ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Εντοπισμός θέσης αυτοκινούμενου ρομπότ με την βοήθεια κάμερας και ταμπελών σήμανσης θέσης αφετηρίας-στόχου»

Γεώργιος Ελευθεριάδης Α.Μ. 66

Εργασία που υποβλήθηκε στο
Πρόγραμμα Μεταπτυχιακών Σπουδών στη Ρομποτική,
του Διεθνούς Πανεπιστημίου της Ελλάδος,
για τη μερική εκπλήρωση υποχρεώσεων για το Δίπλωμα Ειδίκευσης στη Ρομποτική

Επιβλέπων Καθηγητής: Δρ. ΣπυρίδωνΑ. Καζαρλής

Πίνακας περιεχομένων

ΕΥΧΑΡΙΣΤΙΕΣ

Αυτή η εργασία σηματοδοτεί το τέλος των μεταπτυχιακών σπουδών μου. Θα ήθελα να ευχαριστήσω τον Δρ. Σπυρίδωνα Α. Καζαρλή για την εμπιστοσύνη και καθοδήγηση κατά την διάρκεια εκπόνησης της παρούσας διπλωματικής εργασίας. Θα ήθελα ακόμα να ευχαριστήσω κάθε καθηγητή του Π.Μ.Σ ρομποτικής για την προσπάθεια και το ενδιαφέρον που έδειξαν κατά τη διάρκεια των σπουδών μου σε αυτό το τμήμα.

Ακόμα θέλω να ευχαριστήσω τους γονείς μου Θεόφιλο και Στέλλα και την αδερφή μου Μαριάννα που με υποστήριζαν και συνεχίζουν να με υποστηρίζουν με την αμέριστη αγάπη τους, την εμψύχωση μέσα από τα λόγια και τις συμβουλές τους που μου δίνουν πάντα το κουράγιο και τη δύναμη να πετυχαίνω τους στόχους μου.

Τέλος θέλω να ευχαριστήσω την κοπέλα μου Άννα που στάθηκε δίπλα μου υποστηρίζοντας με κάθε τρόπο σε αυτό το ταξίδι μου.

Abstract

This thesis presents a new robot localization system that utilizes a single RGB camera and two colored square tags to determine the position and orientation of a mobile robot in its surroundings.

The system captures multiple images of the environment by rotating the camera, taking pictures and processing these images to extract visual features to be used for identifying the tags' location. The position and orientation of the robot are then calculated based on the relative positions of the tags in the images.

The current implementation has unstable accuracy which can cause the robot make mistaken estimations in certain positions, mostly due to the size of the tags. The system's simplicity, low cost, and applicability make it an attractive approach to robot localization.

Future work could explore the use of multiple cameras for improved accuracy and robustness, as well as the integration of the localization system with other robotic tasks such as mapping and navigation. Another option would be to use a 360° camera and use a stream instead of pictures for continuous localization.

Overall, the proposed robot localization system offers a promising approach for determining the position and orientation of mobile robots using RGB camera(s) and two colored square tags.

Περίληψη

Αυτή η διπλωματική εργασία παρουσιάζει ένα νέο σύστημα εντοπισμού θέσης ρομπότ που χρησιμοποιεί μια μόνο RGB κάμερα και δύο τετράγωνεςταμπέλες διαφορετικού χρώματος για να προσδιορίσει τη θέση και την προσανατολισμό ενός κινητού ρομπότ στο περιβάλλον του.

Το σύστημα αυτό καταγράφει πολλαπλές εικόνες του περιβάλλοντος περιστρέφοντας την κάμερα, λαμβάνοντας φωτογραφίες και επεξεργαζόμενο τις εικόνες αυτές για να εξάγει οπτικά χαρακτηριστικά που θα χρησιμοποιηθούν για την αναγνώριση της θέσης των ταμπελών. Η θέση και ο προσανατολισμός του ρομπότ υπολογίζονται στη συνέχεια βάσει των σχετικών θέσεων των ταμπελών στις εικόνες.

Η τρέχουσα υλοποίηση έχει ασταθή ακρίβεια που μπορεί να οδηγήσει το ρομπότ σε λανθασμένο συμπέρασμα για την τοποθεσία του σε ορισμένες θέσεις. Η απλότητα, το χαμηλή κόστος και η εφαρμοσιμότητα του συστήματος το καθιστούν μια ελκυστική προσέγγιση για τον εντοπισμό ρομπότ.

Μια πιθανή μελλοντική κατεύθυνση για την βελτίωση της ακρίβειας και της αξιοπιστίας του συστήματος θα μπορούσε να είναι η χρήση πολλαπλών καμερών για τον παραλληλισμό των εικόνων και την ενίσχυση της ανθεκτικότητας σε πιθανούς περιορισμούς της μοναδικής κάμερας. Επιπλέον, μια πιο ολοκληρωμένη προσέγγιση θα μπορούσε να ενσωματώσει το σύστημα εντοπισμού θέσης με άλλες ρομποτικές εργασίες, όπως η κατασκευή χαρτών και η πλοήγηση του ρομπότ. Μια άλλη επιλογή θα ήταν η χρήση μιας κάμερας 360 ° και η χρήση ροής εικόνας αντί για φωτογραφίες για συνεχή εντοπισμό.

Συνολικά, το προτεινόμενο σύστημα εντοπισμού θέσης του ρομπότ προσφέρει μια προσέγγιση με προοπτικές για τον προσδιορισμό της θέσης και του προσανατολισμού κινητών ρομπότ χρησιμοποιώνταςκάμερα-εςRGB και δύο ταμπέλες σήμανσης κάποιου χρώματος.

1. Εισαγωγή

1.1 Περιγραφή προβλήματος

Ένα σημαντικό πρόβλημα των κινούμενων ρομπότ είναι η εύρεση της τοποθεσίας τους μέσα στον χώρο. Το πρόβλημα αυτό χρειάζεται λύσεις με ακρίβεια ώστε το ρομπότ να μπορεί να τελέσει τα καθήκοντα του αυτόνομα ή με το βοήθεια κάποιου χειριστή. Υπάρχουν διάφορες προσεγγίσεις σε αυτό το πρόβλημα που ποικίλουν ως εφαρμογές βάση των απαιτήσεων που προκύπτουν.

1.2 Στόχος εργασίας

Η εργασία αυτή χωρίζεται σε δύο τμήματα. Το πρώτο είναι η κατασκευή ενός δίτροχου ρομπότ που θα επικοινωνεί και θα χειρίζεται μέσω ενός υπολογιστή και το δεύτερο είναι η ανάπτυξη λογισμικού που θα επεξεργάζεται τις εικόνες που θα λαμβάνονται από μια rgb κάμερα συνδεδεμένη στο παραπάνω ρομπότ εντός ενός πειραματικού χώρου που θα χρησιμοποιεί ταμπέλες σήμανσης για να βρεθεί η θέση του ρομπότ στον χώρο. Η επικοινωνία του ρομπότ με τον υπολογιστή επιλέχθηκε να επιτευχθεί μέσω τοπικού δικτύου με τη χρήση του προγράμματος RealVNC. Το ρομπότ λειτουργεί βασισμένο στο Raspberry Pi 4 single-board computer και για τους κινητήρες χρησιμοποιείται ο μικροελεγκτής Arduino Uno. Για την ανάπτυξη λογισμικού χρησιμοποιείται η γλώσσα Arduino, η γλώσσα Pythonμαζί με τη βιβλιοθήκη της OpenCV.

1.3 Δομή εργασίας

Η παρούσα εργασία αποτελείται από 5 κεφάλαια τα οποία εξετάζουν το πρόβλημα της εργασίας. Τα κεφάλαια καθορίζονται ως εξής:

Κεφάλαιο πρώτο: Σύντομη περιγραφή του προβλήματος που καλείται να λύσει η παρούσα διπλωματική εργασία, εξετάζονται οι στόχοι της εργασίας και γίνεται επεξήγηση για τον τρόπο δόμησης της εργασίας.

Κεφάλαιο δεύτερο:Το δεύτερο κεφάλαιο περιέχει μια ανασκόπηση στην λειτουργία των ρομπότ διαφορικής κίνησης και σε διάφορους τρόπους εντοπισμού θέσης ενός ρομπότ με έμφαση στον εντοπισμό με χρήση κάμερας και σήμανσης.

Κεφάλαιο τρίτο: Αυτό το κεφάλαιο περιλαμβάνει το γνωστικό υπόβαθρο της επιστήμης της υπολογιστικής όρασης καθώς και γνωστά προβλήματα όπως αυτό της βαθμονόμησης κάμερας.

Κεφάλαιο τέταρτο: Το κεφάλαιο αυτό αποτελεί την επεξήγηση και διαδικασία μέσω της οποίας λύνεται το πρόβλημα της παρούσας διπλωματικής εργασίας.

Κεφάλιο πέμπτο: Το κεφάλαιο αυτό αποτελείται από αποτελέσματα που πάρθηκαν μετά από πειράματα με τη χρήση του αλγορίθμου που αναπτύχθηκε για τη λύση του προβλήματος εντοπισμού θέσης. Ακόμα γίνεται μια σύντομη αναφορά στα συμπεράσματα που προκύπτουν από την παρούσα εργασία και προτάσεις για μελλοντική επέκταση του projectσε ένα πιο «έξυπνο» και εκσυγχρονισμένο σύστημα και πλαίσιο.

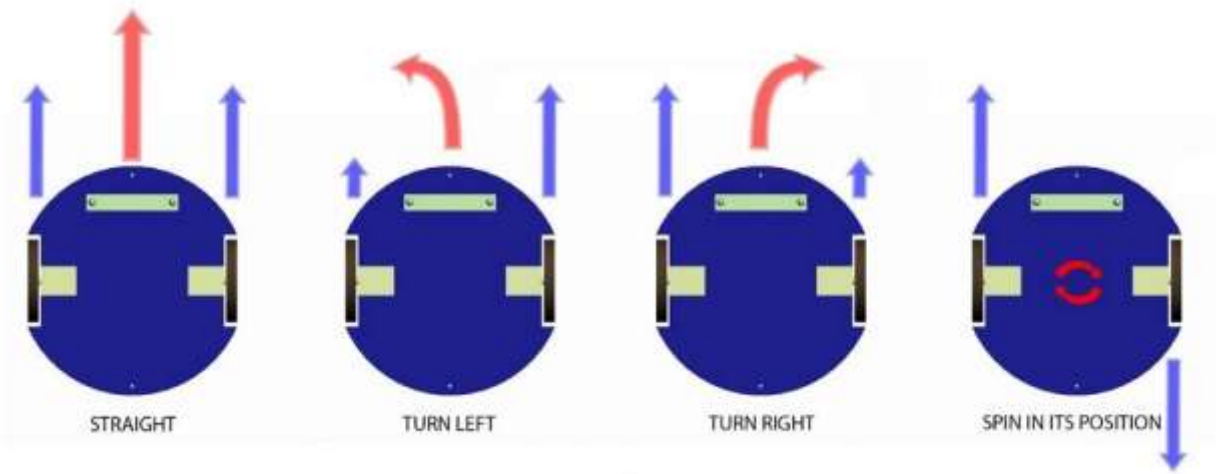
2. Ανασκόπηση ερευνητικής περιοχής

2.1 Ρομπότ διαφορικής κίνησης

Τα οχήματα διαφορικής κίνησης είναι μια κατηγορία κινούμενων ρομπότ που κινούνται με τη βοήθεια δύο τροχών καθ' ένας από τους οποίους περιστρέφεται με τη βοήθεια ενός κινητήρα. Αυτό επιτρέπει τον ακριβή έλεγχο της ταχύτητας και της κατεύθυνσης του ρομπότ στον χώρο. Οι τροχοί τοποθετούνται παράλληλα στο σώμα του ρομπότ και το ρομπότ κινείται ελέγχοντας την ταχύτητα και κατεύθυνση περιστροφής του κάθε τροχού ξεχωριστά. Ένα σημαντικό πλεονέκτημα αυτού του είδους κινούμενου ρομπότ είναι η δυνατότητα του να περιστραφεί γύρω από τον εαυτό του κάτι που ονομάζεται «ολονομική κίνηση». Αυτό πετυχαίνεται με την ταυτόχρονη περιστροφή των δύο τροχών του προς αντίθετες κατευθύνσεις με την ίδια σταθερή ταχύτητα. Αυτή η δυνατότητα κάνει τα ρομπότ διαφορικής κίνησης κατάλληλα για δυναμικούς χώρους με απρόβλεπτο περιβάλλον ή χώρους με πολλά εμπόδια λόγω της δυνατότητας τους για εύκολη αλλαγή κατεύθυνσης. Άλλο ένα θετικό αυτής της κατηγορίας ρομπότ είναι ότι η κατασκευή τους είναι εύκολη. Οι τροχοί και οι κινητήρες είναι εξαρτήματα που μπορούν να διατεθούν εύκολα και συχνά έχουν χαμηλό κόστος. Το σώμα / βάση του ρομπότ μπορεί να κατασκευαστεί από πληθώρα υλικών όπως πλαστικό, μέταλλο ή ξύλο. Τα ρομπότ διαφορικής κίνησης έχουν εφαρμογές σε διάφορους τομείς. Ενδεικτικά, κάποιοι από αυτούς είναι:

- Βιομηχανικός αυτοματισμός και μεταφορά υλικών
- Ρομπότ παροχής υπηρεσιών όπως για καθαριότητα και διανομήαγαθών
- Έρευνα και εκπαίδευση στον τομέα της ρομποτικής
- Στρατιωτικές εφαρμογές και εφαρμογές ασφάλειας

Για τον έλεγχο τους υπάρχουν διάφορες σουίτες λογισμικού όπως το ROS (Robot Operating System), το ArduPilot και το NvidiaISAAC που παρέχουν βιβλιοθήκες, εργαλεία και διεπαφές χρήστη για τον έλεγχο των κινητήρων-τροχών, των αισθητήρων και άλλων ηλεκτρικών εξαρτημάτων. Ακόμα οι σουίτες αυτές προσφέρουν υψηλού επιπέδου αλγόριθμους ελέγχου που μπορούν να χρησιμοποιηθούν γιατων έλεγχο αισθητήρων, την πλοήγηση του ρομπότ στον χώρο, καθώς και για τον εντοπισμό θέσης του ρομπότ και για σχεδιασμό πορείας ρομπότ.[1]



Εικόνα 1 Όχημα διαφορικής κίνησης
R. Sanketh, "MANUAL ROBOTICS," *Medium*. [Online]. Available: https://medium.com/manual-robotics/drives-76c2b2dac97c.

2.2 Εντοπισμός θέσης κινούμενου οχήματος

Το πρόβλημα εντοπισμού θέσης ενός ρομπότ είναι ένα σημαντικό πρόβλημα της ρομποτικής που έχει διαφορετικές λύσεις και προσεγγίσεις που εξαρτώνται από την εφαρμογή, το κόστος και την ακρίβεια που θέλουμε να πετύχουμε. Με τον όρο "εντοπισμός θέσης" νοείται η δυνατότητα του ρομπότ να αναγνωρίσει σε ποια θέση βρίσκεται σε μια δεδομένη στιγμή μέσα στον χώρο. Ο χώρος αυτός μπορεί να είναι ολόκληρη η επιφάνεια της Γης, κάτι που μπορεί να βρεθεί από το ρομπότ για παράδειγμα με τη χρήση GPS, ή μπορεί να είναι ένα σχετικός χώρος που καθορίζεται από εμάς, ή που καθορίζεται από το φυσικό περιβάλλον μέσα στο οποίο υπάρχει το ρομπότ. Υπάρχουν αρκετοί τρόποι για τη λύση αυτού του προβλήματος και κάποιοι από τους σημαντικότερους είναι οι παρακάτω [1]:

- Εντοπισμόςθέσηςμεχρήση GPS (Global Positioning System)
- Εντοπισμός θέσης μέσω οδομετρίας
- Εντοπισμός θέσης με λοιπούς αισθητήρες μέτρησης απόστασης (HC-SR04, lidar)
- Εντοπισμός θέσης με κάμερα
- Εντοπισμός θέσης με συνδυασμό των παραπάνω.

2.3 Εντοπισμός με χρήση GPS

Χρησιμοποιώντας έναν πομποδέκτη GPS, το ρομπότ μπορεί με τη βοήθεια δορυφόρων της Γης να ενημερώνεται ζωντανά για τη θέση, τον προσανατολισμό, το υψόμετρο, την ταχύτητα και την κατεύθυνση προς την οποία κινείται. Αυτό συμβαίνει είτε κινείται σε εξωτερικό είτε σε εσωτερικό χώρο. Όπως αναφέρθηκε προηγουμένως όμως, οι διάφορες προσεγγίσεις στη λύση του προβλήματος εντοπισμού θέσεις έχουν και διαφορετικές εφαρμογές. Με αυτή τη μέθοδο, το ρομπότ είναι σε θέση να λάβει τις πληροφορίες θέσεις του με μικρή ακρίβεια, δηλαδή σε μια ακτίνα κάποιων μέτρων. Επομένως σαν μέθοδος είναι ακατάλληλη για τον εντοπισμό θέσης όταν τα περιθώρια ανακρίβειας και λάθους είναι πολύ μικρά. Ταυτόχρονα δεν μπορεί να χρησιμοποιηθεί σε κλειστούς και σκεπασμένους χώρους προσθέτοντας άλλο ένα σημαντικό μειονέκτημα για αρκετές εφαρμογές. Ακόμα η συγκεκριμένη μέθοδος ενώ μας ενημερώνει για τις απόλυτες πληροφορίες σχετικά με την τοποθεσία του ρομπότ στη Γη, δεν μπορεί να μας δώσει επιπλέον πληροφορίες που να συσχετίσουν τη θέση του, με εμπόδια, ανθρώπους και άλλα ρομπότ που δεν χρησιμοποιούν GPS. [1]

2.4 Εντοπισμός με χρήση οδομετρίας

Με τον όρο οδομετρία νοείται η χρήση αισθητήρων κίνησης για τη μέτρηση της κίνησης του ρομπότ με αποτέλεσμα τον υπολογισμό της σχετικής θέσης του ρομπότ σε σύγκριση με την αρχική του θέση. Με αυτό τον τρόπο εντοπισμού θέσης προκειμένου να γνωρίζουμε τη θέση του ρομπότ στον χώρο πρέπει να γνωρίζουμε το σημείο αφετηρίας κίνησης του ρομπότ. Ακόμα με αυτή τη μέθοδο ο εντοπισμός θέσης γίνεται συσσωρευτικά σε βάθος χρόνου λειτουργίας και κίνησης. Σε ένα ρομπότ διαφορικής κίνησης όπως αυτό που κατασκευάστηκε για την εν λόγω εργασία, με τη χρήση οπτικών και μαγνητικών κωδικοποιητών συνδεδεμένων στους τροχούς, μπορεί να ελέγχεται και να καταγράφεται η περιστροφή των δύο τροχών. Βάση των στοιχείων περιστροφής των τροχών μπορεί να λυθεί το ευθύ κινηματικό πρόβλημα και να υπολογίζεται η θέση στην οποία έχει κινηθεί το ρομπότ. Ένα πρόβλημα με αυτή τη μέθοδο είναι πως με τη συσσωρευτική λειτουργία αυτής της μεθόδου, τα σφάλματα από τους αισθητήρες και ο θόρυβος συσσωρεύεται επίσης με αποτέλεσμα σε βάθος χρόνου η συσσώρευση λάθους να δίνει μεγάλη απόκλιση στην ακρίβεια της μέτρησης.

