ΑΝΩΤΑΤΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΕΙΡΑΙΑ
ΤΕΧΝΟΛΟΓΙΚΟΥ ΤΟΜΕΑ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΑΥΤΟΜΑΤΙΣΜΟΥ

ΘΕΜΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

" ΕΦΑΡΜΟΓΕΣ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ ΣΤΗ ΡΟΜΠΟΤΙΚΗ "

ΟΝΟΜΑ ΦΟΙΤΗΤΗ: ΔΗΜΗΤΡΙΟΣ ΚΟΪΚΑΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΓΡΗΓΟΡΙΟΣ ΝΙΚΟΛΑΟΥ

ΑΙΓΑΛΕΩ, ΦΕΒΡΟΥΑΡΙΟΣ 2017

[0]Εικόνα εξωφύλλου: http://cdn.phys.org/newman/gfx/news/hires/2015/thedrivetowa.jpg

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο / Η κάτωθι υπογεγραμμένος / η **Δημήτριος Κόικας**

του **Παναγιώτη**, με αριθμό μητρώου **41895**, φοιτητής / τρια του Τμήματος **Μηχανικών Αυτοματισμού Τ.Ε.** του Α.Ε.Ι. Πειραιά Τ.Τ. πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

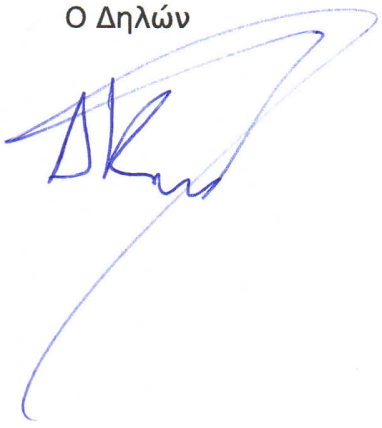
«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

Επίσης δηλώνω υπεύθυνα ότι έχω παρακολουθήσει το σεμινάριο συγγραφής και εκπόνησης πτυχιακής εργασίας που διοργανώνεται από το Τμήμα Μηχανικών Αυτοματισμού Τ.Ε. κατά το Χειμερινό/Εαρινό Εξάμηνο του Ακ. Έτους **2015-2016**

Ο Δηλών



Ημερομηνία

23/01/2017

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω το τμήμα Υπολογιστικής Όρασης του Πανεπιστημίου Koblenz-Ländau, Γερμανία (Computer Visualistics Department, University of Koblenz-Ländau, Germany) για τη παροχή του εξοπλισμού που χρησιμοποιήθηκε στην ανάπτυξη και τη περάτωση της παρούσας μελέτης.

Τους Nicolai Wojke, Frank Neuhaus, Viktor Seib και Raphael Memmesheimer για τις συμβουλές τους περί του εξοπλισμού και την βοήθεια που προσέφεραν σε πολλές περιπτώσεις.

Τον επιβλέποντα καθηγητή, Δρ. Γρηγόριο Νικολάου για την ενθάρρυνση, τη καθοδήγηση, τις συμβουλές και την επίλυση αποριών πάσης φύσεως.

Τέλος, την οικογένεια μου για την υποστήριξη τους κατά την διάρκεια των σπουδών μου και όσων θέλησα να κάνω.

Είχε πλάκα! :-)

ΠΕΡΙΛΗΨΗ

Ύστερα από μελέτη θεμάτων Τεχνητής Νοημοσύνης και Υπολογιστικής Όρασης, αποφασίστηκε η ανάπτυξη μιας εφαρμογής σχετικής με τον εντοπισμό ανθρώπων. Συγκεκριμένα, έχει αναπτυχθεί μία μέθοδος για την αναγνώριση ανθρώπων σε πραγματικό χρόνο. Το σύστημα είναι ικανό να αναγνωρίζει, σε ικανοποιητικό βαθμό, ποια άτομα από αυτά που βρίσκονται στο οπτικό του πεδίο, έχουν ξαναβρεθεί στο παρελθόν.

Η επιλογή των λογισμικών εργαλείων, επηρεάστηκε από τον διαθέσιμο εξοπλισμό του ρομπότ που χρησιμοποιήθηκε.

Συνοπτικά, η διαδικασία που ακολουθήθηκε είναι η παρακάτω:
Αρχικά, μελετήθηκαν υπάρχοντα συστήματα που εξυπηρετούν εφαρμογές εντοπισμού ανθρώπων. Επιλέχθηκε ένα σύστημα που κάνει χρήση δισδιάστατου λέιζερ (2D-laser). Ύστερα, χρησιμοποιήθηκαν τα δεδομένα από την έξοδο του, σε συνδυασμό με κατάλληλη κάμερα RedGreenBlue-Depth (RGB-D). Το λογισμικό που προέκυψε από την παρούσα μελέτη, δέχεται στην είσοδο του τα παραπάνω και δημιουργεί τις συνθήκες για τον τελικό σκοπό.

