκετά περιβάλλοντα τα οποία μοιάζουν σε ένα ξενοδοχειακό περιβάλλον. Τέτοια είναι χώροι όπου ο κόσμος καλείται να συνυπάρξει και την ίδια στιγμή είναι δύσκολη η διαχείριση των όμβριων υδάτων. Για παράδειγμα σε ένα πλοίο που κινείται στη θάλασσα ασταμάτητα ή σε κάποια απομονωμένη ερευνητική εγκατάσταση. Επόμενο πιθανό περιβάλλον που πιθανότατα χρειάζεται τέτοιου είδους ρομποτικά συστήματα είναι οι απομακρυσμένες πλατφόρμες εξόρυξης αργού πετρελαίου και αερίου.

1.5 Εφαρμογή σε κρίσεις πανδημιών

Η τεχνητή νοημοσύνη σίγουρα δεν είναι ακόμα έτοιμη να κατακτήσει τον κόσμο, όμως είναι αρκετά προηγμένη ώστε να πραγματοποιήσει ορισμένες εργασίες που μέχρι τώρα έκανε ο άνθρωπος. Στις αρχές της δεύτερης δεκαετίας του 2000, έκανε την εμφάνισή του στην ανθρωπότητα ένας φονικός ιός (covid-19), ο οποίος θεωρήθηκε πανδημία. Ευτυχώς,

η προηγμένη ιατρική κοινότητα κατάφερε να περιορίσει σημαντικά τις επιπτώσεις της πανδημίας, χωρίς να υπάρχουν τα ποσοστά θανάτων των προηγούμενων μεγάλων πανδημιών που έπληξαν την υφήλιο (Ισπανική γρίπη, χολέρα, Μαύρη πανώλη). Παρόλα αυτά, 2 έτη μετά την εξάπλωσή της ακόμα δεν έχει εξαλειφθεί και συνεχίζει να εμφανίζει νέες μεταλλάξεις. Σύμφωνα με την παγκόσμια ιατρική κοινότητα, θα συνεχίσει να αποτελεί κίνδυνο για την παγκόσμια υγεία τα επόμενα χρόνια.

Δίνεται ιδιαίτερη έμφαση στη παρούσα πανδημία, γιατί το μέσο της μετάδοσής της είναι ο άνθρωπος και χωρίς αυτόν στο χώρο ως εργαζόμενο, θα μπορέσει να σπάσει εύκολα η αλυσίδα της μετάδοσης. Μελετάται η περίπτωση όπου τα λευκά είδη συγκεντρώνονται όλα μαζί σε ένα χώρο και οι υπάλληλοι είναι αυτοί που θα κάνουν τη διαλογή αυτών χωρίζοντάς τα σε πολύ ή λίγο λερωμένα ή ανάλογα με τη σύστασή τους σε διαφορετικούς τρόπους καθαρίσματος. Οι εργαζόμενοι αυτοί, φυσικά μετά την εργασία τους θα επιστρέψουν στο σπίτι τους και σε περίπτωση που έχουν κολλήσει τον ιό από κάποιο ύφασμα, πολύ πιθανό να το μεταδώσουν σε συγγενή τους και αυτός με τη σειρά του σε κάποιον επόμενο. Όμως αν ο εργαζόμενος βγει από τη θέση του διαχωρισμού, τότε η αλυσίδα μετάδοσης σταματάει στο αρχικό της κιόλας στάδιο.

1.6 Δομή της διπλωματικής

Στο δεύτερο κεφάλαιο παρουσιάζονται εργασίες σχετικές με το αντικείμενο της διπλωματικής. Στο τρίτο κεφάλαιο ακολουθεί η παρουσίαση των υλικών του συστήματος και δίνεται έμφαση στα σημαντικά χαρακτηριστικά των υλικών. Ακόμη αναφέρεται ο τρόπος λειτουργίας αυτών. Στο τέταρτο κεφάλαιο εισάγονται έννοιες, παρουσιάζονται οι δυνατότητες κίνησης και όρασης που χρησιμοποιούνται για την επίτευξη του στόχου. Στο πέμπτο κεφάλαιο ξεκινά η υλοποίηση του συστήματος και παρουσιάζονται λεπτομέρειες της εγκατάστασης. Παράλληλα γίνεται ανάπτυξη στις σημαντικές συναρτήσεις που θα χρησιμοποιηθούν και στην εκμάθηση του μοντέλου. Στο έκτο κεφάλαιο και τελευταίο κεφάλαιο, ακολουθούν τα συμπεράσματα του παρόντος πειράματος. Τέλος ακολουθεί η βιβλιογραφία.

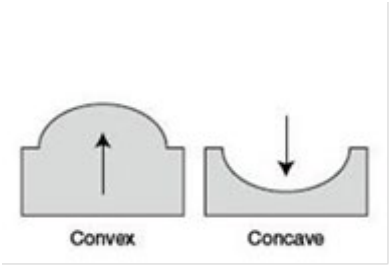
2 Σχετικές εργασίες

Αρχικά η ιδέα ενός ρομποτικού βραχίονα ήρθε έπειτα από έρευνα και συζητήσεις με το ανθρώπινο δυναμικό ξενοδοχειακών μονάδων, αναφορικά με εργαλεία που θα διευκόλυναν την εργασία τους κατά την περίοδο του κορονοϊού. Φυσικά το προσωπικό ήθελε ένα ρομπότ που θα αναλάμβανε την καθαριότητα εξ' ολοκλήρου, όμως κάτι τέτοιο δεν ήταν εφικτό, οπότε η ιδέα επικεντρώθηκε στο τομέα καθαριότητας των λευκών ειδών. Έπειτα υπήρξε μελέτη αναφορικά με παλαιότερες εργασίες και έρευνες που έχουν πραγματοποιηθεί με ρομποτικούς βραχίονες και παρακάτω παρουσιάζονται αυτές που αποτέλεσαν το κεντρικό πυλώνα, της παρούσας μελέτης.

2.1 Picking towels in point clouds [1]

Το paper του τίτλου, αναφέρεται σε μια μελέτη επιλογής ρούχων μέσα από ένα καλάθι ή πάνω από ένα τραπέζι. Στόχος των συγγραφέων ήταν να μπορέσουν να πιάσουν σωστά ένα ύφασμα, ώστε να μπορέσει ο βραχίονας να το σηκώσει. Δεν είχαν ως στόχο την τοποθέτηση αυτού σε μια στοίβα ή σε ένα δεύτερο καλάθι. Για να επιτύχουν κάτι τέτοιο, επικαλέστηκαν τη μηχανική μάθηση και μέσω κάμερας, μπόρεσαν να αναγνωρίσουν τις καμπύλες ενός υφάσματος, ώστε να το πιάσει ο βραχίονας σε εκείνο το σημείο. Ο τρόπος τους μάλιστα είχε επιτυχία 80%. Το paper είναι του Δεκεμβρίου του 2018 και δημοσιεύτηκε το Φεβρουάριου του 2019. Παρακάτω ακολουθεί αναλυτικότερα η μέθοδος.

Η παραπάνω μελέτη είναι διαφορετική από άλλες, καθώς αναγνωρίζει τη δυσκολία πιασίματος ενός μόνο υφάσματος. Ως υλικό δοκιμών χρησιμοποιεί πετσέτες, οι οποίες όμως δε διαφέρουν χρωματικά, καθώς κάτι τέτοιο χρειάζεται επιπρόσθετους αλγορίθμους ανάγνωσης. Κατέληξαν στο συμπέρασμα πως κάθε ύφασμα πιάνεται ευκολότερα με διαφορετικό τρόπο αναλόγως της σύστασής του (π.χ. οι πετσέτες πιάνονται ευκολότερα από τα τραπεζομάντηλα, λόγω πάχους). Ακόμα στοχεύοντας να πιάσουν με μεγαλύτερη επιτυχία την ανώτερη πετσέτα από τη στοίβα, χρησιμοποίησαν αλγορίθμους αναγνώρισης της καμπυλότητας των πετσετών. Αν δηλαδή μια πετσέτα έχει τη καμπύλη προς τα πάνω, έχει κυρτή καμπύλη, ενώ αν έχει καμπύλη προς τα κάτω, έχει κοίλη καμπύλη (Σχήμα 1). Με αυτό τον οδηγό κατέληξαν στην εύρεση πως το σημείο της πετσέτας που είναι ευκολότερο να πιαστεί από το βραχίονα είναι η ανώτερη κυρτή καμπύλη της. Τέλος, η δαγκάνα ήταν σε θέση να περιστραφεί ώστε να πιάσει ορθά αυτή την καμπύλη.

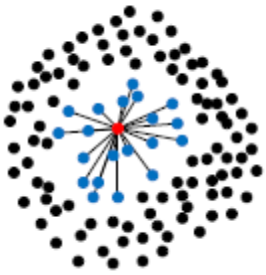


Σχήμα 1: (αριστερά) φαίνεται η κυρτή καμπύλη, (δεξιά) φαίνεται η κοίλη καμπύλη

Για να μπορέσει το ρομπότ να αντιληφθεί με επιτυχία το βάθος, χρειάζεται το χρώμα του φόντου να είναι διαφορετικό από το χρώμα των υφασμάτων. Έπειτα για να καταφέρει να αναγνωρίσει τα αντικείμενα δημιουργεί ένα γράφο και τον χρησιμοποιεί ώστε να κατανοήσει ποιο είναι το υψηλότερο σημείο της πετσέτας που έχει μπροστά του. Για να το κάνει αυτό πρέπει να εξαλείψει το περιβάλλον πίσω από τις πετσέτες.

Το ρομπότ αντιλαμβάνεται το είδος της κοιλότητας που βλέπει ανάλογα με τη διάθλαση των ακτινών φωτός. Στην περίπτωση που οι ακτίνες συγκλίνουν ως προς το σημείο εκπομπής, αντιλαμβάνεται κυρτή καμπύλη, ενώ στην περίπτωση που οι ακτίνες αποκλίνουν αντιλαμβάνεται κοίλη καμπύλη.

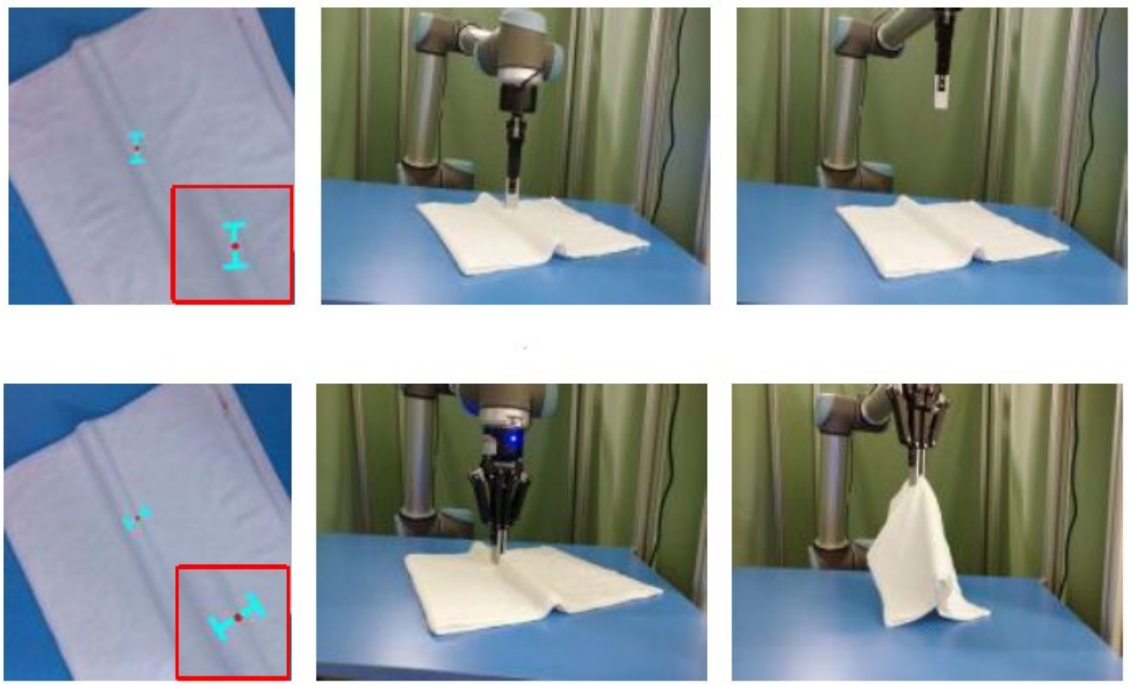
Λόγω του ότι οι πετσέτες του πειράματος είναι του ίδιου χρώματος, το ρομπότ δε μπορεί να χρησιμοποιήσει το δεδομένο αυτό για να καταλάβει ποια από τα σημεία της πετσέτας που έχει μπροστά του είναι το ψηλότερο. Για να αναγνωρίσει το ψηλότερο σημείο ακολουθεί την τεχνική των γειτόνων στα σημεία. Λαμβάνει συνεπώς ως παράμετρο μέσα στις συναρτήσεις το πλήθος των γειτόνων ενός πιθανού σημείου, ώστε να αναγνωρίσει αν πρόκειται για μια μεγάλη ομοιόμορφη κυρτή κοιλότητα (Σχήμα 2) και όχι απλώς μια γραμμική κυρτή κοιλότητα. Όλη αυτή η διαδικασία γίνεται διότι οι βραχίονες με διπλή δαγκάνα, όπως και του πειράματος, είναι δύσκολο να πιάσουν με επιτυχία ένα ύφασμα από μια γραμμική κυρτή καμπύλη του.



Σχήμα 2: Φαίνονται οι κοντινοί γείτονες στη στρογγυλή ομοιογένεια.

Στην περίπτωση που η κυρτή καμπύλη δεν έχει στρογγυλή ομοιογένεια ως προς το ψηλότερο σημείο, τότε πρέπει η διπλή δαγκάνα να περιστραφεί ανάλογα με τη κυρτή γραμμική καμπύλη, ώστε να μπορέσει να την πιάσει με επιτυχία. Αυτό επιβάλλεται μάλιστα, διότι αν δεν καταφέρει να την πιάσει με την πρώτη, είναι σχεδόν σίγουρο ότι θα αλλάξει η κυρτότητα της καμπύλης. Όλο αυτό λοιπόν ενδέχεται να επιφέρει

αλυσιδωτές αποτυχίες με αποτέλεσμα την απομάκρυνση από το στόχο. Για τα καλύτερα μάλιστα δυνατά αποτέλεσμα πρέπει η διπλή δαγκάνα να τοποθετηθεί και να πιάσει κάθετα την κυρτή γραμμική καμπύλη. Στο σχήμα 3 φαίνεται στην πρώτη περίπτωση μια αποτυχημένη προσπάθεια πιασίματος, ενώ στη δεύτερη γραμμή μια επιτυχημένη. Οι πρώτες εικόνες, με τη γαλάζια αναγνώριση δείχνουν την καθετότητα που αναφέρθηκε.



Σχήμα 3: (επάνω) αποτυχημένη προσπάθεια πιασίματος πετσέτας, λόγω λάθους καθετότητας στη καμπύλη, (κάτω) επιτυχημένη προσπάθεια πιασίματος πετσέτας, ακολουθώντας τη σωστή καθετότητα

Καταλήγοντας αναφέρεται ότι έγιναν δύο πειράματα, στο πρώτο οι πετσέτες βρισκόταν επάνω σε ένα τραπέζι, ενώ στο δεύτερο μέσα σε ένα καλάθι. Το σύνολο των πετσετών ήταν 20, οι οποίες ήταν ισο-μοιρασμένες σε δύο μεγέθη και σε δύο πάχη. Παρατηρήθηκε ότι το πείραμα είχε πολύ μεγάλο αρχικό ποσοστό επιτυχίας περίπου 80%. Οι ανεπιτυχείς προσπάθειες ήταν είτε λόγω λάθος αναγνώρισης του ύψους με αποτέλεσμα να μη δημιουργείται τόσο μεγάλη καμπύλη και συνεπώς να μην υπάρχει αρκετό υλικό για να πιάσει η δαγκάνα και να γλιστρά η πετσέτα, είτε να υπάρχουν πολλαπλά ψηλά σημεία δίπλα στο ψηλότερο σημείο, με αποτέλεσμα όταν κατεβαίνει η δαγκάνα για να πιάσει το ψηλότερο σημείο να ακουμπά τα διπλανά ψηλά σημεία, προκαλώντας την υποχώρηση του ψηλότερου σημείο, πέρα από το υπολογισμένο εύρος πιασίματος της δαγκάνας.

2.2 Intelligent mobile robot controller design for hotel room service with deep learning arm-based elevator manipulator [2]

Το paper του τίτλου, αναφέρεται σε ένα ρομπότ το οποίο είναι σε θέση να χρησιμοποιεί το ασανσέρ πατώντας τα πλήκτρα. Οι συγγραφείς αυτού του paper, δέχονται ως δεδομένη την ύπαρξη ρομπότ που μπορούν να κινηθούν σε έναν όροφο, όμως για να είναι σε θέση να κινηθούν αποτελεσματικά σε μεγάλα συγκροτήματα κρίνεται απαραίτητο να μπορούν να χρησιμοποιήσουν αποτελεσματικά τα ασανσέρ. Συνεπώς ένα τέτοιο ρομπότ, πρέπει να είναι σε θέση να αναγνωρίζει τα πλήκτρα του ασανσέρ και να τα μεταφράζει όπως ο άνθρωπος, μέσα από τη κάμερά του, ώστε να φτάσει στο στόχο του. Παράλληλα αναφέρονται παρακάτω επιπλέον τεχνικές παρόμοιων μελετών, οι οποίες χρησιμοποιήθηκαν κυρίως για ιδεασμό.

Στην πρώτη εξ αυτών, μελέτη των P. Mukhopadhyay και B. B. Chaudhuri [3], τα κουμπιά αναγνωρίζονταν από τα άκρα τους και οι αριθμοί μέσα τους από μια βάση συμβόλων.

Στη μελέτη των X. Yu, D. Li, L. Liyuan, και H. Kah Eng [4], χρησιμοποιήθηκαν δύο αισθητήρες υπερήχων για να χειριστούν το ασανσέρ.

Στη μελέτη των W. J. Wang, C. H. Huang, I. H. Lai, και H. C. Chen [5], προτάθηκε ένα σύστημα γραμμών για να βρίσκονται εύκολα τα κουμπιά στους πίνακες των ασανσέρ. Επίσης, κάνοντας τις εικόνες δυαδικές και συγκρίνοντας τελικά χαρακτήρες το ρομπότ τους μπορούσε να αναγνωρίσει τα κουμπιά. Υπήρχε ακόμα κάμερα και αισθητήρες αφής στην άκρη του ρομποτικού βραχίονα και χρησιμοποιώντας αντίστροφη κινηματική για έλεγχο και κίνηση ήταν σε θέση να πατήσει τα κουμπιά.

Στη μελέτη των E. Klingbeil, B. Carpenter, O. Russakovsky, και A. Y. Ng [6], συνδυάστηκε αλγόριθμος EM, αναγνώριση οπτικής κύλισης παραθύρου, τεχνικές OCR και HMM για αναγνώριση των πινάκων και των κουμπιών στα ασανσέρ. Η αναγνώριση των χαρακτήρων έγινε μέσα από τη χρήση νευρωνικών δικτύων και χρησιμοποιήθηκαν 3D αισθητήρες και δύο κάμερες για την επίτευξη του στόχου.

Στη μελέτη των Z. Dong, D. Zhu, και M. Q. H. Meng [7], παρουσιάστηκε μια μέθοδος αναγνώρισης των αντικειμένων με τη χρήση νευρωνικών δικτύων (CNN) που λάμβαναν το περίγραμμα των πλήκτρων αναγνωρίζοντας τις άκρες αυτών.

Στη μελέτη των Troniak, D., Sattar, J., Gupta, A., Little, J. J., Chan, W., Calisgan, E. και Van der Loos [8], χρησιμοποιήθηκε αναγνώριση κοινών χαρακτηριστικών για την αναγνώριση των κουμπιών και μια 3D κάμερα για την εύρεση των τοποθεσιών των κουμπιών.

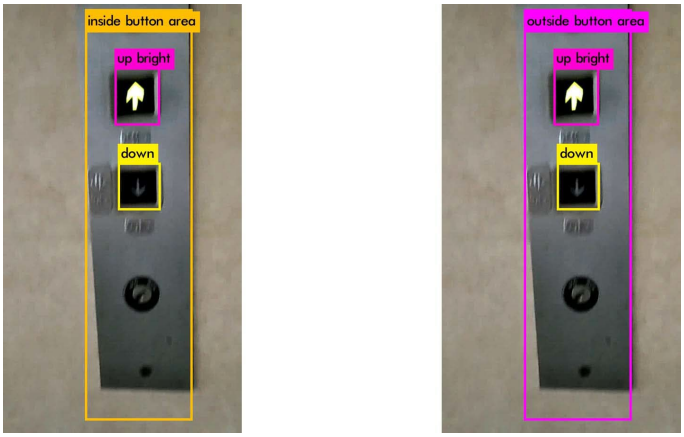
Στη μελέτη των Abdulla, A. A., Liu, H., Stoll, N. και Thurow, K [9], συνδυάστηκε επεξεργασία χρώματος, μείωση θορύβου και εξαγωγή του αριθμού του κουμπιού σε ιδιαίτερες περιπτώσεις, όπως οι γυάλινοι ανελκυστήρες. Επιπλέον αναφέρεται ότι αν

υπάρχει διάφανο υλικό πάνω από τα κουμπιά ή κάποια ανακλαστική επίστρωση, η αναγνώριση γίνεται δυσκολότερη.

Όμως στο παρόν paper, δημιουργήθηκε μια μεγάλη βάση δεδομένων από πίνακες και κουμπιά ασανσέρ, που σε αντίθεση με τις προηγούμενες μελέτες, είναι σε θέση να αναγνωρίσει πληθώρα διαφορετικών κουμπιών και πινάκων. Χρησιμοποιείται αλγόριθμος deep learning, ανεστραμμένης προβολής των αριθμών και κάμερα για να επιτύχουν την αναγνώριση των κουμπιών σε τρεις διαστάσεις. Μάλιστα λόγω των deep learning αλγορίθμων το ρομπότ μπορεί να κάνει την απαιτούμενη δουλειά με αισθητήρες χαμηλού κόστους και με αρκετά μειωμένο βάρος.

Μάλιστα το ρομπότ του πειράματος είναι σε θέση να αναγνωρίζει επίσης αν το ασανσέρ έχει κληθεί να μεταβεί σε άλλο όροφο αναγνωρίζοντας τον πίνακα και τα κουμπιά του ασανσέρ, εντός και εκτός αυτού, από την αλλαγή της φωτεινότητας σε αυτά. Η βελτίωση του μοντέλου επιτυγχάνεται μέσα από περιορισμούς της λανθασμένης ανάγνωσης των αριθμών. Η εκτίμηση των τρισδιάστατων συντεταγμένων είναι σε θέση να υπερνικήσει τα προβλήματα που προκύπτουν από τους χαμηλού κόστους αισθητήρες και παράλληλα μπορεί να προσπελάσει και πιθανά εμπόδια που προκύπτουν από μικρές αποκλίσεις της απόστασης των κουμπιών από τον βραχίονα.

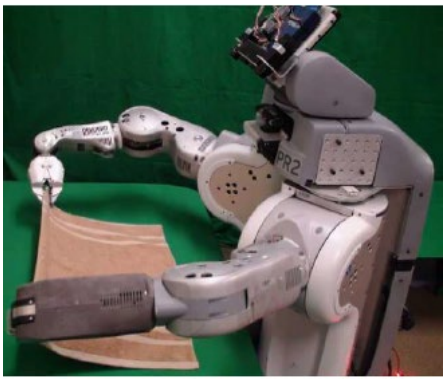
Σχήμα 4: Αναγνώριση του πίνακα και των κουμπιών με διαχωρισμό των φωτεινών από τα μη φωτεινά



Σχήμα 5: Αναγνώριση του εξωτερικού πίνακα και των κουμπιών με διαχωρισμό των φωτεινών από τα μη φωτεινά

2.3 Cloth grasp point detection based on multiple – view geometric cues with application to robotic towel folding [10]

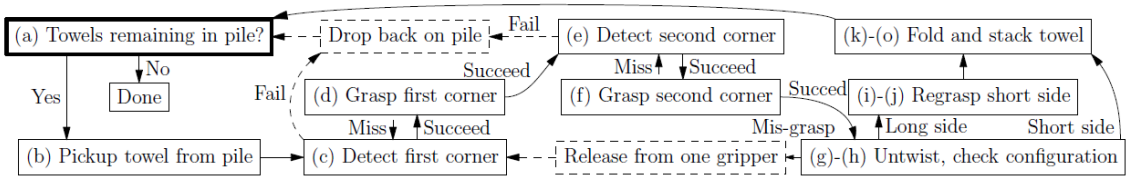
Η μελέτη των Maitin-Shepard, J., Cusumano-Towner, M., Lei, J., και Abbeel, P., αναφέρεται σε ένα ρομπότ ικανό να ανασηκώσει πετσέτες από μία στοίβα στη συνέχεια να τις διπλώσει και τέλος να τις αφήσει σε συγκεκριμένο μέρος. Για να επιτευχθεί αυτό το ρομπότ χρειάζεται μόνο δύο βραχίονες και ένα ζεύγος καμερών (Σχήμα 6). Το πρώτο βήμα του ρομπότ είναι να ανασηκώσει την πετσέτα, τυχαία από μια στοίβα που βρίσκεται μπροστά του. Στη συνέχεια αναγνωρίζει, αν είναι εφικτό, μία από τις γωνίες της πετσέτας και την πιάνει. Έπειτα κάνοντας ορισμένες περιστροφικές κινήσεις προσπαθεί να πιάσει κατάλληλα την πετσέτα και από δεύτερη γωνία. Τέλος με συγχρονισμένες κινήσεις καταφέρνει να τη διπλώσει πάνω σε ένα τραπέζι, χωρίς τη χρήση βοηθητικών εργαλείων και τέλος την τοποθετεί σε στοίβα. Μάλιστα ο αλγόριθμος, έχει τέτοια επιτυχία που είναι η πρώτη φορά που ρομπότ καταφέρνει να διπλώσει με επιτυχία 50 πετσέτες που δεν έχει "ξαναδεί" και να τις τοποθετήσει σε πέντε στοίβες.



Σχήμα 6: Ρομπότ του 4ου paper

Οι συγγραφείς τονίζουν, όπως είναι λογικό, ότι η λειτουργία διπλώματος των ρούχων είναι κάτι πραγματικά δύσκολο για ένα ρομπότ, καθώς κάθε φορά που καλείται να διπλώσει μια πετσέτα από ένα πάγκο για παράδειγμα, είναι σχεδόν σίγουρο ότι η πετσέτα θα έχει μια μοναδική τοποθέτηση απέναντι στο ρομπότ, λόγω των καμπυλοτήτων που δημιουργούνται κάθε φορά. Σε αντίθεση με ένα ρομπότ σε ένα αυστηρό περιβάλλον, που κάθε φορά θα δέχεται με τον ίδιο τρόπο τα αντικείμενα που θέλει να επεξεργαστεί.

Όλες αυτές ο πιθανές λοιπόν δυσκολίες πρέπει να αντιμετωπιστούν με επιτυχία, ώστε ένα ρομπότ διπλώματος ρούχων να κρίνεται επιτυχημένο και να είναι πραγματικά αυτόματο, απέναντι σε μια στοίβα με πετσέτες. Στο παρόν paper λοιπόν παρουσιάζεται ένας αλγόριθμος, ο οποίος επικεντρώνεται στη διαχείριση του σωστού πιασίματος της πετσέτας, μέσα από την αναγνώριση των γωνιών, του σωστού τεντώματος και τέλος του διπλώματος, ως προς τη μικρότερη πλευρά.