Μέσω του προγράμματος που θα δημιουργηθεί για τον έλεγχο θέσης του ρομπότ με την παραπάνω μέθοδο, με τη χρήση φίλτρων όπως το φίλτρο Kalman, η συσσώρευση λάθους μπορεί να μειωθεί ως ένα βαθμό αλλά η μέθοδος αυτή συνεχίζει να αποκτά σφάλματα μετρήσεων σε βάθος χρόνου. [1]

2.5 Εντοπισμός με χρήση αισθητήρων μέτρησης απόστασης

Αυτή η μέθοδος μπορεί να χρησιμοποιηθεί για τον εντοπισμό θέσης ενός ρομποτικού οχήματος μόνο κάτω από συγκεκριμένες συνθήκες. Για να βρεθεί η θέση ενός ρομπότ με αισθητήρες

μέτρησης απόστασης, πρέπει ο χώρος εργασίας να είναι γνωστός και οι αισθητήρες κατάλληλα τοποθετημένοι ώστε να λαμβάνουν πληροφορίες που είναι προμελετημένες για να δώσουν την λύση στο πρόβλημα εντοπισμού θέσης. Ακόμα πρέπει να δημιουργηθεί κάποιου είδους μνήμη ή εξελισσόμενος χάρτης της γνωστής περιοχής ώστε το ρομπότ να χρησιμοποιεί τη γνώση των προηγούμενων μετρήσεων για να συνεχίσει να γνωρίζει τη θέση του και να μη χάνεται μέσα στον χώρο.
Είναι μια μέθοδος περιορισμένη ως προς τις δυνατότητες της, λόγω της δυσκολίας εντοπισμού θέσης σε ένα δυναμικό μεταβαλλόμενο περιβάλλον και λόγω της συσσωρευτικής λειτουργίας της. [1]

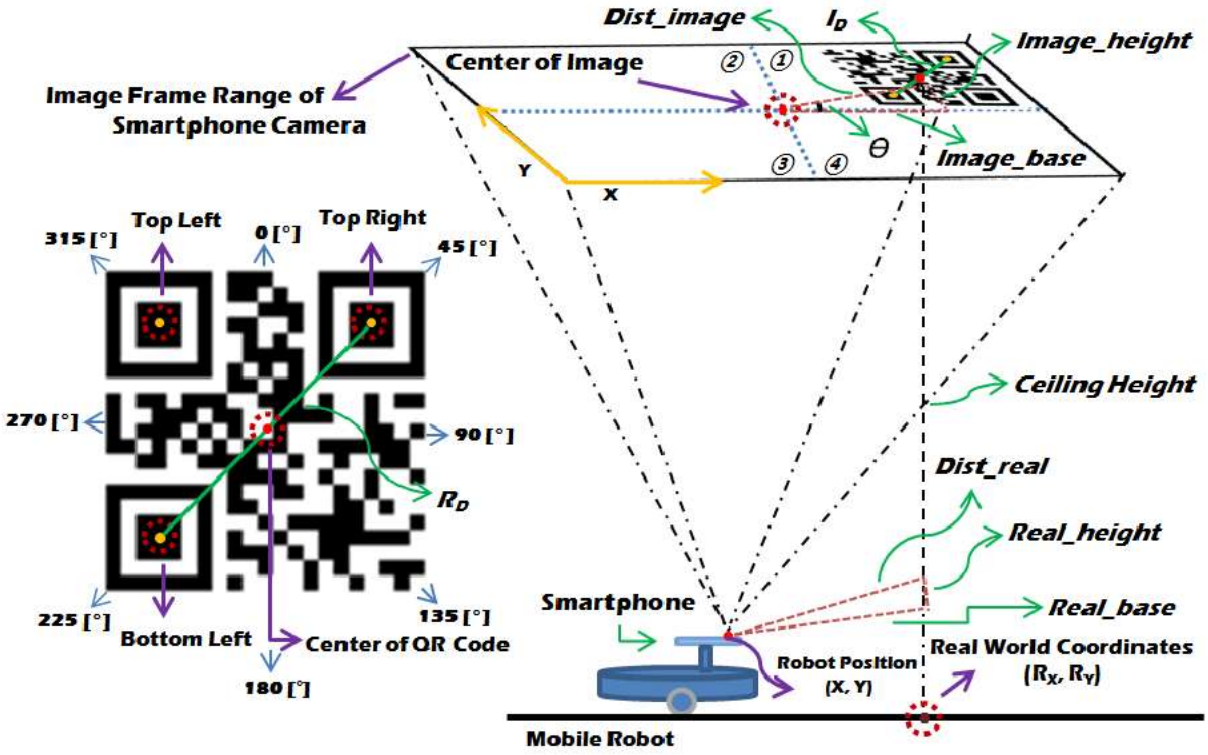
2.6 Εντοπισμός με χρήση κάμερας

Μια δημοφιλής μέθοδος εντοπισμού θέσης είναι με τη χρήση εικόνων ή ροής βίντεο που γίνονται από μία κάμερα ή και περισσότερες τοποθετημένες πάνω στο ρομπότ. Οι κάμερες δεν είναι ικανές να μετρήσουν την απόσταση όπως άλλοι αισθητήρες, όμως μπορούν να χρησιμοποιηθούν για να παρθούν χρήσιμες πληροφορίες που θα βοηθούν να φτάσουμε στην επίλυση του προβλήματος εντοπισμού θέσης.Ένα σημαντικό θετικό αυτής της μεθόδου είναι ότι δεν λειτουργεί συσσωρευτικά και κάθε φορά που γίνεται λήψη εικόνων ή επεξεργασία της ροής βίντεο ο εντοπισμός θέσης υπολογίζεται εκ νέου, εξαλείφοντας τα λάθη και τον θόρυβο που προκύπτουν με το πέρασμα του χρόνου με κάποιες από τις παραπάνω μεθόδους. Σαν μέθοδος μπορεί να έχει αποτελέσματα με μεγάλη ακρίβεια για τον εντοπισμό της θέσης και του προσανατολισμού του ρομπότ. Τα αρνητικά αυτής της μεθόδου είναι πως η ποιότητα της εικόνας που λαμβάνεται από τις κάμερες επηρεάζεται σημαντικά από τις συνθήκες φωτισμού και τις ενδεχόμενες σκιάσεις ή αντανακλάσεις που μπορούν να κάνουν τον εντοπισμό των περιοχών, γωνιών και χρωμάτων στην εικόνα δύσκολα ή αδύνατα. Αυτή η προϋπόθεση πρέπει να λαμβάνεται υπ’ όψη πριν την εφαρμογή αυτής της μεθόδου.

Εντός αυτής την κατηγορίας μπορεί να δημιουργηθούν δύο υποκατηγορίες. Η πρώτη υποκατηγορία είναι ο εντοπισμός αντικειμένων δίχως κάποιο σημείο αναφοράς και ο εντοπισμός αντικειμένων με χρήση σημείων αναφοράς. Το πρόβλημα που καλείται να λύσει η εν λόγω εργασία τοποθετείται στην δεύτερη κατηγορία, αυτή δηλαδή στην οποία κάποια σημεία αναφοράς στον χώρο προσδίδουν περεταίρω πληροφορίες ώστε να βοηθήσουν στην επίλυση του προβλήματος εντοπισμού τοποθεσίας.

Στην παρακάτω έρευνα [2] οι Lee, Tewolde, Lim και Kwon χρησιμοποιούν ως σημείο αναφοράς 4 QR codes τοποθετημένα στην οροφή του χώρου εργασίας του ρομπότ με στόχο να πάρουν τις πληροφορίες που χρειάζεται με τη χρήση μιας κάμερας που είναι περιστραμένη ως προς την οροφή.Χρησιμοποιούν τις πραγματικές γνωστές τοποθεσίες των QRcodes στον χώρο ώστε να

επιτρέψουν στο ρομπότ να λύσει το πρόβλημα εντοπισμού θέσης όπως φαίνεται στην εικόνα 2.



Εικόνα 2 Εντοπισμός θέσης και προσανατολισμού με QRcodes με την μέθοδο [4]
S. J. Lee, G. Tewolde, J. Lim, and J. Kwon, "QR-code based localization for indoor mobile robot with validation using a 3D optical tracking instrument," 2015 IEEE International Conference on Advanced Intelligent Mechatronics (AIM), 2015

Αντιμετωπίζουν πρόβλημα ανακρίβειας λόγω του focus της κάμερας κατά την κίνηση του ρομπότ το οποίο επηρεάζει τον σωστό εντοπισμό των σημείων του QRcode το οποίο λύνουν σχεδιάζοντας κύκλους γύρω από τα QRcodes, οι οποίοι ανιχνεύονται εύκολα και τότε το ρομπότ μειώνει την ταχύτητα κίνησης με αποτέλεσμα πιο καθαρές εικόνες για επεξεργασία και άρα καλύτερη ακρίβεια στον υπολογισμό της τοποθεσίας και του προσανατολισμού.

3. Έννοιες και γνωστικό υπόβαθρο

3.1 Computer vision (Υπολογιστική Όραση)

Η μηχανική όραση είναι κλάδος της τεχνητής νοημοσύνης εστιάζει στη δημιουργία συστημάτων που μπορούν να ερμηνεύουν και να κατανοούν οπτικά δεδομένα από τον κόσμο γύρω τους, όπως εικόνες και βίντεο.Αυτή η τεχνολογία έχει ένα ευρύ φάσμα εφαρμογών, συμπεριλαμβανομένων των αυτοοδηγούμενων αυτοκινήτων, της αναγνώρισης προσώπου και της ανάλυσης ιατρικής εικόνας.
Μία από τις βασικές προκλήσεις στην όραση υπολογιστή είναι η ανάπτυξη αλγορίθμων που μπορούν να κατανοήσουν και να ερμηνεύσουν οπτικά δεδομένα με τρόπο που μιμείται την ανθρώπινη αντίληψη.Αυτό απαιτεί την επίλυση προβλημάτων όπως:

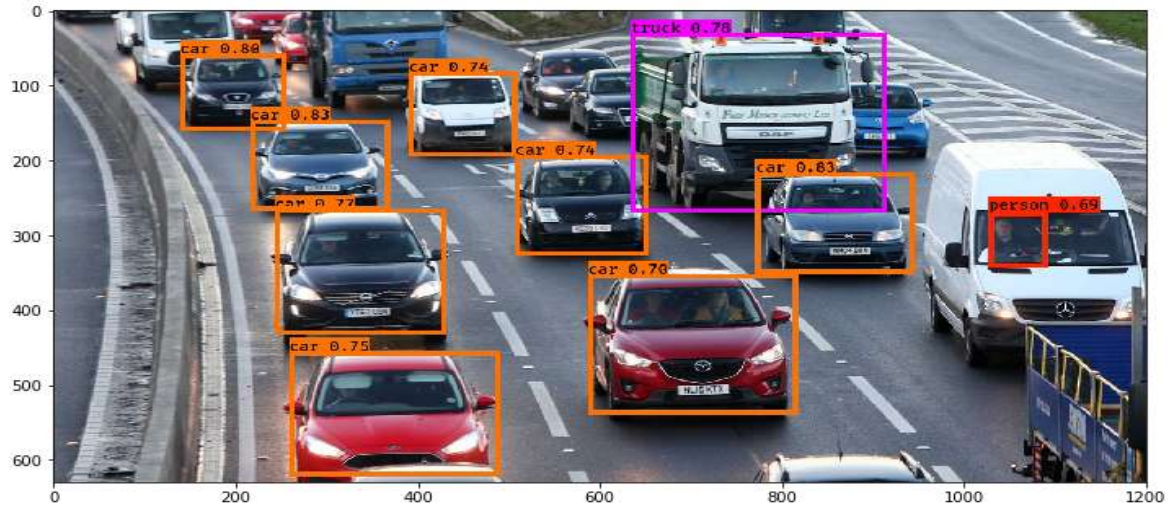
- Κατάτμηση εικόνας (Image segmentation)που χωρίζει μια εικόνα σε κομμάτια η τμήματα για περαιτέρω επεξεργασία
- Ανίχνευση ακμών (Edge detection) που αναζητεί τις ακμές ενός αντικειμένου ή περιοχής σε μια εικόνα για να καταλήξει σε κάποιο συμπέρασμα
- Ανίχνευση μοτίβου (Pattern detection) που ψάχνει χαρακτηριστικά που επαναλαμβάνονται μέσα στην εικόνα
- Ανίχνευσης αντικειμένων (Object detection) που ανιχνεύει τα αντικείμενα ενδιαφέροντος μέσα στην εικόνα
- Αντιστοίχιση χαρακτηριστικών (Feature matching) που αντιστοιχεί κοινά χαρακτηριστικά μέσα σε μια εικόνα ή μεταξύ εικόνων
- Ταξινόμηση εικόνων (Image classification) που ταξινομεί τις εικόνες σε ομάδες
- Τρισδιάστατης ανακατασκευής (3D reconstruction) που ένα αντικείμενο ή σκηνή δημιουργείται σε τρισδιάστατη μορφή με τη χρήση πληροφοριών και δεδομένων που υπάρχουν για αυτό

Μια από τις πιο σημαντικές ανακαλύψεις στον τομέα της όρασης υπολογιστών είναι η ανάπτυξη συνελικτικώννευρωνικών δικτύων (CNN).Πρόκειται για έναν τύπο αλγορίθμου βαθιάς μάθησης που έχει αποδειχθεί ιδιαίτερα αποτελεσματικός σε εργασίες όπως η αναγνώριση αντικειμένων και η ταξινόμηση εικόνων.
Ένα από τα βασικά πλεονεκτήματα των CNN είναι η ικανότητά τους να μαθαίνουν ιεραρχικές αναπαραστάσεις οπτικών δεδομένων.Αυτό σημαίνει ότι μπορούν να μάθουν να αναγνωρίζουν απλά χαρακτηριστικά σε μια εικόνα, όπως άκρες, και στη συνέχεια να χρησιμοποιούν αυτά τα χαρακτηριστικά για να αναγνωρίζουν πιο πολύπλοκα σχήματα και αντικείμενα.
Μια άλλη σημαντική εξέλιξη στην όραση υπολογιστών είναι η χρήση των GenerativeAdversarialNetworks (GAN) για τη δημιουργία εικόνων.Τα GAN αποτελούνται από δύο νευρωνικά δίκτυα: ένα δίκτυο γεννήτριας που δημιουργεί νέες εικόνες και ένα δίκτυο διαχωρισμού που προσπαθεί να κάνει διάκριση μεταξύ των παραγόμενων εικόνων και των πραγματικών εικόνων.Τα GAN έχουν χρησιμοποιηθεί για τη δημιουργία ρεαλιστικών εικόνων,

όπως πρόσωπα και τοπία, και έχουν πολλές πιθανές εφαρμογές σε τομείς όπως η τέχνη και το σχέδιο.

Τα τελευταία χρόνια, η όραση υπολογιστή έχει επίσης εφαρμοστεί στην ιατρική απεικόνιση, όπως αξονική τομογραφία και μαγνητική τομογραφία, για να βοηθήσει τους γιατρούς να διαγνώσουν και να θεραπεύσουν ασθένειες.Για παράδειγμα, οι αλγόριθμοι μηχανικής μάθησης μπορούν να χρησιμοποιηθούν για την αυτόματη αναγνώριση όγκων σε ιατρικές εικόνες, γεγονός που μπορεί να εξοικονομήσει χρόνο και να βελτιώσει την ακρίβεια σε σύγκριση με τη μη αυτόματη ανάλυση.

Συμπερασματικά, η υπολογιστική όραση είναι ένας ταχέως εξελισσόμενος τομέας με ένα ευρύ φάσμα πιθανών εφαρμογών.Η ανάπτυξη των CNN και των GAN οδήγησε σε σημαντική πρόοδο σε εργασίες όπως η αναγνώριση αντικειμένων και η δημιουργία εικόνων, και το πεδίο αναμένεται να συνεχίσει να αναπτύσσεται στο μέλλον.[1][3]



Εικόνα 3 Αναγνώριση αντικειμένων μεσυνελικτικόνευρωνικό δίκτυο YOLO
P. Sharma, "A Practical Guide to Object Detection using the Popular YOLO Framework – Part III (with Python codes)," *Analytics Vidhya*, 06-Dec-2018. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framewor-python/.

3.2 Επεξεργασία Εικόνας

Μια απαραίτητη προϋπόθεση για την μηχανική όραση είναι η επεξεργασία της εικόνας με τους κατάλληλους τρόπους ώστε να είναι εφικτή η εξαγωγή χρήσιμων πληροφοριών.Η επεξεργασία εικόνας σαν πεδίο περιλαμβάνει ένα ευρύ φάσμα εφαρμογών και αλγορίθμων όπως η βελτιστοποίηση εικόνας και μέθοδοι που αναφέρθηκαν παραπάνω όπως η κατάτμηση και ανίχνευση ακμών, μοτίβων και αντικειμένων.

Για να γίνει εξαγωγή των πληροφοριών συχνά χρειάζεται να βελτιωθεί η εικόνα με την αφαίρεση θορύβου και των παραμορφώσεων που προκύπτουν. Τεχνικές βελτίωσης εικόνας,

όπως το τέντωμα αντίθεσης (contrast stretching) και η εξίσωση ιστογράμματος (histogram equalization), χρησιμοποιούνται για τη βελτίωση της οπτικής ποιότητας μιας εικόνας προσαρμόζοντας τη φωτεινότητα, την αντίθεση και την ισορροπία χρωμάτων της.Τεχνικές αποκατάστασης εικόνας, όπως η αποθάμβωση(deblurring) και η αποθορυβοποίηση (denoising), χρησιμοποιούνται για την αφαίρεση του θορύβου και της παραμόρφωσης από μια εικόνα, όπως το θάμπωμα που προκαλείται από το κούνημα της κάμερας ή ο θόρυβος που προκαλείται από συνθήκες χαμηλού φωτισμού. Μια άλλη σημαντική πτυχή της επεξεργασίας εικόνας είναι η κατάτμηση εικόνας (imagesegmentation), η οποία είναι η διαδικασία διαίρεσης μιας εικόνας σε πολλαπλές περιοχές ή τμήματα, με βάση ορισμένες ιδιότητες όπως το χρώμα, η υφή ή το σχήμα.Η κατάτμηση εικόνας χρησιμοποιείται σε ένα ευρύ φάσμα εφαρμογών, συμπεριλαμβανομένης της αναγνώρισης αντικειμένων, της ιατρικής απεικόνισης και των δορυφορικών εικόνων. Μία από τις πιο επιτυχημένες τεχνικές για την κατάτμηση εικόνας είναι η χρήση συνελικτικώννευρωνικών δικτύων (CNN).Τα CNN είναι ένας τύπος αλγόριθμου βαθιάς μάθησης που έχει αποδειχθεί ιδιαίτερα αποτελεσματικός σε εργασίες όπως η ταξινόμηση εικόνων και η αναγνώριση αντικειμένων.Τα CNN μπορούν να μάθουν να αναγνωρίζουν μοτίβα και χαρακτηριστικά σε μια εικόνα, όπως άκρες και σχήματα, τα οποία στη συνέχεια μπορούν να χρησιμοποιηθούν για να τμηματοποιήσουν την εικόνα σε πολλές περιοχές.[1][3]

Εικόνα 4 Lena, αποθορύβωση εικόνας με φίλτρα PSNR, SSIM, IEF U. Erkan and L. Gökrem, "A new method based on pixel density in salt and pepper noise removal," *TÜBİTAK Academic Journals*. [Online]. Available: https://journals.tubitak.gov.tr/elektrik/vol26/iss1/15/.

3.3 Μοντέλο κάμερας σημειακής οπής και πίνακας προβολής κάμερας 3D Projection

Για να προβάλλουμε μια σκηνή τριών διαστάσεων σε μια εικόνα δύο διαστάσεων χρειαζόμαστε σχέσεις μεταξύ των σημείων σκηνής και εικόνας. Οι αλγόριθμοι της βιβλιοθήκης OpenCV που θα χρησιμοποιηθούν στην εργασία στηρίζονται στο γνωστό μοντέλο κάμερας σημειακής οπής (pinhole model). Σε αυτό το μοντέλο μια σκηνή σχηματίζεται προβάλλοντας τα τρισδιάστατα σημεία στο επίπεδο της εικόνας χρησιμοποιώντας τον παρακάτω μετασχηματισμό.