Τα παραπάνω θα αναλυθούν εκτενώς στα επόμενα κεφάλαια του βιβλίου.

ABSTRACT

After studying Artificial Intelligence and Computer Vision subjects, the development of an application related to people detection was decided. Particularly, a method for people identification in real time has been developed. The system is capable to recognise, in a satisfactory degree, which persons in the current range of view have been seen in the past.

Software tools choice was affected by the available robot equipment.

In short, the process that was followed is the following:

At first, existing people detection systems were studied and one of them, using 2D-laser data, was chosen. Afterwards, these data were used combined with an appropriate RedGreenBlue-Depth (RGB-D) sensor. The software that came of as a result from this study, is using the data described above as input and creates the conditions to reach the target.

The above will be analysed extensively in the following chapters.

ΠΕΡΙΕΧΟΜΕΝΑ

ΛΙΣΤΑ ΕΙΚΟΝΩΝ

ΛΙΣΤΑ ΠΙΝΑΚΩΝ

ΕΙΣΑΓΩΓΗ

Ο τομέας της Τεχνητής Νοημοσύνης έχει γνωρίσει ραγδαία εξέλιξη τα τελευταία χρόνια. Έχουν αναπτυχθεί αλγόριθμοι και τεχνικές που αποδίδουν εξαιρετικά αποτελέσματα σε κάθε λογής προβλήματα. Εφαρμόζεται σε ποικίλλα πεδία, όπως τα μαθηματικά, την ψυχολογία, τη νευροεπιστήμη, την ιατρική κ.ά. Ενδεικτικά προβλήματα που προσεγγίζονται με Τεχνητή Νοημοσύνη είναι προβλήματα λήψης αποφάσεων, σχεδιασμού, μάθησης από σύνολα δεδομένων και αντίληψης ερεθισμάτων από το περιβάλλον.

Τα παραπάνω έχουν βρει ευρεία ανταπόκριση στην Ρομποτική και σε εφαρμογές Υπολογιστικής Όρασης, καθώς αποτελούν προβλήματα αντίληψης. Επίσης, υπάρχουν συνδυαστικά προβλήματα σε αυτούς τους τομείς. Για παράδειγμα, είναι πολύ χρήσιμο για ένα ρομπότ να μπορεί να πλοηγηθεί αυτόνομα σε ένα χώρο χωρίς να συγκρούεται με αντικείμενα ή να προκαλεί φθορές σε ο,τιδήποτε. Λύσεις προσφέρονται και όταν ο στόχος είναι ο χειρισμός αντικειμένων με ένα ρομποτικό βραχίονα.

Όσον αφορά την Υπολογιστική Όραση, βασικός στόχος είναι η λήψη και η ανάλυση οπτικών εικόνων με τέτοιο τρόπο, ώστε το σύστημα να αποκτά γνώση που πλησιάζει αυτή του ανθρώπου. Φυσικά, στο πεδίο αυτό υπάγονται τεχνικές για φαινομενικά πιο απλές διεργασίες, όπως επεξεργασία εικόνας (με κάθε έννοια) και κατασκευή βίντεο. Το πιο διαδεδομένο πρόβλημα, όπου έχει πέσει μεγάλο βάρος της ερευνητικής δραστηριότητας, είναι αυτό της αναγνώρισης (αντικειμένων, προσώπων, ανθρώπων). Όταν λέμε αναγνώριση, εννοούμε τον εντοπισμό, την κατηγοριοποίηση, την ταυτοποίηση και (αν απαιτείται από την εφαρμογή) τον προσανατολισμό.

Σε αυτή τη μελέτη λοιπόν, γίνεται μία μελέτη των παραπάνω, κατασκευάζοντας ένα σύστημα που συνδυάζει τα παραπάνω πεδία.

Κεφάλαιο 1

ΕΞΟΠΛΙΣΜΟΣ

Ο εξοπλισμός που χρησιμοποιήθηκε για την εκπόνηση της παρούσας πτυχιακής εργασίας, ανήκει στο τμήμα Μηχανικής Όρασης (Computer Visualistics Department), που βρίσκεται στο Πανεπιστήμιο του Κόμπλεντς-Λαντάου(Κόμπλεντς) στη Γερμανία. (University of Koblenz-Ländau, Campus Koblenz, Germany)

Στο κεφάλαιο αυτό, παρουσιάζονται τα κύρια στοχεία του εν λόγω εξοπλισμού. Επίσης, γίνεται μία σύντομη αναφορά στην ιστορία του.