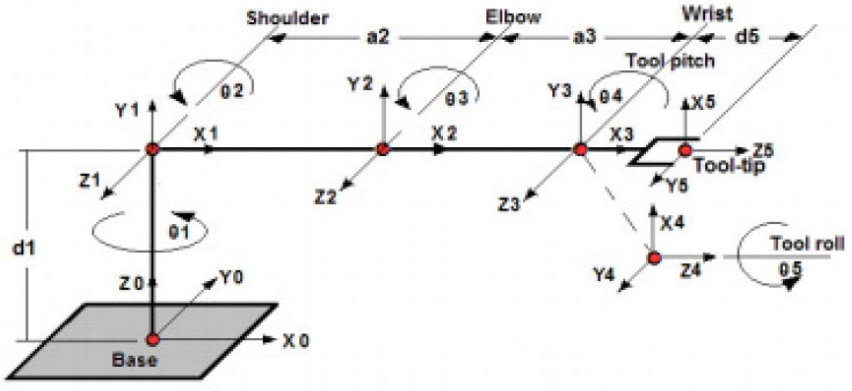
Σχήμα 7: Κύριος αλγόριθμος του paper

Παρόλο που η κεντρική λειτουργία του παρόντος ρομπότ στηρίζονταν κυρίως σε γεωμετρικά δεδομένα τα αποτελέσματα που έφερε ήταν πάρα πολύ ικανοποιητικά. Καταλήγουν επίσης στο συμπέρασμα ότι αυτός ο αλγόριθμος αναγνώρισης γωνιών, θα μπορούσε να χρησιμοποιηθεί και με επιτυχία και σε άλλα ήδη ρουχισμού, πέρα από τις πετσέτες.

2.4 Inverse Kinematics Analysis and Simulation of a 5 DOF Robotic Arm using MATLAB [16]

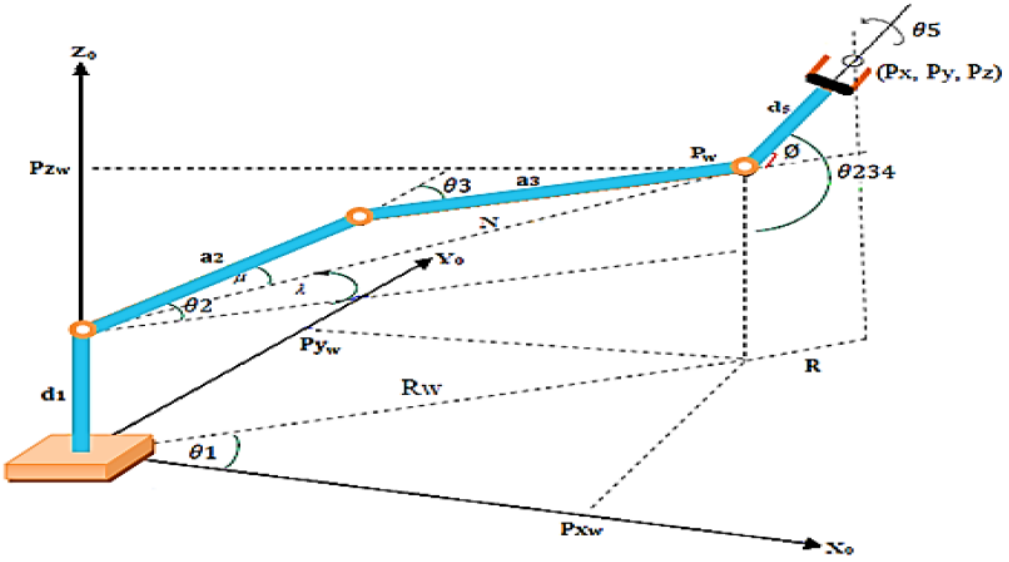
Το paper του τίτλου παρουσιάσει την αντίστροφη κινηματική ενός 5-DoF ρομποτικού βραχίονα χρησιμοποιώντας την εργαλειοθήκη ρομποτικής του MATLAB και τις παραμέτρους της μεθόδου Denavit-Hartenberg που χρησιμοποιήθηκαν για να αναπαραστήσουν τους συνδέσμους και τις αρθρώσεις του ρομποτικού βραχίονα. Χρησιμοποιήθηκε μια γεωμετρική προσέγγιση για τη λύση των εξισώσεων αντίστροφης κινηματικής που επιστρέφουν τις τιμές των παραμέτρων των αρθρώσεων ώστε να μετακινηθεί ο end-effector στις δοθείσες συντεταγμένες. Επίσης, αξίζει να αναφερθεί ότι παρόν paper, βοήθησε και ενέπνευσε στη δημιουργία των εξισώσεων αντίστροφης κινηματικής των ρομποτικών βραχιόνων του προτεινόμενου συστήματος, οι οποίες έχουν αρκετά κοινά σημεία με αυτές του paper.

Για την προσομοίωση της κίνησης του ρομποτικού βραχίονα χρησιμοποιήθηκε το MATLAB. Επίσης, χρησιμοποιήθηκε το MATLAB (GUI) για την προβολή της θέσης κάθε άρθρωσης. Τα αποτελέσματα έδειξαν ότι μέγιστο σφάλμα στις συντεταγμένες X,Y,Z του end-effector ήταν 0,0251%, 0,0239% και 0,1085% αντίστοιχα.



Σχήμα 8: Πλαίσιο συντεταγμένων ρομποτικού βραχίονα

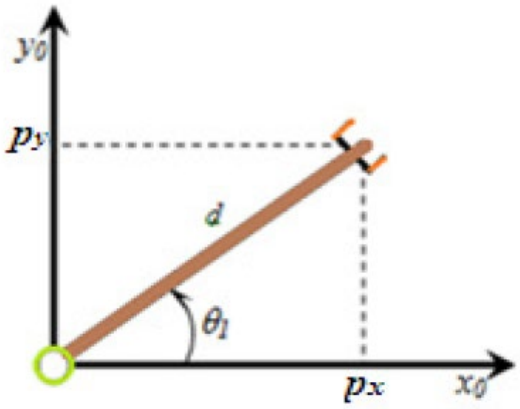
Πολύ σημαντικό σημείο του παρόντος paper είναι η εξής συνάρτηση $\theta_{234} = \theta_2 + \theta_3 + \theta_4$, όπου συμπληρώνεται από την παρακάτω πρόταση: "where θ_{234} *can be calculated based on pitch wrist orientation angle Ø"*[16, page 4]. Η πρόταση αυτή στην παρούσα υλοποίηση αποτέλεσε σημαντικότατη βάση, καθώς μέσα από αυτή λαμβάνεται η πρώτη σταθερά, ώστε να μπορέσουν αργότερα να επιλυθούν όλες οι συναρτήσεις αντίστροφης κινηματικής. Από το σχήμα 9 προκύπτουν όλες οι γεωμετρικές σχέσεις του συστήματος.



Σχήμα 9: Αρθρωτό ρομπότ με 4 συνδέσμους

Παρατηρείται επίσης ότι το παραπάνω σχήμα δεν αποτελεί ούτε κατακόρυφη, αλλά ούτε και πλάγια όψη του ρομποτικού βραχίονα. Είναι μια πολύ αξιόλογη προσπάθεια αναπαράστασης και των δύο σχημάτων μαζί.

Τέλος στο σχήμα 10 φαίνεται η κατακόρυφη όψη του ρομποτικού βραχίονα, η οποία ενέπνευσε και τη κατακόρυφη όψη του συστήματος της παρούσας μελέτης.



Σχήμα 10: Κατακόρυφη όψη ρομποτικού βραχίονα

2.5 A Method of Stereo Vision Matching Based on OpenCV [18]

Το paper του τίτλου αναφέρεται στη στερεοσκοπική όραση που είναι ένας σημαντικός κλάδος της ερευνητικής περιοχής στην υπολογιστική όραση. Μεταξύ των τεχνικών στερεοσκοπικής όρασης, η διόφθαλμη στερεοσκοπική όραση που βασίζεται στην επεξεργασία δυο εικόνων παραμένει το επίκεντρο της έρευνας. Η διόφθαλμη στερεοσκοπική όραση προσομοιώνει άμεσα τον τρόπο με τον οποίο τα ανθρώπινα μάτια παρατηρούν μια σκηνή από δύο διαφορετικές γωνίες. Χρησιμοποιώντας την αρχή του τριγωνισμού, υπολογίζονται οι ανισότητες ενός αριθμού τρισδιάστατων σημείων που αντιστοιχίζονται σε εικονοστοιχεία σε δυο εικόνες. Στη συνέχεια ανακτώνται και οι οπτικές πληροφορίες βάθους. Επίσης, μπορεί να βρεθεί το σχήμα της επιφάνειας του αντικειμένου χρησιμοποιώντας αυτές τις ανισότητες. Σε αυτό το paper, χρησιμοποιείται ένα απλό ζευγάρι καμερών για τη συλλογή των εικόνων. Γίνεται χρήση της OpenCV για τη βαθμονόμηση του αλγορίθμου της στερεοσκοπικής όρασης. Επίσης, εφαρμόζονται γρήγορα και αποτελεσματικά, ο αλγόριθμος στερεοσκοπικής διόρθωσης και στερεοσκοπικής αντιστοίχισης. Τέλος, λαμβάνονται οι πληροφορίες του βάθους του αντικείμενου.

Το στερεοσκοπικό σύστημα όρασης αποτελείται από μια διαδικασία 6 βημάτων.

Το πρώτο βήμα αφορά τη λήψη εικόνας χρησιμοποιώντας ένα ζεύγος καμερών. Αυτές καταγράφουν την ίδια σκηνή την ίδια στιγμή. Η έξοδος αυτού του βήματος είναι στεροσκοπικές εικόνες.

Το δεύτερο βήμα έχει να κάνει με τη βαθμονόμηση της κάμερας. Περιγράφεται η θέση της κάμερας και οι εγγενείς παράμετροί της. Η κύρια ιδέα αυτής της διαδικασίας είναι να διασφαλίσει τη σχέση μεταξύ του τρισδιάστατου σημείου του αντικειμένου στις συντεταγμένες του κόσμου και του δισδιάστατου σημείου στο επίπεδο της εικόνας. Η μέθοδος που χρησιμοποιήθηκε είναι αυτή του ελαχίστου τετραγώνου για τον υπολογισμό του πίνακα μετασχηματισμού που αντιπροσωπεύει την αντιστοίχιση από το τρισδιάστατο σημείο του αντικειμένου στο δισδιάστατου σημείο της εικόνας προκειμένου να δημιουργηθεί ένα αποτελεσματικό μοντέλο.

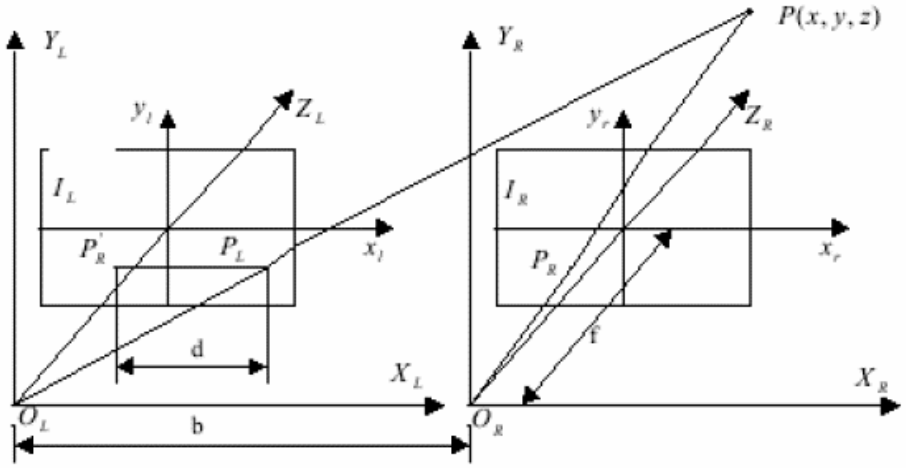
Το επόμενο βήμα αφορά την εξαγωγή χαρακτηριστικών. Ο στόχος της εξαγωγής χαρακτηριστικών είναι να προσδιοριστούν τα σχετικά εικονοστοιχεία τόσο στην αριστερή όσο και στη δεξιά προβολή εικόνας που έχουν αντιστοιχιστεί από το ίδιο τρισδιάστατο σημείο του αντικειμένου στη σκηνή. Επιλέχθηκαν ορισμένα κατάλληλα χαρακτηριστικά εικόνας για την πραγματοποίηση της αντιστοίχισης εικονοστοιχείων μεταξύ της αριστερής προβολής εικόνας στη δεξιά. Το ζεύγος των εικονοστοιχείων μετά την αντιστοίχιση μπορεί να χρησιμοποιηθεί για τον υπολογισμό του βάθους χρησιμοποιώντας τη γνωστή ανισότητα.

Το τέταρτο βήμα έχει να κάνει με τη στερεοσκοπική αντιστοίχιση. Η στερεοσκοπική αντιστοίχιση προσπαθεί να βρει το αντίστοιχο ζεύγος εικονοστοιχείων από δυο εικόνες που αντανακλούν το ίδιο σημείο του αντικειμένου στην τρισδιάστατη σκηνή. Η

στερεοσκοπική αντιστοίχιση είναι η πιο δύσκολη πρόκληση στον τομέα της στερεοσκοπικής όρασης.

Το προτελευταίο βήμα έχει να κάνει με τον υπολογισμό πληροφοριών βάθους. Το βάθος του τρισδιάστατου σημείου του αντικειμένου υπολογίζεται με βάση την αρχή του τριγωνισμού χρησιμοποιώντας το ήδη γνωστό μοντέλο στερεοσκοπικής απεικόνισης της διόφθαλμης κάμερας και την αντίστοιχη ανισότητα.

Το τελικό βήμα αφορά τη συλλογή πληροφοριών και της ακρίβειας όσον αφορά το βάθος για κάθε σημείο της σκηνής. Το Σχήμα 11 απεικονίζει ένα διόφθαλμο στερεοσκοπικό σύστημα.



Σχήμα 11: Διόφθαλμο στερεοσκοπικό σύστημα

Τα συστήματα συντεταγμένων των καμερών είναι κατασκευασμένα αντίστοιχα για τις δυο κάμερες στην αριστερή και τη δεξιά πλευρά. Το P είναι σημείο σε συντεταγμένες του κόσμου και το P προβάλλεται σε δυο επίπεδα εικόνας των καμερών ταυτόχρονα. Τα P1και P2 είναι τα δυο σημεία των προβολών στα δυο επίπεδα. Θυμηθείτε ότι τα επίπεδα εικόνας είναι ευθυγραμμισμένα, άρα οι συντεταγμένες των P1 και P2 θα πρέπει να έχουν το ίδιο y. Η τιμή διαφοράς στο x ονομάζεται ανισότητα και σημειώνεται ως d. Η απόσταση μεταξύ των οπτικών κέντρων των δυο καμερών ονομάζεται μήκος βάσης b. Το b μπορεί να μετρηθεί απευθείας σε συντεταγμένες του φυσικού κόσμου, ενώ οι συντεταγμένες εικονοστοιχείων των P1 και P2 μπορούν να ανακτηθούν από τις εικόνες. Επομένως το d μπορεί να υπολογιστεί. Μετά από τα παραπάνω βήματα, οι συντεταγμένες του P στις συντεταγμένες του κόσμου, μπορεί να υπολογιστεί με βάση την τιμή βάθους που λαμβάνεται χρησιμοποιώντας την ανισότητα d.

Κλείνοντας, οι συγγραφείς του paper καταλήγουν ότι η μέθοδος αντιστοίχισης στερεοσκοπικής όρασης που βασίζεται στην βιβλιοθήκη OpenCV επαληθεύεται από πειραματικές δοκιμές, είναι σταθερή και εύκολη στην εφαρμογή. Χρησιμοποιώντας αυτή τη μέθοδο το σύστημα μπορεί να ολοκληρώσει τον τρισδιάστατο εντοπισμό και άλλες πρακτικές εργασίες με τη χρήση μιας διόφθαλμης στερεοσκοπικής κάμερας.

3 Προτεινόμενο Σύστημα

Για τη δημιουργία του προτεινόμενου ρομποτικού συστήματος χρησιμοποιήθηκαν υλικά χαμηλού κόστους που υπάρχουν διαθέσιμα από πολλούς κατασκευαστές και είναι εύκολη η εύρεσή τους. Δόθηκε έμφαση στην ευκολία σύνδεσης αυτών και στη μελλοντική επεκτασιμότητα του συστήματος επιτρέποντας τη σύνδεση νέων αισθητήρων και, συνεπώς, υποστηρίζοντας την παρακολούθηση των νέων δεδομένων.

3.1 Raspberry Pi 4 Model B

Σχήμα 12: Η βασική πλακέτα Raspberry Pi 4 Model B.

(Εικόνα από προσωπικό αρχείο)

Συγκεκριμένα, επιλέχθηκε το Raspberry Pi 4 4GB Official Desktop Kit, που περιλαμβάνει, εκτός από το Raspberry Pi 4 Model B, ένα ποντίκι και πληκτρολόγιο, καλώδιο micro HDMI σε Standard HDMI, ένα τροφοδοτικό, μια θήκη και μια κάρτα μνήμης 16GB με το λειτουργικό σύστημα NOOBS Raspbian.

3.2 Servo κινητήρες

Η κύρια χρήση ενός servo κινητήρα είναι να περιστρέψει κάτι γύρω από τον άξονά του. Η υλοποίηση περιλαμβάνει στο σύνολό της 12 servo κινητήρες, από 6 σε κάθε έναν από τους δύο βραχίονες. Οι έξι εξ αυτών έχουν ροπή 2.8kg.cm, ενώ οι υπόλοιποι έξι έχουν ροπή 11kg.cm. Όπως εύκολα αντιλαμβάνεται κάποιος, παρόλο που τα νούμερα φαίνονται μεγάλα, δε μπορούν σε καμία περίπτωση να σηκώσουν ολόκληρο το βάρος μιας πετσέτας

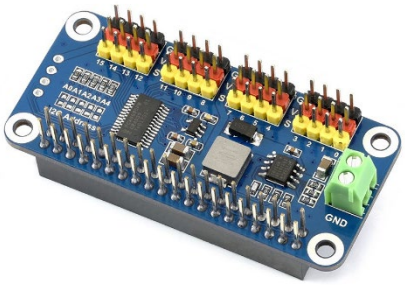
ξενοδοχείου. Ο λόγος είναι ότι η δύναμη που μπορεί να παράγει ο κάθε κινητήρας αναφέρεται στην απόσταση από το κέντρο περιστροφής του.

Σχήμα 13: Servo Motor 2.8kg.cm

Σχήμα 14: Servo Motor 11kg.cm

3.3 Servo Driver HAT

Το Servo Driver HAT, είναι μια πλακέτα που έχει ως στόχο την απροβλημάτιστη σύνδεση περισσότερων servo κινητήρων στο Raspberry Pi. Κουμπώνει κατευθείαν πάνω στο Raspberry Pi και δέχεται μέχρι και 16 servo κινητήρες. Σε περίπτωση που οι κινητήρες χρειαστεί να τραβήξουν περισσότερο ρεύμα, δέχεται σύνδεση 5V. Μάλιστα αν κάποιος θέλει να χρησιμοποιήσει περισσότερους από 16 servo κινητήρες μπορεί να συνδέσει και δεύτερη πλακέτα HAT, πάνω από την αρχική. Η εικόνα του φαίνεται στο Σχήμα 15.



Σχήμα 15: Servo Driver HAT

3.4 Κάμερες

Για την αναγνώριση των λευκών ειδών χρησιμοποιήθηκαν μια απλή κάμερα με σύνδεση USB, κατευθείαν πάνω στο Raspberry Pi και μία ακόμα κάμερα – module Raspberry Pi Camera (8MP,1080p). Η συσκευή Raspberry Pi, διαθέτει μόνο μία θύρα για να δεχθεί κάμερα με καλωδιοταινία. Υπάρχει η δυνατότητα τοποθέτησης περισσότερων αλλά μόνο με τη χρήση κάποιας πρόσθετης πλακέτας (Σχήμα 18). Εφόσον δεν υπήρχε τέτοια πλακέτα, η χρήση μιας USB κάμερας ήταν μονόδρομος.

Σχήμα 16: Κάμερα USB

Σχήμα 17: Raspberry Pi Camera

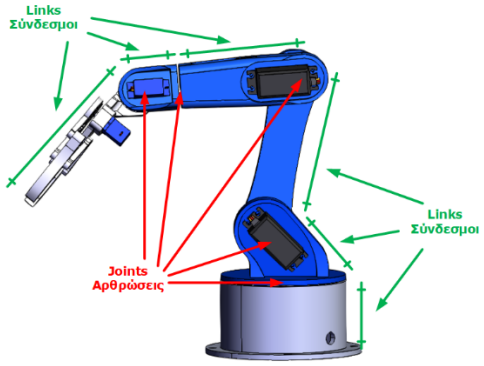
Σχήμα 18: Raspberry Pi multiCamera adapter module

3.5 Βραχίονες

Το μοντέλο βραχίονα που χρησιμοποιήθηκε στη παρούσα μελέτη αφορά ένα 5-DoF βραχίονα (Σχήμα 19). Είναι όμοιος με το βραχίονα που υπάρχει στην παρούσα ηλεκτρονική σελίδα που αναφέρεται στη βιβλιογραφία [15]. Παρόλα αυτά έχουν γίνει ορισμένες διαφοροποιήσεις σε κάποια από τα εκτυπώσιμα τμήματα ώστε να επιτευχθεί ομαλότερη λειτουργία.

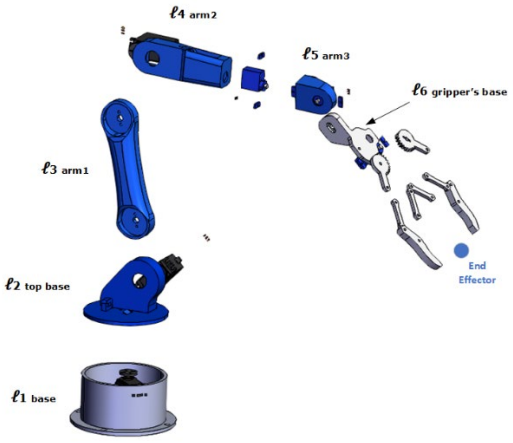
Σχήμα 19: Βραχίονας 5-DoF

Το ακρωνύμιο DoF, μεταφράζεται ως Degrees of Freedom και αναφέρεται στο σύνολο των αρθρώσεων του βραχίονα. Ο κάθε σύνδεσμος ενώνεται με τουλάχιστον μία άρθρωση, η οποία είναι υπεύθυνη για την κίνηση του αμέσως επόμενου συνδέσμου, δεδομένου ότι ο σύνδεσμος που είναι πιο κοντά στη βάση παραμένει σταθερός (Σχήμα 20). Αυτό έχει σαν αποτέλεσμα κατά τη μεταβολή των μοιρών μίας μόνο άρθρωσης να επηρεάζεται η τελική θέση της λαβής (gripper) του ρομποτικού βραχίονα. Όμως καθότι αναφερόμαστε σε τρισδιάστατο χώρο, η οποιαδήποτε μεταβολή επηρεάζει και τους τρεις άξονες.



Σχήμα 20: Joints & Links του βραχίονα

Για την ομαλότερη κατανόηση του σχεδίου, καθώς και για να επιτευχθεί ορθή αναφορά των συνδέσμων - μερών στα επόμενα τμήματα της μελέτης έχει ακολουθηθεί ο επόμενος τρόπος αναφοράς (Σχήμα 21).



Σχήμα 21: Κομμάτια του βραχίονα

Όπως εύκολα παρατηρείται ο ρομποτικός βραχίονας αποτελείται από πολύ λίγα μέρη, τα περισσότερα είναι εκτυπωμένα από 3D – εκτυπωτή, ενώ τα υπόλοιπα αποτελούν οι servo-κινητήρες και οι βίδες. Η συνολική ώρα που χρειάστηκε για να εκτυπωθούν όλα τα

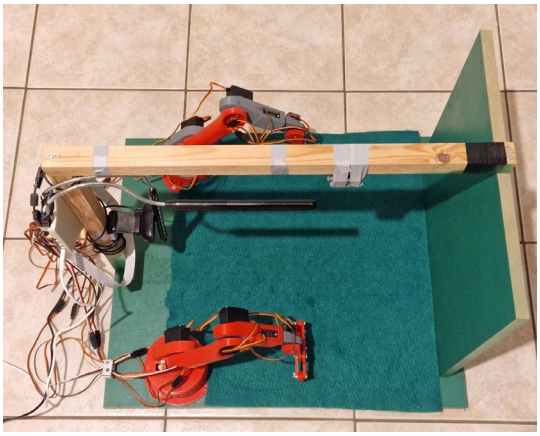
τμήματα για ένα βραχίονα ανέρχεται σε 32 ώρες και το υλικό που χρειάζεται είναι περίπου 350γρ. συμπεριλαμβανομένου και του υλικού που πρέπει να εκτυπωθεί για την υποστήριξη την εκτυπώσιμων αρχείων. Σε όλα τα τμήματα του ρομποτικού βραχίονα χρησιμοποιήθηκαν απλές βίδες, με εξαίρεση το τελευταίο τμήμα που συγκρατεί τη λαβή του, στο οποίο χρειάστηκαν βίδες με παξιμάδια ασφαλείας, ώστε να μη ξεβιδώνουν από τις επαναλαμβανόμενες κινήσεις.

3.6 Κατασκευή

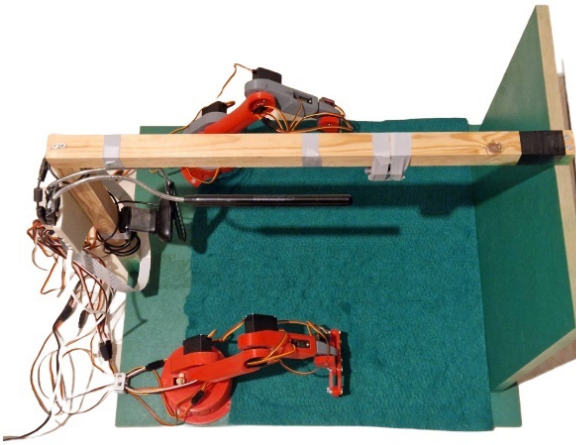
Η κατασκευή του ρομποτικού συστήματος έγινε σταδιακά. Οι αρχικές μελέτες για τις δυνατότητες των βραχιόνων, έγιναν στο πρόγραμμα visio, ώστε να μην σπαταληθούν πόροι υλικών. Έπειτα μετρήθηκε η δυναμική του κάθε βραχίονα αναφορικά με το εύρος στο οποίο μπορεί να φτάσει από τη βάση του και μελετήθηκε η δυναμική του καθενός εκ των δύο ώστε να περιστρέφονται ελεύθερα / απροβλημάτιστα, καθώς η πληθώρα των καλωδίων εγκυμονούσε κινδύνους είτε αποσύνδεσης αυτών από τη πλακέτα HAT, είτε αυτό-τραυματισμού του ίδιου του κινητήρα.