$s\, x = K[R \mid t]\, X$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Εικόνα 1 Μαθηματικό μοντέλο pinhole camera
"Camera calibration and 3D reconstruction," *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html.

Όπου:

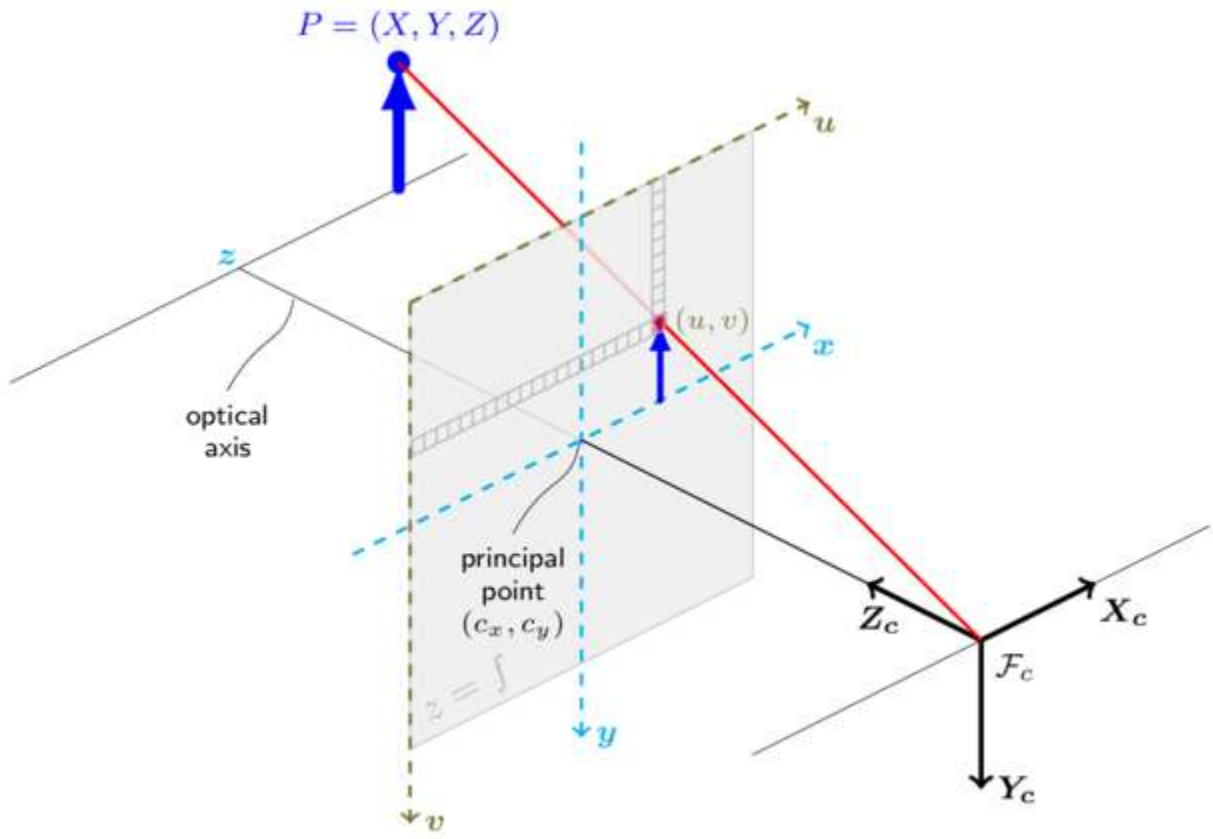
- sείναι ο συντελεστής κλιμάκωσης του x
- x =είναι οι συντεταγμένες του σημείου προβολής σε εικονοστοιχεία
- K είναι ο εσωτερικός πίνακας κάμερας ή αλλιώς πίνακας εγγενών παραμέτρων
- f_x, f_y είναι οι εστιακές αποστάσεις που εκφράζονται σε εικονοστοιχεία
- c_x, c_yείναι τα κεντρικό σημείο (principalpoint) στην εικόνα
- R|t είναι ο 4x4 πίνακας περιστροφής και μετατόπισης
- X, Z, Yείναι οι συντεταγμένες ενός τρισδιάστατου σημείου στο παγκόσμιο χώρο συντεταγμένων

Οι παραπάνω παράμετροι της κάμερας μπορούν να κατηγοριοποιηθούν σε εγγενείς και εξωγενείς παραμέτρους.

Οι εγγενείς παράμετροι (intrinsicparameters) που υπάρχουν μέσα στον εσωτερικό πίνακα κάμερας K περιλαμβάνουν την εστιακή απόσταση, τη μορφή του αισθητήρα που έγινε λήψη εικόνας και το κεντρικό σημείο κάμερας. Οι παράμετροι $f_x = f * m_x$και$f_y = f * m_y$αντιπροσωπεύουν την εστιακή απόσταση σε εικονοστοιχεία με τα m_xκαι m_yνα είναι τα

αντίστροφα πλάτους και ύψους ενός εικονοστοιχείου στο επίπεδο προβολής και f να είναι η εστιακή απόσταση μετρήσιμη σε απόσταση.

Οι εξωτερικές παράμετροι R|t δηλώνουν τους μετασχηματισμούς του συστήματος συντεταγμένων από συντεταγμένες του κόσμου σε συντεταγμένες κάμερας. Αυτές οι παράμετροι καθορίζουν τη θέση του κέντρου της κάμερας και την κατεύθυνση της κάμερας στις συντεταγμένες κόσμου. Το άνυσμα 3x1 T είναι η θέση προέλευσης του παγκόσμιου συστήματος συντεταγμένων που εκφράζεται σε συντεταγμένες του συστήματος συντεταγμένων με επίκεντρο την κάμερα. Ο πίνακας 3x3 R είναι ο πίνακας περιστροφής στον Ευκλείδιο χώρο.[4]



Εικόνα 6 Μοντέλο κάμερας OpenCV
"Camera calibration and 3D reconstruction," *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html.

3.4 Παραμόρφωση

Ορισμένες κάμερες pinhole εισάγουν σημαντική παραμόρφωση στις εικόνες. Δύο κύρια είδη παραμόρφωσης είναι η ακτινική παραμόρφωση και η εφαπτομενική παραμόρφωση. Η ακτινική παραμόρφωση κάνει τις ευθείες γραμμές να φαίνονται καμπύλες. Η ακτινική

παραμόρφωση γίνεται μεγαλύτερη όσο πιο μακριά είναι τα σημεία από το κέντρο της εικόνας. Οι συντελεστές παραμόρφωσης ανήκουν στις εγγενείς παραμέτρους της κάμερας και παραμένουν ίδιες ανεξάρτητα από την ανάλυση της εικόνας ή τη σκηνή. Για τον λόγο αυτό είναι σημαντική η βαθμονόμηση της κάμερας πριν χρησιμοποιηθεί για εφαρμογές ρομποτικής όρασης ώστε να βρεθούν οι παράμετροι παραμόρφωσης και να μειωθούν τα λάθη των υπολογισμών.

Η ακτινικήπαραμόρφωση μπορεί να αναπαρασταθεί ως:

$x_{distorted} = x (1 + k_1r^2+k_2r^4+k_3r^6)$
$y_{distorted} = y (1 + k_1r^2+k_2r^4+k_3r^6)$

Η εφαπτομενική παραμόρφωση μπορεί να αναπαρασταθεί ως:

$x_{distorted}=x+[2p_1xy+p_2(r^2+2x^2)]$
$y_{distorted}=y+[p_1(r^2+2y^2)+2p_2xy]$

όπουγια τις παραπάνω σχέσεις ισχύει:

- x, y είναι οι μη παραμορφωμένες θέσεις εικονοστοιχείων. Τα x και y βρίσκονται σε κανονικοποιημένες συντεταγμένες εικόνασκαι είναι αδιάστατα.
- k_1, k_2, και k_3είναι οι συντελεστές ακτινικής παραμόρφωσης του φακού.
- p_1καιp_2, και k_3είναι οισυντελεστέςεφαπτομενικής παραμόρφωσης του φακού.
- $r^2 = x^2 + y^2$

Όπως φαίνεται από τις παραπάνω σχέσεις οι παράμετροι που πρέπει να βρεθούν κατά τη διάρκεια της βαθμονόμησης κάμερας είναι παράμετροι παραμόρφωσης = (k_1 $k_2$$p_1$$p_2$$k_3$).[4]

Εικόνα 7 Παραμορφώσεις εικόνας
"Camera calibration and 3D reconstruction," *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html.

3.5 Βαθμονόμηση κάμερας

Η βαθμονόμηση κάμερας είναι ένα σημαντικό βήμα σε πολλές εφαρμογές υπολογιστικής όρασης και ρομποτικής που απαιτούν ακριβείς πληροφορίες πόζας κάμερας και γεωμετρίας. Η διαδικασία βαθμονόμησης κάμερας περιλαμβάνει έναν συνδυασμό τεχνικών επεξεργασίας εικόνας και βελτιστοποίησης και οι συγκεκριμένοι αλγόριθμοι που χρησιμοποιούνται θα εξαρτηθούν από τη συγκεκριμένη εφαρμογή και την κάμερα. Η βαθμονόμηση κάμερας είναι η διαδικασία προσδιορισμού των εγγενών και εξωτερικών παραμέτρων μιας κάμερας, οι οποίες είναι απαραίτητες για ακριβή 3D ανακατασκευή και εκτίμηση πόζας από εικόνες 2D. Οι εγγενείς παράμετροι περιλαμβάνουν την εστιακή απόσταση, το κύριο σημείο και την παραμόρφωση του φακού, ενώ οι εξωτερικές παράμετροι περιγράφουν τη θέση και τον προσανατολισμό της κάμερας στον τρισδιάστατο χώρο.

Η διαδικασία βαθμονόμησης της κάμερας περιλαμβάνει συνήθως τα ακόλουθα βήματα:

1. Λήψη εικόνων βαθμονόμησης: Ένα σύνολο εικόνων βαθμονόμησης λαμβάνεται χρησιμοποιώντας την κάμερα που πρόκειται να βαθμονομηθεί. Αυτές οι εικόνες θα πρέπει να καλύπτουν ένα ευρύ φάσμα στάσεων και προσανατολισμών και θα πρέπει να περιέχουν έναν γνωστό στόχο βαθμονόμησης, όπως μία σκακιέρα ή ένα πλέγμα.
2. Ανίχνευση στόχου βαθμονόμησης: Σε κάθε εικόνα βαθμονόμησης, ο στόχος βαθμονόμησης ανιχνεύεται και οι συντεταγμένες δισδιάστατης εικόνας του εξάγονται χρησιμοποιώντας τεχνικές όρασης υπολογιστή, όπως η ανίχνευση χαρακτηριστικών και η τμηματοποίηση εικόνας.
3. Υπολογισμός παραμέτρων κάμερας: Χρησιμοποιώντας τις συντεταγμένες εικόνας 2D του στόχου βαθμονόμησης και τις γνωστές 3D συντεταγμένες τους στο παγκόσμιο σύστημα συντεταγμένων, οι παράμετροι της κάμερας μπορούν να εκτιμηθούν χρησιμοποιώντας έναν αλγόριθμο βαθμονόμησης. Ένας δημοφιλής αλγόριθμος για το σκοπό αυτό είναι η μέθοδος του Zhang, η οποία χρησιμοποιεί μια βελτιστοποίηση ελαχίστων τετραγώνων για να εκτιμήσει τις εγγενείς και εξωγενείς παραμέτρους της κάμερας.
4. Βελτιστοποίηση παραμέτρων κάμερας: Μετά την αρχική εκτίμηση, οι παράμετροι της κάμερας μπορούν να βελτιωθούν περαιτέρω χρησιμοποιώντας έναν αλγόριθμο βελτιστοποίησης για να ελαχιστοποιηθεί το σφάλμα επαναπροβολής μεταξύ των σημείων του τρισδιάστατου κόσμου και των αντίστοιχων προβολών 2D εικόνας.

Οι παράμετροι βαθμονόμησης της κάμερας που προκύπτουν μπορούν στη συνέχεια να χρησιμοποιηθούν για διάφορες εφαρμογές όρασης υπολογιστή, όπως ανακατασκευή 3D, εκτίμηση πόζας και διόρθωση εικόνας.[4]

3.6 Βαθμονόμηση με μέθοδο Zhang

Η μέθοδος του Zhang[5] είναι ένας δημοφιλής αλγόριθμος για τη βαθμονόμηση της κάμερας χρησιμοποιώντας ένα σύνολο γνωστών σημείων 3D αντικειμένων και των αντίστοιχων σημείων 2D εικόνας τους. Τα βήματα αυτής της μεθόδου φαίνονται παρακάτω:

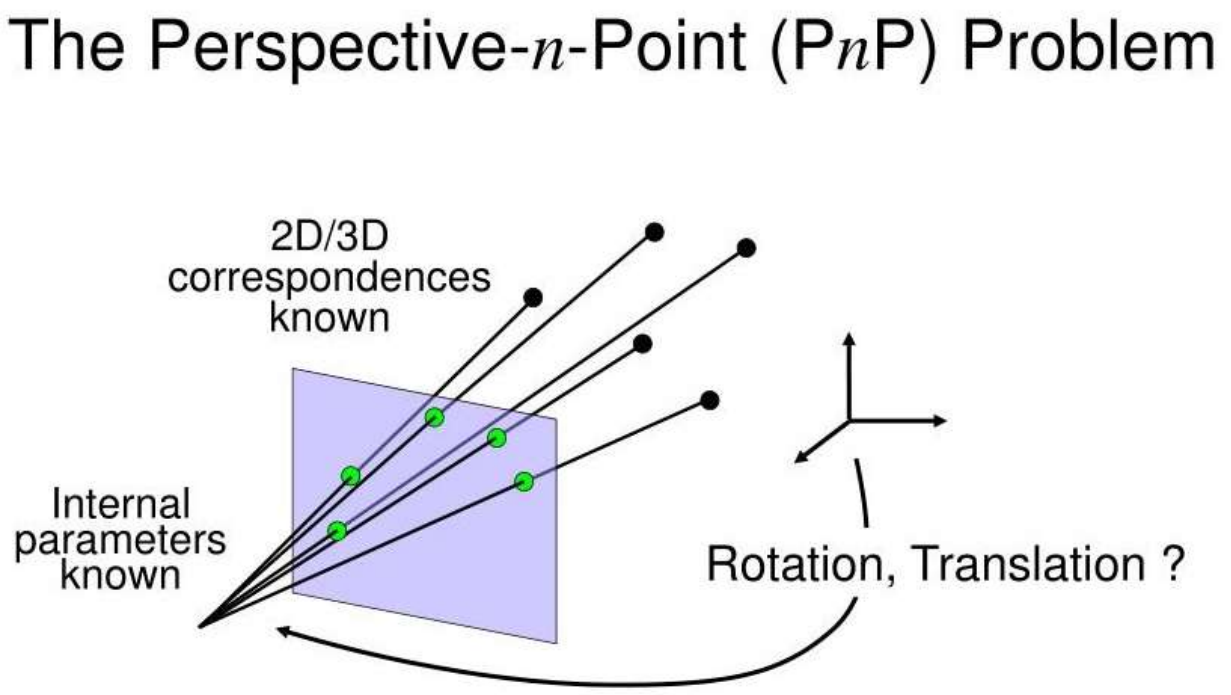
1. Λήψη εικόνων βαθμονόμησης: Καταγράψτε ένα σύνολο εικόνων ενός αντικειμένου βαθμονόμησης με γνωστές τρισδιάστατες συντεταγμένες, όπως ένα μοτίβο σκακιέρας ή ένα πλέγμα βαθμονόμησης.
2. Ανίχνευση σημείων χαρακτηριστικών: Ανίχνευση σημείων χαρακτηριστικών σε κάθε εικόνα βαθμονόμησης, όπως οι γωνίες του αντικειμένου βαθμονόμησης.
3. Υπολογισμός ομογραφιών: Υπολογίστε τον πίνακα ομογραφίας που αντιστοιχίζει τα σημεία 3D αντικειμένου στα αντίστοιχα σημεία δισδιάστατης εικόνας για κάθε εικόνα.
4. Επίλυση για εγγενείς παραμέτρους: Επίλυση για τις εγγενείς παραμέτρους της κάμερας χρησιμοποιώντας τις ομογραφίες και το σύνολο των τρισδιάστατων σημείων αντικειμένων. Αυτό μπορεί να γίνει ελαχιστοποιώντας το σφάλμα επαναπροβολής, το οποίο είναι η διαφορά μεταξύ των πραγματικών σημείων εικόνας και των προβαλλόμενων σημείων εικόνας χρησιμοποιώντας τις εκτιμώμενες παραμέτρους της κάμερας.
5. Επίλυση για εξωτερικές παραμέτρους: Μόλις γίνουν γνωστές οι εσωτερικές παράμετροι, λύστε τις εξωτερικές παραμέτρους της κάμερας για κάθε εικόνα αποσυνθέτοντας τη μήτρα ομογραφίας.
6. Βελτιώστε τις παραμέτρους: Προαιρετικά, βελτιώστε τις εκτιμώμενες παραμέτρους της κάμερας ελαχιστοποιώντας το σφάλμα επαναπροβολής χρησιμοποιώντας έναν αλγόριθμο βελτιστοποίησης.

Το αποτέλεσμα της μεθόδου του Zhang είναι ένα σύνολο εκτιμώμενων παραμέτρων κάμερας, συμπεριλαμβανομένων των εγγενών παραμέτρων (εστιακή απόσταση, κύριο σημείο και συντελεστές παραμόρφωσης φακού) και των εξωτερικών παραμέτρων (θέση και προσανατολισμός κάμερας).Η μέθοδος του Zhang είναι ένας ευρέως χρησιμοποιούμενος και ισχυρός αλγόριθμος για τη βαθμονόμηση της κάμερας, καθώς απαιτεί μόνο ένα σύνολο αντιστοιχιών 2D-3D και δεν βασίζεται σε υποθέσεις σχετικά με το αντικείμενο βαθμονόμησης ή το μοντέλο της κάμερας. Η μέθοδος έχει εφαρμοστεί σε πολλές βιβλιοθήκες υπολογιστικής όρασης, συμπεριλαμβανομένων των OpenCV και MATLAB.

3.7 Πρόβλημα P-n-P

Το πρόβλημα p-n-p είναι ένα πολύ γνωστό πρόβλημα στην όραση υπολογιστών και στην υπολογιστική γεωμετρία, το οποίο πραγματεύεται εάν ένα δεδομένο τρισδιάστατο αντικείμενο μπορεί να ανακατασκευαστεί από μια συλλογή 2D εικόνων. Συγκεκριμένα, το πρόβλημα είναι να προσδιοριστεί η θέση και ο προσανατολισμός του αντικειμένου στον τρισδιάστατο χώρο (δηλαδή, η θέση του) με δεδομένες τις αντίστοιχες δισδιάστατες προβολές σε κάθε εικόνα. Το όνομα "P-n-P" σημαίνει "προοπτική-n-σημείο" επειδή το πρόβλημα συνήθως περιλαμβάνει τον προσδιορισμό της προοπτικής προβολής του αντικειμένου σε ένα επίπεδο εικόνας και στη συνέχεια την επίλυση της στάσης του αντικειμένου χρησιμοποιώντας αντιστοιχίες μεταξύ των σημείων εικόνας και των σημείων 3D στο αντικείμενο.Για την επίλυση αυτού του προβλήματος χρειάζεται να έχει προηγηθεί βαθμονόμηση κάμερας ώστε να είναι γνωστές οιεγγενείς παράμετροι της κάμερας.