Μάλιστα αφού τα πειραματικά αποτελέσματα ήταν ολοκληρωμένα, συνέχεια είχε η κατασκευή του πλαισίου, το οποίο αποσκοπεί σε πολλαπλές χρήσεις. Αρχικά χρειάζονται δύο επίπεδες επιφάνειες, κάθετα τοποθετημένες μεταξύ τους. Οι δύο αυτές επιφάνειες όμως λόγου του λείου των επιφανειών τους, προκαλούν αντανακλάσεις του φωτός με αποτέλεσμα, να μην είναι κατάλληλες οι εικόνες που λαμβάνονται για επεξεργασία. Για το λόγο αυτό τοποθετήθηκε ύφασμα τσόχας ώστε να αποτρέπει τις αντανακλάσεις.

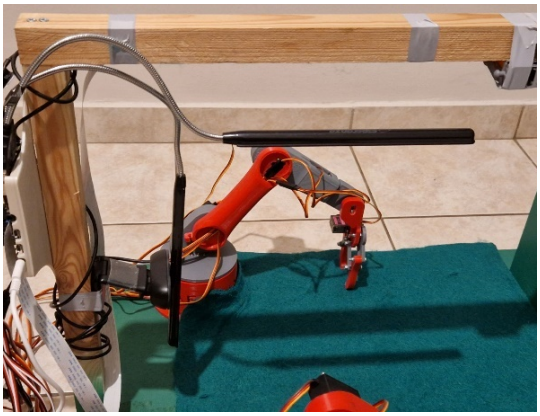
Κρίθηκε επίσης αναγκαίο να τοποθετηθούν δύο ξύλινοι άξονες οι οποίοι θα συγκρατούσαν τις κάμερες, πρόσθετο φωτισμό, καθώς επίσης θα πρόσδιδαν στιβαρότητα σε ολόκληρη τη κατασκευή. Επιλέχθηκε να τοποθετηθεί πρόσθετος φωτισμός από αυτό του περιβάλλοντα χώρου, ώστε το ρομποτικό σύστημα να είναι ικανό να αποδίδει σταθερά κάτω υπό οποιασδήποτε συνθήκες. Στις εικόνες που ακολουθούν φαίνεται με λεπτομέρεια η κατασκευή και στα σχήματα 22 - 27 φαίνεται ο τρόπος εφαρμογής της πλακέτας HAT επάνω στο Raspberry.



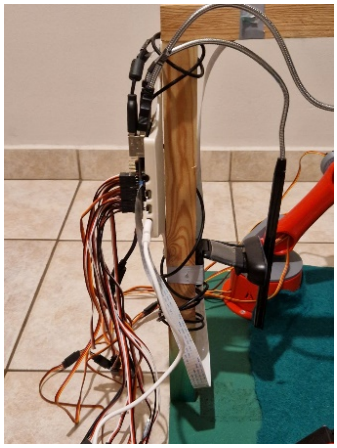
Σχήμα 22: Κατασκευή (1)



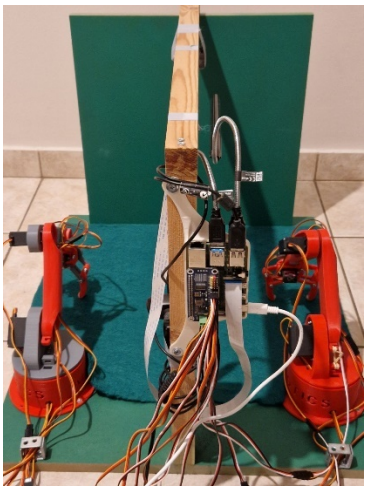
Σχήμα 23: Κατασκευή (2)



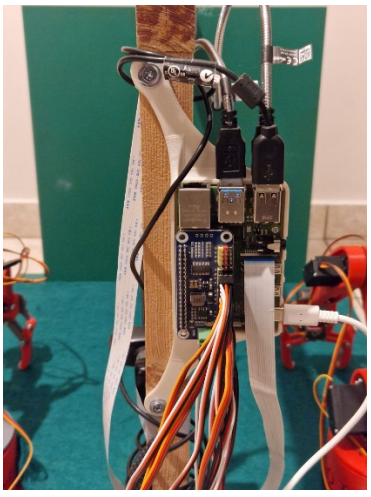
Σχήμα 24: Κατασκευή (3)



Σχήμα 25: Κατασκευή (4)



Σχήμα 26: Κατασκευή (5)

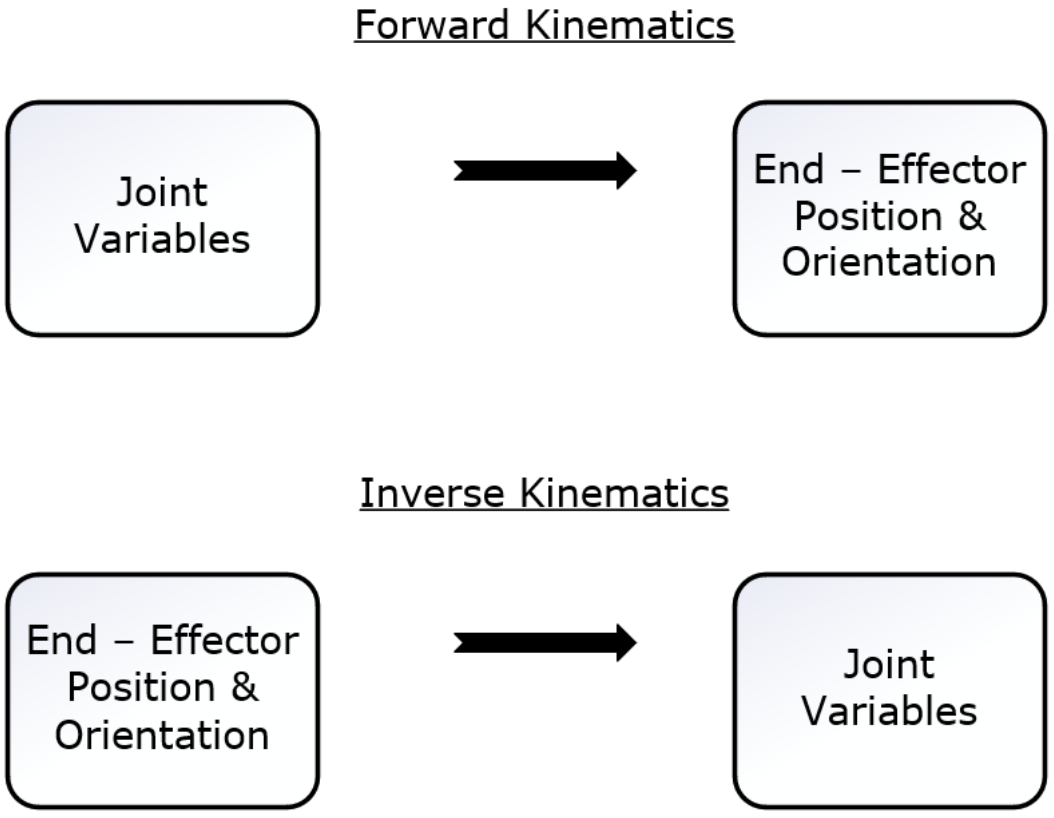


Σχήμα 27: Κατασκευή (6)

4 Κίνηση και Όραση

4.1 Κινηματική

Στην ενότητα αυτή μελετάται η κινηματική του προτεινόμενου ρομποτικού συστήματος. Υπάρχουν δύο είδη κινηματικής, η ευθεία κινηματική (forward kinematics) και η αντίστροφη κινηματική (inverse kinematics). Στην ευθεία κινηματική δίνει ο χρήστης τις τιμές των μεταβλητών των αρθρώσεων και λαμβάνει την τελική θέση της λαβής / ακίδας (end-effector). Αντιθέτως, στην αντίστροφη κινηματική, ο χρήστης ξέροντας που θέλει να στείλει τον end-effector, πρέπει να υπολογίσει τις τιμές των μεταβλητών των αρθρώσεων. Παρακάτω αναφέρονται οι βασικές ορολογίες που απαιτούνται για την κατανόηση των δύο μοντέλων κινηματικής.

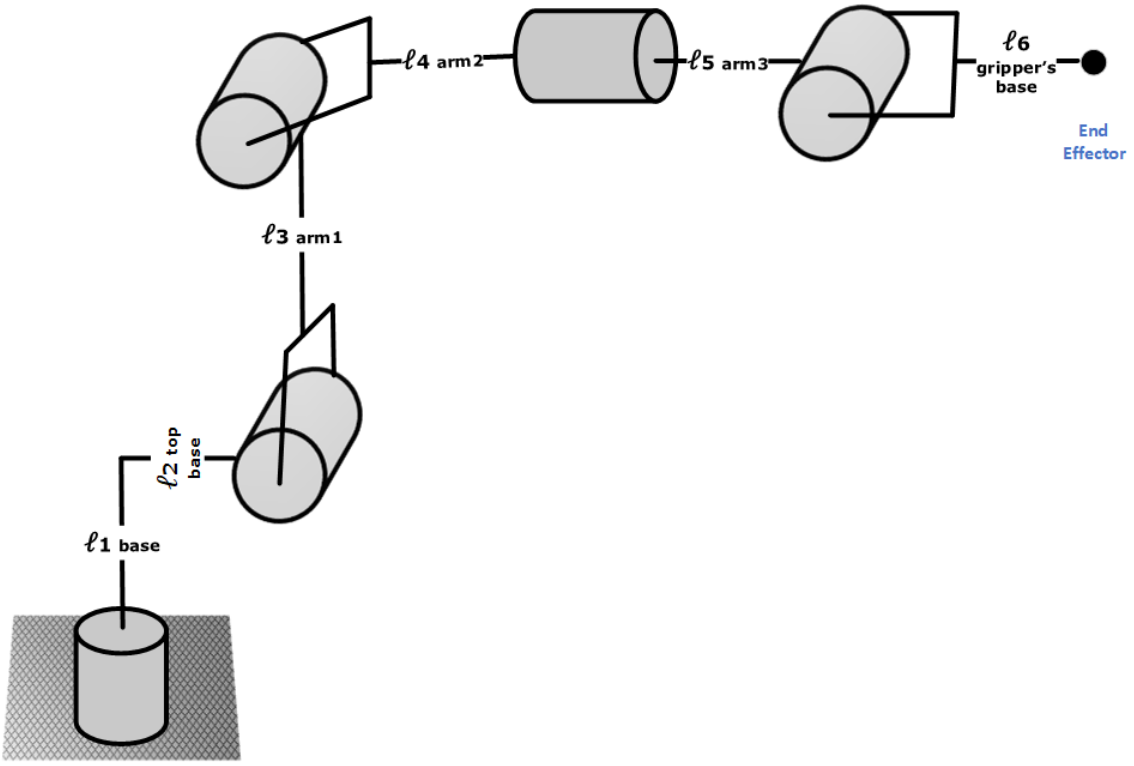


Σχήμα 28: Είδη κινηματικής

4.1.1 Μέθοδος Denavit-Hartenberg

Η μέθοδος Denavit-Hartenberg δίνει μια τυπική μεθοδολογία για τη σύνταξη των κινηματικών εξισώσεων ενός ρομποτικού βραχίονα. Η εφαρμογή της μεθόδου ολοκληρώνεται σε 4 βήματα.

Πριν όμως από το πρώτο βήμα της μεθόδου Denavit-Hartenberg, πρέπει να δημιουργηθεί το κινηματικό διάγραμμα. Ένα κινηματικό διάγραμμα είναι ένα είδος σχεδίου που χρησιμοποιείται στη ρομποτική για να βρεθούν οι εξισώσεις που χρειάζονται για τον χειρισμό ενός ρομπότ. Στην ουσία ένα κινηματικό διάγραμμα περιγράφει σχηματικά το ρομπότ που έχει κατασκευάσει και θέλει να χρησιμοποιήσει ο εκάστοτε χρήστης. Το κινηματικό διάγραμμα από το ρομπότ της παρούσας μελέτης παρουσιάζεται στο Σχήμα 29, και με βάση αυτό θα εφαρμοστεί το πρώτο βήμα της μεθόδου Denavit-Hartenberg.



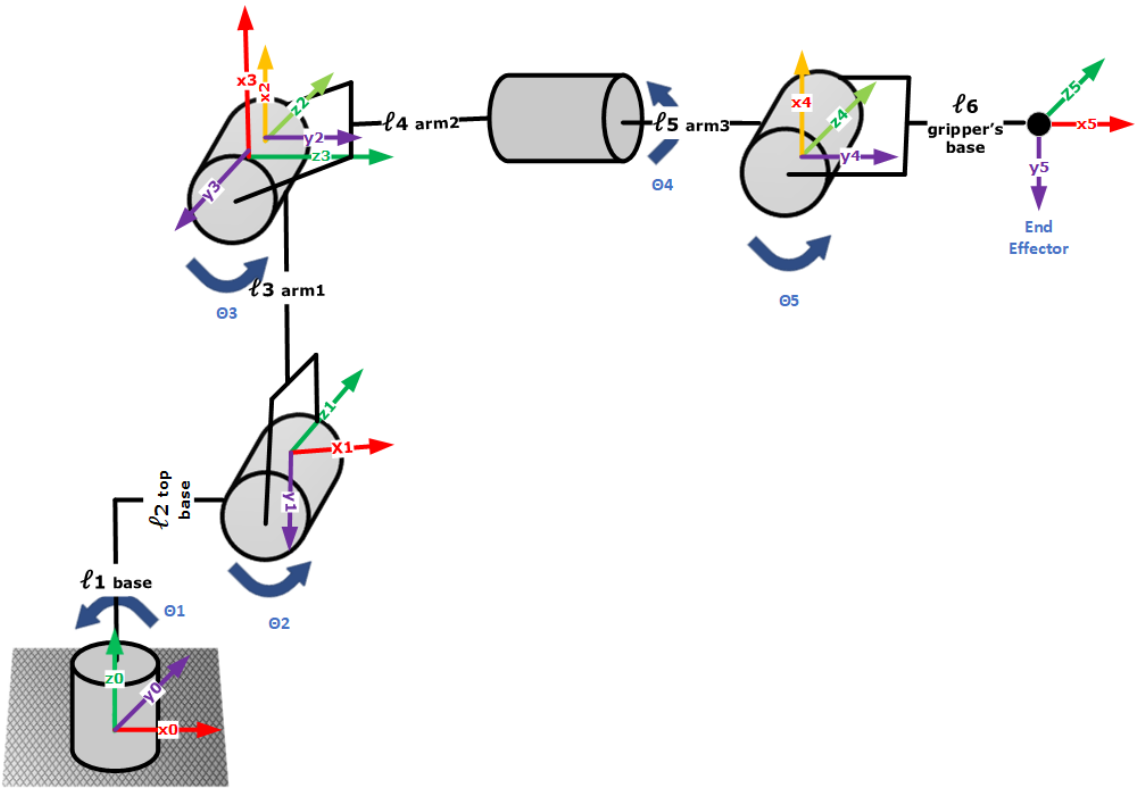
Σχήμα 29: Kinematic diagram

Το πρώτο βήμα αναφέρει το σχεδιασμό πλαισίων σύμφωνα με τους 4 κανόνες των Denavit-Hartenberg.

- Ο πρώτος κανόνας για το σχεδιασμό των πλαισίων, αναφέρει ότι ο άξονας Z πρέπει να είναι ο άξονας περιστροφής της άρθρωσης.
- Ο δεύτερος κανόνας αφορά τον άξονα X, και ότι θα πρέπει να είναι κάθετος με τον άξονα Z του προηγούμενο πλαισίου.
- Ο τρίτος κανόνας αναφέρει ότι ο άξονας X πρέπει να τέμνει τον άξονα Z του προηγούμενο πλαισίου.

- Ο τελευταίος κανόνας αφορά το σχεδιασμό του άξονα Y, ώστε όλα τα πλαίσια να ακολουθούνε τον κανόνα του δεξιού χεριού. Ο αντίχειρας δείχνει τον άξονα Z, ο δείκτης τον άξονα X και η παλάμη τον άξονα Y.

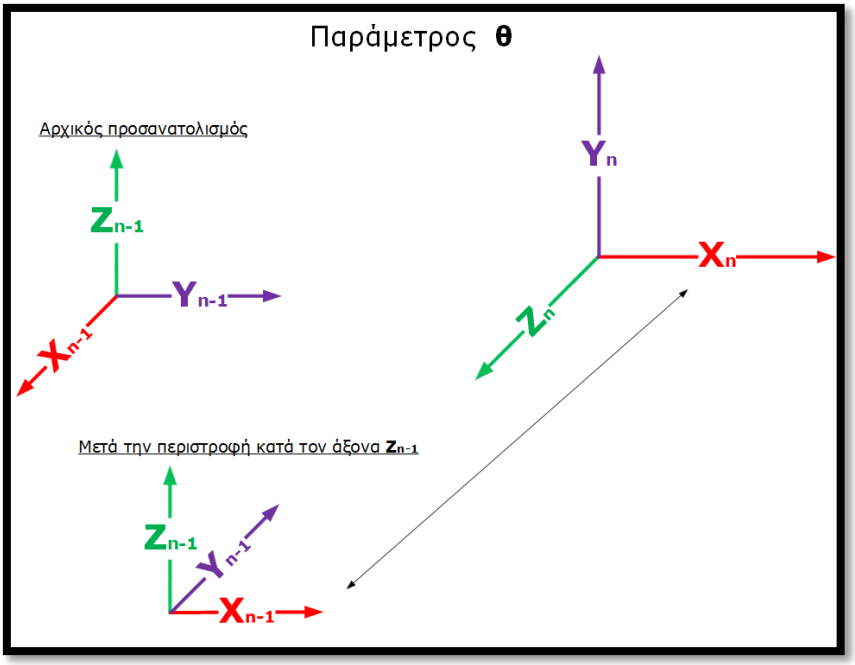
Εφαρμόζοντάς τους παραπάνω κανόνες θα προκύψει το κινηματικό διάγραμμα του Σχήματος 30.



Σχήμα 30: Kinematic diagram

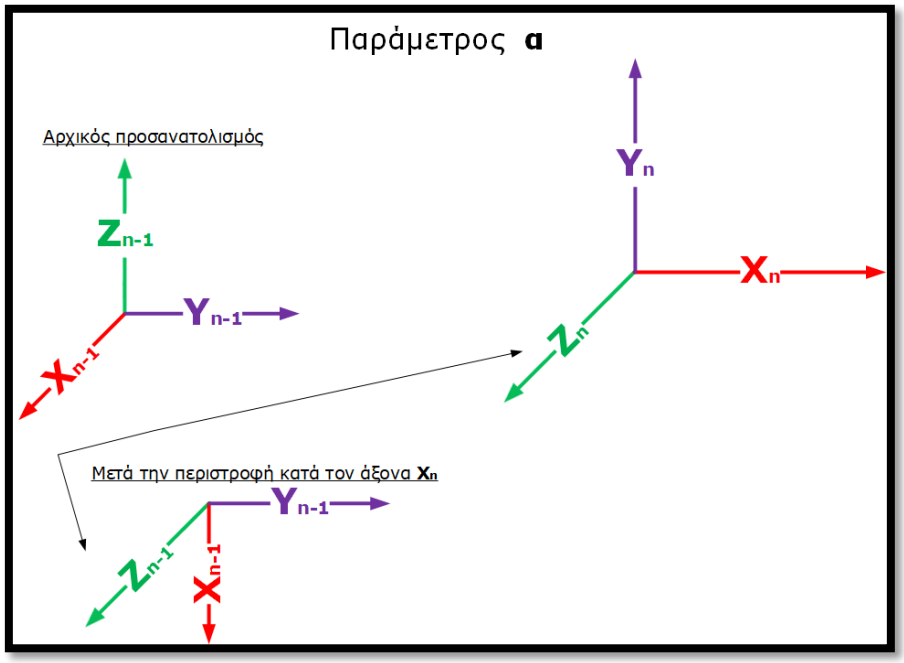
Το δεύτερο βήμα της μεθόδου Denavit-Hartenberg αναφέρεται στη δημιουργία του πίνακα παραμέτρων. Στον πίνακα παραμέτρων θα γίνει καταγραφή όλων των σχέσεων μετατόπισης (displacement) και περιστροφής (rotation) μεταξύ κάθε ζεύγους πλαισίων. Είναι ένας πίνακας που έχει μια γραμμή λιγότερη από τα πλαίσια του κινηματικού διαγράμματος, επειδή έχει μια γραμμή για κάθε ζεύγος από πλαίσια. Επίσης, ο πίνακας αποτελείται από τέσσερεις στήλες, μια για κάθε παράμετρο. Αυτές οι παράμετροι περιλαμβάνουν ότι χρειάζεται για να εκφραστεί η μετατόπιση και η περιστροφή ανάμεσα σε δυο πλαίσια. Δυο παράμετροι καταγράφουν περιστροφή και δυο μετατόπιση. Η ονομασία των παραμέτρων είναι για τις παραμέτρους περιστροφής θ και α, και αντίστοιχα τις παραμέτρους μετατόπισης r και d. Στις επόμενες παραγράφους αναλύεται τι περιγράφει κάθε παράμετρος.

Η πρώτη παράμετρος που αναλύεται είναι η θ, που περιγράφει το κατά πόσο πρέπει να περιστραφεί το πλαίσιο n-1 (προηγούμενο πλαίσιο) γύρω από τον άξονα Z_{n-1}, ώστε ο άξονας X_{n-1} να ταιριάξει με τον άξονα X_n (Σχήμα 31). Επίσης, πρέπει να συμπεριληφθεί σε αυτή την παράμετρο και όποια περιστροφή της άρθρωσης.



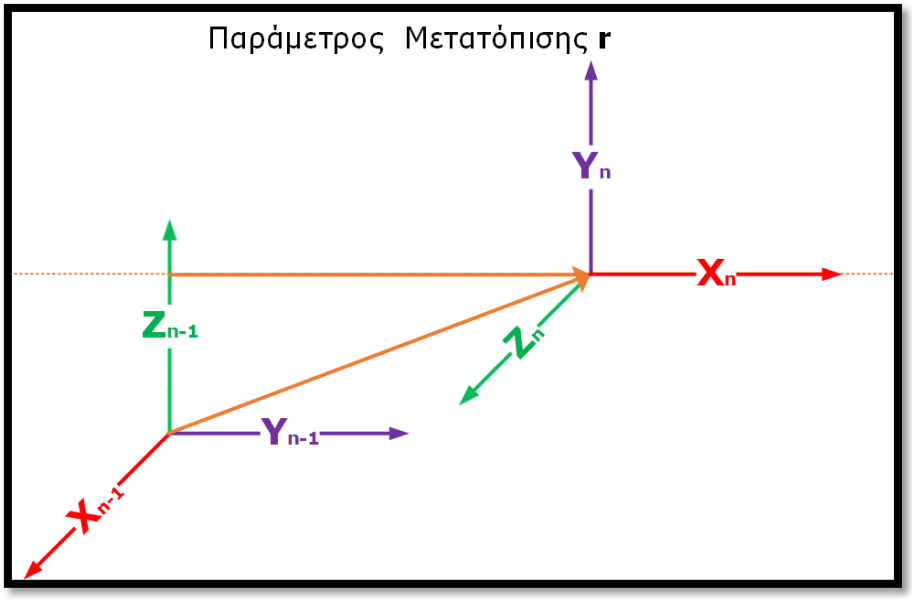
Σχήμα 31: Παράμετρος περιστροφής **θ**

Η δεύτερη παράμετρος, η α, είναι και αυτή μια παράμετρος περιστροφής. Η α περιγράφει το πόσο πρέπει να περιστραφεί το πλαίσιο n-1 (προηγούμενο πλαίσιο) γύρω από τον άξονα X_n, ώστε ο άξονας Z_{n-1} να ταιριάξει με τον άξονα Z_n (Σχήμα 32).



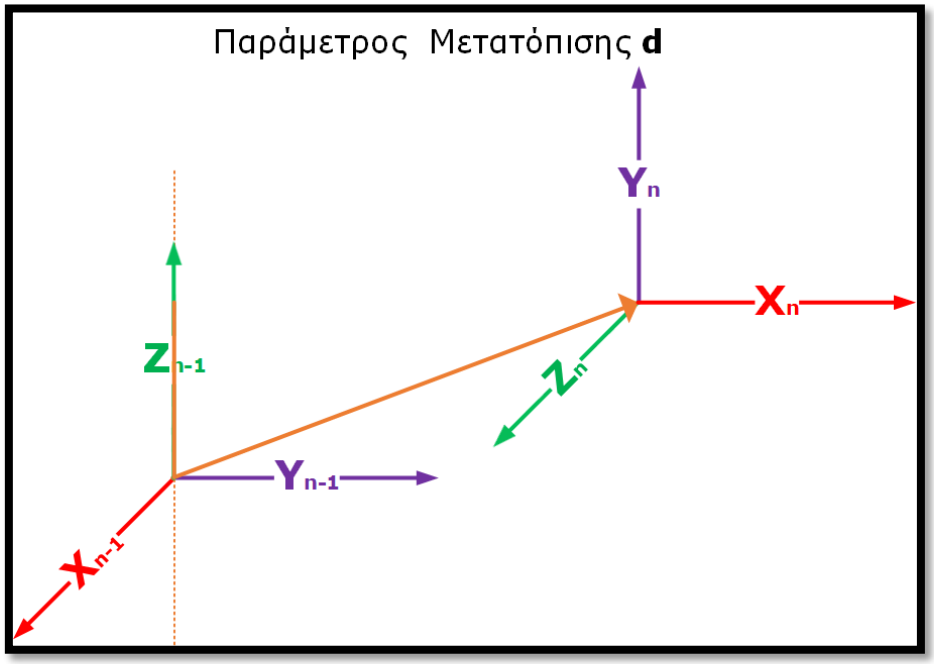
Σχήμα 32: Παράμετρος περιστροφής **α**

Η επόμενη παράμετρος, η r, είναι η πρώτη από τις δυο παραμέτρους μετατόπισης. Η r ορίζεται ως η μετατόπιση από το κέντρο του πλαισίου n-1 στο πλαίσιο n, μετρώντας μόνο κατά την κατεύθυνση του άξονα X_n.



Σχήμα 33: Παράμετρος μετατόπισης **r**

Η επόμενη και τελευταία παράμετρος είναι η *d*. Η *d* ορίζεται ως η μετατόπιση από το κέντρο του πλαισίου *n-1* στο πλαίσιο *n*, μετρώντας μόνο κατά την κατεύθυνση του άξονα Z_{n-1}.



Σχήμα 34: Παράμετρος μετατόπισης **d**

Συνεπώς, ο πίνακας παραμέτρων για το ρομποτικό βραχίονα της μελέτης απεικονίζεται στον παρακάτω πίνακα.

Πίνακας 1: Πίνακας παραμέτρων

Frames	θ	α	r	d
1	θ_1	-90	ℓ_2	ℓ_1
2	-90 + θ_2	0	ℓ_3	0
3	θ_3	-90	0	0
4	θ_4	90	0	$\ell_4 + \ell_5$
5	90 + θ_5	0	ℓ_6	0

Το τρίτο βήμα της μεθόδου Denavit-Hartenberg αφορά την εισαγωγή των τιμών στον πίνακα μετασχηματισμού (homogeneous transformation matrix) (Πίνακας 2).