Το πρόβλημα P-n-P είναι σημαντικό σε διάφορους τομείς, όπως η ρομποτική, η επαυξημένη πραγματικότητα και η αυτόνομη πλοήγηση. Υπάρχουν πολλοί αλγόριθμοι που έχουν προταθεί για την επίλυση του προβλήματος P-n-P,ένας των οποίων είναι ο επαναληπτικός αλγόριθμος EPnPκαι ένας μη επαναληπτικός είναι ο Direct Linear Transformation (DLT).[7]



Εικόνα 2 Perspective-n-Point (PnP)
V. Lepetit , "Low complexity keypoint recognition and pose estimation - [PPT PowerPoint]," *vdocuments.mx*. [Online]. Available: https://vdocuments.mx/low-complexity-keypoint-recognition-and-pose-estimation-vincent-lepetit.html. [Accessed: 07-Mar-2023].

3.8 Direct Linear Transformation(DLT)

Ο DLT είναι ένας δημοφιλής αλγόριθμος για την επίλυση του προβλήματος P-n-P στην όραση υπολογιστή και στη φωτογραμμετρία. Ο αλγόριθμος λειτουργεί κατασκευάζοντας πρώτα ένα σύστημα γραμμικών εξισώσεων που συσχετίζει τα τρισδιάστατα σημεία αντικειμένου με τα αντίστοιχα σημεία δισδιάστατης εικόνας. Αυτή η εξίσωση μπορεί να γραφτεί ως:

s * x' = P * X**ή** $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathsf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$ **ή** $\begin{aligned} x &= \frac{p_{11}X + p_{12}Y + p_{13}Z + p_{14}}{p_{31}X + p_{32}Y + p_{33}Z + p_{34}} \\ y &= \frac{p_{21}X + p_{22}Y + p_{23}Z + p_{24}}{p_{31}X + p_{32}Y + p_{33}Z + p_{34}} \end{aligned}$

όπου s είναι ένας συντελεστής κλιμάκωσης, x' είναι το σημείο εικόνας σε ομογενείς συντεταγμένες, P είναι ο πίνακας κάμερας (που περιέχει τις εγγενείς και εξωγενείς παραμέτρους της κάμερας) και το X είναι το ομογενές σημείο 3D στις συντεταγμένες του κόσμου.

Στη συνέχεια, ο αλγόριθμος DLT λύνει τον πίνακα P της κάμερας γραμμικά χρησιμοποιώντας τη μέθοδο των ελαχίστων τετραγώνων. Συγκεκριμένα, ο αλγόριθμος κατασκευάζει έναν πίνακα A που περιέχει τους συντελεστές των γραμμικών εξισώσεων για όλες τις αντιστοιχίες και ένα διάνυσμα b που περιέχει τις δεξιές πλευρές των εξισώσεων. Ο πίνακας της κάμερας P προκύπτει στη συνέχεια λύνοντας το γραμμικό σύστημα:

A * p = b

όπου p είναι ένα διάνυσμα που περιέχει τα στοιχεία του P.

Μόλις ληφθεί ο πίνακας της κάμερας P, η πόζα του αντικειμένου μπορεί να υπολογιστεί αποσυνθέτοντας το P στις εγγενείς και εξωγενείς παραμέτρους του.Ένα πλεονέκτημα του αλγόριθμου DLT είναι ότι μπορεί να χειριστεί μεγάλο αριθμό αντιστοιχιών. Ωστόσο, είναι ευαίσθητος στο θόρυβο και στα ακραία σημεία στις αντιστοιχίες και μπορεί να παράγει ανακριβή αποτελέσματα εάν οι αντιστοιχίες δεν είναι ξεκάθαρες. [6]

3.9Εργαλεία

Για την υλοποίηση της εργασίας θα χρησιμοποιηθούν τα παρακάτω λογισμικά και σουίτες λογισμικού:

VNC Viewer: Τοπρόγραμμα αυτό διατίθεται δωρεάν και όπως προκύπτει και απ' το όνομα είναι το λογισμικό που θα επιτρέψει στο ρομπότ να επικοινωνεί με τον κεντρικό υπολογιστή. Το Raspberry Pi θα λειτουργεί σαν VNC server και ο υπολογιστής θα είναι ο VNC viewer με δυνατότητα να χειρίζεται απομακρυσμένα τον server. Για να επιτευχθεί ή επικοινωνία βάση αυτού του προγράμματος πρέπει οι συσκευές να είναι συνδεδεμένες στο ίδιο δίκτυο και οι διευθύνσεις τους να είναι γνωστές ώστε να επιτραπεί η επικοινωνία.

ArduinoIDE: Το πρόγραμμα αυτό διατίθεται δωρεάν. Πρόκειται για μια σουίτα προγραμματισμού για μικροελεγκτές. Η γλώσσα προγραμματισμού αυτής της πλατφόρμας αυτής βασίζεται και είναι παραπλήσια στη C++. Περιέχονται βιβλιοθήκες με έτοιμο κώδικα και αλγορίθμους για τη διευκόλυνση του προγραμματισμού.Για το arduinoscriptτης εργασίας θα χρησιμοποιηθεί η βιβλιοθήκη TinyStepper.h για τον έλεγχο του βηματικού κινητήρα.

PyCharm Community Edition: Το πρόγραμμα αυτό διατίθεται δωρεάν. Είναι μια σουίτα ανάπτυξης λογισμικού με τη γλώσσα Pythonκαιπαρέχει εργαλεία για testing και debugging κώδικα. Για την εργασία θα χρησιμοποιηθούν οι παρακάτω βιβλιοθήκες:

- Pynput: μια βιβλιοθήκη Python που επιτρέπει τον έλεγχο και παρακολούθηση συσκευών εισόδου όπως το πληκτρολόγιο και το ποντίκι. Παρέχει έναν τρόπο προσομοίωσης πατημάτων πλήκτρων και κλικ του ποντικιού, καθώς και παρακολούθησης και χειρισμού συμβάντων που συμβαίνουν σε αυτές τις συσκευές.
- Time:είναι μια βιβλιοθήκη Python που παρέχει διάφορες λειτουργίες που σχετίζονται με το χρόνο. Περιλαμβάνει λειτουργίες για λήψη της τρέχουσας ώρας, λειτουργία sleep για καθορισμένο χρονικό διάστημα και μετατροπή μεταξύ διαφορετικών μορφών ώρας.
- Serial: είναι μια βιβλιοθήκη Python που επιτρέπει την επικοινωνία με σειριακές συσκευές όπως μικροελεγκτές και αισθητήρες. Παρέχει έναν τρόπο ανοίγματος και διαμόρφωσης μιας σειριακής σύνδεσης, ανάγνωσης και εγγραφής δεδομένων και χειρισμού διαφόρων σφαλμάτων που μπορεί να προκύψουν κατά την επικοινωνία.
- Picamera: είναι μια βιβλιοθήκη Python που παρέχει μια διεπαφή για τον έλεγχο και την πρόσβαση στην κάμερα Raspberry Pi. Επιτρέπει τη λήψη εικόνων και βίντεο, εφαρμογή εφέ και φίλτρα και τον έλεγχο ρυθμίσεων κάμερας, όπως φωτεινότητα, έκθεση και ισορροπία λευκού.
- Os: είναι βιβλιοθήκη Python που παρέχει διάφορες λειτουργίες που σχετίζονται με λειτουργίες λειτουργικού συστήματος. Περιλαμβάνει λειτουργίες για το χειρισμό αρχείων και καταλόγων, τη λήψη πληροφοριών σχετικά με το τρέχον σύστημα και την αλληλεπίδραση με το υποκείμενο λειτουργικό σύστημα.

- cv2: Η OpenCV2 είναι μια βιβλιοθήκη Python που παρέχει λειτουργίες υπολογιστικής όρασης και επεξεργασίας εικόνας. Επιτρέπει τη φόρτωση, τον χειρισμό και την ανάλυση εικόνας και βίντεο,την επεξεργασία εικόνας όπως το φιλτράρισμα, το όριο και ανίχνευση ακμών, και πολλούς άλλους υψηλού επιπέδου αλγορίθμους του τομέα της υπολογιστικής όρασης
- numpy: Είναι μια βιβλιοθήκη Python που παρέχει διάφορες λειτουργίες που σχετίζονται με αριθμητικούς υπολογισμούς και ανάλυση δεδομένων. Παρέχει έναν τρόπο εργασίας με πολυδιάστατους πίνακες, εκτέλεση διαφόρων μαθηματικών πράξεων σε αυτούς τους πίνακες και χειρισμό και ανάλυση μεγάλων συνόλων δεδομένων. Συχνά χρησιμοποιείται σε συνδυασμό με άλλες βιβλιοθήκες όπως η cv2 για την εκτέλεση προηγμένων εργασιών ανάλυσης εικόνων και δεδομένων.

Samba: Το λογισμικό αυτό είναι ένα δωρεάν λογισμικό ανοιχτού κώδικα που επιτρέπει σε λογισμικά τύπου unix την επικοινωνία με άλλους υπολογιστές στα πλαίσια του network file-sharing. Θα χρησιμοποιηθεί ώστε ο κεντρικός προσωπικός υπολογιστής να έχει πρόσβαση σε έναν κοινό διαμοιρασμένο φάκελο που θα αποθηκεύονται οι εικόνες που θα λαμβάνονται από το Raspberry Pi.

4. Μέθοδος επίλυσης προβλήματος

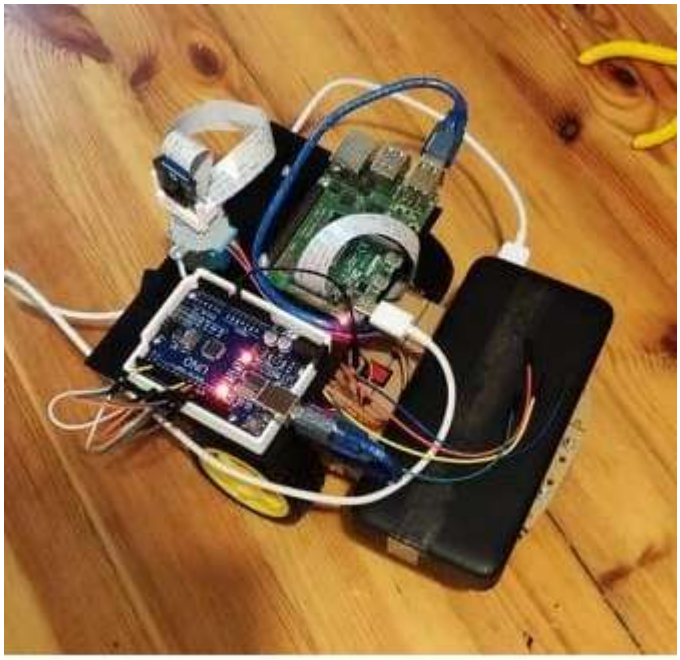
4.1 Κατασκευήδίτροχου ρομπότ και κόστος

Τα υλικά που χρησιμοποιήθηκαν για την κατασκευή του ρομπότ είναι ένας σκελετός από kit δίτροχου ρομπότ για ανάπτυξη με Arduino μαζι με τροχούς, τροχό κάστερ και 2 μοτέρ συνεχούς ρεύματος. Τα μοτέρ λειτουργούν με 3V-12V ενώ η ένδειξη προτεινόμενης λειτουργίας είναι στο εύρος 6V-8V. Για την τροφοδοσία των μοτέρ χρησιμοποιήθηκαν 3 μπαταρίες Panasonic NCR18650B Li-ion 3400 mAh 3.7V σε σειρά.Οι μπαταρίες αυτές έχουν μέγιστη τάση 4.2Vμετά από πλήρη φόρτιση καιμπορούν να προσφέρουν μέγιστη τάση 12.6Vσε σειρά. Όπως θα αναφερθεί αργότερα λόγω της πτώσης τάσης από τον ελεγκτή L298Nη τάση δεν ξεπερνάει ποτέ τα 12Vπου είναι ανώτερη τάση των μοτέρ. Για τον έλεγχο των μοτέρ χρησιμοποιείται ο ελεγκτής L298N ο οποίος θα επικοινωνεί με έναν ελεγκτή UNOR3 ATmega328P (Arduino UNO clone). Ακόμα θα χρησιμοποιηθεί το RaspberryPi 4 και ένα Powerbank Xiaomi Redmi 18W 20000 mAh για την τροφοδοσία του RaspberryPi. Λόγω έλλειψης χώρου στον αρχικό σκελετό του kit, χρησιμοποιούνται βίδες m3x50 και αποστάτες από υλικό PLA εκτυπωμένοι από 3D Printer μαζί με μια βάση 10x16x2 για να τοποθετηθούν πάνω σε αυτή ο ελεγκτής Arduino, το Raspberry Pi και ένα βηματικό μοτέρ πάνω στο οποίο θα βρίσκεται η κάμερα RaspberryPi 8MP Camera Module. Για τον έλεγχο του βηματικού μοτέρ χρησιμοποιείται ο ελεγκτής ULN2003.

BILL OF MATERIALS	
ΥΛΙΚΑ	**ΤΙΜΗ**
Arduino robot kit car	€ 10.00
UNO R3 Atmega 328P	€ 7.50
2x Panasonic NCR18650B Li-ion 3400mAh 3.7V	€ 18.00
Plastic Battery Holder / Case for 3x 18650	€ 1.00
Stepper Driver L298N Dual H Bridge	€ 4.00
Powerbank Xiaomi Redmi 18W 20000mAh	€ 20.00
βίδες + περικόχλια	€ 2.00
28BYJ-48-5V Stepper Motor + 5V ULN2003 Stepper Motor Board	€ 4.00
Raspberry Pi 4 MODEL B, 2GB	€ 36.00
Official Camera Board για Raspberry Pi 4 8MP v2	€ 30.00
Jumper Cables Male - Male + Female - Female	€ 7.00
ΣΥΝΟΛΙΚΟ ΧΡΗΜΑΤΙΚΟ ΠΟΣΟ ΥΛΙΚΩΝ	**€ 139.50**

Εικόνα 3 Robot bill of materials

Εικόνα 4Πλάγια λήψη ρομπότ

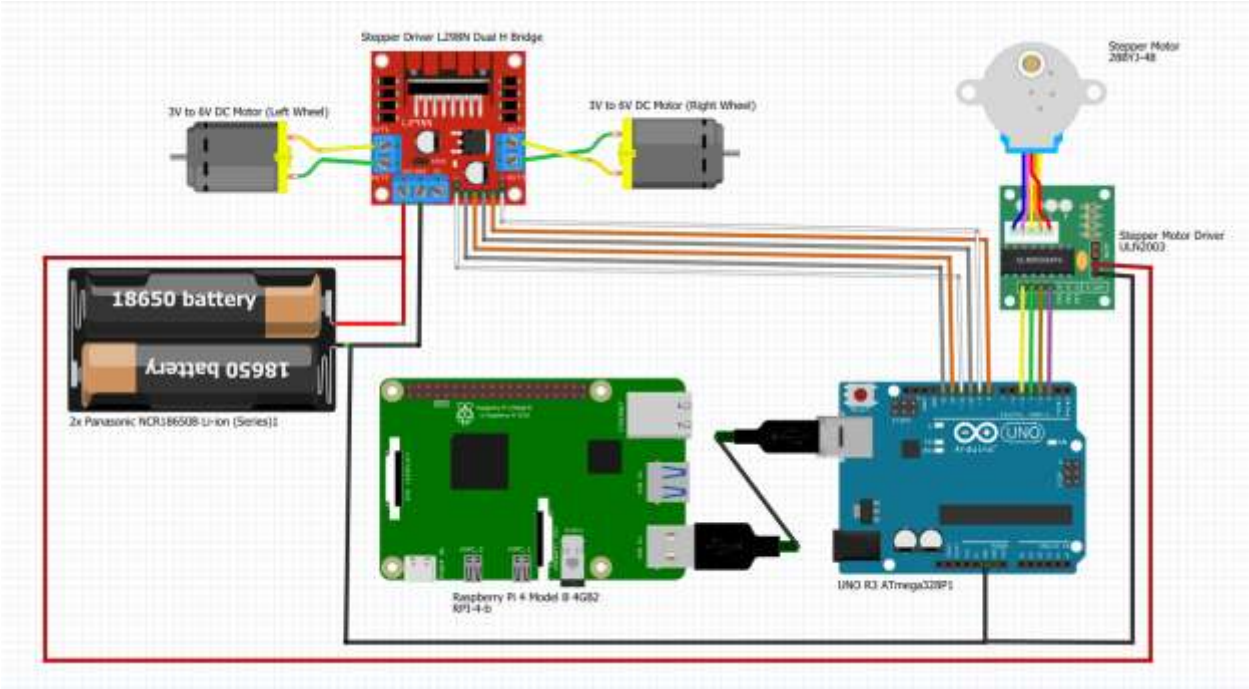


Εικόνα11 Κάτοψη ρομπότ

4.2 Ελεγκτές και κινητήρες

Ο ελεγκτής L298N επιτρέπει την χρήση δύο κινητήρων συνεχούς ρεύματος με έλεγχο της φοράς κατεύθυνσης και της ταχύτητας των κινητήρων. Ο ελεγκτής αυτός χρησιμοποιεί δύο γέφυρες τύπου Η για να ελέγξει την περιστροφή των κινητήρων με τους ακροδέκτες IN1 – IN4 και σήματα PWM ώστε να ελέγξει την ταχύτητα περιστροφής των τροχών βάση του πλάτους παλμού που θα δέχεται με τους ακροδέκτες ENA και ENB. Τροφοδοτείται με τάση 5V-35V. Για τον βηματικό κινητήρα που θα ελέγχει την περιστροφή της κάμερας θα χρησιμοποιηθεί ο ελεγκτής ULN2003.

Μετά απο πειραματική λειτουργία του ρομπότ και μετρήσεις τάσεως στην είσοδο και έξοδο του ελεγκτή L298N, παρατηρήθηκε πτώση τάσης ≈ 2V που δεν επιτρέπει τη λειτουργία των τροχών με τροφοδοσία δύο μπαταριών 18650. Για τον λόγο αυτό, προστέθηκε μια τρίτη μπαταρία Panasonic NCR18650B σε σειρά με αποτέλεσμα να γίνεται σωστή λειτουργία των μοτέρ συνεχούς ρεύματος που είναι υπεύθυνα για την περιστροφή των τροχών.



Εικόνα12 Συνδεσμολογία συστήματος

4.3 Επικοινωνία και προγραμματισμός συστήματος

4.3.1 RaspberryPi

Το RaspberryPi θα επικοινωνεί σειριακά (UARTprotocol) με τον μικροελεγκτή UNOR3 ATmega328P με ένα pythonscript που θα τρέχει μόνιμα στο RaspberryPi και η επικοινωνία θα επιτυγχάνεται μέσω καλωδίου USB. Η σειριακή επικοινωνία με το πρωτόκολλο UART μεταξύ των δύο είναι εφικτό να επιτευχθεί και με τη χρήση καλωδίων και GPIOpins μεταξύ του RaspberryPi και του UNOR3. Στην περίπτωση αυτή είναι απαραίτητος ένας level-shifter για να προστατέψει τα GPIO του Raspberry που δεν δέχονται τάση μεγαλύτερη από 3.3V ενώ τα pins του UNOR3 λειτουργούν στα 5V. Έτσι για λόγους απλότητας επιλέχθηκε η επικοινωνία με USB.
Μέσω του pythonscript θα λαμβάνονται απ' το δίτροχο ρομπότ οι αποφάσεις για την κίνηση των μοτέρ βάση των δεδομένων που θα δέχεται από τον χρήστη. Τα δεδομένα που θα δίνει ο χρήστης για την κίνηση του ρομπότ επιλέχθηκαν όπως φαίνεται παρακάτω:

Πλήκτρο W πληκτρολογίου -> κίνηση εμπρός

Πλήκτρο A πληκτρολογίου -> στροφή αριστερά

Πλήκτρο S πληκτρολογίου -> κίνηση πίσω

Πλήκτρο D πληκτρολογίου -> στροφή δεξιά

Πλήκτρο R πληκτρολογίου -> εντοπισμός θέσης (περιστροφές βηματικού μηχανισμού, λήψη φωτογραφιών και λειτουργία αλγορίθμου για εντοπισμό θέσης)

Οποιοδήποτε άλλο πλήκτρο του πληκτρολογίου -> Τέλος W, A, S, D κινήσεων

Για το pythonscript θα χρησιμοποιηθεί η βιβλιοθήκη pynput για να αναγνωρίζονται πλήκτρα από το πληκτρολόγιο ως είσοδοι για να γίνεται λήψη αποφάσεων. Ακόμα θα χρησιμοποιηθούν οι βιβλιοθήκες picamera που είναι υπεύθυνη για τον χειρισμό της κάμερας RaspberryPi, η time που θα χρησιμοποιηθεί για να γίνονται παύσεις μερικών δευτερολέπτων όπου θεωρηθεί απαραίτητη και η serial για τη σειριακή επικοινωνία μεταξύ RPi και Arduino.

Για την αρχικοποίηση της σειριακής επικοινωνίας θα χρησιμοποιηθεί η εντολή:

```
ser = serial.Serial('/dev/ttyUSB0',115200, timeout=1
```

Με την παράμετρο '/dev/ttyUSBO' ορίζουμε το σειριακό όνομα του UNOR3 για την επικοινωνία με το RaspberryPi, με την παράμετρο 115200 ορίζουμε το ρυθμό μετάδοσης (baudrate) ο οποίος πρέπει να είναι ίδιος μεταξύ του RaspberryPi και του UNOR3 για να είναι

επιτυχής η επικοινωνία και με την παράμετρο timeout = 1, ορίζουμε το τέλος χρόνου ανάγνωσης bytes στο ένα δευτερόλεπτο.

Για τον έλεγχο του ρομπότ δημιουργείται μια συνάρτηση on_press(key) η οποία θα είναι υπεύθυνη για το διάβασμα των πλήκτρων και την αποστολή μιας σειριακής εντολής στο UNOR3. Προς το παρόν οι εντολές που θα γραφτούν θα έχουν κενό περιεχόμενο το οποίο στην πορεία θα καλυφθεί μέσω του Arduinoscript.

```
defon_press(key):
if key == keyboard.Key.esc:
print('STOP')
ser.write(b'escape\n')
try:
      k = key.char# single-char keys
except:
      k = key.name  # other keys
if k in ['w', 'a', 's', 'd', 'r']: #, 'q']:  # Movement keys
      # self.keys.append(k)  # store it in global-like variable
      #print('Key pressed: ' + k)
if k == 'w':
print('Moving Forward')
ser.write(b'forward\n')
```

Αυτό το τμήμα της συνάρτησης είναι υπεύθυνο για την αποστολή σήματος σχετικά με την κίνηση των τροχών. Όπως φαίνεται από τον κώδικα, όταν πατηθεί το πλήκτρο w του πληκτρολογίου τότε το πρόγραμμα θα εμφανίσει το stringMovingForward και θα γράψει σειριακά το string χαρακτήρων ‘forward’ που θα αναγνωσθεί από το Arduino. Με την ίδια λογική λειτουργούν και οι υπόλοιπες κινήσεις.

Ακόμα μένει η λειτουργία του βηματικού μηχανισμού και της κάμερας που θα περιστρέφονται για να φωτογραφίζουν το περιβάλλον. Λόγω του καλωδίου της κάμερας δεν έχουμε τη δυνατότητα να περιστρέφουμε επ’ αορίστου τον μηχανισμό με αποτέλεσμα η υλοποίηση να γίνεται με δύο περιπτώσεις. Η μια περίπτωση θα είναι η κάμερα που είναι τοποθετημένη στο επάνω μέρος του βηματικού μηχανισμού να περιστρέφεται με τη φορά του ρολογίου και να λαμβάνει φωτογραφίες που θα ορίσουμε πως έχουν γωνία απ’ την πρώτη εως την τελευταία 0, 60, 120, 180, 240 και 300. Στην επόμενη περίπτωση η κάμερα θα περιστρέφεται αντίθετα από τη φορά του ρολογιού με γωνία 0, 300, 240, 180, 120, 60. Για να επιτευχθεί αυτός ο διαχωρισμός γίνεται στην αρχή του κώδικα αρχικοποίηση μιας μεταβλητής μετρητή:

```
count = 0
```

```
elifk == 'r':
global count
camera.start_preview()
ser.write(b'stop\n')
print('Wait, taking pictures for self-localization')

if (count % 2) == 0:
print('Clockwise Rotation')
for iinrange (6):
        sleep(2)
camera.capture('/home/pi/Desktop//PHOTOS/image%s.jpg' % i)
ser.write(b'rr\n')
# print('even ' , count)
count = count + 1
camera.stop_preview()

elif:
print('Counter Clockwise Rotation')
for iinrange (6):
        sleep(2)
camera.capture('/home/pi/Desktop//PHOTOS/image%s.jpg' % i)
ser.write(b'rl\n')
# print('odd ' , count)
count = count + 1
camera.stop_preview()
```

Με το πάτημα του πλήκτρου r του πληκτρολογίου στέλνονται σειριακά οι χαρακτήρες rr ή rl, η κάμερα προετοιμάζεται, σταματάει η κίνηση των τροχών εάν βρίσκονται σε κίνηση, και γίνεται έλεγχος για το εάν ο μετρητής είναι μονός ή ζυγός αριθμός. Εάν είναι ζυγός, γίνονται 6 δεξιόστροφες περιστροφές (rr), κάθε περιστροφή θα πρέπει να γίνεται για 60 μοίρες και στο τέλος κάθε περιστροφής γίνεται λήψη φωτογραφίας. Εάν ο αριθμός του μετρητή είναι μονός η ίδια διαδικασία γίνεται αριστερόστροφα (rl).

Τέλος στο τέλος της συνάρτησης χρησιμοποιείται από τη βιβλιοθήκη pynput το function για την αναγνώριση των πλήκτρων και γίνεται η εκκίνηση του αναγνώστη πλήκτρων.

```
listener = keyboard.Listener(on_press=on_press)#, on_release=on_release)
listener.start()  # start to listen on a separate thread
```

Το python script για την επικοινωνία μεταξύ του RPi και του UNOR3 έχει τελειώσει με επιτυχία στο σημείο αυτό. Επόμενο στάδιο είναι να δημιουργηθεί ένα arduino sketch που να λαμβάνει τις σειριακές εντολές και να τις μεταφράζει σε εντολές στα μοτέρ.

4.3.2 UNOR3 ATmega328P

Ο ελεγκτής UNOR3 θα είναι υπεύθυνος για τη μετάφραση των αποφάσεων που θα δέχεται από το pythonscript ως κίνηση τροχών. Αυτό θα γίνει με την ανάγνωση σειριακών δεδομένων και αντιστοίχιση ενεργειών στο H-bridge που είναι υπεύθυνο για τον έλεγχο των τροχών. Για το arduino sketch θα χρησιμοποιηθεί μόνο η βιβλιοθήκη TinyStepper. Θα γίνει αρχικοποίηση και αντιστοίχισητων IN1, IN2, IN3, IN4 του βηματικού μοτέρ και θα δημιουργηθεί μια σταθερά HALFSTEPS = 4096. Η σταθερά αυτή θα χρησιμοποιηθεί για να πετύχουμε μεγαλύτερη ακρίβεια με το βηματικό μοτέρ που έχει τη δυνατότητα 2048 βημάτων με half-stepping. Ακόμα πρέπει να αρχικοποιηθούν και να αντιστοιχιστούν σε pins τα 6 pins του H-bridge, δηλαδή τα EnableA και EnableB που είναι υπεύθυνα για την ταχύτητα περιστροφής του αριστερού και δεξιού τροχού αντίστοιχα και των pin1, pin2, pin3, pin4 που είναι υπεύθυνα για την φορά περιστροφής του αριστερού και του δεξιού τροχού. Ακόμα θα δημιουργηθεί ένα string χαρακτήρων “command” που θα χρησιμοποιηθεί μετέπειτα για τις εντολές που θα δέχεται ο μικροελεγκτής απ’ το Raspberry και μια σταθερά speed που θα χρησιμοποιηθεί για την ταχύτητα των τροχών.

```
#include <TinyStepper.h>// Load the Stepper library

#define IN1 13
#define IN2 12
#define IN3 9
#define IN4 8
#define HALFSTEPS 4096

String command;

// Initialize counter to use for clockwise and counterclockwise rotation
int count = 0;

//Left Motor Pins
intena= 11;
int in1 = 4;
int in2 = 3;
//Right Motor Pins
int in3 = 7;
int in4 = 6;
```

```
intenb= 10;
// wheel speed
int speed = 60;
TinySteppermyStepper(HALFSTEPS,13,12,9,8);
```

Στη συνέχεια αφού γίνουν οι απαραίτητες αρχικοποιήσεις, εντός της συνάρτησης voidsetup() θα ξεκινήσει η σειριακή επικοινωνία με το Raspberry Pi επιλέγοντας για ρυθμό μετάδοσης (baudrate) την ίδια τιμή που επιλέχθηκε και στο RaspberryPi. Ακόμα Δηλώνουμε τα τερματικά της πλακέτας L298N ως εξόδους και αρχικοποιούμε τη λειτουργία του βηματικού μοτέρ.

```
void setup() {
Serial.begin(115200); //9600

  // H bridge Pins for DC motors
pinMode(ena, OUTPUT);
pinMode(in1, OUTPUT);
pinMode(in2, OUTPUT);
pinMode(enb, OUTPUT);
pinMode(in3, OUTPUT);
pinMode(in4, OUTPUT);

myStepper.Enable();
delay(1000);
}
```

Σε αυτό το σημείο ετοιμάζουμε τη συνάρτηση voidloop() που θα τρέχει συνεχώς και θα είναι ο λήπτης των αποφάσεων κίνησης για το σύστημα του ρομπότ. Στο pythonscript που θα τρέχει στο RaspberryPi δημιουργήσαμε καταστάσεις που κατά το πάτημα ενός πλήκτρου γράφουν σειριακά ένα string χαρακτήρων. Σε αυτό το σημείο θα πρέπει να επιλέξουμε τι επιλογές και πως θα παίρνει αυτές τις επιλογές ο μικροελεγκτήςATmega328P.

Τα σενάρια που δημιουργηθούν σχετικά με την κίνηση των τροχών θα είναι 5. Τα σενάρια κίνησης τροχών είναι τα παρακάτω:

- Κίνηση μπροστά
- Κίνηση προς τα πίσω
- Περιστροφή προς τα δεξιά
- Περιστροφή προς τα αριστερά
- Τέλος κίνησης

Ακόμα, πρέπει να δημιουργηθούν 2 σενάρια για την περιστροφή του βηματικού μοτέρ που περιστρέφει την κάμερα.

- Περιστροφή βηματικού μηχανισμού 60 μοίρες δεξιά
- Περιστροφή βηματικού μηχανισμού 60 μοίρες αριστερά

Σε κάθε περίπτωση κίνησης των τροχών η ταχύτητα που θα χρησιμοποιηθεί θα είναι η μεταβλητή speed που ισούται με 60.

Για την κίνηση εμπρός, με την ανάγνωση του string χαρακτήρων “forward” οι δύο τροχοί θα περιστρέφονται εμπρός και αντίστοιχα με την ανάγνωση του string “backward” οι δύο τροχοί θα περιστρέφονται προς τα πίσω.

```
if (command.equals("forward")) {
// LEFT MOTOR FORWARD
digitalWrite(in1,HIGH);
digitalWrite(in2,LOW);
analogWrite(ena, speed);
// RIGHT MOTOR FORWARD
digitalWrite(in3,HIGH);
digitalWrite(in4,LOW);
analogWrite(enb, speed);
}


////////////////// Backward movement
else if (command.equals("backward")) {
// LEFT MOTOR BACKWARD
digitalWrite(in1,LOW);
digitalWrite(in2,HIGH);
analogWrite(ena, speed);
// RIGHT MOTOR BACKWARD
digitalWrite(in3,LOW);
digitalWrite(in4,HIGH);
analogWrite(enb, speed);
}
```

Η περιστροφή του ρομπότ θα γίνεται με άξονα το κέντρο της απόστασης μεταξύ των δύο τροχών. Για να γίνει δεξιόστροφη περιστροφή, ο δεξιός τροχός θα κινείται προς τα πίσω και ο αριστερός τροχό θα κινείται προ τα εμπρός μετά την ανάγνωση του string “right”. Αντίστοιχα

για να γίνει αριστερόστροφη περιστροφή, ο δεξιός τροχός θα κινείται προς τα εμπρός και ο αριστερός τροχός θα κινείται προς τα πίσω μετά την ανάγνωση του string “left”.

Για να σταματήσει η κίνηση με την ανάγνωση του string “stop”, θα παύει η περιστροφή των τροχών.

```
////////////////////// Left movement
else if (command.equals("left")) {
// LEFT MOTOR BACKWARD
digitalWrite(in1,LOW);
digitalWrite(in2,HIGH);
analogWrite(ena, speed);
// RIGHT MOTOR FORWARD
digitalWrite(in3,HIGH);
digitalWrite(in4,LOW);
analogWrite(enb, speed);
}


////////////////////// Right movement
else if (command.equals("right")) {
// LEFT MOTOR FORWARD
digitalWrite(in1,HIGH);
digitalWrite(in2,LOW);
analogWrite(ena, speed);
// RIGHT MOTOR BACKWARD
digitalWrite(in3,LOW);
digitalWrite(in4,HIGH);
analogWrite(enb, speed);
}


////////////////////// Stop movement
else if (command.equals("stop")) {
digitalWrite(in1, LOW);
digitalWrite(in2, LOW);
digitalWrite(in3, LOW);
digitalWrite(in4, LOW);
delay(2000);
}
```

Για την περιστροφή του βηματικού μηχανισμού με την ανάγνωση του string “rr” θα γίνεται περιστροφή προς τα δεξιά κατά 60 μοίρες και με την ανάγνωση του string “rl” θα γίνεται περιστροφή του μηχανισμού 60 μοίρες προς τα αριστερά.

```
//////////////////// rotate right for pictures and localization
else if (command.equals("rr")){
myStepper.Move(60);
}


//////////////////// rotate right for pictures and localization
else if (command.equals("rl")){
myStepper.Move(-60);
```

Αυτές είναι όλες οι αποφάσεις για τις οποίες θα είναι υπεύθυνος ο μικροελεγκτής.

ΟΛΟΚΛΗΡΩΜΕΝΟ ΣΥΣΤΗΜΑ ΑΠΟΦΑΣΕΩΝ – ΚΙΝΗΣΗΣ

Συνοψίζοντας τα παραπάνω, το σύστημα θα αποτελείται αρχικά από έναν χρήστη που θα δίνει τις εντολές - επιθυμητές κινήσεις χρησιμοποιώντας το πληκτρολόγιο. Το Python script θα είναι υπεύθυνο για την ανάγνωση των πλήκτρων που πληκτρολογούνται από τον χρήστη και για την «αντιστοίχιση» των πλήκτρων με την κατάλληλη πράξη. Η αντιστοίχιση γίνεται όταν αφού αναγνωρίσει πως πατήθηκε κάποιο πλήκτρο ενδιαφέροντος στέλνει σειριακά τις κατάλληλες πληροφορίες στον μικροελεγκτή UNOR3. Ο μικροελεγκτής αφού δεχτεί τις σειριακές πληροφορίες κάνει τη δική του «αντιστοίχιση» και για την ανάγνωση πληροφοριών ενδιαφέροντος, στέλνει τα κατάλληλα ρεύματα στο H-Bridge το οποίο κινεί τους τροχούς.

4.4 Βαθμονόμηση κάμερας

Το πρώτο βήμα πριν προχωρήσουμε στην επίλυση του προβλήματοςεντοπισμού θέσης θα είναι να γίνει βαθμονόμηση της κάμερας. Όπως αναφέρθηκε προηγουμένωςη βαθμονόμηση είναι ένας σημαντικός παράγοντας για τη χρήση της κάμερας ως αισθητήρα ώστε να βρεθεί ο πίνακας εγγενών παραμέτρων (camera matrix) και οι συντελεστές παραμόρφωσης.

Εικόνα13 Δείγμα εικόνας για βαθμονόμηση κάμερας

Η βαθμονόμηση θα γίνει με τη βοήθεια της βιβλιοθήκης OpenCV που παρέχει αλγόριθμο για την αναγνώριση μοτίβου σκακιέρας και χρήση των χαρακτηριστικών της σκακιέραςκαι για τη βαθμονόμηση της κάμερας που στηρίζεται στη μέθοδο Zhang[5]. Θα χρησιμοποιηθούν 12 εικόνες με τη σκακιέρα σε διαφορετικές τοποθεσίες στον χώρο. Αρκεί να γνωρίζουμε το μέγεθος των τετραγώνων της σκακιέρας (25mm στην προκειμένη περίπτωση) και να επιλέξουμε το επιθυμητό ποσό γωνιώνπου θέλουμε να εντοπισθεί από τετράγωνο σε τετράγωνο που επιλέχθηκε να είναι 6x9.

Αφού δημιουργηθούν μεταβλητές για τα ανύσματα της τοποθεσίας των σημείων της εικόνας και της σκηνής, βρίσκονται οι γωνίες με την εντολή:

```
ret, corners = cv2.findChessboardCorners(gray, CHECKERBOARD)
```

Και στο επόμενο στάδιο οι μεταβλητές ανυσμάτων εικόνας και σκηνής γεμίζουν με τις αντίστοιχες 2D και 3D θέσεις των γωνιών των τετραγώνων της σκακιέρας.

Στο σημείο αυτό μπορεί να χρησιμοποιηθεί ο αλγόριθμος calibrateCamera της βιβλιοθήκης OpenCV που θα κάνει τα παρακάτω βήματα:

- Υπολογισμός εγγενών παραμέτρων και δημιουργία κενού πίνακα συντελεστών παραμόρφωσης
- Εκτίμηση πόζας χρησιμοποιώντας τον ενσωματωμένο αλγόριθμοsolvePnP() για την εύρεση πίνακα εσωτερικών παραμέτρων και συντελεστών παραμόρφωσης
- Εφαρμογή αλγορίθμου βελτίωσης για την μείωση σφαλμάτων επαναπροβολής.

```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1],None,None)
```

Από τη βαθμονόμηση της κάμερας προκύπτουν τα παρακάτω δεδομένα που μπορούν να χρησιμοποιηθούν στο επόμενο στάδιο της εργασίας.