Πίνακας 2: Πίνακας HTM

$$H_n^{n-1} = \begin{bmatrix} \cos(\Theta_n) & -\sin(\Theta_n)\cos(a_n) & \sin(\Theta_n)\sin(a_n) & r_n\cos(\Theta_n) \\ \sin(\Theta_n) & \cos(\Theta_n)\cos(a_n) & -\cos(\Theta_n)sin(a_n) & r_n\sin(\Theta_n) \\ 0 & \sin(a_n) & \cos(a_n) & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Οι πίνακες που θα προκύψουν παρουσιάζονται παρακάτω:

$$H_1^0 = \begin{bmatrix} \cos(\Theta_1) & 0 & -\sin(\Theta_1) & \ell_2\cos(\Theta_1) \\ \sin(\Theta_1) & 0 & \cos(\Theta_1) & \ell_2\sin(\Theta_1) \\ 0 & -1 & 0 & L1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_2^1 = \begin{bmatrix} \cos(\Theta_2 - 90) & -\sin(\Theta_2 - 90) & 0 & \ell_3\cos(\Theta_2 - 90) \\ sin(\Theta_2 - 90) & \cos(\Theta_2 - 90) & 0 & \ell_3\sin(\Theta_2 - 90) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_3^2 = \begin{bmatrix} \cos(\Theta_3) & 0 & -\sin(\Theta_3) & 0 \\ \sin(\Theta_3) & 0 & \cos(\Theta_3) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_4^3 = \begin{bmatrix} \cos(\Theta_4) & 0 & \sin(\Theta_4) & 0 \\ \sin(\Theta_4) & 0 & -\cos(\Theta_4) & 0 \\ 0 & 1 & 0 & \ell_4 + \ell_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_5^4 = \begin{bmatrix} \cos(\Theta_5 + 90) & -\sin(\Theta_5 + 90) & 0 & \ell_6 \cos(\Theta_5 + 90) \\ sin(\Theta_5 + 90) & \cos(\Theta_5 + 90) & 0 & \ell_6 \sin(\Theta_5 + 90) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Το τέταρτο και τελευταίο βήμα της μεθόδου Denavit-Hartenberg, αφορά τον πολλαπλασιασμό των πινάκων του προέκυψαν από το τρίτο βήμα, ώστε να βρεθεί η συσχέτιση του πρώτου πλαισίου με το τελευταίο.

$$H_5^0 = H_1^0 H_2^1 H_3^2 H_4^3 H_5^4$$

4.1.2 Ευθεία κινηματική

Στη ρομποτική κίνηση, η ευθεία κινηματική αναφέρεται στη χρήση εξισώσεων για τον υπολογισμό της θέσης του end-effector για συγκεκριμένες τιμές των παραμέτρων των αρθρώσεων. Συνεπώς, για να υπολογιστεί η θέση του end-effector του ρομποτικού βραχίονα, το μόνο που χρειάζεται είναι να χρησιμοποιηθεί ο πίνακας μετασχηματισμού H_5^0 και να δοθούν τιμές στις παραμέτρους. Τα $\ell_1, \ell_2, \ell_3, \ell_4, \ell_5$ και ℓ_6 είναι τα μήκη των μερών του ρομποτικού βραχίονα. Για παράδειγμα, αν για τιμές παραμέτρων δοθεί θ_1=50°, θ_2=20°, θ_3=10°, θ_4=0° και θ_5=0°, το αποτέλεσμα που θα προκύψει απεικονίζεται στο Σχήμα 35.

```
[[ 0.5566704  -0.3213938  -0.76604444 17.28382053]
 [ 0.66341395 -0.38302222  0.64278761 20.59805521]
 [-0.5        -0.8660254   0.          9.27631145]
 [ 0.          0.          0.          1.        ]]
```

Σχήμα 35: Παράδειγμα τιμών ευθείας κινηματικής

Από το παραπάνω σχήμα μπορεί να εξαχθεί η πληροφορία για τη θέση του end-effector, που θα βρίσκεται στο X=17.28, Y=20.59 και Z=9.27.

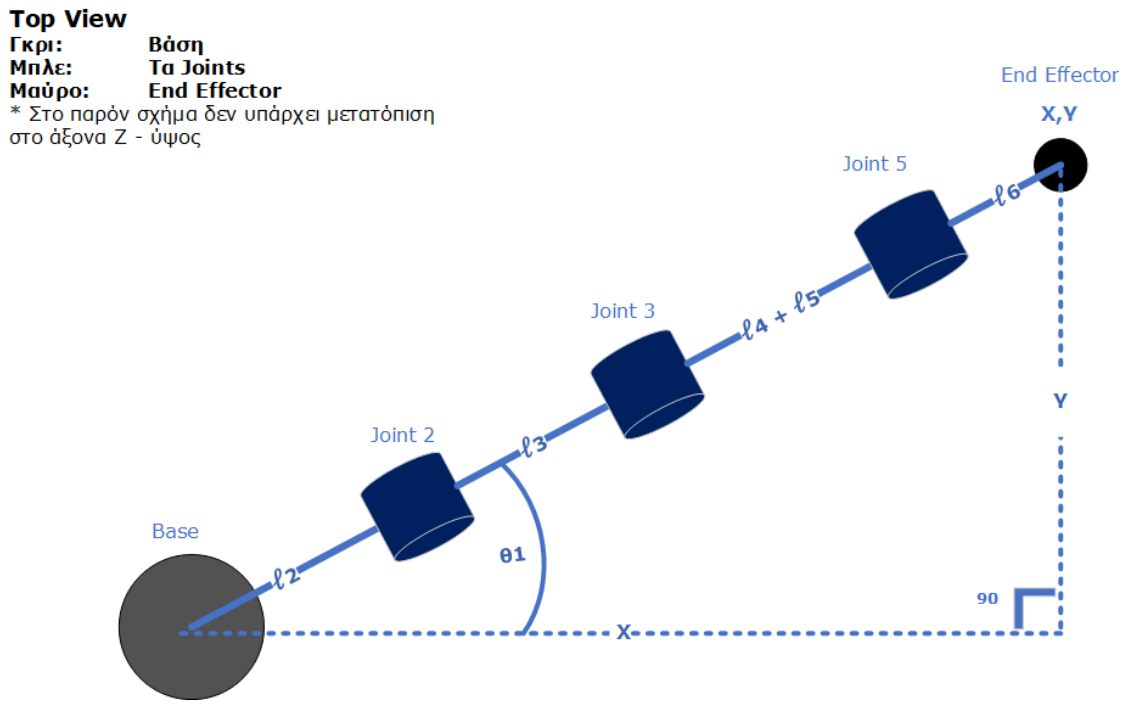
Στην περίπτωση της εργασία, η ευθεία κινηματική χρησιμοποιήθηκε για την επαλήθευση των εξισώσεων της αντίστροφης κινηματικής. Η εύρεση των εξισώσεων της ευθείας κινηματικής, έχει μια ακολουθία από βήματα, που ακολουθώντας τα ο χρήστης καταλήγει στη λύση του προβλήματος. Αυτό όμως δεν ισχύει για τη λύση των εξισώσεων της αντίστροφης κινηματικής.

4.1.3 Αντίστροφη κινηματική

Σε αυτή την ενότητα μελετάται ένα από τα πιο δύσκολα κομμάτια αυτής της εργασίας, η δημιουργία των εξισώσεων της αντίστροφης κινηματικής, που δοθέντος της επιθυμητής θέσης του end-effector θα επιστρέφουν τις τιμές των παραμέτρων των αρθρώσεων, με σκοπό να μετακινηθεί ο end-effector σε αυτές τις συντεταγμένες.

Υπάρχουν διάφοροι τρόποι για τη δημιουργία εξισώσεων αντίστροφης κινηματικής [19]. Για την παρούσα εργασία επιλέχθηκε η γεωμετρική προσέγγιση [21]. Η γεωμετρική προσέγγιση απαιτεί τη δημιουργία, εκτός του κινηματικού διαγράμματος, επιπλέον όψεων του ρομποτικού βραχίονα. Σκοπός των εξισώσεων είναι η εύρεση των θ_1, θ_2, θ_3, θ_4 και θ_5. Η θ_4 δεν επηρεάζει τη θέση του end-effector στην υλοποίηση του προτεινόμενου ρομποτικού συστήματος.

Ξεκινώντας από την πάνω όψη (top view), λόγω της απλότητας της. Αυτή απεικονίζεται στο Σχήμα 36.

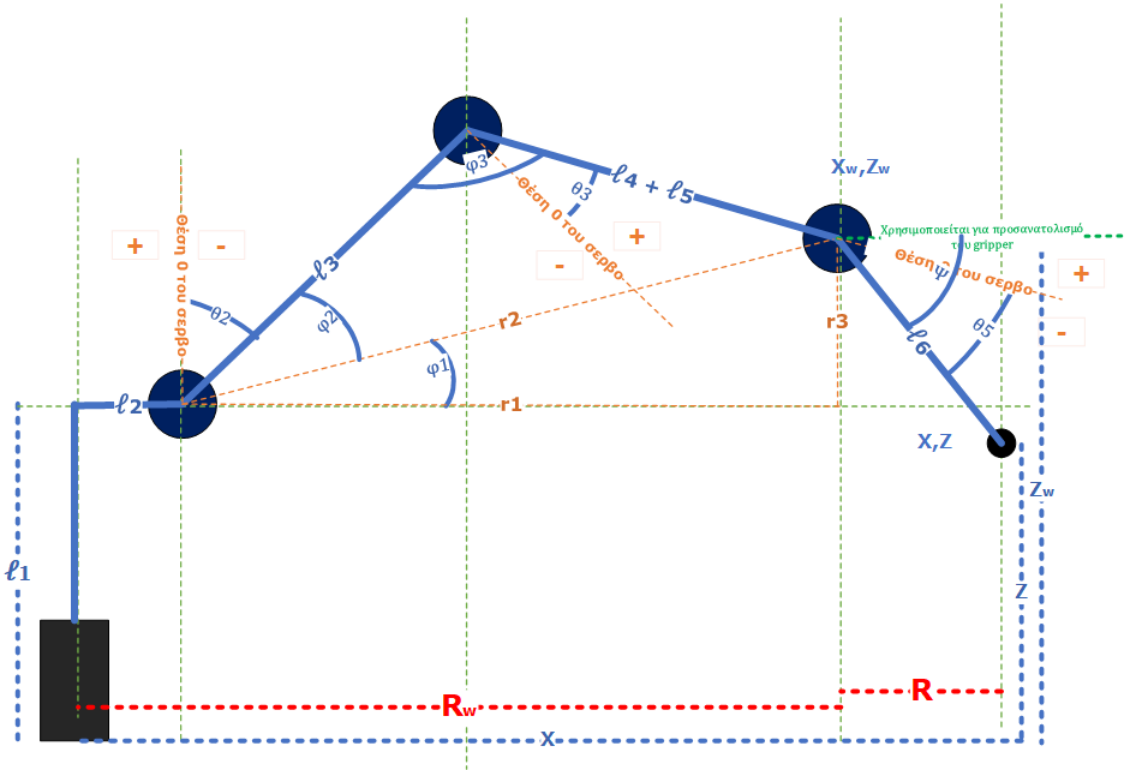


Σχήμα 36: top view

Χρησιμοποιώντας το σχήμα της πάνω όψης, παρατηρείται ότι μπορεί να υπολογιστεί η θ_1 γεωμετρικά, καταλήγοντας στον παρακάτω τύπο:

$$\theta_1 = \tan^{-1}\left(\frac{Y}{X}\right) \quad (1)$$

Στη συνέχεια αναλύεται η πλάγια όψη του ρομποτικού βραχίονα, η οποία απεικονίζεται στο Σχήμα 37.



Σχήμα 37: side view

Όπως παρατηρείται, εδώ για να φτάσει κάποιος στις λύσεις των θ_2, θ_3 και θ_5, θα χρειαστεί βοηθητικές εξισώσεις οι οποίες θα προκύψουν γεωμετρικά από το σχήμα. Λόγω των τριών αρθρώσεων, για να είναι εφικτή η λύση του προβλήματος γεωμετρικά, χρησιμοποιείται η γωνία ψ, η οποία θα είναι γνωστή και θα δείχνει τον προσανατολισμό του end-effector. Έχοντας σαν δεδομένη τη γωνία ψ, μπορεί να υπολογιστούν οι συντεταγμένες του σημείου w. Υπολογίζοντας αυτές τις συντεταγμένες, καταλήγουμε στη λύση των θ_2 και θ_3.

Πρώτα υπολογίζεται το R και στη συνέχεια τα x_w, y_w και z_w, όπως φαίνεται στις εξισώσεις 2, 3, 4 και 5.

$$R = l6 \cdot \cos\psi \quad (2)$$

$$Xw = X - R \cdot \cos\theta 1 \quad (3)$$

$$Yw = Y - R \cdot \sin\theta 1 \quad (4)$$

$$Zw = Z + l6 \cdot \sin\psi \quad (5)$$

Στη συνέχεια υπολογίζονται κάποιες από τις βοηθητικές εξισώσεις, 6, 7 ,8 ,9 ,10 και 11.

$$Rw = \sqrt{Xw^2 + Yw^2} \quad (6)$$

$$r1 = Rw - l2 \quad (7)$$

$$r3 = |Zw - l1| \quad (8)$$

$$r2 = \sqrt{r1^2 + r3^2} \quad (9)$$

$$\varphi 1 = cos^{-1}\left(\frac{r1^2 + r2^2 - r3^2}{2 \cdot r1 \cdot r2}\right) (10)$$

$$\varphi 2 = cos^{-1}\left(\frac{r2^2 + l3^2 - (l4 + l5)^2}{2 \cdot r2 \cdot l3}\right) (11)$$

Πλέον, υπάρχει ότι χρειάζεται για τον υπολογισμό της θ_2, εξίσωση 12.

$$\theta 2 = -((\varphi 1 + \varphi 2) - \frac{\Pi}{2}) (12)$$

Η επόμενη εξίσωση που υπολογίστηκε, η 13, θα χρησιμεύσει στην εύρεση της θ_3, εξίσωση 14.

$$\varphi 3 = cos^{-1}\left(\frac{l3^2 + (l4 + l5)^2 - r2^2}{2 \cdot l3 \cdot (l4 + l5)}\right) (13)$$

$$\theta 3 = -(\varphi 3 - \frac{\Pi}{2}) (14)$$

Όπως προαναφέρθηκε, η θ_4 δεν επηρεάζει τη θέση του end-effector.

$$\theta 4 = 0 (15)$$

Τέλος, η εύρεση της θ_5 προκύπτει από την εξίσωση 16.

$$\theta_5 = \psi - \theta 2 - \theta 3 (16)$$

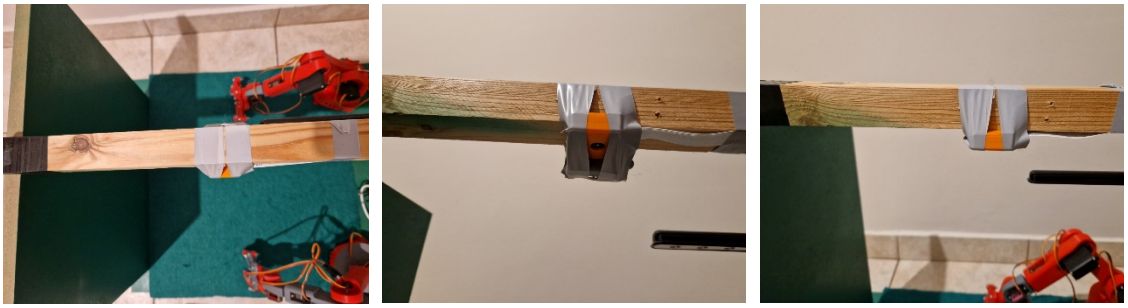
Έχοντας συνεπώς βρει όλες τις εξισώσεις με την βοήθεια των διαγραμμάτων, μένει να ερευνηθεί η όραση του συστήματος.

4.2 Υπολογιστική όραση

Στις προηγούμενες ενότητες μελετήθηκε η μετακίνηση του ρομποτικού βραχίονα σε ένα σημείο, έχοντας τις συντεταγμένες του. Αυτές τις συντεταγμένες, το ρομποτικό σύστημα θα πρέπει να τις αναγνωρίζει. Μια από τις δυο κάμερες του ρομποτικού συστήματος, το τροφοδοτεί με εικόνες, οι οποίες στη συνέχεια επεξεργάζονται κατάλληλα, ώστε να γίνει η εξαγωγή των συντεταγμένων.

4.2.1 Μετατροπή εικονοστοιχείων σε εκατοστά

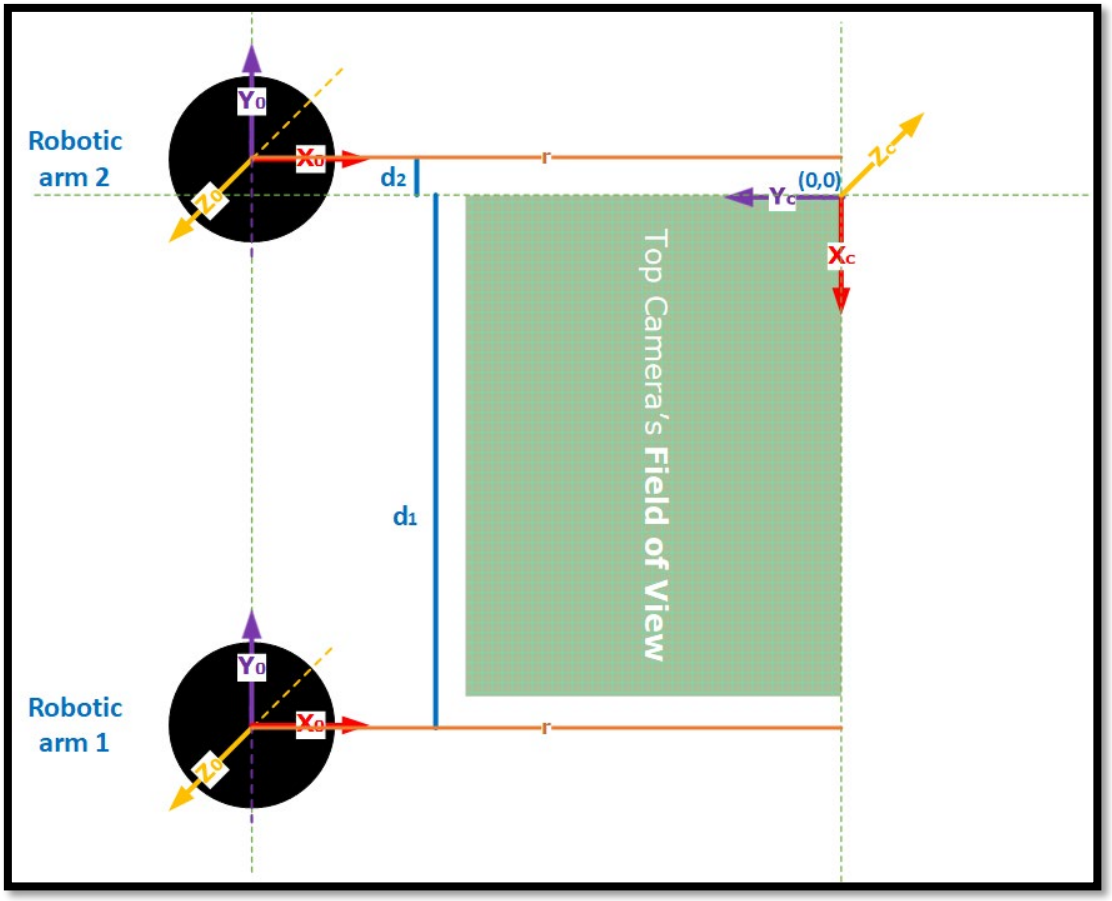
Χρησιμοποιώντας την κάμερα που έχει τοποθετηθεί στο πάνω μέρος της κατασκευής (Σχήμα 38), λαμβάνεται μία εικόνα. Ξεκινώντας, θα χρειαστεί να γίνει η μετατροπή των εικονοστοιχείων (pixel) σε εκατοστά. Για να γίνει η μετατροπή χρειάζεται ένας χάρακας ώστε να μετρηθεί το οπτικό πεδίο της κάμερας σε εκατοστά. Το μέγεθος της εικόνας σε εικονοστοιχεία είναι 640x480. Συνεπώς, η μετατροπή είναι μια απλή διαίρεση των εκατοστών με τα εικονοστοιχεία. Επειδή η κάμερα είναι παράλληλα τοποθετημένη με το επίπεδο / έδαφος και τα εικονοστοιχεία είναι τετράγωνα, δε χρειάζεται να μετρηθούν και κάθετα και οριζόντια το οπτικό πεδίο της κάμερας.



Σχήμα 38: Εικόνες κατασκευής

4.2.2 Μετασχηματισμός συντεταγμένων

Για καλύτερη κατανόηση, πως βλέπει η κάμερα αξίζει να σημειωθεί ότι το σημείο (0,0) της κάμερας είναι το πάνω αριστερά, όπως απεικονίζεται στο Σχήμα 39. Όμως, αν τροφοδοτηθεί το σύστημα με βάση το σημείο (0,0) της κάμερας, τότε οι ρομποτικοί βραχίονες θα κατευθυνθούν σε λάθος σημείο. Η λύση είναι η μετατροπή της θέσης ενός αντικειμένου από τις συντεταγμένες του πλαισίου της κάμερας στις συντεταγμένες του πλαισίου των ρομποτικών βραχιόνων. Για να γίνει αυτό θα χρειαστεί να βρεθεί ο πίνακας μετασχηματισμού από το πλαίσιο της βάσης του κάθε βραχίονα στο πλαίσιο της κάμερας και στη συνέχεια να πολλαπλασιαστεί με τη θέση που χρειάζεται να αλληλοεπιδράσει ο κάθε βραχίονας.



Σχήμα 39: Top camera's field of view

Το πρώτο βήμα για τη δημιουργία του πίνακα μετασχηματισμού είναι να ταιριάξει ο άξονας Z της κάμερας με τον άξονα Z των βραχιόνων. Για να γίνει αυτό θα περιστραφεί το πλαίσιο της βάσης των βραχιόνων κατά τον άξονα X κατά 180º. Ο πίνακας περιστροφής είναι ο παρακάτω. Ο πίνακας αυτός είναι όμοιος και για τους δυο βραχίονες.

$$R_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(180) & -\sin(180) \\ 0 & \sin(180) & \cos(180) \end{bmatrix}$$

Στο επόμενο βήμα περιστρέφεται το πλαίσιο των βραχιόνων κατά τον άξονα Z κατά 89º. Με αυτό το τελευταίο βήμα θα ταιριάξει το πλαίσιο των βραχιόνων με το πλαίσιο της κάμερας. Ο πίνακας περιστροφής είναι ο παρακάτω. Επίσης, ο πίνακας είναι όμοιος και για τους δυο βραχίονες.

$$R_Z = \begin{bmatrix} \cos(89) & -\sin(89) & 0 \\ \sin(89) & \cos(89) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Έχοντας βρει τους πίνακες περιστροφής, ακολουθεί ο πολλαπλασιασμό τους.

$$R_c^0 = R_X R_Z$$

Στη συνέχεια πρέπει να βρεθεί η μετατόπιση από κάθε βραχίονα σε σχέση με το πλαίσιο της κάμερας. Για το δεξιό βραχίονα η μετατόπιση είναι η παρακάτω:

$$dR_c^0 = [42.5 \quad 43.0 \quad 0.0]$$

Και για τον αριστερό βραχίονα είναι η παρακάτω:

$$dL_c^0 = [40.0 \quad 4.0 \quad 0.0]$$

Οι πίνακες μετασχηματισμού παρουσιάζονται παρακάτω:

$$Hr_c^0 = \begin{bmatrix} R_c^0[1,1] & R_c^0[1,2] & R_c^0[1,3] & dR_c^0[1] \\ R_c^0[2,1] & R_c^0[2,2] & R_c^0[2,3] & dR_c^0[2] \\ R_c^0[3,1] & R_c^0[3,2] & R_c^0[3,3] & dR_c^0[3] \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Hl_c^0 = \begin{bmatrix} R_c^0[1,1] & R_c^0[1,2] & R_c^0[1,3] & dL_c^0[1] \\ R_c^0[2,1] & R_c^0[2,2] & R_c^0[2,3] & dL_c^0[2] \\ R_c^0[3,1] & R_c^0[3,2] & R_c^0[3,3] & dL_c^0[3] \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Τέλος, πολλαπλασιάζεται ο πίνακας μετασχηματισμού με το σημείο που θέλει να αλληλεπιδράσει ο κάθε ρομποτικός βραχίονας.

Έστω οι συντεταγμένες του σημείου ενδιαφέροντος,

$$PC = [X_{Location}, Y_{Location}, Z_{Location}, 1]$$

Άρα, η μετατροπή θα είναι της μορφής,

$$PO = H_c^0 \cdot PC$$

Πλέον, το ρομποτικό σύστημα είναι σε θέση να μετακινήσει τους end-effectors των βραχιόνων στις συντεταγμένες που θα τροφοδοτηθεί.

4.2.3 Αλγόριθμος ανίχνευσης γωνιών Harris

Το επόμενο μέρος του αλγορίθμου όρασης αφορά την εύρεση των συντεταγμένων των γωνιών ενός λευκού είδους, καθώς εκεί θα στοχεύσουν οι ρομποτικοί βραχίονες. Πρώτα, θα χρειαστεί να απομονώσουμε το λευκό είδος από τη φωτογραφία που θα πάρει το σύστημα με την κάμερα. Αυτό θα γίνει για να εφαρμοστεί με μεγαλύτερη επιτυχία ο αλγόριθμος εύρεσης των γωνιών. Συνεπώς, εφαρμόζεται μια μάσκα που ανιχνεύει το

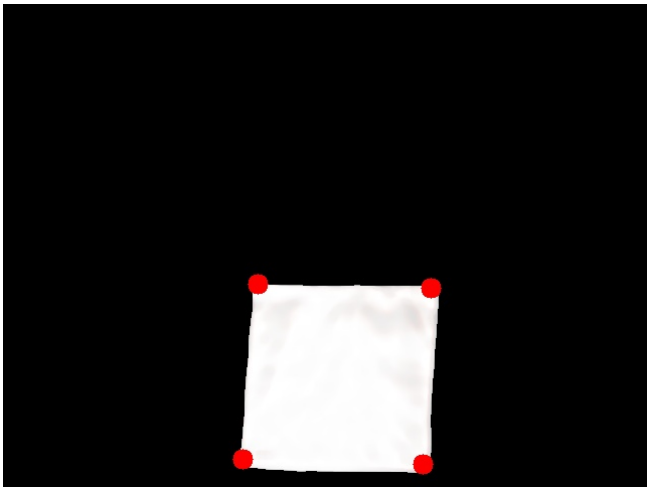
λευκό χρώμα. Αφού έχει εφαρμοστεί η μάσκα θα εφαρμοστεί στην εικόνα και ένα φίλτρο μεσαίας τιμής για να καθαρίσει ο θόρυβος και να παραχθεί ένα καλύτερο αποτέλεσμα.

Τέλος, για την εύρεση των γωνιών, χρησιμοποιείται ο αλγόριθμος ανίχνευσης γωνιών Harris (Harris Corner Detection).

Η ιδέα πίσω από αυτόν τον αλγόριθμο είναι να εξετάσει ένα μικρό παράθυρο γύρω από κάθε εικονοστοιχείο σε μια εικόνα. Χρειάζεται να εντοπιστούν όλα τα παράθυρα που είναι μοναδικά. Η μοναδικότητα μπορεί να μετρηθεί μετατοπίζοντας κάθε παράθυρο κατά μια μικρή ποσότητα σε μια δεδομένη κατεύθυνση, μετρώντας το μέγεθος της αλλαγής στις τιμές των εικονοστοιχείων. Ο αλγόριθμος δουλεύει ως εξής:

1. Παίρνει σαν είσοδο μια εικόνα σε κλίμακα του γκρι της αρχικής εικόνας.
2. Εφαρμόζει ένα Γκαουσιανό φίλτρο για να εξομαλύνει το θόρυβο.
3. Εφαρμόζει τον τελεστή Sobel για να βρει τις τιμές κλίσης των X και Y για κάθε εικονοστοιχείο στην εικόνα.
4. Για κάθε εικονοστοιχείο στην εικόνα, θεωρεί ένα παράθυρο γύρω του και υπολογίζει τη συνάρτηση δύναμης της γωνίας. Αυτή ονομάζεται τιμή Harris.
5. Βρίσκει όλα τα εικονοστοιχεία που υπερβαίνουν ένα συγκεκριμένο όριο και αποτελούν τοπικά μέγιστα σε ένα συγκεκριμένο παράθυρο.
6. Για κάθε εικονοστοιχείο που ικανοποιεί αυτά τα κριτήρια, επιστρέφει τα X και Y.

Ένα παράδειγμα εφαρμογής αυτού του αλγορίθμου σε μια εικόνα που έχει εφαρμοστεί και η μάσκα αναγνώρισης του λευκού χρώματος, απεικονίζεται στο Σχήμα 40.



Σχήμα 40: Εικόνα με μάσκα αναγνώρισης

Όπως παρατηρείτε στο Σχήμα 40, υπάρχουν τέσσερεις γωνίες. Άρα, θα πρέπει να κατευθυνθεί ο αριστερός βραχίονας στην πάνω αριστερή γωνία και ο δεξιός βραχίονας στην πάνω δεξιά γωνία. Για να δουλέψει σωστά ο αλγόριθμος που υλοποιήθηκε, το σύστημα αναγκάστηκε να συνεχίσει την εύρεση μόνο εφόσον ανιχνεύσει 4 γωνίες. Αυτό έγινε λόγω των περιορισμών τους διαθέσιμου υλικού που υπήρχε στην κατοχή μας. Το πρόβλημα της μη ύπαρξης στερεοσκοπικών καμερών και τα οφέλη αυτών, αναλύεται στο επόμενο κεφάλαιο 5.4. Μόλις γίνει η ανίχνευση των τεσσάρων γωνιών, δίνεται η εντολή σε κάθε βραχίονα να πιάσει την κατάλληλη γωνία.

Μόλις οι βραχίονες πιάσουν και ανασηκώσουν το λευκό είδος, το φέρνουν στο οπτικό σημείο της δεύτερης κάμερας του ρομποτικού συστήματος. Στο επόμενο βήμα γίνεται η κατηγοριοποίηση του λευκού είδος σε λερωμένο ή όχι.

Αν και ένας άνθρωπος μπορεί να καταλάβει αμέσως τη διαφορά, δε γίνεται το ίδιο και από μια μηχανή.

« Μία μηχανή, θα πρέπει να μάθει να καταλαβαίνει αυτό που βλέπει. »

Εδώ προκύπτει ένα πρόβλημα, το οποίο αντιμετωπίστηκε με επιτυχία. Το πρόβλημα είναι ότι για να μάθει κάτι μια μηχανή πρέπει να εφαρμοστεί ένας αλγόριθμος μηχανικής μάθησης. Στη προκειμένη περίπτωση εφαρμόζεται ένα νευρωνικό δίκτυο, καθώς αυτό δουλεύει εξαιρετικά με δεδομένα από εικόνες, αλλά εφόσον δεν υπάρχει έτοιμο κάποιο σύνολο από εικόνες με λερωμένες και καθαρές πετσέτες για να δοθεί ως είσοδος στο νευρωνικό δίκτυο, πρέπει να δημιουργηθεί από το μηδέν, όπως και έγινε. Με αυτό το νευρωνικό δίκτυο μαθαίνει το σύστημα να ξεχωρίζει ένα λερωμένο από ένα καθαρό λευκό είδος. Το σύστημα απόφασης του ρομποτικού συστήματος αναλύεται με λεπτομέρεια στο επόμενο κεφάλαιο.

5 Υλοποίηση

Σε αυτό το κεφάλαιο μελετάται πως από τη θεωρία, εφαρμόζονται στην πράξη οι αλγόριθμοι του προηγούμενου κεφαλαίου. Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την υλοποίηση είναι η Python.

5.1 Κίνηση ρομποτικού βραχίονα

Το πρώτο βήμα για την υλοποίηση του ρομποτικού συστήματος αφορούσε την κίνηση των ρομποτικών βραχιόνων. Για το λόγο αυτό χρησιμοποιήθηκε η βιβλιοθήκη adafruit_servokit. Αυτή δίνει τη δυνατότητα να δημιουργηθεί ένα αντικείμενο kit με την εντολή kit = ServoKit(channels=16). Η επιλογή 16 χρησιμοποιήθηκε λόγω του HAT που υπήρχε, καθώς χρειάζονταν να συνδεθούν 12 σερβοκινητήρες στο raspberry pi. Οι σερβοκινητήρες του ρομποτικού βραχίονα μπορούν να περιστρέφουν έως 180 μοίρες. Άρα, χρησιμοποιώντας την εντολή kit.servo[0].angle = 90, μπορεί να περιστραφεί ένας σερβοκινητήρας κατά 90 μοίρες, στη συγκεκριμένη περίπτωση αυτός που είναι συνδεδεμένος στη θέση 0 του HAT. Χρησιμοποιώντας την παραπάνω εντολή για όλους του σερβοκινητήρες μπορεί να μετακινηθεί ο ρομποτικός βραχίονας στην επιθυμητή θέση.

Ενώ όλα φαινόταν ότι πηγαίνανε σύμφωνα με το πλάνο, προέκυψε ένα πρόβλημα που λίγο έλειψε να καταστρέψει όλους του σερβοκινητήρες. Μετά από λίγη ώρα αρχίζαν να ζεσταίνονται, σε σημείο που ένας από αυτούς χρειάστηκε μια μικρή επισκευή καθώς λιώσανε τα καλώδια και ενωθήκαν μεταξύ τους. Δυστυχώς, έπειτα από έρευνα στο διαδίκτυο για θέματα αυξημένης θερμοκρασίας υλικού, δε βρέθηκε κάποια λύση στο πρόβλημα. Μετά από αρκετή σκέψη, και αφού λύθηκε - αποσυναρμολογήθηκε ο ρομποτικός βραχίονας για ακόμη μια φορά φάνηκε ότι δεν είχαν βαθμονομηθεί σωστά οι σερβοκινητήρες. Λόγω έλλειψης εμπειρίας, κατά την συναρμολόγηση του ρομποτικού βραχίονα την πρώτη φόρα, η τοποθέτηση των σερβοκινητήρων έγινε χωρίς να ελεγχθεί αν μπορούν να κάνουν την κίνηση 180 μοιρών που απαιτείται. Έτσι, κάθε φορά που προγραμματίζονταν ένας σερβοκινητήρας να περιστραφεί κατά κάποιες μοίρες και δε μπορούσε να κάνει την περιστροφή, συνέχιζε να προσπαθεί με αποτέλεσμα να υπερθερμαίνεται.

Έχοντας βρει τη λύση στο πρόβλημα που παρουσιάστηκε, πραγματοποιήθηκαν κάποιες πρόσθετες δοκιμές και ολοκληρώθηκαν με επιτυχία.

Το κινηματικό διάγραμμα του ρομποτικού βραχίονα που παρουσιάστηκε, δείχνει τη θέση μηδέν των σερβοκινητήρων. Συνεπώς, για να μετακινηθεί ένα κομμάτι του βραχίονα, θα πρέπει να το τροφοδοτηθεί με τιμές που κυμαίνονται από -90 μέχρι 90. Οι σερβοκινητήρες αντιλαμβάνονται τιμές από 0 μέχρι και 180. Άρα, το επόμενο βήμα έχει να κάνει με τη μετατροπή των τιμών από το διάστημα [-90, 90] στο διάστημα [0,180]. Γι' αυτό το λόγο δημιουργήθηκαν οι συναρτήσεις Theta1, Theta2, Theta3, Theta4 Theta5, Theta7, Theta8, Theta9, Theta10 και Theta11.

Για την κίνηση των ρομποτικών βραχιόνων σε ένα συγκεκριμένο σημείο, δημιουργήθηκε η συνάρτηση inverse_kinematics, που δέχεται σαν είσοδο τα X,Y,Z ενός σημείου και υπολογίζει τις γωνίες των αρθρώσεων, ώστε να μπορέσει να το φτάσει ο end-effector.

Ακόμη, δημιουργήθηκε η συνάρτηση smooth_move, με την οποία ρυθμίζεται η ταχύτητα κίνησης των βραχιόνων. Άλλη μια συνάρτηση, η initial_position, επαναφέρει τους βραχίονες στην αρχική τους θέση, δηλαδή στη θέση που όλοι οι σερβοκινητήρες έχουν τιμή μηδέν, όπως παρουσιάζονται στο κινηματικό διάγραμμα.

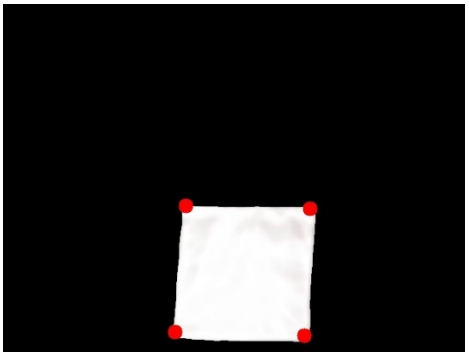
Τέλος, για την κίνηση των ρομποτικών βραχιόνων δημιουργήθηκαν οι συναρτήσεις close_gripper και open_gripper, οι οποίες κλείνουν και ανοίγουν τον end-effector αντίστοιχα, και οι συναρτήσεις take_position, take_position_left_1, take_position_left_2, take_position_left_3, take_position_left_4, take_position_right_1, take_position_right_2, take_position_right_3 και take_position_right_4 οι οποίες, αφού οι βραχίονες έχουν πιάσει το λευκό είδος, το μετακινούνε στο πεδίο θέασης της δεύτερης κάμερας.

5.2 Όραση Συστήματος

Η ενότητα αυτή αφιερώνεται στην υλοποίηση των αλγορίθμων όρασης του ρομποτικού συστήματος. Η σύνδεση των καμερών στο Raspberry Pi θα τροφοδοτεί με εικόνες το σύστημα, ώστε να εξάγει τις συντεταγμένες των γωνιών του λευκού είδους και να τα κατηγοριοποιεί σε λερωμένα και καθαρά.

5.2.1 Εύρεση των γωνιών του λευκού είδους

Η πρώτη κάμερα του ρομποτικού συστήματος που είναι τοποθετημένη στο πάνω μέρος της κατασκευής, τροφοδοτεί με εικόνες το σύστημα από τις οποίες γίνεται η εύρεση των γωνιών. Αφού τροφοδοτηθεί το σύστημα από την κάμερα, στη συνέχεια χρησιμοποιείται η συνάρτηση color_recognition, η οποία εφαρμόζει μια μάσκα που απομονώνει το λευκό είδος. Η εικόνα που θα επιστρέψει η συνάρτηση color_recognition, θα δοθεί ως είσοδος στη συνάρτηση harris, η οποία εφαρμόζει τον αλγόριθμο εύρεσης γωνιών Harris (harris corner detection), επιστρέφει τις συντεταγμένες των γωνιών και αποθηκεύει μια εικόνα με ζωγραφισμένες τις γωνίες που εντόπισε (Σχήμα 41).



Σχήμα 41: Εντοπισμός γωνιών

5.2.2 Μοντέλο Κατηγοριοποίησης

Ένα από τα πιο δύσκολα κομμάτια που έπρεπε να λυθεί, για να δημιουργηθεί ένα πλήρως λειτουργικό σύστημα, ήταν πως θα καταλαβαίνει το ρομπότ αν το λευκό είδος είναι λερωμένο ή όχι.

Αποφασίστηκε να χρησιμοποιηθεί ένα προεκπαιδευμένο νευρωνικό δίκτυο, δεδομένου ότι έχουν εξαιρετική απόδοση στην κατηγοριοποίηση εικόνας.

5.2.3 Εκπαίδευση μοντέλου

Επιλέχθηκε να χρησιμοποιηθεί η βιβλιοθήκη ImageAI που παρέχει διάφορους αλγορίθμους βαθιάς μάθησης, όπως οι SqueezeNet, ResNet, InceptionV3 και DenseNet. Μάλιστα χρησιμοποιώντας το σύνολο από εικόνες που δημιουργήθηκε για τους σκοπούς της μελέτης και με λιγοστό κώδικα δημιουργήθηκε το προσαρμοσμένο μοντέλο. Για τη δημιουργία αυτού του μοντέλου έπρεπε να ξεπεραστούν αρκετά προβλήματα, τα οποία αναφέρονται παρακάτω.

Η κλάση ClassificationModelTrainer επιτρέπει να εκπαιδευτεί οποιοσδήποτε από τους 4 αλγόριθμους, MobileNetV2, ResNet50, InceptionV3 και DenseNet121, και για να δουλέψει σωστά θα πρέπει να υπάρχουν τουλάχιστον 500 εικόνες ανά κλάση. Η διαδικασία εκπαίδευσης δημιουργεί ένα αρχείο JSON που αντιστοιχίζει τους τύπους κλάσεων που υπάρχουν στο σύνολο εικόνων με ένα αριθμό.

Το σύνολο δεδομένων τοποθετήθηκε σε ένα φάκελο, μέσα στον οποίο δημιουργήθηκαν άλλοι δύο φάκελοι με ονόματα train και test. Σε κάθε έναν από αυτούς δημιουργήθηκαν ακόμα δύο φάκελοι με ονόματα clean και dirty. Από το σύνολο δεδομένων κρατήθηκε το 80% των εικόνων για εκπαίδευση και το υπόλοιπο 20% των εικόνων χρησιμοποιήθηκε για δοκιμή.

Έχοντας ετοιμάσει το σύνολο δεδομένων, δημιουργήθηκε ένα αντικείμενο της κλάσης ModelTraining.

```
model_trainer = ClassificationModelTrainer()
```

Στη συνέχεια, δηλώθηκε ο αλγόριθμος που θα εκπαιδεύσει το μοντέλο.

```
model_trainer.setModelTypeAsInceptionV3()
```

Και επίσης, δηλώθηκε η διαδρομή που έχει αποθηκευτεί το σύνολο με τις εικόνες.

```
model_trainer.setDataDirectory("/content/drive/MyDrive/final_dataset_v2")
```

Τέλος, η ακόλουθη εντολή αρχίζει την εκπαίδευση του μοντέλου.

```
model_trainer.trainModel(num_objects=2, num_experiments=20, enhance_data=False, batch_size=8, show_network_summary=True, save_full_model = True, training_image_size = 600)
```

Το num_objects δηλώνει των αριθμό των κλάσεων, το num_experiments των αριθμό των εποχών, το enhance_data χρησιμοποιείται για τη μετατροπή του συνόλου των εικόνων προκειμένου να δημιουργηθούν περισσότερες για την εκπαίδευση του μοντέλου, στην περίπτωση μας επιλέχθηκε να μην χρησιμοποιηθεί αυτή η επιλογή, batch_size ο αλγόριθμος εκπαιδεύεται παράλληλα σε ένα σύνολο εικόνων, show_network_summary εμφανίζει τη δομή του αλγόριθμου εκπαίδευσης, save_full_model αποθηκεύει το πλήρες μοντέλο και το training_image_size αλλάζει το μέγεθος των εικόνων που πρόκειται να εκπαιδεύσουν το μοντέλο.

Το μοντέλο που παράχθηκε, όπως και το αρχείο JSON υπάρχει διαθέσιμο στον παρακάτω σύνδεσμο:

https://drive.google.com/drive/folders/1j0XPrVqJ__wsazXPYnpd_vIIE9OxWn-B?usp=sharing

5.2.4 Πρόβλεψη με βάση μοντέλο

Έχοντας εκπαιδεύσει το μοντέλο, μπορούμε να το χρησιμοποιήσουμε για να προβλέψουμε αν κάποια άγνωστη εικόνα ανήκει σε μια από τις δυο κλάσεις, clean ή dirty. Πριν φτάσουμε σε αυτό, θα χρησιμοποιήσουμε τη συνάρτηση take_picture, η οποία δημιουργήθηκε για να τροφοδοτεί το σύστημα με εικόνες από το λευκό είδος. Η take_picture εκτελεί επί σειρά τις ακόλουθες λειτουργίες. Πρώτα εφαρμόζει ένα φίλτρο μεσαίας τιμής στην εικόνα για να καθαρίσει τον θόρυβο. Έπειτα χρησιμοποιώντας τη συνάρτηση color_recognition_2 αναγνωρίζει το λευκό είδος, μετατρέπει την εικόνα σε κλίμακα του γκρι και στη συνέχεια σε δυαδική. Αλλάζει το μέγεθος της εικόνας σε 600x600 και την αποθηκεύει σε ένα νέο αρχείο. Αυτή η εικόνα θα χρησιμοποιηθεί από τη συνάρτηση prediction για να αποφασίσει την κατηγοριοποίηση του λευκού είδους.

Στη συνάρτηση prediction, αρχικά δημιουργούμε ένα αντικείμενο CustomImageClassification().

prediction = ImageClassification()

Στη συνέχεια, δηλώνουμε ότι θα χρησιμοποιήσουμε ένα προεκπαιδευμένο InceptionV3.

prediction.setModelTypeAsInceptionV3()

Και το φορτώνουμε από εκεί που το έχουμε αποθηκεύσει, όπως και το αρχείο JSON.

prediction.setModelPath("/home/pi/Desktop/CNN_Model/Final_Model.h5")
prediction.setJsonPath("/home/pi/Desktop/CNN_Model/final_model_class.json")
prediction.loadModel(num_objects=2)

Τέλος, υπολογίζουμε τις πιθανότητες που έχει μια άγνωστη εικόνα να ανήκει σε κάθε μια κλάση.

predictions, probabilities = prediction.classifyImage("//home/pi/Desktop/Prediction_Pictures/frame.png", result_count=2)

Τα αποτέλεσμα κατηγοριοποίησης ενός καθαρού λευκού είδους απεικονίζεται στο Σχήμα 42.

```
from imageai.Classification.Custom import CustomImageClassification
import os

execution_path = os.getcwd()

prediction = CustomImageClassification()
prediction.setModelTypeAsInceptionV3()
prediction.setModelPath("binary_model.h5")
prediction.setJsonPath("binary_model_class.json")
prediction.loadModel(num_objects=2)

predictions, probabilities = prediction.classifyImage("clean_37.png", result_count=2)

for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
clean  :  94.68600153923035
dirty  :  5.313992127776146
```

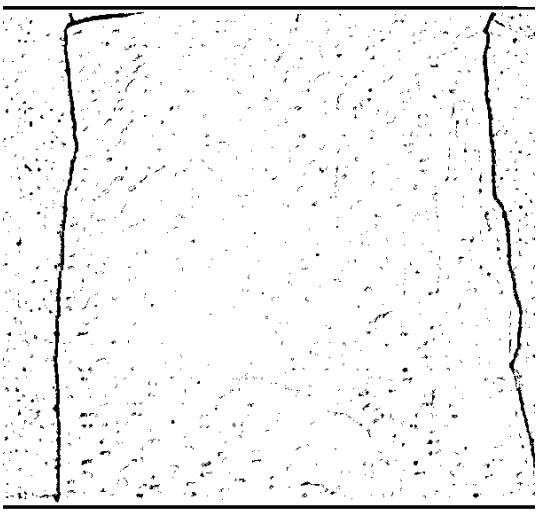
Σχήμα 42: Αποτέλεσμα κατηγοριοποίησης ενός καθαρού λευκού είδους

5.2.5 Δημιουργία του συνόλου δεδομένων

Για να μπορέσει να εκπαιδευτεί το μοντέλο χρειάζεται τα κατάλληλα δεδομένα, τα οποία δυστυχώς δεν υπάρχουν έτοιμα και χρειάστηκε να δημιουργηθούν.

Για τη δημιουργία του συνόλου εικόνων, τραβήχτηκαν 10 φωτογραφίες για κάθε κλάση, διαστάσεων 600x600. Από αυτές τις φωτογραφίες, με τις παρακάτω τεχνικές, δημιουργήθηκε ένα σύνολο δεδομένων 4800 φωτογραφιών.

Η πρώτη τεχνική που εφαρμόστηκε ήταν η rotate. Πιο συγκεκριμένα, έγινε rotate σε κάθε εικόνα κατά 45, 90, 135, 180, 225, 270 και 315 μοίρες. Άρα, από τις 10 φωτογραφίες για κάθε κλάση, πλέον έχουμε 80. Στη συνέχεια, αυτές τις κάναμε FLIP_TOP_BOTTOM και FLIP_LEFT_RIGHT, φτάνοντας στις 240 φωτογραφίες για κάθε κλάση. Η επόμενη τεχνική αφορά τη μετακίνηση της εικόνας αριστερά, δεξιά, πάνω και κάτω. Εφαρμόζοντας αυτές οι φωτογραφίες φτάνουν πλέον στις 1200 φωτογραφίες για κάθε κλάση. Τέλος, σε κάθε φωτογραφία που έχει παραχθεί εφαρμόζεται θόρυβος salt and pepper. Πλέον, έχει δημιουργηθεί ένα σύνολο δεδομένων που αποτελείται από 4800 φωτογραφίες. Για την ολοκλήρωση του συνόλου δεδομένων, έγινε μετατροπή σε κλίμακα του γκρι και στη συνέχεια εφαρμόστηκε ένα φίλτρο median. Τέλος, έγινε μετατροπή των εικόνων σε δυαδικές. Το αποτέλεσμα αυτών απεικονίζεται στα Σχήματα 43 και 44. Κρατήθηκαν 3840 για την εκπαίδευση του μοντέλου και οι υπόλοιπες 960 χρησιμοποιήθηκαν για τη δοκιμή του. Για κάθε κλάση υπάρχει ακριβώς ο ίδιος αριθμός εικόνων τόσο στο σύνολο εκπαίδευσης όσο και στο σύνολο δοκιμής.

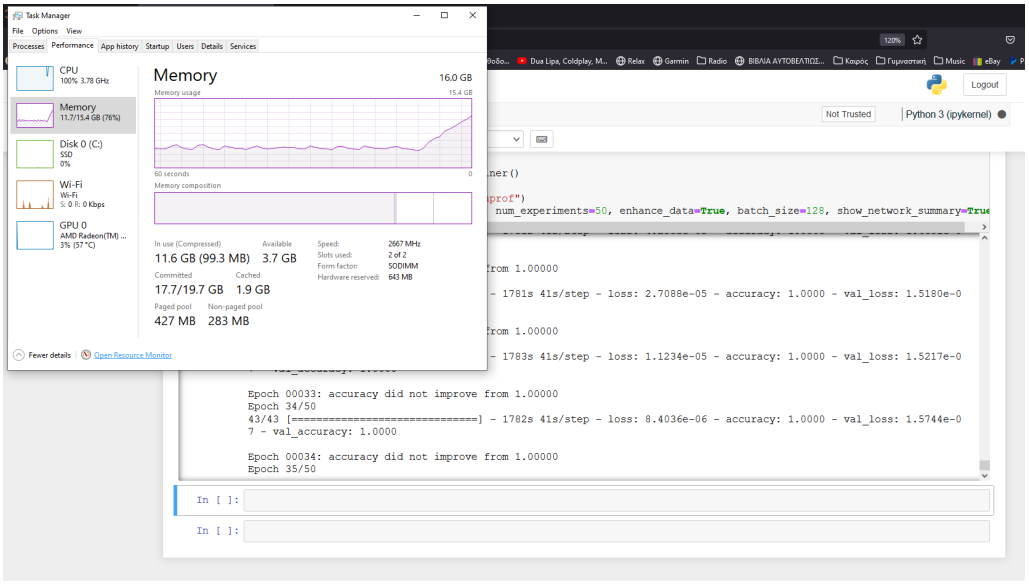


Σχήμα 43: Δυαδική εικόνα - clean

Σχήμα 44: Δυαδική εικόνα - dirty

5.2.6 Πλατφόρμα εκπαίδευσης του μοντέλου

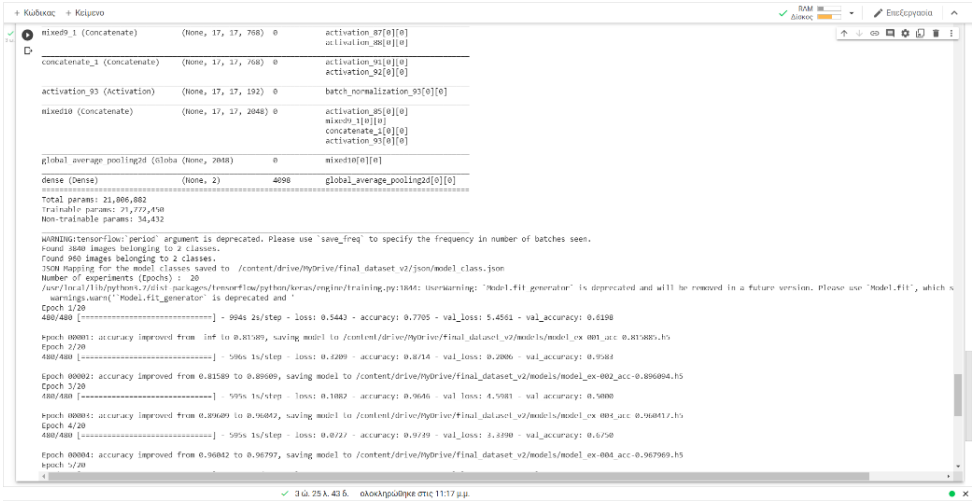
Ένας αλγόριθμος βαθιάς μάθησης έχει τρομερές απαιτήσεις από υπολογιστική ισχύ. Η πρώτη δοκιμή που πραγματοποιήθηκε για την εκπαίδευση του μοντέλου, χρησιμοποιώντας τους προσωπικούς υπολογιστές, διήρκησε περίπου 25 ώρες για 50 εποχές. Στις επόμενες προσπάθειες χαμηλώθηκε ο αριθμός των εποχών, αλλά ακόμη και τότε ο χρόνος ήταν πολύ μεγάλος.