Camera Matrix =

[1.51687387e+03	0.00000000e+00	1.00245102e+03]
[0.00000000e+00	1.51304681e+03	5.22219286e+02]
[0.00000000e+00	0.00000000e+00	1.00000000e+00]

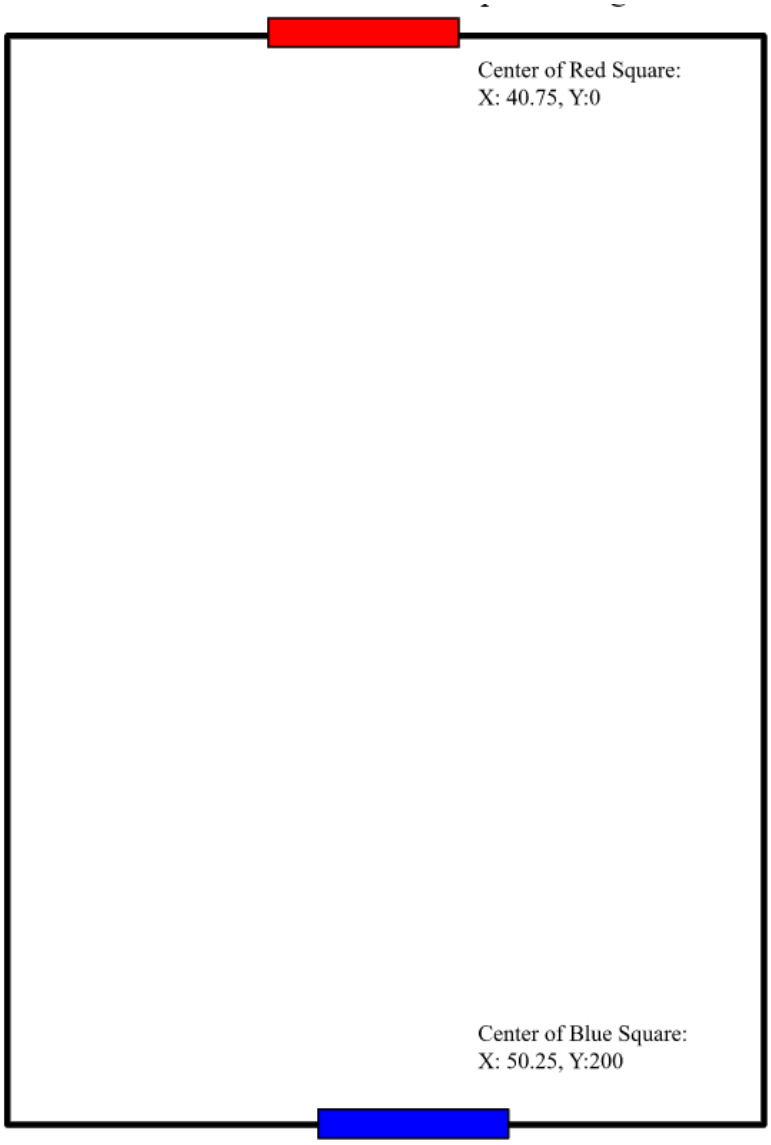
DistortionCoefficients =

[[0.17372084 0.10634241 -0.00611513 0.00981087 -4.07591309]]

Η βαθμονόμηση της κάμερας έχει πραγματοποιηθεί επιτυχώς και ο πίνακας εγγενών παραμέτρων και οι συντελεστές παραμόρφωσης αποθηκεύονται σαν πίνακες σε αρχείο για τη χρήση τους στην επίλυση του προβλήματος εντοπισμού θέσης.

4.5 Επίλυση προβλήματος εντοπισμού θέσης

Για να λυθεί το συγκεκριμένο πρόβλημα πρέπει να εντοπισθεί η θέση και ο προσανατολισμός του ρομπότ μέσα σε ένα γνωστό χώρο εργασίας με τη χρήση μονάχα μιας κάμεραςrgb. Η κάμερα που βρίσκεται τοποθετημένη πάνω στο ρομπότ περιστρέφεται από τον βηματικόμηχανισμόκάνει λήψη εικόνας και περιστρέφεται για 60 μοίρες κάνοντας λήψεις μέχρι να επιστρέψει στην αρχική της θέση, καλύπτοντας έτσι περιστροφικά όλο το περιβάλλον. Για να το πετύχει αυτό πρέπει να χρησιμοποιηθούν οι γνωστές πληροφορίες για το περιβάλλον στο οποίο βρίσκεται το ρομπότ. Οι πληροφορίες αυτές είναι το μέγεθος, η τοποθεσία και το χρώμα των τετραγώνων που λειτουργούν ως σημάδια για τον εντοπισμό θέσης τουρομπότ. Το περιβάλλον στο οποίο τοποθετείται το ρομπότ φαίνεται στην παρακάτω κάτοψη:



Εικόνα14Οπτικοποίηση κάτοψης χώρου

Για την οπτικοποίηση του προβλήματος θα δημιουργηθεί στην Python ένα αρχείο μεμια κλάσηCanvas η οποία θα λειτουργήσει σαν κανβάς / πίνακας κάτοψης στον οποίο καμβά θα δημιουργηθούν συναρτήσεις για την οπτικοποίηση των θέσεων των τετραγώνων και της σχετικής θέσης του ρομπότκαιεπίσης οι συναρτήσεις που θα χρησιμοποιηθούν για τους υπολογισμούς της θέσης.

Το πρόβλημα αυτό χωρίζεται σε δύο μέρη. Το πρώτο είναι η εύρεση της τοποθεσίας του ρομπότ στους άξονες Χ και Υ και το δεύτερο η εύρεση του προσανατολισμού του ρομπότ στον χώρο.

Για την εύρεση των τετραγώνων θα εφαρμοστούν τα παρακάτω βήματα:

1. Φόρτωση εικόνας
2. Επεξεργασία εικόνας (θόλωμα με σκοπό να είναι ευκολότερος ο εντοπισμός των τετραγώνων)
3. Μετατροπή του χρωματικού χώρου της εικόνας από RGBσε HSVγια την καλύτερη λήψη πληροφοριών από την εικόνα
4. Τμηματοποίηση της εικόνας στον χρωματικό χώρο HSVμε επιθυμητές τιμές για κόκκινο και μπλε ώστε να μας επιστραφούν οι κόκκινες και μπλέ περιοχές (περιοχές ενδιαφέροντος, θα μας επιστρέψουν μια μάσκα)
5. Εύρεση περιγραμμάτων τετραγώνων και αφαίρεση ψευδών θετικών
6. Χρήση των επεξεργασμένων περιγραμμάτων για εύρεση του κεντρικού σημείου των τετραγώνων.

Εύρεση θέσης

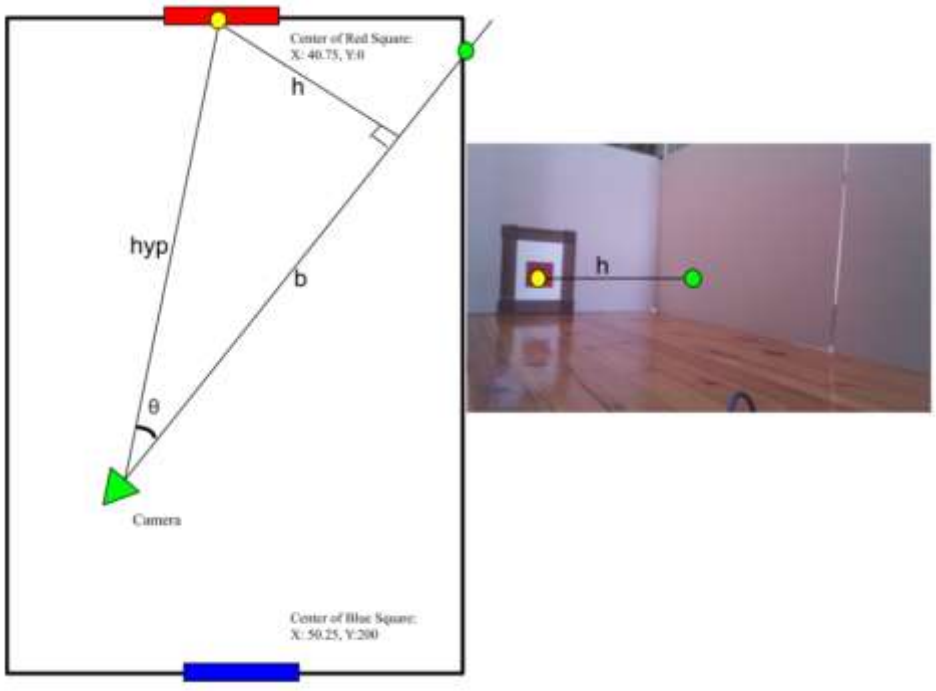
Για την εύρεση της θέσης κάμερας γίνονται τα παρακάτω βήματα:

1. Εύρεση κεντρικού σημείου κόκκινου και μπλε τετραγώνου όπως αναφέρεται στο τελευταίο βήμα της εύρεσης των τετραγώνων παραπάνω
2. Εύρεση οπτικού πεδίου (Field of view) της κάμερας ως προς τις κατευθύνσεις Χκαι Υμε τη βοήθεια του πίνακα της κάμερας που βρέθηκε από τη βαθμονόμηση κάμερας(camera matrix) με χρήση της παρακάτω φόρμουλας:

   Fov_x = 2 * arctan(Image Width / 2 * focal length x)

   Fov_y = 2 * arctan(Image Height / 2 * focal length y)

3. Αφού γίνει ανίχνευση τετραγώνου, το κεντρικό σημείο του τετραγώνου της εικόνας χρησιμοποιείται για να βρεθεί η γωνία που δημιουργείται μεταξύ αυτού και του κεντρικού σημείου της εικόνας. Στην παρακάτω εικόνα[15] γίνεται η αναπαράσταση του οπτικού πεδίου της κάμερας κατά την ανίχνευση ενός τετραγώνου στη λήψη και μεταφράζεται και σε κάτοψη. Στην αναπαράσταση ο κίτρινος κύκλος αναπαριστά το κέντρο του τετραγώνου και ο πράσινος κύκλος αναπαριστά το κέντρο της εικόνας. Το μέγεθος του τετραγώνου είναι γνωστό σε εκατοστά στο φυσικό του μέγεθος αλλά και σε πίξελς μέσω της φωτογραφίας. Αυτό μας επιτρέπει να δημιουργήσουμε μια μεταβλητή που θα μετατρέπει τα εικονοστοιχεία σε εκατοστά. Η απόσταση (h) μεταξύ του κέντρου της εικόνας και του κέντρου του τετραγώνου υπολογίζεται σε εικονοστοιχεία από την εικόνα και μετατρέπεται σε εκατοστά με τη προαναφερθείσα μεταβλητή. Αυτό φαίνεται στη δεξιά εικόνα 2 διαστάσεων. Αυτός ο τρόπος επιλέχθηκε για τη δημιουργία ενός νοητού ορθογώνιου τριγώνου που θα μας επιτρέψει να κάνουμε υπολογισμούς για τη λύση του προβλήματος. Η απόσταση μεταξύ του κέντρου της εικόνας και του κέντρου του τετραγώνου που έχει ανιχνευτεί είναι η βάση του τριγώνου στην οποία αντιστοιχείται το γράμμα h όπως φαίνεται στην παραπάνω εικόνα. Ακόμα πρέπει να βρεθεί η γωνία θ του παραπάνω τριγώνου. Το οπτικό πεδίο με τον τρόπο που υπολογίστηκε προηγουμένως μπορεί να μας βοηθήσει να λύσουμε αυτό το πρόβλημα.



Εικόνα15 Αναπαράσταση τριγώνου ρομπότ-κεντρικού σημείου-κέντρου τετραγώνου

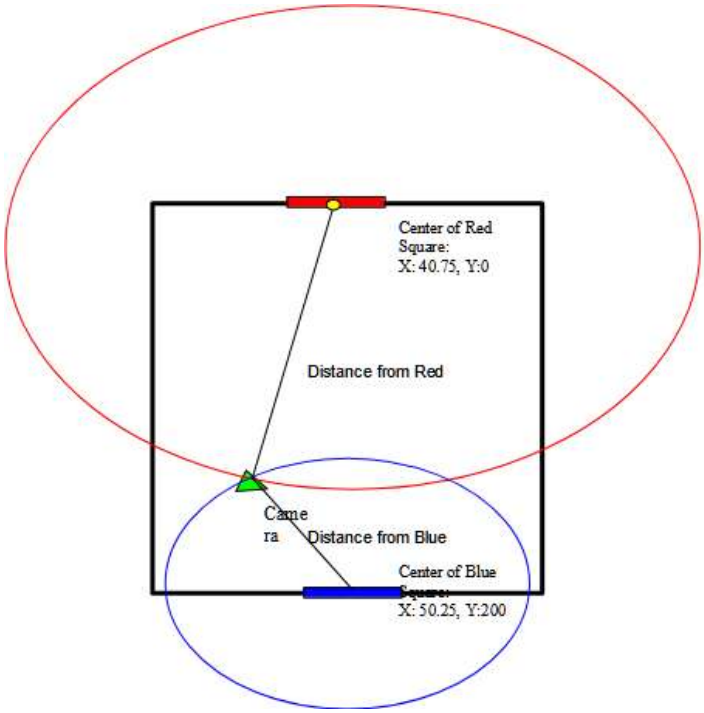
Για να βρεθεί η γωνία θ αρκεί να διαιρέσουμε την απόσταση h με το πλάτος την εικόνας και να πολλαπλασιάσουμε με το οπτικό πεδίο FOV_x όπως φαίνεται από την σχέση: $\theta = (h(pixels) / Image\ width) * FOV_x$

Βρίσκοντας τη γωνία θ μπορούμε με ευκολία να βρούμε την υποτείνουσα με την τριγωνομετρική σχέση:

$hyp = h/\arcsin(\theta)$

όπου η υποτείνουσα θα είναι η απόσταση από την κάμερα στο κέντρο του ανιχνευμένου τετραγώνου. Αυτό το βήμα επαναλαμβάνεται μετά από την ανίχνευση του δεύτερου τετραγώνου για να υπολογισθεί η απόσταση της κάμερας και από τα δύο τετράγωνα.

4. Αφού υπολογισθούν οι αποστάσεις των δύο τετραγώνων από την κάμερα, σχεδιάζονται δύο νοητοί κύκλοι με κέντρα τα τετράγωνα των οποίων οι αποστάσεις υπολογίστηκαν και ακτίνα κύκλου τις αντίστοιχες αποστάσεις. Οι κύκλοι αυτοί θα τέμνονται σε δύο σημεία και αυτά τα σημεία είναι οι δύο πιθανές θέσεις x,yτου ρομπότ στον χώρο.



Εικόνα16Εύρεσηπιθανώνθέσεων

Εύρεση προσανατολισμού

Για την εύρεση του προσανατολισμού χρησιμοποιούνται τα παραπάνω αποτελέσματα από την εύρεση θέσης και προστίθενται τα παρακάτω βήματα:

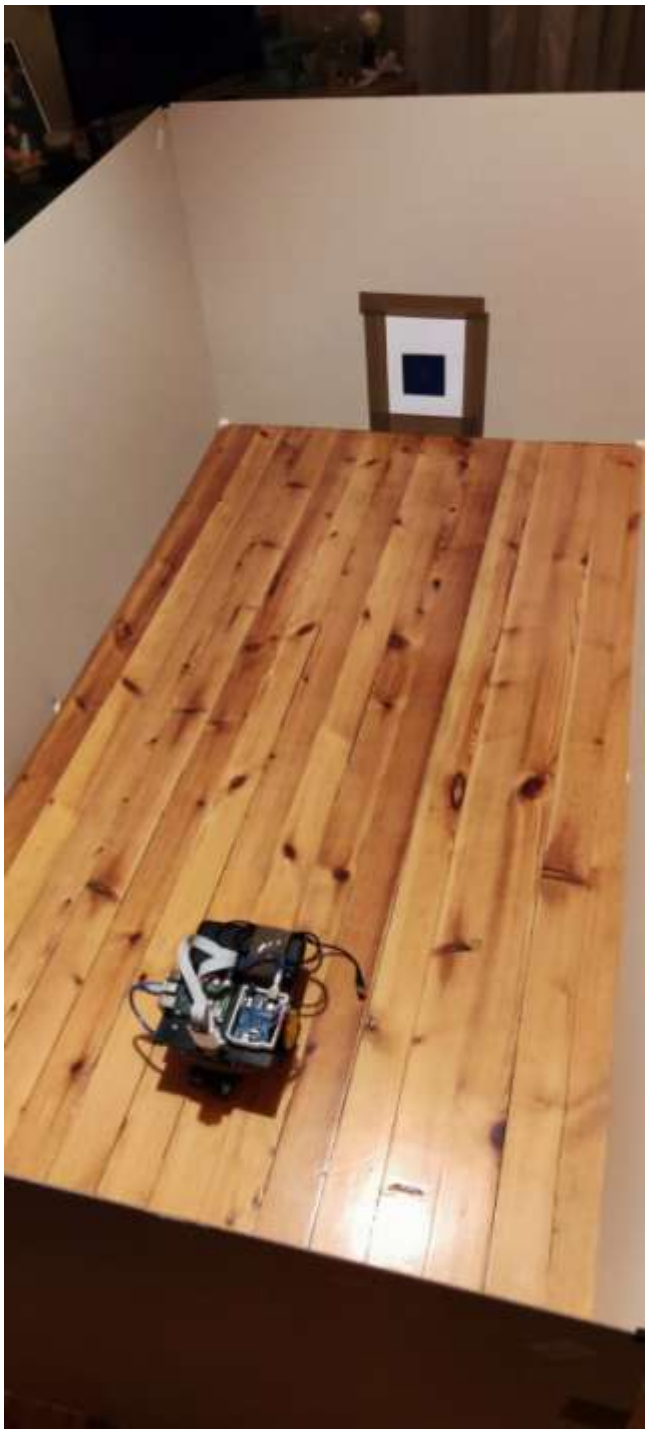
1. Οι έξι εικόνες που θα τραβήξει η κάμερα του ρομπότ έχουν το όνομα image”x” όπου το “x” είναι η γωνία λήψης της φωτογραφίας σε σχέση με το παγκόσμιο σύστημα συντεταγμένων. Αφού λοιπόν γίνουν όλες οι περιστροφές των 60 μοιρών και οι λήψειστα ονόματα των εικόνων είναι image0, image60, image120, image180, image240, image 270 και image300.

2. Αφού βρεθεί η σχετική γωνία περιστροφής χρησιμοποιούνται ήδη υπάρχουσες γνωστές πληροφορίες για να βρεθεί η σωστή αρχική γωνία προσανατολισμού. Από τα δεδομένα του προβλήματος είναι γνωστό που όταν το ρομπότ αντικρίζει το μπλε τετράγωνο η γωνία είναι 0 και όταν αντικρίζει το κόκκινο τετράγωνο η γωνία είναι 180. Έτσι στη σχετική γωνία του προηγούμενου βήματος προστίθενται 180 μοίρες για το κόκκινο τετράγωνο και στην περίπτωση του μπλε η σχετική γωνία παραμένει ίδια. Αυτό δίνει δύο πιθανές γωνίες ως λύσεις.

3. Τέλος παίρνεται ο μέσος όρος των δύο γωνιών για να βρεθεί η τελική γωνία.

5. ΑΠΟΤΕΛΕΣΜΑΤΑ

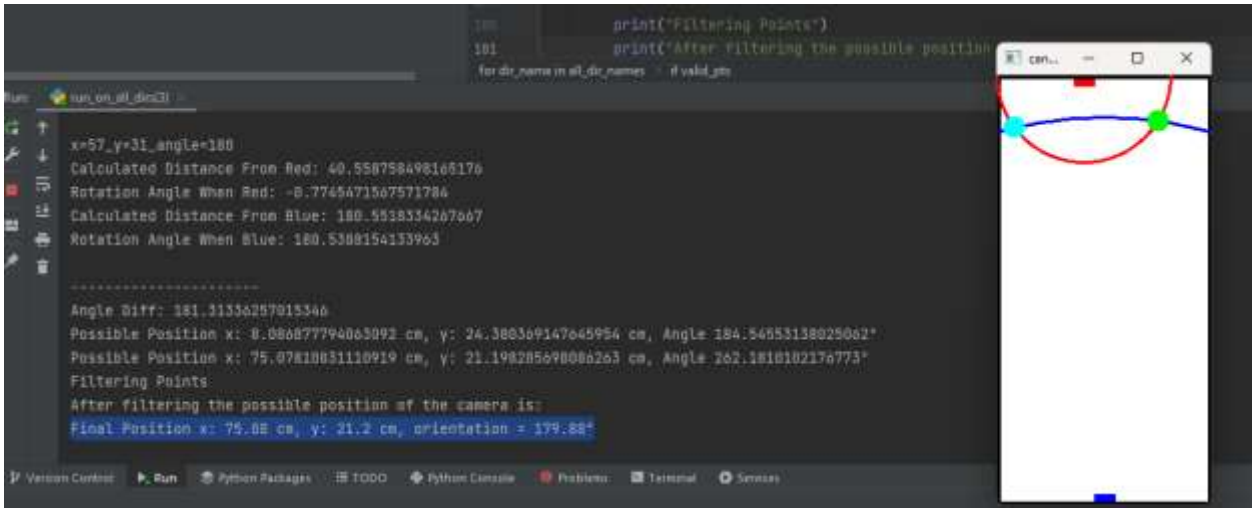
Ο πειραματικός χώρος για την χρήση του συστήματος παρουσιάζεται από την παρακάτω εικόνα [Εικόνα 17]:



Εικόνα17Πειραματικός χώρος εργασίας

5.1 Dataset 1

Οι παραπάνω λήψεις έγιναν με την τοποθεσία της κάμερας στο σημείο X,Y =(57,31) στον χώρο εργασίας και τον προσανατολισμό του 180° όπου 0 μοίρες θεωρείται η κατακόρυφη κατεύθυνση κοιτώντας το μπλε τετράγωνο και αυξάνεται δεξιόστροφα προς τις 180° όταν φτάνει να κοιτάζει κάθετα το κόκκινο τετράγωνο.