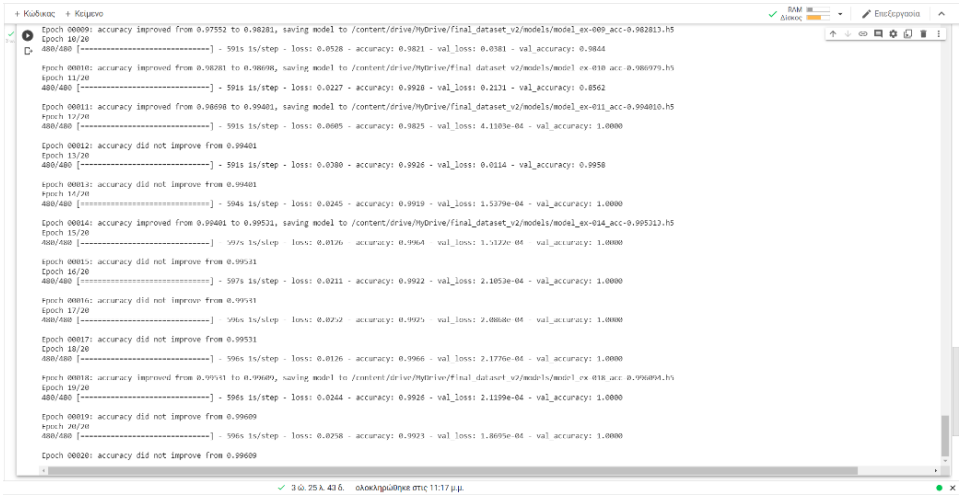
Σχήμα 45: Στιγμιότυπο λήψης προσωπικού υπολογιστή κατά τη διάρκεια της εκπαίδευσης

Θέλοντας να βελτιωθεί ο χρόνος εκτέλεσης των μοντέλων έγινε μια περαιτέρω έρευνα και διαπιστώθηκε ότι μπορούν να χρησιμοποιηθούν πυρήνες από την κάρτα γραφικών εφόσον ήταν συμβατή. Η nvidia 1650 mobile (κάρτα γραφικών προσωπικού υπολογιστή) είναι συμβατή, αλλά μετά από αρκετές προσπάθειες δεν κατάφερε να γίνει η εκπαίδευση του μοντέλου από την gpu. Οπότε σε συνέχεια αναζητήσεων εναλλακτικών λύσεων,

συναντήθηκε το Google Colab. Το Google Colab είναι μια εφαρμογή υπολογιστικού νέφους που δίνει την επιλογή να χρησιμοποιηθούν αρκετά ισχυρές κάρτες γραφικών για την εκπαίδευση αλγορίθμων βαθιάς μάθησης, είτε δωρεάν με κάποιους περιορισμούς, είτε επί πληρωμή [22]. Χρησιμοποιήθηκε η δωρεάν έκδοσή, η οποία παρόλο των περιορισμών βοήθησε να παραχθούν αρκετά πιο γρήγορα τα μοντέλα εκπαίδευσης. Τα Σχήματα 46 και 47 απεικονίζουν το περιβάλλον που έγινε η εκπαίδευση του μοντέλου.



Σχήμα 46: Περιβάλλον εφαρμογής Colab



Σχήμα 47: Περιβάλλον εφαρμογής Colab

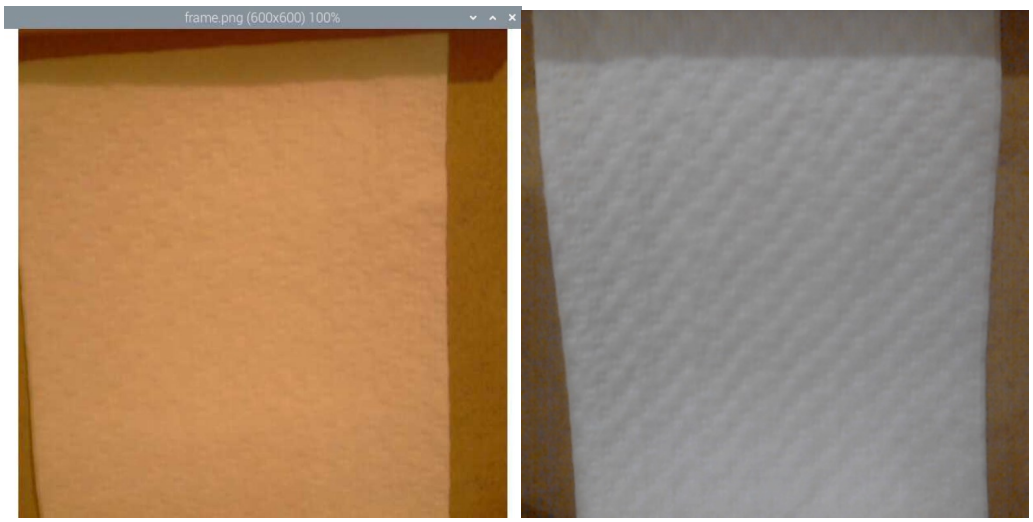
5.2.7 Δυσκολίες εκπαίδευσης του μοντέλου

Κατά την υλοποίηση του συστήματος στο κομμάτι της όρασης, προέκυψαν αρκετά προβλήματα. Τα περισσότερα προβλήματα είχαν να κάνουν με τη δημιουργία του μοντέλου που θα κατηγοριοποιεί τα λευκά είδη σε καθαρά και λερωμένα. Ένα από τα προβλήματα ήταν η έλλειψη του κατάλληλου συνόλου από δεδομένα για την εκπαίδευση του μοντέλου. Αυτό το πρόβλημα λύθηκε δημιουργώντας ένα καινούργιο σύνολο δεδομένων από την αρχή. Επειδή, όμως κάθε φορά που τελείωνε η εκπαίδευση του μοντέλου, το αποτέλεσμα ήταν να προβλέπει πάντα μια κλάση, συνεχίστηκε η

αναδιαμόρφωση του συνόλου δεδομένων ευελπιστώντας να δώσει λύση στο πρόβλημα. Κάποια από τα σύνολα δεδομένων που δημιουργήθηκαν μπορείτε να τα βρείτε στους παρακάτω συνδέσμους:

1. https://drive.google.com/drive/folders/1A-XaR9evOEHHLVWsguq69vBTcL-_xcWc?usp=sharing
2. https://drive.google.com/drive/folders/195dZmbsSYwsWlOHXSFHsmiet3G9s64xr?usp=sharing
3. https://drive.google.com/drive/folders/1Ta37e8hPDsideQz7arATSdubD-JbNnGq?usp=sharing
4. https://drive.google.com/drive/folders/18NIWrksF0jzGZiiWxpA7bV0q7bKl-17U?usp=sharing

Στους παραπάνω συνδέσμους υπάρχουν και τα μοντέλα που δημιουργήθηκαν για κάθε σύνολο δεδομένων.
Βλέποντας ότι το πρόβλημα που είχαν όλα τα μοντέλα και κατηγοριοποιούσαν τις εικόνες μόνο σε μια κλάση, παρέμενε, αποφασίστηκε να αλλαγθεί το μοντέλο βαθιάς μάθησης που χρησιμοποιούνταν στην αρχή των πειραμάτων, το ResNet50. Από τις πρώτες κιόλας προσπάθειες, υπήρξε διαφορά στην απόδοση του μοντέλου χρησιμοποιώντας το InceptionV3. Για την εκπαίδευση του χρησιμοποιήθηκε το final_dataset_v2 που περιέχει έγχρωμες φωτογραφίες μεγέθους 600x600. Όλα δούλευαν όπως έπρεπε, οι κατηγοριοποιήσεις γίνονταν σωστά φτάνοντας στο τελευταίο βήμα, την τοποθέτηση του μοντέλου στο raspberry. Εκεί δυστυχώς η απόδοση δεν ήταν η αναμενόμενη. Για τη δημιουργία του συνόλου φωτογραφιών τοποθετήθηκε η κάμερα σε έναν υπολογιστή με λειτουργικό σύστημα Windows 11. Κάτι που δεν είχε ληφθεί υπόψη ήταν η διαφορά της απόδοσης της κάμερα σε Windows σε σχέση με το raspberry. Συνεπώς, κάθε φορά που λαμβάνονταν μια φωτογραφία με την κάμερα στο raspberry, η διαφορά στην εικόνα ήταν αρκετή ώστε να μην γίνεται σωστά η κατηγοριοποίηση. Στο Σχήμα 48 απεικονίζεται αυτή η διαφορά. Οι φωτογραφίες τραβήχτηκαν σε ίδιες συνθήκες φωτισμού.



Σχήμα 48: Διαφορά εικόνας στο raspberry (αριστερή) με τα Windows (δεξιά)

Η επόμενη κίνηση ήταν να δημιουργηθεί ένα σύνολο δεδομένων σε κλίμακα του γκρι. Δυστυχώς, ούτε αυτό είχε το αναμενόμενο αποτέλεσμα.

Μια τελευταία απόπειρα που δοκιμάστηκε ήταν η μετατροπή του συνόλου δεδομένων σε δυαδικό. Πριν από την μετατροπή εφαρμόστηκε και ένα median φίλτρο για την αφαίρεση του θορύβου. Ακόμη και έτσι, λόγω της χαμηλότερης απόδοσης της κάμερας στο raspberry, δεν γινόταν σωστή κατηγοριοποίηση των καθαρών λευκών ειδών. Παρατηρήθηκε ότι ακόμη και με τη μετατροπή σε δυαδική, η εικόνα είχε αρκετό θόρυβο ο οποίος έβλαπτε την διαδικασία της κατηγοριοποίησης. Για να λυθεί αυτό, στο φίλτρο median που χρησιμοποιούνταν, μεγάλωσε το παράθυρο εφαρμογής από 5 στο 8. Με αυτή τη ρύθμιση το σύστημα που υλοποιήθηκε ήταν σε θέση να κατηγοριοποιήσει σωστά όλα τα λευκά είδη που του δίνονταν.

5.3 Κώδικες υλοποίησης

Η παρούσα ενότητα είναι αφιερωμένη στον κώδικα του ρομποτικού συστήματος και σε σημαντικές λεπτομέρειες αυτού.

5.3.1 Κώδικας υλοποίησης ευθείας κινηματικής

Η ευθεία κινηματική, όπως και προαναφέρθηκε, δεν έχει χρησιμότητα στο σύστημα που υλοποιήθηκε, αλλά ήταν κρίσιμη, καθώς βοήθησε στη δημιουργία των σωστών εξισώσεων αντίστροφης κινηματικής. Οι εξισώσεις της αντίστροφης κινηματικής δεν ήταν εύκολες στη δημιουργία τους και έπρεπε να δοκιμάζονται συνεχώς ως προς την ορθότητά τους. Έχοντας δημιουργήσει τις εξισώσεις της ευθείας κινηματικής, επιτεύχθηκε η δυνατότητα δοκιμής τιμών των παραμέτρων των αρθρώσεων που επέστρεφαν οι προς δοκιμή εξισώσεις αντίστροφης κινηματικής και αν συμπίπταν τα αποτελέσματα X,Y,Z, τότε αυτό αλάνθαστα επαλήθευε την ορθότητά τους και στη συνέχεια καθίσταται δυνατό να χρησιμοποιηθούν στους ρομποτικούς βραχίονες. Ο κώδικας των εξισώσεων της ευθείας κινηματικής παρουσιάζεται παρακάτω:

```
import numpy as np
```

```
l1=10.0
l2=1.8
l3=12.0
l4=9.0
l5=3.5
l6=12.0
```

```
theta1=50
theta2=20
theta3=10
theta4=0.0
theta5=0.0

theta1=np.deg2rad(theta1)
theta2=np.deg2rad(theta2)
theta3=np.deg2rad(theta3)
theta4=np.deg2rad(theta4)
theta5=np.deg2rad(theta5)
```

```
H_0_1=[[np.cos(theta1),0,-np.sin(theta1),l2*np.cos(theta1)],[np.sin(theta1),0,np.cos(theta1),l2*np.sin(theta1)],
       [0,-1,0,l1],[0,0,0,1]]

H_1_2=[[np.cos(theta2-np.pi/2),-np.sin(theta2-np.pi/2),0,l3*np.cos(theta2-np.pi/2)],
       [np.sin(theta2-np.pi/2),np.cos(theta2-np.pi/2),0,l3*np.sin(theta2-np.pi/2)],[0,0,1,0],[0,0,0,1]]

H_2_3=[[np.cos(theta3),0,-np.sin(theta3),0],[np.sin(theta3),0,np.cos(theta3),0],[0,-1,0,0],[0,0,0,1]]

H_3_4=[[np.cos(theta4),0,np.sin(theta4),0],[np.sin(theta4),0,-np.cos(theta4),0],[0,1,0,l4+l5],[0,0,0,1]]

H_4_5=[[np.cos(theta5+np.pi/2),-np.sin(theta5+np.pi/2),0,l6*np.cos(theta5+np.pi/2)],
       [np.sin(theta5+np.pi/2),np.cos(theta5+np.pi/2),0,l6*np.sin(theta5+np.pi/2)],[0,0,1,0],[0,0,0,1]]
```

```
H_0_2=np.dot(H_0_1,H_1_2)
H_0_3=np.dot(H_0_2,H_2_3)
H_0_4=np.dot(H_0_3,H_3_4)
H_0_5=np.dot(H_0_4,H_4_5)
```

```
print(np.matrix(H_0_5))
```

```
[[ 0.5566704  -0.3213938  -0.76604444 17.4335982 ]
 [ 0.66341395 -0.38302222  0.64278761 20.77655329]
 [-0.5        -0.8660254   0.          9.02631145]
 [ 0.          0.          0.          1.        ]]
```

Σχήμα 49: Κώδικας ευθείας κινηματικής

Αν για παράδειγμα δοθούν οι τιμές των παραμέτρων των αρθρώσεων 50, 20, 10, 0 και 0, τότε ο end-effector θα βρεθεί στο σημείο με συντεταγμένες (17.4,20.7,9.0)

5.3.2 Κώδικας υλοποίησης

Στην επόμενη ενότητα αναφέρεται ο κώδικας του ρομποτικού συστήματος.
Στο Σχήμα 50 απεικονίζονται όλες οι απαραίτητες βιβλιοθήκες για τη λειτουργία του συστήματος.

```
import numpy as np
import cv2
import time
from scipy import ndimage
import math
import os
import imagehash
from PIL import Image
import matplotlib.pyplot as plt
import threading
from imageai.Classification.Custom import CustomImageClassification
from adafruit_servokit import ServoKit
```

Σχήμα 50: Κώδικας εισαγωγής βιβλιοθηκών

Στο επόμενο σχήμα 51 βαθμονομούνται οι σερβοκινητήρες, ώστε να μπορούν να περιστραφούν σε όλο το διαθέσιμο εύρος [0,180]. Η βαθμονόμηση έγινε χρησιμοποιώντας ένα μοιρογνωμόνιο και αλλάζοντας τις τιμές στη συνάρτηση set_pulse_width_range μέχρι να έρθει το επιθυμητό αποτέλεσμα.

```
kit = ServoKit(channels=16)

################## Right Arm ###################
kit.servo[0].set_pulse_width_range(550, 2500) #
kit.servo[1].set_pulse_width_range(550, 2500) #
kit.servo[2].set_pulse_width_range(550, 2500) #
kit.servo[3].set_pulse_width_range(550, 2500) #
kit.servo[4].set_pulse_width_range(550, 2500) #
kit.servo[5].set_pulse_width_range(550, 2500) #
################################################

################## Left Arm ###################
kit.servo[6].set_pulse_width_range(550, 2500) #
kit.servo[7].set_pulse_width_range(550, 2500) #
kit.servo[8].set_pulse_width_range(550, 2500) #
kit.servo[9].set_pulse_width_range(550, 2500) #
kit.servo[10].set_pulse_width_range(550, 2500)#
kit.servo[11].set_pulse_width_range(550, 2500)#
################################################
```

Σχήμα 51: Κώδικας βαθμονόμησης servo κινητήρων

Στο Σχήμα 52 παρουσιάζεται ο κώδικας που μετατρέπει τις τιμές των μοιρών από το διάστημα [-90,90] στο διάστημα [0,180], που μπορούν να καταλάβουν οι σερβοκινητήρες. Δημιουργήθηκαν 10 τέτοιες συναρτήσεις ώστε να μπορέσουμε να κάνουμε την μετατροπή και το σύστημα υλοποίησης να είναι πιστό με το κινηματικό διάγραμμα. Επίσης, να μπορούν να εφαρμοστούν οι εξισώσεις αντίστροφης κινηματικής αυτούσιες, χωρίς αλλαγές.

```
################################### Theta1 ###################################
def Theta1(angle):                                                           #
    min_comp=0                                                               #
    max_comp=180                                                             #
    min_angle=-90                                                            #
    max_angle=90                                                             #
    compare=((max_comp-min_comp)/(max_angle-min_angle))*(angle-min_angle)+min_comp #
                                                                             #
    return compare                                                           #
##############################################################################

################################### Theta2 ###################################
def Theta2(angle):                                                           #
    min_comp=0                                                               #
    max_comp=180                                                             #
    min_angle=90                                                             #
    max_angle=-90                                                            #
    compare=((max_comp-min_comp)/(max_angle-min_angle))*(angle-min_angle)+min_comp #
                                                                             #
    return compare                                                           #
##############################################################################

################################### Theta3 ###################################
def Theta3(angle):                                                           #
    min_comp=0                                                               #
    max_comp=180                                                             #
    min_angle=-90                                                            #
    max_angle=90                                                             #
    compare=((max_comp-min_comp)/(max_angle-min_angle))*(angle-min_angle)+min_comp #
                                                                             #
    return compare                                                           #
##############################################################################

################################### Theta4 ###################################
def Theta4(angle):                                                           #
    min_comp=0                                                               #
    max_comp=180                                                             #
    min_angle=-90                                                            #
    max_angle=90                                                             #
    compare=((max_comp-min_comp)/(max_angle-min_angle))*(angle-min_angle)+min_comp #
                                                                             #
    return compare                                                           #
##############################################################################

################################### Theta5 ###################################
def Theta5(angle):                                                           #
    min_comp=0                                                               #
    max_comp=180                                                             #
    min_angle=90                                                             #
    max_angle=-90                                                            #
    compare=((max_comp-min_comp)/(max_angle-min_angle))*(angle-min_angle)+min_comp #
                                                                             #
    return compare                                                           #
##############################################################################
```

```
################################### Theta7 ###################################
def Theta7(angle):                                                           #
    min_comp=0                                                               #
    max_comp=180                                                             #
    min_angle=-90                                                            #
    max_angle=90                                                             #
    compare=((max_comp-min_comp)/(max_angle-min_angle))*(angle-min_angle)+min_comp #
                                                                             #
    return compare                                                           #
##############################################################################

################################### Theta8 ###################################
def Theta8(angle):                                                           #
    min_comp=0                                                               #
    max_comp=180                                                             #
    min_angle=90                                                             #
    max_angle=-90                                                            #
    compare=((max_comp-min_comp)/(max_angle-min_angle))*(angle-min_angle)+min_comp #
                                                                             #
    return compare                                                           #
##############################################################################

################################### Theta9 ###################################
def Theta9(angle):                                                           #
    min_comp=0                                                               #
    max_comp=180                                                             #
    min_angle=-90                                                            #
    max_angle=90                                                             #
    compare=((max_comp-min_comp)/(max_angle-min_angle))*(angle-min_angle)+min_comp #
                                                                             #
    return compare                                                           #
##############################################################################

################################### Theta10 ##################################
def Theta10(angle):                                                          #
    min_comp=0                                                               #
    max_comp=180                                                             #
    min_angle=-90                                                            #
    max_angle=90                                                             #
    compare=((max_comp-min_comp)/(max_angle-min_angle))*(angle-min_angle)+min_comp #
                                                                             #
    return compare                                                           #
##############################################################################

################################### Theta11 ##################################
def Theta11(angle):                                                          #
    min_comp=0                                                               #
    max_comp=180                                                             #
    min_angle=90                                                             #
    max_angle=-90                                                            #
    compare=((max_comp-min_comp)/(max_angle-min_angle))*(angle-min_angle)+min_comp #
                                                                             #
    return compare                                                           #
##############################################################################
```

Σχήμα 52: Κώδικας μετατροπής μοιρών

Οι επόμενες συναρτήσεις επιτελούν την ίδια λειτουργία με μικρές αλλαγές. Και οι δυο δημιουργούν και εφαρμόζουν μια μάσκα σε μια εικόνα που ανιχνεύει το λευκό χρώμα. Η πρώτη διαφορά τους αφορά την εφαρμογή του φίλτρου μεσαίας τιμής, που στην πρώτη περίπτωση εφαρμόζεται στο τέλος και στη δεύτερη περίπτωση στην αρχή. Η άλλη αλλαγή έχει να κάνει με τις διαφορετικές τιμές της μάσκας.

```
################## color_recognition ##################
def color_recognition(img):                           #
    # Resizing the image                              #
    image = cv2.resize(img, (640, 480))               #
                                                      #
    lower = np.array([165, 165, 165], dtype = "uint8")#
    upper = np.array([255, 255, 255], dtype = "uint8")#
                                                      #
    # Defining mask for detecting color               #
    mask1 = cv2.inRange(image, lower, upper)          #
    res = cv2.bitwise_and(image,image, mask= mask1)   #
                                                      #
    res = ndimage.median_filter(res, size=10)         #
                                                      #
    return res                                        #
#######################################################
```

```
################## color_recognition_2 ################
def color_recognition_2(img):                         #
    # Resizing the image                              #
    image = cv2.resize(img, (640, 480))               #
                                                      #
    image = ndimage.median_filter(image, size=7)      #
                                                      #
    lower = np.array([110, 110, 110], dtype = "uint8")#
    upper = np.array([255, 255, 255], dtype = "uint8")#
                                                      #
    # Defining mask for detecting color               #
    mask1 = cv2.inRange(image, lower, upper)          #
    res = cv2.bitwise_and(image,image, mask= mask1)   #
                                                      #
    return res                                        #
#######################################################
```

Σχήμα 53: Κώδικας ανίχνευσης λευκού χρώματος **54**

Η συνάρτηση του επόμενου σχήματος υλοποιεί τον αλγόριθμο αναγνώρισης γωνιών και επιστρέφει τις συντεταγμένες αυτών. Πρώτα, μετατρέπει την εικόνα που δέχεται σε κλίμακα του γκρι. Στη συνέχεια εφαρμόζει ένα φίλτρο για την εξομάλυνση της εικόνας μειώνοντας το θόρυβο καθώς διατηρεί τις άκρες. Πλέον, η εικόνα είναι έτοιμη για την εφαρμογή της συνάρτησης που υλοποιεί τον αλγόριθμο εύρεσης γωνιών. Αυτή είναι ενσωματωμένη σε βιβλιοθήκη της python. Η cornerHarris δέχεται τέσσερεις τιμές. Η πρώτη είναι η εικόνα, η δεύτερη το μέγεθος του πλαισίου για την εύρεση γωνιών, η τρίτη αφορά τον τελεστή sobel και η τελευταία είναι η ελεύθερη παράμετρος εύρεσης του Harris. Για καλύτερα αποτελέσματα, ανάλογα με τον τύπο ρούχου ή υφάσματος, χρησιμοποιούνται οι τιμές 0.02, 0.04 και 0.05. Μόλις εφαρμοστεί, αποθηκεύονται τα αποτελέσματα, δημιουργείται μια μάσκα που θα ανιχνεύει τις γωνίες, εφαρμόζεται η μάσκα και δημιουργείται μια λίστες που έχει όλες τις γωνίες. Έπειτα δημιουργείται ένα κατώφλι, ώστε να κρατηθούν οι γωνίες που ενδιαφέρουν, ζωγραφίζοντας τις γωνίες πάνω στην εικόνα, αποθηκεύεται και επιστρέφονται οι συντεταγμένες των γωνιών.

```
######################################### harris ###########################################
def harris(img):                                                                           #
    # Convert the image to grayscale                                                       #
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)                                            #
                                                                                           #
    # Apply a bilateral filter.                                                            #
    # This filter smooths the image, reduces noise, while preserving the edges             #
    bi = cv2.bilateralFilter(gray, 5, 75, 75)                                              #
                                                                                           #
    # Apply Harris Corner detection.                                                       #
    # The four parameters are:                                                             #
    #   The input image                                                                    #
    #   The size of the neighborhood considered for corner detection                       #
    #   Aperture parameter of the Sobel derivative used.                                   #
    #   Harris detector free parameter                                                     #
    #   --You can tweak this parameter to get better results                               #
    #   --0.02 for tshirt, 0.04 for washcloth, 0.02 for jeans, 0.05 for contour_thresh_jeans#
    #   Source: https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html        #
    dst = cv2.cornerHarris(bi, 15, 3, 0.04)                                                #
                                                                                           #
    # Dilate the result to mark the corners                                                #
    dst = cv2.dilate(dst,None)                                                             #
                                                                                           #
    # Create a mask to identify corners                                                    #
    mask = np.zeros_like(gray)                                                             #
                                                                                           #
    # All pixels above a certain threshold are converted to white                          #
    mask[dst>0.4*dst.max()] = 255                                                          #
                                                                                           #
    # Create an array that lists all the pixels that are corners                           #
    coordinates = np.argwhere(mask)                                                        #
                                                                                           #
    # Convert array of arrays to lists of lists                                            #
    coordinates_list = [l.tolist() for l in list(coordinates)]                             #
                                                                                           #
    # Convert list to tuples                                                               #
    coordinates_tuples = [tuple(l) for l in coordinates_list]                              #
                                                                                           #
    # Create a distance threshold                                                          #
    thresh = 100                                                                           #
                                                                                           #
    # Compute the distance from each corner to every other corner.                         #
    def distance(pt1, pt2):                                                                #
        (x1, y1), (x2, y2) = pt1, pt2                                                      #
        dist = math.sqrt( (x2 - x1)**2 + (y2 - y1)**2 )                                    #
        return dist                                                                        #
                                                                                           #
    # Keep corners that satisfy the distance threshold                                     #
    coordinates_tuples_copy = coordinates_tuples                                           #
    i = 1                                                                                  #
    for pt1 in coordinates_tuples:                                                         #
        for pt2 in coordinates_tuples[i::1]:                                               #
            if(distance(pt1, pt2) < thresh):                                               #
                coordinates_tuples_copy.remove(pt2)                                        #
        i+=1                                                                               #
                                                                                           #
    # Place the corners on a copy of the original image                                    #
    img2 = img.copy()                                                                      #
    for pt in coordinates_tuples:                                                          #
        #print(tuple(reversed(pt))) # Print corners to the screen                          #
        cv2.circle(img2, tuple(reversed(pt)), 10, (0, 0, 255), -1)                         #
    #cv2.imshow('Image with 4 corners', img2)                                              #
    cv2.imwrite('harris_corners_towel.jpg', img2)                                          #
                                                                                           #
    return coordinates_tuples                                                              #
############################################################################################
```

Σχήμα 55: Κώδικας αναγνώρισης γωνιών και επιστροφής συντεταγμένων

Στο Σχήμα 56 παρουσιάζεται η συνάρτηση όπου εφαρμόζονται οι εξισώσεις αντίστροφης κινηματικής, δοθέντος των συντεταγμένων του σημείου που καλείται να σταλεί ο end-effector και του προσανατολισμού του, και επιστρέφει τις τιμές των παραμέτρων των αρθρώσεων.

```
######################## inverse_kinematics ######################
def inverse_kinematics(X_Location,Y_Location,Z_Location,psi_Value):#
    X=X_Location                                                  #
    Y=Y_Location                                                  #
    Z=Z_Location                                                  #
                                                                  #
    l1=10.5                                                       #
    l2=1.8                                                        #
    l3=12.0                                                       #
    l4=9.0+3.5                                                    #
    l5=3.5                                                        #
    l6=12.0                                                       #
    psi=psi_Value                                                 #
    theta4=np.deg2rad(1)                                          #
    theta1=np.arctan2(Y,X)                                        #
                                                                  #
    R=l6*np.cos(psi)                                              #
                                                                  #
    Xw=X-R*np.cos(theta1)                                         #
                                                                  #
    Yw=Y-R*np.sin(theta1)                                         #
                                                                  #
    Zw=Z+l6*np.sin(psi)                                           #
                                                                  #
    Rw=np.sqrt((Xw)**2+(Yw)**2)                                   #
                                                                  #
    r1=Rw-l2                                                      #
    r3=abs(Zw-l1)                                                 #
    r2=np.sqrt(r1**2+r3**2)                                       #
    phi1=np.arccos((r1**2+r2**2-r3**2)/(2*r1*r2))                 #
    c=(r2**2+l3**2-l4**2)/(2*r2*l3)                               #
    phi2=np.arccos(c)                                             #
                                                                  #
    theta2=((phi1+phi2)-(np.pi/2))*(-1)                           #
                                                                  #
    c2=(l3**2+l4**2-r2**2)/(2*l3*l4)                              #
    phi3=np.arccos(c2)                                            #
                                                                  #
    theta3=(phi3-(np.pi/2))*(-1)                                  #
                                                                  #
    theta5=(psi-theta2-theta3)                                    #
                                                                  #
    return theta1,theta2,theta3,theta4,theta5                     #
##################################################################
```

Σχήμα 56: Κώδικας εξισώσεων αντίστροφης κινηματικής

Οι επόμενες δυο συναρτήσεις που παρουσιάζονται στο Σχήμα 57, κλείνουν και ανοίγουν αντίστοιχα τον end-effector.

```
############## close_gripper ##############
def close_gripper(gripper_select):        #
    kit.servo[gripper_select].angle = 0   #
###########################################

############## open_gripper ###############
def open_gripper(gripper_select):         #
    kit.servo[gripper_select].angle = 180 #
###########################################
```

Σχήμα 57: Κώδικας End-Effector

Η take_position του Σχήματος 58, τροφοδοτεί το ρομποτικό σύστημα με τις κατάλληλες τιμές στις παραμέτρους των αρθρώσεων, ώστε να μεταφέρουν το λευκό είδος στην γωνία θέασης της δεύτερης κάμερας.

```
###### take_position ######
def take_position():      #
    a1=25                 #
    a2=30                 #
    a3=-40                #
    a4=1                  #
    a5=0                  #
                          #
    return a1,a2,a3,a4,a5 #
#########################
```

Σχήμα 58: Κώδικας τροφοδότησης παραμέτρων αρθρώσεων

Οι επόμενες συναρτήσεις του Σχήματος 59 ρυθμίζουν την ομαλή μετακίνηση του αριστερού βραχίονα, που κρατάει τη μια γωνία του λευκού είδους, προς τη γωνία θέασης της δεύτερης κάμερας.

```
############################ take_position_left_2 ############################
def take_position_left_2():                                                 #
    global pt2_left,t2_left,pt3_left,t3_left,pt5_left,t5_left,pt1_left,t1_left #
                                                                            #
    smooth_move(pt5_left,t5_left,10) #Servo 11                              #
    pt5_left=t5_left                                                        #
##############################################################################

############################ take_position_left_1 ############################
def take_position_left_1():                                                 #
    global pt2_left,t2_left,pt3_left,t3_left,pt5_left,t5_left,pt1_left,t1_left #
                                                                            #
    smooth_move(pt2_left,t2_left,7) #Servo 8                                #
    pt2_left=t2_left                                                        #
##############################################################################

############################ take_position_left_3 ############################
def take_position_left_3():                                                 #
    global pt2_left,t2_left,pt3_left,t3_left,pt5_left,t5_left,pt1_left,t1_left #
                                                                            #
    smooth_move(pt3_left,t3_left,8) #Servo 9                                #
    pt3_left=t3_left                                                        #
##############################################################################

############################ take_position_left_4 ############################
def take_position_left_4():                                                 #
    global pt2_left,t2_left,pt3_left,t3_left,pt5_left,t5_left,pt1_left,t1_left #
                                                                            #
    smooth_move(pt1_left,(-1)*t1_left,6) #Servo 7                           #
    pt1_left=t1_left                                                        #
##############################################################################
```

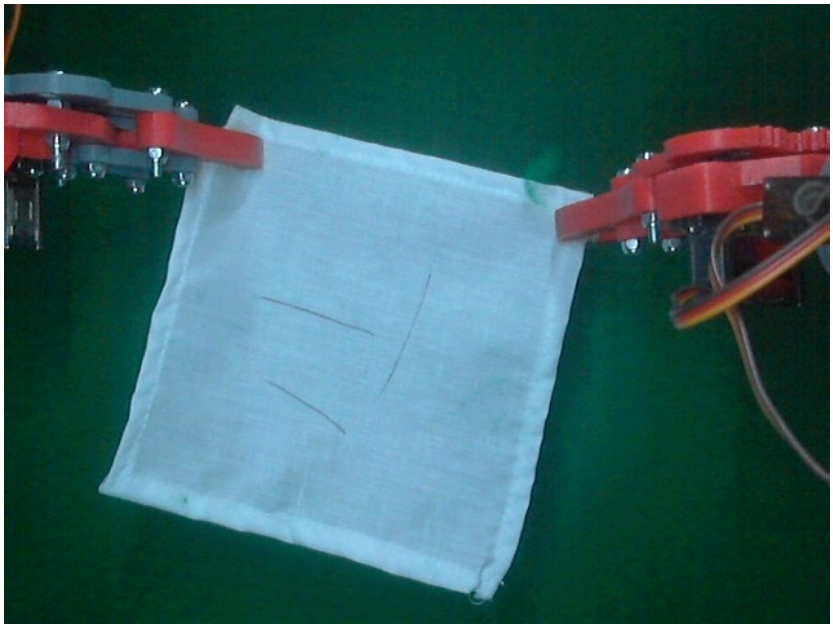
Σχήμα 59: Κώδικας ομαλής μετακίνησης αριστερού βραχίονα

Και οι επόμενες συναρτήσεις του Σχήματος 60 ρυθμίζουν την ομαλή μετακίνηση του δεξιού βραχίονα, που κρατάει την άλλη γωνία του λευκού είδους, προς τη γωνία θέασης της δεύτερης κάμερας.

```
################################ take_position_right_2 ################################
def take_position_right_2():                                                          #
    global pt2_right,t2_right,pt3_right,t3_right,pt5_right,t5_right,pt1_right,t1_right#
                                                                                      #
    smooth_move(pt5_right,t5_right,4) #Servo 5                                        #
    pt5_right=t5_right                                                                #
#######################################################################################

################################ take_position_right_1 ################################
def take_position_right_1():                                                          #
    global pt2_right,t2_right,pt3_right,t3_right,pt5_right,t5_right,pt1_right,t1_right#
                                                                                      #
    smooth_move(pt2_right,t2_right,1) #Servo 2                                        #
    pt2_right=t2_right                                                                #
#######################################################################################

################################ take_position_right_3 ################################
def take_position_right_3():                                                          #
    global pt2_right,t2_right,pt3_right,t3_right,pt5_right,t5_right,pt1_right,t1_right#
                                                                                      #
    smooth_move(pt3_right,t3_right,2) #Servo 3                                        #
    pt3_right=t3_right                                                                #
#######################################################################################

################################ take_position_right_4 ################################
def take_position_right_4():                                                          #
    global pt2_right,t2_right,pt3_right,t3_right,pt5_right,t5_right,pt1_right,t1_right#
                                                                                      #
    smooth_move(pt1_right,t1_right,0) #Servo 1                                        #
    pt1_right=t1_right                                                                #
#######################################################################################
```

Σχήμα 60: Κώδικας ομαλής μετακίνησης δεξιού βραχίονα

Στο σχήμα 61 παρέχεται μια εικόνα μετάβασης των βραχιόνων στις δύο πάνω γωνίες του υφάσματος που έχει μπροστά του.



Σχήμα 61: Πιάσιμο λευκού είδους από δύο βραχίονες

Οι παρακάτω συναρτήσεις χρησιμοποιούνται, αφού γίνει η κατηγοριοποίηση του λευκού είδους σε καθαρό ή λερωμένο, για τη μεταφορά του στον αντίστοιχο κάδο.

```
################## move_right #################
def move_right():                             #
    global pt1_right,t1_right                 #
                                              #
    t1_right=-60                              #
                                              #
    smooth_move(pt1_right,t1_right,0) #Servo 1 #
    pt1_right=t1_right                        #
###############################################

################## move_left ##################
def move_left():                              #
    global pt1_left,t1_left                   #
                                              #
    t1_left=60                                #
                                              #
    smooth_move(pt1_left,t1_left,6) #Servo 7  #
    pt1_left=t1_left                          #
###############################################
```

Σχήμα 62: Κώδικας μεταφοράς του υφάσματος

Το επόμενο κομμάτι κώδικα, του Σχήματος 63, αφορά την αρχικοποίηση κάποιον μεταβλητών με την τιμή μηδέν.

```
pt1_right=pt2_right=pt3_right=pt4_right=pt5_right=0
pt1_left=pt2_left=pt3_left=pt4_left=pt5_left=0
```

Σχήμα 63: Κώδικας αρχικοποίησης μεταβλητών

Η συνάρτηση του Σχήματος 64, μετακινεί τους ρομποτικούς βραχίονες στην αρχική τους θέση.

```
########################################## initial_position ##########################################
def initial_position():                                                                              #
    global pt2_right,t2_right,pt3_right,t3_right,pt5_right,t5_right,pt1_right,t1_right,pt4_right,t4_right#
    global pt2_left,t2_left,pt3_left,t3_left,pt5_left,t5_left,pt1_left,t1_left,pt4_left,t4_left      #
                                                                                                     #
    t1_right=t2_right=t3_right=t4_right=t5_right=0                                                   #
    t1_left=t2_left=t3_left=t4_left=t5_left=0                                                        #
                                                                                                     #
    smooth_move(pt1_left,np.rad2deg(t1_left),6) #Servo 7                                             #
    pt1_left=np.rad2deg(t1_left)                                                                     #
                                                                                                     #
    smooth_move(pt5_left,(np.rad2deg(t5_left)),10) #Servo 11                                         #
    pt5_left=np.rad2deg(t5_left)                                                                     #
                                                                                                     #
    smooth_move(pt3_left,np.rad2deg(t3_left),8) #Servo 9                                             #
    pt3_left=np.rad2deg(t3_left)                                                                     #
                                                                                                     #
    smooth_move(pt2_left,np.rad2deg(t2_left),7) #Servo 8                                             #
    pt2_left=np.rad2deg(t2_left)                                                                     #
                                                                                                     #
    smooth_move(pt4_left,np.rad2deg(t4_left),9) #Servo 10                                            #
    pt4_left=np.rad2deg(t4_left)                                                                     #
                                                                                                     #
    smooth_move(pt1_right,np.rad2deg(t1_right),0) #Servo 1                                           #
    pt1_right=np.rad2deg(t1_right)                                                                   #
                                                                                                     #
    smooth_move(pt5_right,(np.rad2deg(t5_right)),4) #Servo 5                                         #
    pt5_right=np.rad2deg(t5_right)                                                                   #
                                                                                                     #
    smooth_move(pt3_right,np.rad2deg(t3_right),2) #Servo 3                                           #
    pt3_right=np.rad2deg(t3_right)                                                                   #
                                                                                                     #
    smooth_move(pt2_right,np.rad2deg(t2_right),1) #Servo 2                                           #
    pt2_right=np.rad2deg(t2_right)                                                                   #
                                                                                                     #
    smooth_move(pt4_right,np.rad2deg(t4_right),3) #Servo 4                                           #
    pt4_right=np.rad2deg(t4_right)                                                                   #
                                                                                                     #
    open_gripper(5)                                                                                  #
    open_gripper(11)                                                                                 #
#####################################################################################################
```

Σχήμα 64: Κώδικας μετακίνησης των βραχιόνων στις αρχικές θέσεις

Η συνάρτηση του Σχήματος 65, βοηθάει στην ομαλή κίνηση των σερβοκινητήρων, καθώς μεταβάλλει τις τιμές των μεταβλητών των αρθρώσεων κατά μια μικρή τιμή και εφαρμόζοντας και μια μικρή καθυστέρηση.

```
########################## smooth_move ############################
def smooth_move(pt,t,s):                                          #
                                                                  #
    increment=0.2                                                 #
                                                                  #
    while(pt>=t and (pt<=(90-increment) and pt>=(-90+increment))): #
        kit.servo[s].angle = eval("Theta"+str(s+1))(pt)           #
        pt=pt+increment*(-1)                                      #
        time.sleep(0.01)                                          #
                                                                  #
    while(pt<=t and (pt<=(90-increment) and pt>=(-90+increment))): #
        kit.servo[s].angle = eval("Theta"+str(s+1))(pt)           #
        pt=pt+increment*(1)                                       #
        time.sleep(0.01)                                          #
###################################################################
```

Σχήμα 65: Κώδικας ομαλής κίνησης κινητήρων

Η επόμενη συνάρτηση, του Σχήματος 66, αφορά την υλοποίηση ενός αλγόριθμου ταξινόμησης, και συγκεκριμένα τον αλγόριθμο ευθείας ανταλλαγής ή αλλιώς φυσαλίδας. Χρησιμοποιείται για την εύρεση των κατάλληλων γωνιών που θα πιάσουν οι βραχίονες.

```
############################## bubble_sort ############################
def bubble_sort(list_x,list_y):                                       #
    n = len(list_y)                                                   #
    # optimize code, so if the array is already sorted, it doesn't need #
    # to go through the entire process                                #
    swapped = False                                                   #
    # Traverse through all array elements                             #
    for i in range(n-1):                                              #
        # range(n) also work but outer loop will                      #
        # repeat one time more than needed.                           #
        # Last i elements are already in place                        #
        for j in range(0, n-i-1):                                     #
            # traverse the array from 0 to n-i-1                      #
            # Swap if the element found is greater                    #
            # than the next element                                   #
            if list_y[j] > list_y[j + 1]:                             #
                swapped = True                                        #
                list_y[j], list_y[j + 1] = list_y[j + 1], list_y[j]   #
                list_x[j], list_x[j + 1] = list_x[j + 1], list_x[j]   #
                                                                      #
        if not swapped:                                               #
            # if we haven't needed to make a single swap, we          #
            # can just exit the main loop.                            #
            return                                                    #
#########################################################################
```

Σχήμα 66: Κώδικας αλγορίθμου ευθείας ανταλλαγής

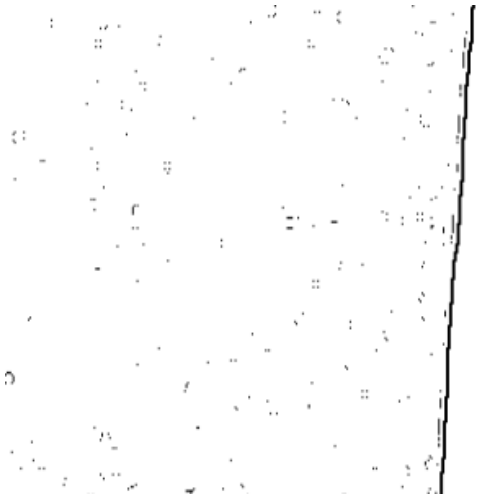
Η συνάρτηση του Σχήματος 67, φτιάχτηκε για να τροφοδοτείται με εικόνες από τη δεύτερη κάμερα του συστήματος. Από αυτές τις εικόνες θα αποφασίζει το σύστημα αν ένα λευκό είδος είναι καθαρό ή λερωμένο, αφού φυσικά τις τροφοδοτήσει στον αλγόριθμό απόφασης του συστήματος. Για να το επιτύχει αυτό εφαρμόζει, στην εικόνα που δέχεται, ένα φίλτρο μεσαίας τιμής, χρησιμοποιώντας τη συνάρτηση

color_recognition_2 για την απομόνωση του λευκού χρώματος. Έτσι μετατρέπει την εικόνα σε κλίμακα του γκρι, στη συνέχεια σε δυαδική, αλλάζει το μέγεθος της, κάνει μια μικρή περικοπή ώστε να μην φαίνονται οι end-effectors και την αποθηκεύει.

```
################################################ take_picture ################################################
def take_picture(frame2):                                                                                    #
                                                                                                             #
    frame2 = ndimage.median_filter(frame2, size=4)                                                           #
                                                                                                             #
    white_frame2=color_recognition_2(frame2)                                                                 #
                                                                                                             #
    white_frame2 = cv2.cvtColor(white_frame2, cv2.COLOR_BGR2GRAY)                                            #
                                                                                                             #
    white_frame2 = cv2.adaptiveThreshold(white_frame2,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,5,2) #
                                                                                                             #
    white_frame2 = cv2.resize(white_frame2, (600,600))                                                       #
                                                                                                             #
    white_frame2 = white_frame2[200:500,150:450]                                                             #
                                                                                                             #
    img_name = "/home/pi/Desktop/Prediction_Pictures/frame.png"                                              #
    cv2.imwrite(img_name, white_frame2)                                                                      #
                                                                                                             #
    cap2.release()                                                                                           #
############################################################################################################
```

Σχήμα 67: Κώδικας τροφοδότησης εικόνων 2ης κάμερας

Η εικόνα που επιστρέφει η συνάρτηση take_picture απεικονίζεται στο Σχήμα 68.



Σχήμα 68: Εικόνα συνάρτησης take picture

Η τελευταία συνάρτηση του συστήματος, Σχήμα 69, κατηγοριοποιεί το λευκό είδος σε καθαρό ή λερωμένο, χρησιμοποιώντας τον αλγόριθμο του νευρωνικού δικτύου, που αναφέρθηκε παραπάνω και επιστρέφει την πρόβλεψη που έκανε.

```
################################################## prediction ##################################################
def prediction():                                                                                             #
    execution_path = os.getcwd()                                                                              #
                                                                                                              #
    prediction = CustomImageClassification()                                                                  #
    prediction.setModelTypeAsInceptionV3()                                                                    #
    prediction.setModelPath("/home/pi/Desktop/CNN_Model/binary_model.h5")                                     #
    prediction.setJsonPath("/home/pi/Desktop/CNN_Model/binary_model_class.json")                              #
    prediction.loadModel(num_objects=2)                                                                       #
                                                                                                              #
    predictions, probabilities = prediction.classifyImage("/home/pi/Desktop/Prediction_Pictures/frame.png", result_count=2)#
                                                                                                              #
    for eachPrediction, eachProbability in zip(predictions, probabilities):                                   #
        print(eachPrediction , " : " , eachProbability)                                                       #
                                                                                                              #
    if(probabilities[0]>probabilities[1]):                                                                    #
        prediction = predictions[0]                                                                           #
    else:                                                                                                     #
        prediction = predictions[1]                                                                           #
                                                                                                              #
    return prediction                                                                                         #
###############################################################################################################
```

Σχήμα 69: Κώδικας κατηγοριοποίησης λευκού είδους

Στον κώδικα του Σχήματος 70, αρχικά δημιουργείται ο συντελεστής για τη μετατροπή από εικονοστοιχεία σε εκατοστά. Και στη συνέχεια για κάθε ρομποτικό βραχίονα, υπολογίζεται ο πίνακας μετασχηματισμού σε σχέση με την κάμερα.

```
cm_to_pixel=47.0/640.0

########################################### Right Arm ##########################################
R180_X_Right=[[1,0,0],[0,np.cos(np.pi),-np.sin(np.pi)],[0,np.sin(np.pi),np.cos(np.pi)]]           #
r89_Right=np.deg2rad(89)                                                                          #
R_Z_Right=[[np.cos(r89_Right),-np.sin(r89_Right),0],[np.sin(r89_Right),np.cos(r89_Right),0],[0,0,1]]#
R0_C_Right=np.dot(R180_X_Right,R_Z_Right)                                                         #
                                                                                                  #
d0_C_Right=[[42.5],[43.0],[0.0]]                                                                  #
                                                                                                  #
H0_C_Right=np.concatenate((R0_C_Right,d0_C_Right),1)                                              #
H0_C_Right=np.concatenate((H0_C_Right,[[0,0,0,1]]),0)                                             #
###################################################################################################

########################################### Left Arm ###########################################
R180_X_Left=[[1,0,0],[0,np.cos(np.pi),-np.sin(np.pi)],[0,np.sin(np.pi),np.cos(np.pi)]]         #
r89_Left=np.deg2rad(89)                                                                        #
R_Z_Left=[[np.cos(r89_Left),-np.sin(r89_Left),0],[np.sin(r89_Left),np.cos(r89_Left),0],[0,0,1]]#
R0_C_Left=np.dot(R180_X_Left,R_Z_Left)                                                         #
                                                                                               #
d0_C_Left=[[40.0],[4.0],[0.0]]                                                                 #
                                                                                               #
H0_C_Left=np.concatenate((R0_C_Left,d0_C_Left),1)                                              #
H0_C_Left=np.concatenate((H0_C_Left,[[0,0,0,1]]),0)                                            #
################################################################################################
```

Σχήμα 70: Κώδικας μετατροπής εικονοστοιχείων σε εκατοστά, υπολογισμός πίνακα μετασχηματισμού

Στο επόμενο κομμάτι κώδικα δημιουργούνται τέσσερα threads, ώστε όταν οι βραχίονες πιάσουν τις γωνίες του λευκού είδους, να το ανασηκώσουν ταυτόχρονα. Επιπλέον δημιουργούνται δυο αντικείμενα VideoCapture που επιτρέπουν να χρησιμοποιηθούν οι κάμερες.

```
#################################################################################################################
threads_1=[threading.Thread(target=take_position_left_1),threading.Thread(target=take_position_right_1)] #
threads_2=[threading.Thread(target=take_position_left_2),threading.Thread(target=take_position_right_2)] #
threads_3=[threading.Thread(target=take_position_left_3),threading.Thread(target=take_position_right_3)] #
threads_4=[threading.Thread(target=take_position_left_4),threading.Thread(target=take_position_right_4)] #
#################################################################################################################

cap1=cv2.VideoCapture(0)
cap2=cv2.VideoCapture(1)
```

Σχήμα 71: Κώδικας ταυτόχρονης ανασήκωσης

Στον κώδικα του Σχήματος 72, αρχικοποιούνται οι βραχίονες στην αρχική τους θέση. Λαμβάνεται μια εικόνα με την πρώτη κάμερα, μετατρέπεται σε κλίμακα του γκρι και

εμφανίζεται. Μόλις πατηθεί το πλήκτρο ESC, τότε θα τερματιστεί η επανάληψη και θα συνεχιστούν οι επόμενες εντολές του προγράμματος. Επίσης, γίνονται κάποιες αρχικοποιήσεις μεταβλητών με την τιμή μηδέν. Αυτές χρησιμεύουν στην ομαλή κίνηση των βραχιόνων.

```
#########################################################################
while(1):                                                               #
    initial_position()                                                  #
                                                                        #
    _,frame1=cap1.read()                                                #
                                                                        #
    gray_image_11=cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)               #
                                                                        #
    cv2.imshow('Background 1',gray_image_11)                            #
                                                                        #
    k=cv2.waitKey(5)                                                    #
    if(k==27):                                                          #
        break                                                           #
#########################################################################

pt1_left=pt2_left=pt3_left=pt4_left=pt5_left=pt1_right=pt2_right=pt3_right=pt4_right=pt5_right=0
```

Σχήμα 72: Κώδικας συστήματος (1)

Το επόμενο κομμάτι κώδικα του Σχήματος 73, αφορά το κύριο μέρος του προγράμματος. Ξεκινώντας η επανάληψη, αφού δημιουργηθούν κάποιες λίστες που θα αποθηκεύουν τις συντεταγμένες των γωνιών, αρχίζει ο έλεγχος, επάνω στην φωτογραφία που θα τραβήξει η πρώτη κάμερα, για τον εντοπισμό των γωνιών με τη συνάρτηση Harris. Επίσης, σε αυτό το κομμάτι γίνεται και η μετατροπή από εικονοστοιχεία σε εκατοστά. Ο αλγόριθμος ρυθμίστηκε κατάλληλα, ώστε να συνεχίσει στις επόμενες εντολές του προγράμματος μόνο εάν βρει τέσσερεις γωνίες. Αυτό έγινε για την καλή λειτουργία του ρομποτικού συστήματος λόγω των περιορισμών που αναφέρονται στην ενότητα 5.4.