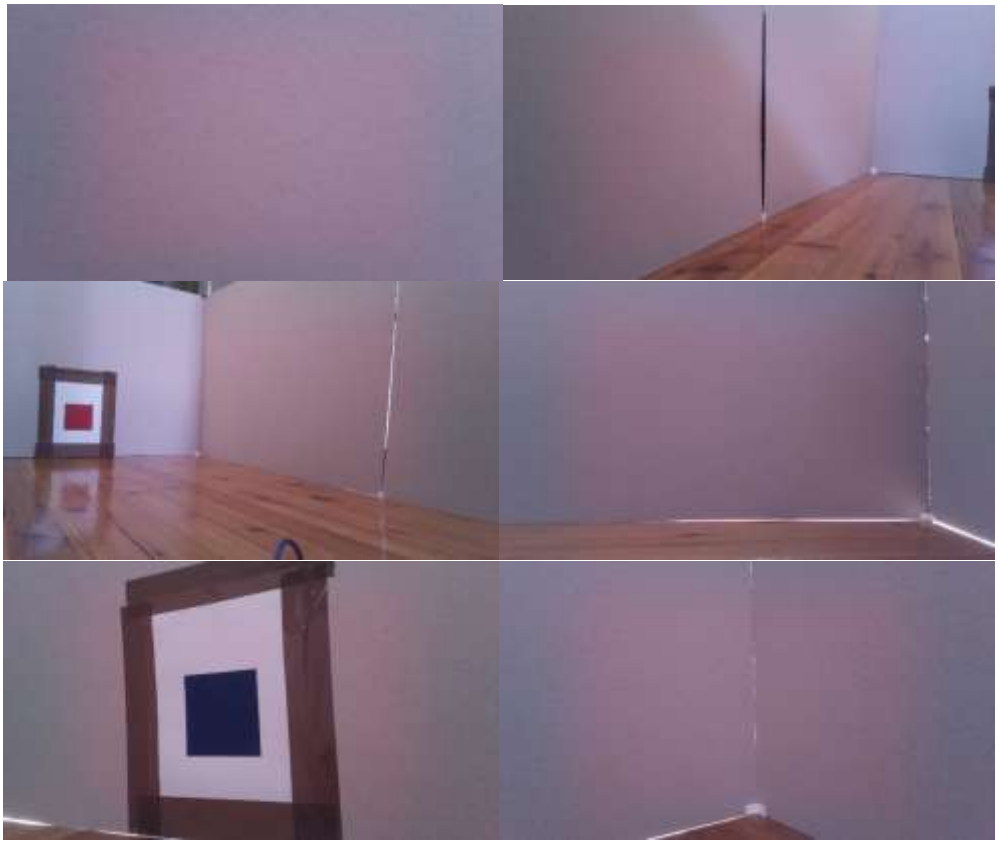


Εικόνα18 Υπολογισμός θέσηςdataset 1 (x57,y31, θ=180)

Για το dataset 1, ο αλγόριθμος υπολογίζει δύο ζεύγη πιθανών θέσεων [**Εικόνα18**]: x_1,y_1 = (8.08,24.38) και x_2,y_2= (75.07, 21.19). Ακόμα υπολογίζει τη γωνία στις 179.88°. Μετά από φιλτράρισμα βάση των ονομάτων των εικόνων και της τοποθεσίας των τετραγώνων μέσα σε αυτές επιλέγει το δεύτερο ζεύγος ως το πιο πιθανό και υπολογίζει την τελική τοποθεσία του ρομπότ ως: x: 75.08 cm, y: 21.2 cm, orientation = 179.88°.

Παρατηρείται πως οι θέσεις x και yέχουν μεγάλη απόκλιση από τις πραγματικές τιμές. Η απόκλιση είναι 18cm στον άξονα x και -10cm στον άξονα y. Η τιμή για τη γωνία του προσανατολισμού βρίσκεται με ακρίβεια και εντός λογικών περιθωρίων απόκλισης.

5.2 Dataset 2



Οι παραπάνω λήψεις έγιναν με την τοποθεσία της κάμερας στο σημείο X,Y =(26,160) και προσανατολισμό στις 270°.

Εικόνα19Υπολογισμός θέσηςdataset2 (x26,y160,θ=270)

Για το dataset2, ο αλγόριθμος υπολογίζει δύο ζεύγη πιθανών θέσεων [Εικόνα 19]: x_1,y_1 = (25.51,153.54) και x_2,y_2= (69.96, 151.43). Ακόμα υπολογίζει τη γωνία στις 256.71°. Μετά από φιλτράρισμα βάση των ονομάτων των εικονών και της τοποθεσίας των τετραγώνων μέσα σε αυτές επιλέγει το πρώτο ζεύγος ως το πιο πιθανό και υπολογίζει την τελική τοποθεσία του ρομπότ ως: x: 25.51cm, y: 153.54cm, orientation = 256.71°.

Αυτή τη φορά οι αποκλίσεις των τιμών είναι -0.5cm στον άξονα x,-6.5cm στον άξονα y και -3.3°, τιμές που δεν απέχουν πολύ από τις πραγματικές.

5.3 Dataset 3

Εικόνα20Υπολογισμός θέσηςdataset 3 (x41, y153, θ=0)

Οι παραπάνω λήψεις έγιναν με την τοποθεσία της κάμερας στο σημείο X,Y =(41,153) και προσανατολισμό στις 0°.

Για το dataset3, ο αλγόριθμος υπολογίζει δύο ζεύγη πιθανών θέσεων [**Εικόνα20**]: x_1,y_1 = (11.73, 166.77) και x_2,y_2= (84.94, 163.29). Ακόμα υπολογίζει τη γωνία στις 173.55°.

Όπως φαίνεται συγκρίνοντας τις εικόνες και το γράφημα του αλγορίθμου η θέση εντοπισμού στο γράφημα είναι λανθασμένη και στις δύο περιπτώσεις.

5.4 Συμπεράσματα

Από τους παραπάνω πειραματισμούς προκύπτει πως ο αλγόριθμος έχει τη δυνατότητα να υπολογίσει με ακρίβεια τη θέση του ρομπότ στον χώρο όμως γίνονται και λανθασμένες εκτιμήσεις απόστασης κάτι που οφείλεται στην αρχική μέτρηση της απόστασης του κέντρου των τετραγώνων από το ρομπότ.

Δύο ακόμα σημαντικοί παράγοντες είναι η ένταση της φωτεινότητας στον χώρο ανίχνευσης των τετραγώνων και το φυσικό μέγεθος του φακού που λόγω του μικρού μεγέθους του δεν έχει τη δυνατότητα να επιτρέψει σε περισσότερο φως να φτάνει στο οπτικό κέντρου του φακού.

Ο πρώτος παράγοντας δηλαδή η ένταση της φωτεινότητα, έχει τη δυνατότητα να κάνει τη λειτουργία αυτού του συστήματος εντοπισμού τελείως αδύνατη αφού χωρίς αρκετό φως ο αισθητήρας της θέσης έχει πρόβλημα να ξεχωρίσει τις χρωματικές τιμές των τετραγώνων και έτσι δεν μπορεί να προσανατολιστεί. Αυτό σημαίνει πως για να εφαρμοστεί ένα τέτοιο σύστημα θα πρέπει να προϋπάρχει μελέτη που να αποδεικνύει πως ο φωτισμός του περιβάλλοντος του χώρου εργασίας θα είναι σε ικανοποιητικά επίπεδα ώστε η λειτουργία να επιτυγχάνεταιμε ακρίβεια.

Ο δεύτερος παράγοντας δηλαδή η κάμερα που χρησιμοποιείται είναι ένα πολύ σημαντικό μέρος για την σωστή επίλυση του προβλήματος εντοπισμού θέσης ενώ την ίδια στιγμή όσο καλύτερος και μεγαλύτερος αισθητήρας χρησιμοποιηθεί τόσο περισσότερο φως θα είναι σε θέση να φτάσει στο οπτικό κέντρο του αισθητήρα όμως η τιμή του αισθητήρα θα είναι υψηλότερη.

5.5Μελλοντική επέκταση

Τα αποτελέσματα δείχνουν πως ο εντοπισμός θέσης με τη χρήση μιας κάμερας μπορεί να επιτευχθεί όμως χρειάζονται κάποιες βελτιώσεις. Για τη χρήση του συγκεκριμένου συστήματος και εξάλειψη των «απρόβλεπτων» λανθασμένων μετρήσεων μια αλλαγή θα ήταν η καθιέρωση λειτουργίας χρήσης σε έναν χώρο εργασίας με ετικέτες αρκετά μεγάλου μεγέθους ώστε να εξαλειφθεί το λάθος κατά τη μέτρηση αποστάσεων.

Ακόμα και μετά από αυτή την αλλαγή όμως, το σύστημα της συγκεκριμένες εργασίας είναι ένα παλαιωμένο σύστημα που χρειάζεται πολύ χρόνο και ακόμα περισσότερο κάποιον χειριστή για να πετύχει τον στόχο του, δηλαδή την κίνηση του μέσα στον χώρο και τον εντοπισμό θέσης του. Ακόμα για να γίνει αυτό το ρομπότ πρέπει να σταματήσει την κίνηση του και να κάνει λήψη 6 φωτογραφιών που έπειτα πρέπει να επεξεργαστούν για να φτάσει στον χρήστη η πληροφορία της θέσης και προσανατολισμού του ρομπότ.

Το παραπάνω σύστημα θα μπορούσε επομένως να εξελιχθεί και να γίνει αυτόνομο με τη χρήση μιας τρισδιάστατης κάμερας που καλύπτει περιστροφικά όλο το οπτικό πεδίο του ρομπότ. Με

τον τρόπο αυτό δεν υπάρχει η ανάγκη της παύσης της κίνησης του ρομπότ και της αναμονής για περιστροφή στον χώρο και λήψη φωτογραφιών. Η ίδια λογική της εργασίας μπορεί να εφαρμοστεί σε μια ροή βίντεο που θα καταγράφεται από το ρομπότ προσφέροντας τη δυνατότητα για εντοπισμό θέσης και προσανατολισμού σε πραγματικό χρόνο με συχνέςανανεώσεις τοποθεσίας για κάθε καρέ που λαμβάνεται και υπολογίζεται από τον αλγόριθμο.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, Introduction to autonomous mobile robots. Cambridge, MA: MIT Press, 2011.

[2] S. J. Lee, G. Tewolde, J. Lim, and J. Kwon, "QR-code based localization for indoor mobile robot with validation using a 3D optical tracking instrument," 2015 IEEE International Conference on Advanced Intelligent Mechatronics (AIM), 2015

[3] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative Adversarial Networks: An overview," IEEE Signal Processing Magazine, vol. 35, no. 1, pp. 53–65, 2018.

[4] "Camera calibration and 3D reconstruction," OpenCV. [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html.

[5] Z. Zhang, "A flexible new technique for camera calibration," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 11, pp. 1330–1334, 2000.

[6] C. Stachniss, "Camera Calibration: Direct Linear Transform," in Photogrammetry & Robotics Lab.

[7] X. X. Lu, "A review of solutions for perspective-N-point problem in camera pose estimation," Journal of Physics: Conference Series, vol. 1087, p. 052009, 2018.

ΕΙΚΟΝΕΣ

Εικόνα 1: R. Sanketh, “MANUAL ROBOTICS,” *Medium*. [Online]. Available: https://medium.com/manual-robotics/drives-76c2b2dac97c.

Εικόνα 2:S. J. Lee, G. Tewolde, J. Lim, and J. Kwon, “QR-code based localization for indoor mobile robot with validation using a 3D optical tracking instrument,” 2015 IEEE International Conference on Advanced Intelligent Mechatronics (AIM), 2015

Εικόνα 3: P. Sharma, “A Practical Guide to Object Detection using the Popular YOLO Framework – Part III (with Python codes),” *Analytics Vidhya*, 06-Dec-2018. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framewor-python/.

Εικόνα 4: U. Erkan and L. Gökrem, “A new method based on pixel density in salt and pepper noise removal,” *TÜBİTAK Academic Journals*. [Online]. Available: https://journals.tubitak.gov.tr/elektrik/vol26/iss1/15/.

Εικόνα 5 & 6& 7: “Camera calibration and 3D reconstruction,” *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html.

Εικόνα 8: V. Lepetit , “Low complexity keypoint recognition and pose estimation - [PPT PowerPoint],” *vdocuments.mx*. [Online]. Available: https://vdocuments.mx/low-complexity-keypoint-recognition-and-pose-estimation-vincent-lepetit.html. [Accessed: 07-Mar-2023].

Εικόνα 9 & 10: Πλάγια όψη και κάτοψη του ρομπότ

Εικόνα 12: Συνδεσμολογία συστήματος

Εικόνα 13: Δείγμα εικόνας για βαθμονόμηση κάμερας

Εικόνα 14:Οπτικοποίησηκάτ

οψης χώρου

Εικόνα 15: Αναπαράσταση τριγώνου ρομπότ-κεντρικού σημείου-κέντρου τετραγώνου

Εικόνα 16: Εύρεση πιθανών θέσεων

Εικόνα 17: Πειραματικός χώρος εργασίας

Εικόνα 18: Υπολογισμός θέσης dataset 1 (x57, y31, θ=180)

Εικόνα 19: Υπολογισμός θέσης dataset 2 (x26, y160, θ=270)

Εικόνα 20: Υπολογισμός θέσης dataset 3 (x41, y153, θ=0)

ΠΑΡΑΡΤΗΜΑ ΚΩΔΙΚΑ

ArduinoScriptγια λειτουργία τροχών και επικοινωνία με RPi

```
// Georgios Eleftheriadis Thesis Project
// MSc in Robotics, IHU Serres
// Robot Remote Keyboard Control

#include <TinyStepper.h>    // Load the Stepper library
#define IN1 13
#define IN2 12
#define IN3 9
#define IN4 8
#define HALFSTEPS 4096

String command;

// Initialize counter to use for clockwise and counterclockwise rotation
int count = 0;

//Left Motor Pins
intena = 11;
int in1 = 4;
int in2 = 3;
//Right Motor Pins
int in3 = 7;
int in4 = 6;
```

```
intenb = 10;
// wheel speed
int speed = 60;

TinySteppermyStepper(HALFSTEPS,13,12,9,8);

void setup() {
Serial.begin(115200); //9600

  // H bridge Pins for DC motors
pinMode(ena, OUTPUT);
pinMode(in1, OUTPUT);
pinMode(in2, OUTPUT);
pinMode(enb, OUTPUT);
pinMode(in3, OUTPUT);
pinMode(in4, OUTPUT);

myStepper.Enable();
delay(1000);
}

void loop() {

if (Serial.available()) {
command = Serial.readStringUntil('\n');
```

```
command.trim();

   ////////////////////// Forward movement
if (command.equals("forward")) {
   // LEFT MOTOR FORWARD
digitalWrite(in1,HIGH);
digitalWrite(in2,LOW);
analogWrite(ena, speed);
   // RIGHT MOTOR FORWARD
digitalWrite(in3,HIGH);
digitalWrite(in4,LOW);
analogWrite(enb, speed);
   }


   ////////////////////// Backward movement
else if (command.equals("backward")) {
   // LEFT MOTOR BACKWARD
digitalWrite(in1,LOW);
digitalWrite(in2,HIGH);
analogWrite(ena, speed);
   // RIGHT MOTOR BACKWARD
digitalWrite(in3,LOW);
digitalWrite(in4,HIGH);
analogWrite(enb, speed);
```

```
    }


    ////////////////////// Left movement
else if (command.equals("left")) {
    // LEFT MOTOR BACKWARD
digitalWrite(in1,LOW);
digitalWrite(in2,HIGH);
analogWrite(ena, speed);
    // RIGHT MOTOR FORWARD
digitalWrite(in3,HIGH);
digitalWrite(in4,LOW);
analogWrite(enb, speed);
    }


    ////////////////////// Right movement
else if (command.equals("right")) {
    // LEFT MOTOR FORWARD
digitalWrite(in1,HIGH);
digitalWrite(in2,LOW);
analogWrite(ena, speed);
    // RIGHT MOTOR BACKWARD
digitalWrite(in3,LOW);
digitalWrite(in4,HIGH);
```

```
analogWrite(enb, speed);
   }


   ////////////////////// Stop movement
else if (command.equals("stop")) {
digitalWrite(in1, LOW);
digitalWrite(in2, LOW);
digitalWrite(in3, LOW);
digitalWrite(in4, LOW);
delay(2000);
   }


   ////////////////////// rotate right for pictures and localization
else if (command.equals("rr")){
myStepper.Move(60);
   }


   ////////////////////// rotate right for pictures and localization
else if (command.equals("rl")){
myStepper.Move(-60);
   }
```

```
else {
Serial.println("error");
}
  }
}
```

PythonScriptγιατηβαθμονόμησητηςκάμερας

```
importnumpy as np

import cv2

import glob


# Defining the dimensions of checkerboard

CHECKERBOARD = (6,9)

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 25, 0.001)
#25mm squares


# Creating vector to store vectors of 3D points for each checkerboard image

objpoints = []

# Creating vector to store vectors of 2D points for each checkerboard image

imgpoints = []


# Defining the world coordinates for 3D points

objp = np.zeros((1, CHECKERBOARD[0] * CHECKERBOARD[1], 3), np.float32)

objp[0,:,:2] = np.mgrid[0:CHECKERBOARD[0], 0:CHECKERBOARD[1]].T.reshape(-1, 2)

prev_img_shape = None


images = glob.glob("F:/ROBOTICS/ThesisPython/Calibration/*.jpg")


forfname in images:

img = cv2.imread(fname)

gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

```
    # Find the chess board corners
ret, corners = cv2.findChessboardCorners(gray, CHECKERBOARD)#,
cv2.CALIB_CB_ADAPTIVE_THRESH + cv2.CALIB_CB_FAST_CHECK +
cv2.CALIB_CB_NORMALIZE_IMAGE)


    # If found, add object points, image points (after refining them)
if ret == True:
objpoints.append(objp)


        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
imgpoints.append(corners2)


        # Draw and display the corners
img = cv2.drawChessboardCorners(img, CHECKERBOARD, corners2, ret)
cv2.imshow('img',img)
cv2.waitKey(500)


cv2.destroyAllWindows()


ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-
1],None,None)


print("Camera matrix - intrinsic parameters - pinakaseswterikwnparametrwn : \n")
print(mtx)
print("distortion coefficients - parametroiparamorfwshs : \n")
```

```
print(dist)


fs_write = cv2.FileStorage('F:/ROBOTICS/ThesisPython/Calibration/CamMat_DistCoef.yml', cv2.FILE_STORAGE_WRITE)

arr = np.random.rand(5, 5)

fs_write.write("K", mtx, )

fs_write.write("D", dist)

fs_write.release()
```

Αρχείο yaml με τα αποτελέσματα της βαθμονόμησης κάμερας

%YAML:1.0

K: !!opencv-matrix

rows: 3

cols: 3

dt: d

data: [1.5168738713829448e+03, 0., 1.0024510197065216e+03, 0.,

1.5130468073952691e+03, 5.2221928642186981e+02, 0., 0., 1.]

D: !!opencv-matrix

rows: 1

cols: 5

dt: d

data: [1.7372083577369418e-01, 1.0634240837267861e-01,

-6.1151331073646380e-03, 9.8108654889859526e-03,

-4.0759130873502478e+00]

PythonScriptγιασειριακήεπικοινωνίαμεArduino, κίνησητωντροχώνκαιτουβηματικούμηχανισμούκαιλήψηεικόνων

```
frompynput import keyboard

from time import sleep

import serial

frompicamera import PiCamera


classCar_script:

sleep(2)

count = 0

camera = PiCamera()


if __name__ == '__main__':

ser = serial.Serial('/dev/ttyUSB0',115200, timeout=1) #9600

ser.flush()


defon_press(key):

if key == keyboard.Key.esc:

print('STOP')

ser.write(b'escape\n')

try:

        k = key.char  # single-char keys

except:

        k = key.name  # other keys

if k in ['w', 'a', 's', 'd', 'r']: #, 'q']:  # Movement keys
```

```
            # self.keys.append(k)  # store it in global-like variable
            #print('Key pressed: ' + k)
if k == 'w':
print('Moving Forward')
ser.write(b'forward\n')

elif k == 's':
print('Moving Backward')
ser.write(b'backward\n')

elif k == 'a':
print('Moving Left')
ser.write(b'left\n')

elif k == 'd':
print('Moving Right')
ser.write(b'right\n')

            # elif k == 'q':
            #     print('Stop')
            #     ser.write(b'stop\n')

elif k == 'r':
global count
camera.start_preview()
```

```
ser.write(b'stop\n')
print('Wait, taking pictures for self-localization')

if (count % 2) == 0:
print('Clockwise Rotation')
fori in range (6):
sleep(2)
camera.capture('/home/pi/Desktop//PHOTOS/image%s.jpg' % i)
ser.write(b'rr\n')
                    # print('even ' , count)
count = count + 1
camera.stop_preview()

else:
print('Counter Clockwise Rotation')
fori in range (6):
sleep(2)
camera.capture('/home/pi/Desktop//PHOTOS/image%s.jpg' % i)
ser.write(b'rl\n')
                    # print('odd ' , count)
count = count + 1
camera.stop_preview()

else:
print('error')
```

```
                # navrw macro giaseira
else:
print('Stop')
ser.write(b'stop\n')



listener = keyboard.Listener(on_press=on_press)#, on_release=on_release)
listener.start()  # start to listen on a separate thread
    #     listener.join()  # remove if main thread is polling self.keys
```

Script για υπολογισμούς και οπτικοποίηση θέσης

```
importnumpy as np

import math

import cv2



class Canvas:
    '''
    This class contains the code for all the drawing and calculations. The main function of this class
is to create an empty image and then drawing where the blue and red markers are placed and finally
calculating the position of the camera and finally drawing it to the empty image in relation to the
blue and red markers
    '''
def __init__(self):
        '''
        This is the constructor function of the class it contains the size of the canvas, the positions of the red
and blue squares and it also contains the size of the squares.