Στις επόμενες σειρές του κώδικα, ταξινομήθηκαν οι συνταγμένες κατά τον άξονα Y, ώστε να αποφασίσει το σύστημα και να στείλει τους βραχίονες να πιάσουν τις πάνω γωνίες και κάθε βραχίονας να πιάσει τη γωνία που είναι κοντύτερα σε αυτόν. Αφού αποφασιστεί με ποια γωνία θα αλληλεπιδράσει ο κάθε βραχίονας, στη συνέχεια γίνεται η μετατροπή των συνταγμένων σε σχέση με τους βραχίονες, αφού μέχρι εκείνη τη στιγμή ο άξονας (0,0) των μετρήσεων ήταν βάση της πρώτης κάμερας του συστήματος.

```
##########################################################################################
while(1):
    print("Starting.............")

    X_Location=[]
    Y_Location=[]
    PC=[]

    corner_count=0
    while(corner_count<4):
        _,frame1=cap1.read()
        white_frame1=color_recognition(frame1)
        X_Location.clear()
        Y_Location.clear()
        for i in harris(white_frame1):
            y,x=i
            x=x*cm_to_pixel
            y=y*cm_to_pixel
            X_Location.append(x)
            Y_Location.append(y)
        corner_count=len(X_Location)
        print("Corners detection.............")

    print("Found ",corner_count," corners")

    bubble_sort(X_Location,Y_Location)

    if(X_Location[0]<=320*cm_to_pixel):
        PC_left=[[X_Location[0]],[Y_Location[0]],[0],[1]]
        PC_right=[[X_Location[1]],[Y_Location[1]],[0],[1]]
    else:
        PC_left=[[X_Location[1]],[Y_Location[1]],[0],[1]]
        PC_right=[[X_Location[0]],[Y_Location[0]],[0],[1]]

    P0_Left=np.dot(H0_C_Left,PC_left)
    X0_left=P0_Left[0]
    Y0_left=P0_Left[1]
    Z0_left=0

    P0_Right=np.dot(H0_C_Right,PC_right)
    X0_right=P0_Right[0]
    Y0_right=P0_Right[1]
    Z0_right=0

    print("Left arm corner coordinates: ","(",X0 left,",",Y0 left,",",Z0 left,")")
```

Σχήμα 73: Κώδικας συστήματος (2)

Η συνέχεια του κυρίου κώδικα απεικονίζεται στο Σχήμα 74. Εδώ τροφοδοτούμε τη συνάρτηση που εκτελεί την αντίστροφη κινηματική με τις συντεταγμένες του σημείου ενδιαφέροντος, και για τους δυο βραχίονες, και στη συνέχεια μετακινείται ο end-effector σε αυτό το σημείο χρησιμοποιώντας τις τιμές των παραμέτρων των αρθρώσεων που έχει επιστρέψει η συνάρτηση της αντίστροφης κινηματικής, σε συνδυασμό με τη συνάρτηση smooth_move για την ομαλή κίνηση αυτών. Ακόμη, οι end-effectors κλείνουν, ώστε να πιάσουν τις γωνίες.

```
print("Right arm corner coordinates: ","(",X0_right,",",Y0_right,",",Z0_right,")")

t1_left,t2_left,t3_left,t4_left,t5_left = inverse_kinematics(X0_left,Y0_left,2.0,np.pi/2)
print("Left arm angles",np.rad2deg(t1_left),np.rad2deg(t2_left),np.rad2deg(t3_left),np.rad2deg(t4_left),np.rad2deg(t5_left))

t1_right,t2_right,t3_right,t4_right,t5_right = inverse_kinematics(X0_right,Y0_right,1.1,np.pi/2)
print("Right arm angles",np.rad2deg(t1_right),np.rad2deg(t2_right),np.rad2deg(t3_right),np.rad2deg(t4_right),np.rad2deg(t5_right))

smooth_move(pt1_left,np.rad2deg(t1_left),6) #Servo 7
pt1_left=np.rad2deg(t1_left)

smooth_move(pt5_left,(np.rad2deg(t5_left)),10) #Servo 11
pt5_left=np.rad2deg(t5_left)

smooth_move(pt3_left,np.rad2deg(t3_left),8) #Servo 9
pt3_left=np.rad2deg(t3_left)

smooth_move(pt2_left,np.rad2deg(t2_left),7) #Servo 8
pt2_left=np.rad2deg(t2_left)

smooth_move(pt4_left,np.rad2deg(t4_left),9) #Servo 10
pt4_left=np.rad2deg(t4_left)

smooth_move(pt1_right,np.rad2deg(t1_right),0) #Servo 1
pt1_right=np.rad2deg(t1_right)

smooth_move(pt5_right,(np.rad2deg(t5_right)),4) #Servo 5
pt5_right=np.rad2deg(t5_right)

smooth_move(pt3_right,np.rad2deg(t3_right),2) #Servo 3
pt3_right=np.rad2deg(t3_right)

smooth_move(pt2_right,np.rad2deg(t2_right),1) #Servo 2
pt2_right=np.rad2deg(t2_right)

smooth_move(pt4_right,np.rad2deg(t4_right),3) #Servo 4
pt4_right=np.rad2deg(t4_right)

time.sleep(2)
close_gripper(5)
close_gripper(11)
time.sleep(2)
```

Σχήμα 74: Κώδικας συστήματος (3)

Στη συνέχεια, αφού οι βραχίονες έχουν πιάσει τις γωνίες, ακολουθεί ο κώδικας του Σχήματος 75, ο οποίος δίνει εντολή και αρχίζουν και ανασηκώνουν το λευκό είδος ώστε να βρεθεί στο πεδίο θέασης της δεύτερης κάμερας.

```
t1_left,t2_left,t3_left,t4_left,t5_left = take_position()
t1_right,t2_right,t3_right,t4_right,t5_right = take_position()

for th1 in threads_1:
    th1.start()

for th1 in threads_1:
    th1.join()

for th2 in threads_2:
    th2.start()

for th2 in threads_2:
    th2.join()

for th3 in threads_3:
    th3.start()

for th3 in threads_3:
    th3.join()

for th4 in threads_4:
    th4.start()

for th4 in threads_4:
    th4.join()
```

Σχήμα 75: Κώδικας συστήματος (4)

Στο Σχήμα 76 παρουσιάζεται το τελευταίο κομμάτι κώδικα. Η δεύτερη κάμερα αφού τραβήξει μια φωτογραφία το λευκό είδος, τροφοδοτεί με την εικόνα τη συνάρτηση take_picture. Αυτή με τη σειρά της επιστρέφει την εικόνα κατάλληλα επεξεργασμένη,

ώστε να τη χρησιμοποιήσει η συνάρτηση prediction και να επιστρέψει αν είναι καθαρό ή λερωμένο το λευκό είδος. Στη συνέχεια, μεταφέρει στον κατάλληλο χώρο συλλογής το λευκό είδος και επιστρέφουν οι βραχίονες στην αρχική τους θέση. Το πρόγραμμα σταματά τη λειτουργία του εκεί καθώς δεν κρίθηκε απαραίτητο στο πλαίσια αυτή της εργασίας, το σύστημα να αναμένει συνεχώς νέα λευκά είδη για διαχωρισμό.

```
        _,frame2=cap2.read()
        take_picture(frame2)

        p=prediction()

        if(p == 'clean'):
            print("Towel is clean")
            open_gripper(11)
            time.sleep(5)
            move_right()
            time.sleep(5)
            open_gripper(5)
        else:
            print("Towel is dirty")
            open_gripper(5)
            time.sleep(5)
            move_left()
            time.sleep(5)
            open_gripper(11)

        time.sleep(2)

        initial_position()

        print("End process")

        break

        k=cv2.waitKey(5)
        if(k==27):
            break

cv2.destroyAllWindows()
```

Σχήμα 76: Κώδικας συστήματος (5)

Στο Σχήμα 77 απεικονίζεται η έξοδος του συστήματος για μια κατηγοριοποίηση ενός λευκού είδους με απόλυτη επιτυχία.

```
Starting.............
Corners detection............
Found  4  corners
Left arm corner coordinates:  ( [19.53692311] , [-14.49900992] , 0 )
Right arm corner coordinates:  ( [22.40037647] , [12.09450614] , 0 )
Left arm angles [-36.58035681] [59.23201199] [-47.04516825] 1.0 [77.81315626]
Right arm angles [28.36568526] [69.70378721] [-62.52310588] 1.0 [82.81931867]
clean  :  72.83953428268433
dirty  :  27.160465717315674
Towel is clean
End process
```

Σχήμα 77: Έξοδος του συστήματος

5.4 Προσθήκη στερεοσκοπικών καμερών & βιομηχανικοί βραχίονες

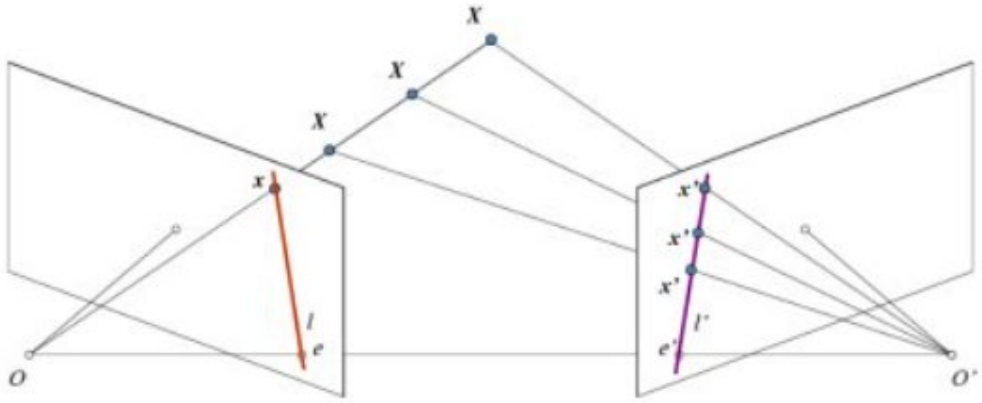
Στο ξεκίνημα του σχεδιασμού και υλοποίησης του προτεινόμενου συστήματος, η ιδέα ήταν, τροφοδοτώντας το σύστημα με ένα λευκό είδος, να μπορεί να βρίσκει τις γωνίες του, ακόμη και αν ήταν τυλιγμένο, να το τεντώνει και να το κατηγοριοποιεί κατάλληλα. Δυστυχώς με τον υπάρχοντα εξοπλισμό αυτό κατέστη αδύνατον. Γι' αυτό το λόγο, το σύστημα κατασκευάστηκε με την παραδοχή ότι θα τροφοδοτείται κάθε φορά με ένα λευκό είδος μικρό σε μέγεθος, με ευδιάκριτες όλες του τις γωνίες του. Στις επόμενες παραγράφους αναλύεται αυτή η παραδοχή.

Κατά τη κατασκευή του ρομποτικού συστήματος, όταν έφτασε η στιγμή να δοκιμαστεί η δυνατότητα να πιάσει μια γωνία, παρατηρήθηκε ότι δεν υπήρχε η απαραίτητη δύναμη για να την πιάσει και να τη συγκρατήσει σφιχτά με αποτέλεσμα, τις περισσότερες φορές να πέφτει κάτω. Όπως γίνεται κατανοητό, για να μπορέσει να λειτουργήσει το σύστημα όπως οραματίστηκε, όταν ένας από του βραχίονες έπιανε μια γωνία θα έπρεπε να περιστρέψει το λευκό είδος με χαμηλή ταχύτητα, ώστε να γίνει πιο ευκολά αντιληπτό μια ακόμη γωνία που θα πιάσει ο άλλος βραχίονας. Αυτό δεν κατέστη δυνατόν καθώς ο end-effector δεν μπορούσε να συγκρατήσει το λευκό είδος. Γι' αυτό το λόγο χρειάζονται καλύτεροι ρομποτικοί βραχίονες.

Ο επόμενος και πιο σημαντικός περιορισμός ήταν η έλλειψη στεροσκοπικών καμερών. Χωρίς αυτές δεν υπάρχει τρόπος να δοθεί στο σύστημα η δυνατότητα να «βλέπει» σε τρεις διαστάσεις. Φυσικά, για να δουλέψει σωστά το σύστημα θα χρειαστεί τουλάχιστον 2 στερεοσκοπικές κάμερες, συνολικά, δηλαδή τουλάχιστον 4 ξεχωριστές κάμερες, σε ζεύγη των δύο. Το ιδανικό θα ήταν τα ζεύγη να αποτελούνταν από το ίδιο μοντέλο καμερών.

Όταν λαμβάνεται μια εικόνα από μια απλή κάμερα, μη στερεοσκοπική, χάνεται σημαντική πληροφορία, δηλαδή το βάθος της εικόνας. Συνεπώς, δε μπορούν να προσδιοριστούν οι συντεταγμένες ενός σημείου στον τρισδιάστατο χώρο. Επομένως, μια σημαντική ερώτηση είναι αν μπορεί να βρεθεί το βάθος με μια τέτοια κάμερα. Όπως αναφέρθηκε και παραπάνω, η απάντηση είναι να χρησιμοποιηθούν περισσότερες από μια κάμερες. Τα μάτια του ανθρώπου λειτουργούν με παρόμοιο τρόπο, σαν δηλαδή να χρησιμοποιούνται δυο κάμερες / μάτια που προσδίδουν στεροσκοπική όραση. Αν υπήρχαν επιπλέον κάμερες διαθέσιμες, ιδανικά ίδια μοντέλα, θα μπορούσε να χρησιμοποιηθεί η βιβλιοθήκη OpenCV της python για την υλοποίηση.

Αξίζει να αναφερθούν πρώτα μερικές βασικές έννοιες στη γεωμετρία πολλαπλών προβολών. Στη συνέχεια αναφέρεται η επιπολική (epipolar) γεωμετρία. Το παρακάτω σχήμα παρουσιάζει μια βασική ρύθμιση με δυο κάμερες που παίρνουν εικόνα της ίδιας σκηνής.



Σχήμα 78: Δύο κάμερες λαμβάνουν την ίδια σκηνή [24]

Αν χρησιμοποιούνταν μόνο η αριστερή κάμερα, δεν θα μπορούσε να βρεθεί το τρισδιάστατο σημείο που αντιστοιχεί στο σημείο X του σχήματος 78, επειδή κάθε σημείο της γραμμής OX προβάλλεται στο ίδιο σημείο στο επίπεδο της εικόνας. Αλλά αν ληφθεί υπόψη και η δεξιά εικόνα, διαφορετικά σημεία της ευθείας OX προβάλλονται σε διαφορετικά σημεία (x') στο δεξί επίπεδο. Έτσι, με αυτές τις δυο εικόνες, μπορεί να τριγωνοποιηθεί το σωστό τρισδιάστατο σημείο. Αυτή είναι η γενική ιδέα.

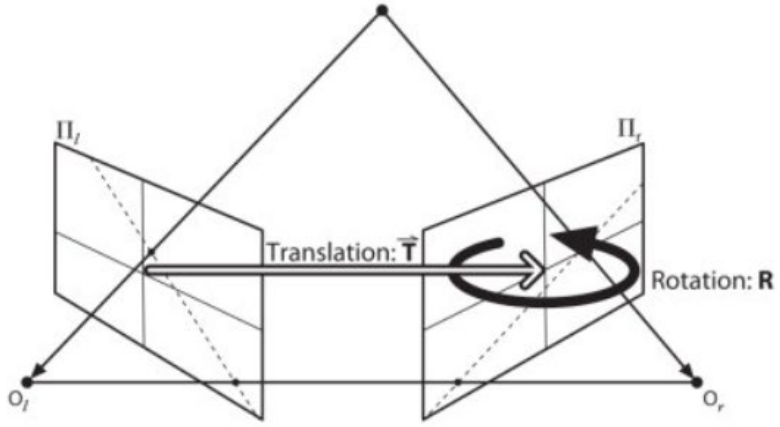
Οι προβολές των διαφορετικών σημείων στην OX σχηματίζουν μια ευθεία στο δεξιό επίπεδο (γραμμή ℓ'). Η ευθεία αυτή ονομάζεται επιπολική γραμμή (epiline) που αντιστοιχεί στο σημείο x. Σημαίνει ότι για να βρεθεί το σημείο x στη δεξιά εικόνα, θα αναζητηθεί κατά το μήκος αυτής της επιπολικής γραμμής και θα πρέπει επίσης να είναι κάπου πάνω σε αυτή τη γραμμή. Άρα, για να βρεθεί το σημείο που ταιριάζει στην άλλη εικόνα, δεν χρειάζεται να γίνει αναζήτηση ολόκληρης της εικόνας, απλώς αναζήτηση κατά μήκος της επιπολικής γραμμής. Θα έχει καλύτερη απόδοση και ακρίβεια. Αυτό ονομάζεται Επιπολικός Περιορισμός (Epipolar Constraint). Όμοια όλα τα σημεία θα έχουν τις αντίστοιχες επιπολικές γραμμές τους στην άλλη εικόνα. Το επίπεδο XOO′ ονομάζεται Επιπολικό Επίπεδο (Epipolar Plane).

Τα O και O′ είναι τα κέντρα των καμερών. Από το Σχήμα 78, παρατηρείται ότι η προβολή της δεξιάς κάμερας O′ φαίνεται στην αριστερή εικόνα σημείο e. Αυτό ονομάζεται επίπολος (epipole). Επίπολος είναι το σημείο τομής της γραμμής από τα κέντρα των καμερών και των επιπέδων των εικόνων. Ομοίως e' είναι ο επίπολος της αριστερής κάμερας. Σε ορισμένες περιπτώσεις δεν θα είναι δυνατός ο εντοπισμός του επιπόλου καθώς μπορεί να βρίσκεται εκτός της εικόνας, το οποίο σημαίνει ότι η μια κάμερα δεν βλέπει την άλλη.

Από τον επίπολο περνούν όλες οι επιπολικές γραμμές. Συνεπώς για να βρεθεί η θέση του επιπόλου, μπορούν να βρεθούν πολλές επιπολικές γραμμές και έπειτα να βρεθεί το σημείο τομής τους.

Άρα θα πρέπει να βρεθούν οι επιπολικές γραμμές και τα επίπολα. Για να βρεθούν αυτά θα χρειαστούν δυο ακόμη συστατικά, το Fundamental Matrix (F) και το Essential Matrix

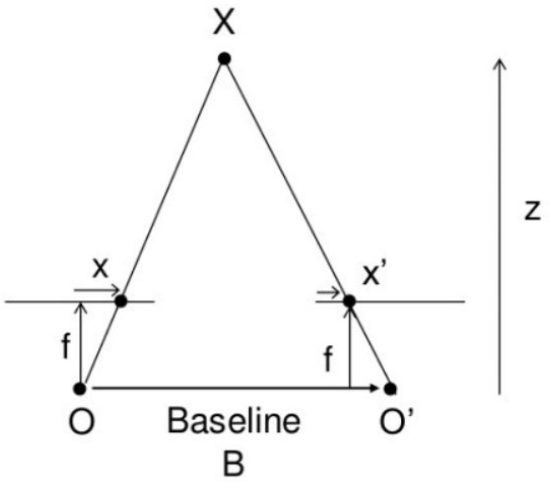
(E). Το Essential Matrix περιέχει πληροφορίες σχετικές με τη μετατόπιση και την περιστροφή, οι οποίες περιγράφουν τη θέση της δεύτερης κάμερας σε σχέση με την πρώτη (Σχήμα 79).



Σχήμα 79: Σχετική θέση της δεύτερης κάμερας σε σχέση με την πρώτη [24]

Το Fundamental Matrix περιέχει τις ίδιες πληροφορίες με το Essential Matrix, και επιπλέον τις πληροφορίες σχετικά με τα εγγενή στοιχεία και των δυο καμερών, έτσι ώστε να μπορούν να συσχετιστούν οι δυο κάμερες σε συντεταγμένες εικονοστοιχείων. Με απλά λόγια, ο Fundamental Matrix, αντιστοιχίζει ένα σημείο σε μια εικόνα σε μια γραμμή (επιπολική) στην άλλη εικόνα. Αυτό υπολογίζεται από τα σημεία που ταιριάζουν και από τις δυο εικόνες. Απαιτούνται τουλάχιστον 8 τέτοια σημεία για να βρεθεί ο Fundamental Matrix. [24]

Στις προηγούμενες παραγράφους αναφέρθηκαν βασικές έννοιες όπως οι επιπολικοί περιορισμοί και άλλοι σχετικοί όροι. Αναφέρθηκε επίσης ότι αν υπάρχουν δυο εικόνες της ίδιας σκηνής, μπορούν να παρθούν πληροφορίες βάθους από αυτές. Στο σχήμα 79 απεικονίζονται μέσω της εικόνας οι σχέσεις των μεταβλητών.



Σχήμα 80: Ισοδύναμα τρίγωνα καμερών [25]

Το παραπάνω σχήμα περιέχει ισοδύναμα τρίγωνα. Γράφοντας τις ισοδύναμες εξισώσεις αυτών των τριγώνων προκύπτει το ακόλουθο αποτέλεσμα:

$$disparity = x - x' = \frac{Bf}{Z}$$

Χ και Χ΄ είναι η απόσταση μεταξύ σημείων στο επίπεδο της εικόνας που αντιστοιχούν στο τρισδιάστατο σημείο της σκηνής και στο κέντρο της κάμερας τους. Το B είναι η απόσταση μεταξύ των δυο καμερών και f είναι η εστιακή απόσταση της κάμερας. Εν συντομία, η παραπάνω εξίσωση λέει ότι το βάθος ενός σημείου σε μια σκηνή είναι αντιστρόφως ανάλογος με τη διαφορά της απόστασης των αντίστοιχων σημείων της εικόνας και των κέντρων της κάμερα τους. Έτσι, με αυτές τις πληροφορίες μπορεί να αντληθεί το βάθος όλων των εικονοστοιχείων σε μια εικόνα. [25]

Με αυτό το τρόπο, λοιπόν βρίσκονται οι αντιστοιχίες μεταξύ δυο εικόνων. Όπως έχει ήδη προαναφερθεί ο περιορισμός της επικλίνης κάνει αυτή τη λειτουργία πιο γρήγορη και ακριβής. Όλα τα παραπάνω υλοποιούνται με τη χρήση της OpenCV.

6 *Συμπεράσματα*

Το ρομπότ της συγκεκριμένης πρότασης, είναι ολοκληρωμένο σε δοκιμαστικό στάδιο. Όμως η απλότητά του, το χαμηλό κόστος σχέσης τιμής - ποιότητας και η μελλοντική δυνατότητα επεκτασιμότητας, είναι τα τρία κύρια στοιχεία, που το θέτουν ως αναγκαιότητα στη ζήτηση της αγοράς. Η πολυχρηστικότητά του σε πολλούς κλάδους, τον καθιστά ιδιαίτερα ανταγωνιστικό, με αποτέλεσμα να μπορεί να βρεθεί εύκολα χρηματοδότηση, καθώς επίσης και η σταδιακή μελλοντική επεκτασιμότητα με βάση προηγούμενες μελέτες, αν αυτό κρίνεται αναγκαίο στο εκάστοτε πεδίο. Φυσικά ένα τέτοιο ρομποτικό σύστημα θα μπορούσε να χρησιμοποιηθεί σε οποιαδήποτε μεγάλη μονάδα πιθανώς και με ελάχιστα διαφοροποιημένες λειτουργίες. Ανάμεσα σε αυτές θα περιλαμβάνονταν τα νοσοκομεία, τα πλοία, οι απομακρυσμένοι χώροι εργασίας και πιθανών μελλοντικά και στο διάστημα.

Ένα ρομπότ αυτού του επιπέδου θα είναι σε θέση πέρα από την εύρεση και το διαχωρισμό των πολύ λερωμένων ρούχων και στο διαχωρισμό αυτών ανάλογα με τις χρωματικές πλύσεις. Όμως πέρα από αυτό θα μπορούσε σίγουρα να ξεχωρίσει και τα είδη (π.χ. τα μπλουζάκια από τις κάλτσες) και συνεπώς να μάθει και τα ανάλογα απορρυπαντικά που χρησιμοποιούνται, ανά είδος ρούχου. Πηγαίνοντας το πλάνο μακρύτερα στο μέλλον θα μπορούσε να αναλάβει πέραν των άλλων, στεγνό καθάρισμα, τόσο σε πανωφόρια, όσα και σε υποδήματα. Ποιος λοιπόν μελλοντικά δε θα ήθελε ένα οικιακό ρομπότ σε θέση να εκτελέσει όλες τις εργασίες και γιατί όχι με καθαρή ενέργεια, καθώς όλη η οικονομία κινείται προς αυτή τη κατεύθυνση; Όσο οδεύει η ανθρωπότητα προς το μέλλον, οι έννοιες της αειφορίας, της βιωσιμότητας και γενικότερα του μηδενικού περιβαλλοντικού αντικτύπου, θα απαρτίζουν μεγαλύτερο μέρος των καθημερινών στόχων

7 *Βιβλιογραφία*

[1] Wang, X., Jiang, X., Zhao, J., Wang, S., Yang, T., & Liu, Y. (2019). Picking towels in point clouds. *Sensors*, *19*(3), 713.

[2] Yang, P. Y., Chang, T. H., Chang, Y. H., & Wu, B. F. (2018, June). Intelligent mobile robot controller design for hotel room service with deep learning arm-based elevator manipulator. In *2018 International Conference on System Science and Engineering (ICSSE)* (pp. 1-6). IEEE.

[3] Mukhopadhyay, P., & Chaudhuri, B. B. (2015). A survey of Hough Transform. *Pattern Recognition*, *48*(3), 993-1010.

[4] Yu, X., Dong, L., Li, L., & Hoe, K. E. (2009, November). Lift-button detection and recognition for service robot in buildings. In *2009 16th IEEE International Conference on Image Processing (ICIP)* (pp. 313-316). IEEE.

[5] Wang, W. J., Huang, C. H., Lai, I. H., & Chen, H. C. (2010, August). A robot arm for pushing elevator buttons. In *Proceedings of SICE Annual Conference 2010* (pp. 1844-1848). IEEE.

[6] Klingbeil, E., Carpenter, B., Russakovsky, O., & Ng, A. Y. (2010, May). Autonomous operation of novel elevators for robot navigation. In *2010 IEEE International Conference on Robotics and Automation* (pp. 751-758). IEEE.

[7] Dong, Z., Zhu, D., & Meng, M. Q. H. (2017, December). An autonomous elevator button recognition system based on convolutional neural networks. In *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)* (pp. 2533-2539). IEEE.

[8] Troniak, D., Sattar, J., Gupta, A., Little, J. J., Chan, W., Calisgan, E., ... & Van der Loos, M. (2013, May). Charlie Rides the Elevator--Integrating Vision, Navigation and Manipulation towards Multi-floor Robot Locomotion. In *2013 international conference on computer and robot vision* (pp. 1-8). IEEE.

[9] Abdulla, A. A., Liu, H., Stoll, N., & Thurow, K. (2016, June). A robust method for elevator operation in semi-outdoor environment for mobile robot transportation system in life science laboratories. In *2016 IEEE 20th Jubilee International Conference on Intelligent Engineering Systems (INES)* (pp. 45-50). IEEE.

[10] Maitin-Shepard, J., Cusumano-Towner, M., Lei, J., & Abbeel, P. (2010, May). Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *2010 IEEE International Conference on Robotics and Automation* (pp. 2308-2315). IEEE.

[11] Αμπελιώτης, Κ. (2016). Το πλύσιμο των ρούχων.

[12] Flores-Hernández, B. R., De Moure-Flores, F., Mayén-Hernández, S. A., & Santos-Cruz, J. (2021, November). Graphene for a green-environmentally methodology with organic surfactants. In 2021 18th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE) (pp. 1-5). IEEE.

[13] Schleich, J. & Hillenbrand, T. (2009). Determinants of residential water demand in Germany. Ecological Economics, 1756-1769

[14] Willis, R.M., Stewart R.A., Giurco, D.P., Talebpour, M.R. & Mousavinejad, A. (2011). End use water consumption in households: impact of socio-demographic factors and efficient devices. Journal of Cleaner Production, 60, 107-115

[15] Robotic manipulator part's model site: thangs.com/designer/m/3d-model/38899

[16] Abaas, T. F., Khleif, A. A., & Abbood, M. Q. (2020). Inverse Kinematics Analysis and Simulation of a 5 DOF Robotic Arm using MATLAB. Al-Khwarizmi Engineering Journal, 16(1), 1-10.

[17] Sivasamy, D., Anand, M. D., & Sheela, K. A. (2019). Robot forward and inverse kinematics research using MATLAB. International Journal of recent technology and engineering, 8(2S3), 29-35.

[18] Zou, L., & Li, Y. (2010, November). A method of stereo vision matching based on OpenCV. In 2010 International conference on audio, language and image processing (pp. 185-190). IEEE.

[19] Kucuk, S., & Bingul, Z. (2006). Robot kinematics: Forward and inverse kinematics. London, UK: INTECH Open Access Publisher.

[20] Aristidou, A., & Lasenby, J. (2009). Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver.

[21] Tokarz, K., & Kieltyka, S. (2010, July). Geometric approach to inverse kinematics for arm manipulator. In *WSEAS International Conference on Systems: Part of the 14th WSEAS CSCC Multiconference* (Vol. 2, pp. 682-687).

[22] Bisong, E., & Bisong, E. (2019). Google colaboratory. Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners, 59-64.

[23] Ye, Z., Pei, Y., & Shi, J. (2009, October). An improved algorithm for Harris corner detection. In 2009 2nd International Congress on Image and Signal Processing (pp. 1-4). IEEE.

[24] Epipolar Geometry. Retrieved from https://docs.opencv.org/3.4/da/de9/tutorial_py_epipolar_geometry.html

[25] Depth Map from Stereo Images. Retrieved from https://docs.opencv.org/3.4/dd/d53/tutorial_py_depthmap.html

[26] Mariottini, G. L., Oriolo, G., & Prattichizzo, D. (2007). Image-based visual servoing for nonholonomic mobile robots using epipolar geometry. IEEE Transactions on Robotics, 23(1), 87-100.

[27] Pomaska, G. (2019). Stereo Vision Applying OpenCV and Raspberry Pi. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, 42, 265-269.

[28] Oshima, T., Yoshimi, T., Mizukawa, M., & Ando, Y. (2014, October). A study of towel folding by a robot arm-Spreading and vertex detection using image processing. In *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)* (pp. 627-631). IEEE.

[29] Rokbani, N., Neji, B., Slim, M., Mirjalili, S., & Ghandour, R. (2022). A Multi-Objective Modified PSO for Inverse Kinematics of a 5-DOF Robotic Arm. *Applied Sciences*, *12*(14), 7091.

[30] AbuQassem, M. R. (2010). Simulation and Interfacing of 5 DOF Educational Robot Arm. *Islamic University of Gaza*.

[31] Agustian, I., Daratha, N., Faurina, R., & Suandi, A. (2021). Robot Manipulator Control with Inverse Kinematics PD-Pseudoinverse Jacobian and Forward Kinematics Denavit Hartenberg. *arXiv preprint arXiv:2103.10461*.

[32] Cheon, S., Ryu, K., & Oh, Y. (2013, October). Object manipulation using robot arm-hand system. In *2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)* (pp. 163-166). IEEE.