        It should be noted that the canvas size is in pixels while the square position and square size are in
meters, therefore, we need to convert the meters into pixels and vice versa to draw things accurately and also
to measure the camera position.
        '''
```

```
        # This is the size of the canvas where everything will be drawn and visualized
self.canvas_W = 200
self.canvas_H = 400

        # This is the x and y position of the red and blue markers in meters
self.blue_start_x = 0.455
self.blue_start_y = 2

self.red_start_x = 0.36
self.red_start_y = 0

        # 'w' is the size of the markers in meters and since we need the dimensions in pixels to accurately draw everything
        # we multiply 'w' by the pixel width of the canvas, this gives us the equivalent size of the square in pixels. The
        # result is saved in 'box_w_scaled'
self.w = 0.095
self.box_w_scaled = int(self.w * self.canvas_W)

defdraw_everything(self):
        '''
        This function takes the parameters of the squares and scales them to pixels and then draws them to the canvas as
rectangles.

        It should be noted that x values are multiplied by width and y values are multiplied by height. This is because the
```

y values determine the height or determine how high things are and the x values determine the horizontal positoin.

One more thing worth noting is that it is important to keep the values in meters between 0 and 1, this way the value

can be used to scale the height and width accurately. Therefore, whenever there is a value in meters that is above 1

it will be normalized so that it lies in between 0 and 1.

'''

Starting with the req square, the x position of the square in meters (red_start_x) is multiplied by the width of the

canvas to get the scaled x position in pixels.

x = int(self.red_start_x * self.canvas_W)

The y position is calculated the same way, however, it should be noted that the value in meters (red_start_y) is first

divided by 2 and then multiplied by the canvas height. This is done to normalize the value of height so that it lies

between 0 and 1 rather than the current 0 and 2 range. This normlaization is done to make sure that the markers are

drawn in the given canvas space.

y = int(self.red_start_y / 2 * self.canvas_H)

box_h is the height of the rectangle in pixels that will be used to drawn on screen to visualize the square. This is done

purely for the sake of visualizatoin and has no other use.

box_h = 10

```
        # This is where the marker is drawn to the canvas. Note that we are using the scaled width in pixels and the x and y positions

        # are also in pixels. The height of the box is set to 'box_h' to make sure that the recangle is visible. The rectangle is colored

        # red

cv2.rectangle(self.canvas,

                (x, y),

                (x + self.box_w_scaled, y + box_h), (0, 0, 255), -1)


        # This is the same process as described above but this time it is for the blue marker.

        x = int(self.blue_start_x * self.canvas_W)

        # Thebox_w is subtracted from the y position here to make sure that the drawn rectangle stays inside the bounds of canvas. This is

        # done for visualization

        y = int(self.blue_start_y / 2 * self.canvas_H) - box_h


        # The blue square is drawn here.

cv2.rectangle(self.canvas,

                (x, y),

                (x + self.box_w_scaled, y + box_h), (255, 0, 0), -1)


        # A black border is draw here around the whole canvas to make things look neater

cv2.rectangle(self.canvas,

                (0, 0),

                (self.canvas_W, self.canvas_H), (0, 0, 0), 5)
```

defget_canvas(self):

'''

This function creates an empty canvas that is all white and returns it. This function also initializes the variables that will

be used to measure the camera position.

'''

This is the position and the radiis of the red and blue circles. They are initialized here but not yet assigned a value.

self.red_circ_x = None

self.blue_circ_x = None

self.red_circ_y = None

self.blue_circ_y = None

self.red_circ_radius = None

self.blue_circ_radius = None

This is where the canvas is created

self.canvas = np.ones((self.canvas_H, self.canvas_W, 3)).astype("uint8") * 255

self.draw_everything() # The visualization part is completed here, the squares are drawn to the canvas and the canvas is then returned

returnself.canvas

defdraw_actual(self, x, y):

'''

This function simply draws a black circle at the given position (x, y)

'''

cv2.circle(self.canvas, (x, y), 5, (0, 0, 0), -1)

returnself.canvas

defdraw_circle(self, radius, square, rescale=True):

'''

This function takes the radius and a square name and then draws a circle to show the possible position of the camera with respect

to a certain colored square.

'''

if rescale:

Rescale parameter specifies whether the radius should be converted from meters into pixels. If the

radius is in pixels it is not changed but if it is in meters or centimeters then it is converted into pixels.

radius = radius * self.box_w_scaled / (self.w * 100)

if square == "red":

This is where the center of the red square is calculated using the dimensions in meters and then the center point is converted into pixels.

x = int((self.red_start_x + self.w / 2) * self.canvas_W)

y = int(self.red_start_y / 2 * self.canvas_H)

cv2.circle(self.canvas, (x, y), int(radius), (0, 0, 255), 2) # This is where the circle is drawn in the appropriate color

ifself.red_circ_radius is None: # If the position is not yet set, it is set here.

self.red_circ_radius = radius

self.red_circ_x = x

```
self.red_circ_y = y


if square == "blue":
        # This is where the center of the blue square is calculated using the dimensions in meters and then the center point is converted into pixels.
        x = int((self.blue_start_x + self.w / 2) * self.canvas_W)
        y = int(self.blue_start_y / 2 * self.canvas_H)
cv2.circle(self.canvas, (x, y), int(radius), (255, 0, 0), 2) # This is where the circle is drawn in the appropriate color


ifself.blue_circ_radius is None:
self.blue_circ_radius = radius
self.blue_circ_x = x
self.blue_circ_y = y


returnself.canvas


deffind_intersection_pts(self):
    '''
    This function finds intersection points between the red and blue circles, this will give us
    2 possible positions.
    '''
    # This is where the maximum distance between the two squares is calculated. The distance is calculated using the
    # euclidean distance formula. This value will be used to find the intersection points
x_val = (self.red_circ_x - self.blue_circ_x) ** 2
```

```
y_val = (self.red_circ_y - self.blue_circ_y) ** 2

        R = np.sqrt(x_val + y_val)



        # Everything from this point on is used to find the x and y positions of the two interesection points between the red

        # and blue circles. The formula has been taken from: https://math.stackexchange.com/questions/256100/how-can-i-find-the-points-at-which-two-circles-intersect

temp_r_sub = (self.red_circ_radius ** 2 - self.blue_circ_radius ** 2)

temp_r_sum = (self.red_circ_radius ** 2 + self.blue_circ_radius ** 2)



        val1 = 1 / 2 * (self.red_circ_x + self.blue_circ_x)

        val2 = temp_r_sub / (2 * R ** 2) * (self.blue_circ_x - self.red_circ_x)

        val3 = 1 / 2 * np.sqrt((2 * temp_r_sum / R ** 2 - (temp_r_sub ** 2) / R ** 4) - 1) * (

self.blue_circ_y - self.red_circ_y)

x1 = val1 + val2 + val3

x2 = val1 + val2 - val3



        val1 = 1 / 2 * (self.red_circ_y + self.blue_circ_y)

        val2 = temp_r_sub / (2 * R ** 2) * (self.blue_circ_y - self.red_circ_y)

        val3 = 1 / 2 * np.sqrt((2 * temp_r_sum / R ** 2 - (temp_r_sub ** 2) / R ** 4) - 1) * (

self.red_circ_x - self.blue_circ_x)
```

```
y1 = val1 + val2 + val3

y2 = val1 + val2 - val3


# This is where we check whether the circles actually intersect eachother or not. If there it no intersection, then we

# increase the circle radiis by 1 pixel until the circles intersect.

ifmath.isnan(x1) or math.isnan(x2) or math.isnan(y1) or math.isnan(y2):

self.blue_circ_radius += 1 # This is where the radii value is increased

self.red_circ_radius += 1

(x1, y1), (x2, y2) = self.find_intersection_pts() # This is where this function is called again, it is a recursion call.

self.canvas = np.ones((self.canvas_H, self.canvas_W, 3)).astype("uint8") * 255 # When the circles actually intersect, we remove the previous circle positions and draw the new ones that intersect


# These functions have been discussed in detail above.

self.draw_everything()

self.canvas = self.draw_circle(radius=self.blue_circ_radius, square="blue", rescale=False)

self.canvas = self.draw_circle(radius=self.red_circ_radius, square="red", rescale=False)


# This is where the points are returned. The order of the points is such that the first point returned is the point to the left and the 2nd point is to the right of the first point.

if x1 < x2:

return (x1, y1), (x2, y2)

else:

return (x2, y2), (x1, y1)
```

```
defget_pts_in_cm(self, pt1, pt2):
    '''
    This function takes the points in pixels and converts them into pixels.
    '''

    # pt[0] represents x and pt[1] represents y. This is where the position in pixels is first divided
    # thethe canvas size to normalize it and then it is multiplied by 100 for x axis and 200 for y axis
    # to scale the normalized value the scaled value will be in CM. We saw earlier that the y position reached
    # 2 meters which is why we multiply the normalized y position value by 200 to make sure that we get the
    # correct scaled value
    pt1 = (pt1[0] / self.canvas_W * 100, pt1[1] / self.canvas_H * 200)
    pt2 = (pt2[0] / self.canvas_W * 100, pt2[1] / self.canvas_H * 200)

return pt1, pt2

defdraw_possible_camera_pos(self, c, color=(0, 255, 255)):
    '''
    This function takes a single point and draws it on the screen as a circle to indicate the camera position.
    '''
ifnp.isnan(c[0]) or np.isnan(c[1]):
        # If the position is invalid then nothing is drawn and 'False' is returned to indicate that the drawing function
```

```
        # was not successful
returnself.canvas, False


    # If the positions are valid then they are drawn to the screen.
c_int = (int(c[0]), int(c[1]))
cv2.circle(self.canvas, c_int, 10, color, -1)


returnself.canvas, True


defget_angle(self, p1):
    '''
    This function takes a single point and finds the angle between that point and the centers of the red and blue squares.
    These angles are then used to find the actual angle of the camera.
    '''
    # The angle of the point with the red and blue squares is calculated here using simple trigonometric formulas
angle_with_red = np.arctan2(p1[1] - self.red_circ_y, p1[0] - self.red_start_x)
angle_with_blue = np.arctan2(p1[1] - self.blue_circ_y, p1[0] - self.blue_circ_x)
    # The angle the robot had to turn to see the two squares is calcualted here. in other words, given that the robot is at point p1
    # and two lines are drawn, one from p1 to red square center and another to blue square center. Then the angle calculated below will be
    # The angle the two drawn lines make at point p1.
angle = angle_with_blue - angle_with_red
angle = angle * 180 / np.pi # The angle is converted to degrees
if angle < 0: # The angle is corrected to ensure a consistent result.
```

```
angle = 360 + angle
returnangle
```

Pythonscriptγια τη λειτουργία του αλγορίθμου εύρεσης θέσης

```
importos
import cv2
importnumpy as np
from canvas import Canvas

## Euclidean distance
defdist(x1, y1, x2, y2):
returnnp.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

## Load Camera Matrix and Distortion coefficient from camera calibration step
defload_matrices(filepath):
fs = cv2.FileStorage(filepath, cv2.FILE_STORAGE_READ)
camera_matrix = fs.getNode("K").mat()
distortion_coeff = fs.getNode("D").mat()
returncamera_matrix, distortion_coeff

## Denoise and segment images
def segment(img, low, high):
mask = cv2.inRange(img, low, high)
mask = cv2.dilate(mask, None, iterations=1)
mask = cv2.erode(mask, None, iterations=1)
    H, W = mask.shape
    m = cv2.resize(mask, (W // 2, H // 2))
    # cv2.imshow("mask", m)
```

```
return mask


## Find Contours and map coordinates
deffind_box(mask):
globalimg
cnts, hierarchy = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
cnts = sorted(cnts, key=cv2.contourArea, reverse=True)
iflen(cnts) == 0:
return None
cnt = cnts[0]

rect = cv2.minAreaRect(cnt)
box = cv2.boxPoints(rect)
box = np.int0(box)

area = cv2.contourArea(cnt)
x, y, w, h = cv2.boundingRect(cnt)
    w = dist(box[0, 0], box[0, 1], box[1, 0], box[1, 1])
    h = dist(box[1, 0], box[1, 1], box[2, 0], box[2, 1])

if area < 1000 or not (0.8 < w / h < 1.2):
return None

return x, y, w, h
```

```
## Find camera distance from the center of the squares and relative angles
deffind_distance(img, square):
globalbox_h
if square == "red":
img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
mask = segment(img, red_low, red_high)
if square == "blue":
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
mask = segment(img, blue_low, blue_high)

    H, W, _ = img.shape
x_img_c = W // 2
y_img_c = H // 2

fov_y = 2 * np.arctan2(H, (2 * camera_matrix[1, 1]))
fov_x = 2 * np.arctan2(W, (2 * camera_matrix[0, 0]))

box = find_box(mask)

if box is None:
return None, None

x, y, w, h = box
```

```
x_red_c = x + w // 2
y_red_c = y + h // 2
cm_per_pixels = (9.5 / w + 9.5 / h) / 2

opposite_pixels = dist(x_red_c, y_red_c, x_img_c, y_red_c)
opposite_cm = opposite_pixels * cm_per_pixels
theta = opposite_pixels / W * fov_x

hyp_cm = opposite_cm / np.arcsin(theta)

angle_c = (x_red_c - x_img_c) / W * fov_x * 180 / np.pi

returnhyp_cm, angle_c



images_dir_path =
"C:/Users/geoel/PycharmProjects/Test2/Measurements/x=57_y=31_angle=180"

red_low = (158, 118, 0)  # HSV
red_high = (255, 255, 255)

blue_low = (0, 0, 0)  # RGB
blue_high = (59, 255, 255)

angle_at_red = None
```

```
angle_at_blue = None

parsing_successful = False


camera_matrix, distortion_coeff =
load_matrices('C:/Users/geoel/PycharmProjects/Test2/CamMat_DistCoef.yml')

print(camera_matrix)


canvas_maker = Canvas()

canvas = canvas_maker.get_canvas()


all_img_names = os.listdir(images_dir_path)


forimg_name in all_img_names:

img_path = os.path.join(images_dir_path, img_name)

image_at_rotation = int(img_name.replace("image", "").split(".")[0])


img = cv2.imread(img_path)

img = cv2.GaussianBlur(img, (9, 9), 0.9)


dist_from_red, ang_r = find_distance(img, square="red")

dist_from_blue, ang_b = find_distance(img, square="blue")


ifdist_from_red is not None:

ifdist_from_red<= 200:

print("\nRed Square Detected.")
```

```
print(f"Calculated Distance From Red: {dist_from_red}")
ifparsing_successful:
        d = dist(40.75, 0, x_pos, y_pos)
print(f"Actual Distance From Red: {d}")
canvas = canvas_maker.draw_circle(radius=dist_from_red, square="red")
angle_at_red = image_at_rotation + ang_r
print(f"Rotation Angle When Red: {angle_at_red}")

ifdist_from_blue is not None:
ifdist_from_blue<= 200:
print("\nBlue Square Detected.")
print(f"Calculated Distance From Blue: {dist_from_blue}")
ifparsing_successful:
        d = dist(50.25, 200, x_pos, y_pos)
print(f"Actual Distance From Blue: {d}")
canvas = canvas_maker.draw_circle(radius=dist_from_blue, square="blue")
angle_at_blue = image_at_rotation + ang_b
print(f"Rotation Angle When Blue: {angle_at_blue}")

pt1, pt2 = canvas_maker.find_intersection_pts()
canvas, valid_pts = canvas_maker.draw_possible_camera_pos(pt1, color=(255, 255, 0))
canvas, valid_pts = canvas_maker.draw_possible_camera_pos(pt2, color=(255, 255, 0))
ifvalid_pts:
print("\n----------------------")
ifangle_at_blue is not None and angle_at_red is not None:
```

```
angle_diff = angle_at_blue - angle_at_red


ifangle_diff< 0:

angle_diff = 360 - angle_diff


print(f"Angle Diff: {angle_diff}")

robot_calculated_angle = (180 + angle_at_red + angle_at_blue) / 2

print(f"Calculated Initial Angle: {robot_calculated_angle}")


    angle1 = canvas_maker.get_angle(pt1)

    angle2 = canvas_maker.get_angle(pt2)

    pt1_, pt2_ = canvas_maker.get_pts_in_cm(pt1, pt2)

print(f"Possible Position x: {pt1_[0]} cm, y: {pt1_[1]} cm, Angle {angle1}")

print(f"Possible Position x: {pt2_[0]} cm, y: {pt2_[1]} cm, Angle {angle2}")


print("Filtering Points")

print("After filtering the possible position of the camera is:")

ifangle_diff< 180:

print(

f"Final Position x: {round(pt1_[0], 2)} cm, y: {round(pt1_[1], 2)} cm, orientation =
{round(robot_calculated_angle, 2)}° ")

canvas, valid_pts = canvas_maker.draw_possible_camera_pos(pt1, color=(0, 255, 0))

else:

print(

f"Final Position x: {round(pt2_[0], 2)} cm, y: {round(pt2_[1], 2)} cm, orientation =
{round(robot_calculated_angle, 2)}° ")
```

```
canvas, valid_pts = canvas_maker.draw_possible_camera_pos(pt2, color=(0, 255, 0))


cv2.imshow("canvas", canvas)
cv2.waitKey(0)
```