1.1 Το ρομπότ Lisa

Το ρομπότ Lisa, είναι μία πλατφόρμα που έχει αναπτυχθεί από φοιτητές του τμήματος, στα πλαίσια του πρακτικού μαθήματος που μπορούν κατ' επιλογή να παρακολουθήσουν. Υπάγεται στην κατηγορία των ρομπότ οικιακού περιβάλλοντος *(domestic robots)*. Είναι ευνόητο λοιπόν, ότι ο σκοπός του είναι η εκτέλεση διαδικασιών χρήσιμων για τον άνθρωπο, μέσα σε ένα οικιακό περιβάλλον.

Για την επίτευξη αυτού του στόχου, επιστρατεύονται αλγοριθμικές τεχνικές για αυτόνομη πλοήγηση *(autonomous navigation)*, εντοπισμός και αναγνώριση αντικειμένων/ανθρώπων *(object/people detection and recognition)* κ.ά.

Όλα τα παραπάνω, δοκιμάζονται σε διαγωνιστικό επίπεδο με διοργανώσεις όπως το Robocup, το RoCKiN και το European Robotics League (ERL), στην αντίστοιχη κατηγορία. Με αυτό το τρόπο, η ομάδα μπορεί να αξιολογήσει καλύτερα την επίδοση του

συστήματος και να παρατηρήσει που χρειάζονται να γίνουν διορθώσεις ή βελτιώσεις.

Τα κύρια εξαρτήματα του ρομπότ είναι τα παρακάτω:

- Δύο (2) δισδιάστατα λέιζερ *(2D-laser)*
- Ρομποτικός βραχίονας
- Pan-tilt unit
- RGB-D camera Kinect v2
- Μικρόφωνο
- Laptop

Όπως ήδη αναφέρθηκε, τα στοιχεία που χρησιμοποιήθηκαν για την εκπόνηση τής παρούσας εργασίας, θα αναλυθούν εκτενέστερα παρακάτω.

Η μορφή του ρομπότ, μέχρι την στιγμή της συγγραφής αυτού του βιβλίου, φαίνεται στην εικόνα 1.1.1

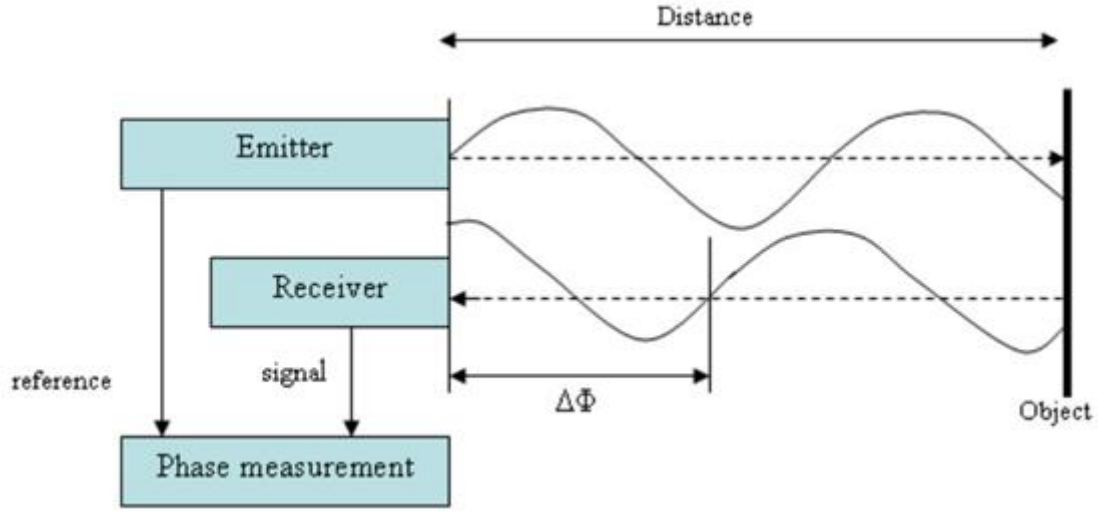
Εικόνα 1.1.1: Το ρομπότ Lisa

1.2 Δισδιάστατο λέιζερ-αποστασιόμετρο *(2D-laser rangefinder)*

Από την εικόνα 1.1.1, παρατηρούμε ότι η Lisa έχει δύο (2) laser. Όπως θα δούμε σε επόμενα κεφάλαια, πρέπει να γίνεται διαρκής εντοπισμός των ποδιών των ανθρώπων. Οπότε, για τις ανάγκες τής εφαρμογής, χρήσιμο ήταν το laser που βρίσκεται στην βάση του ρομπότ. Το συγκεκριμένο λοιπόν, θα παρουσιαστεί σε αυτή την ενότητα, καθώς και ο τρόπος λειτουργίας των laser.

1.2.1 Λειτουργία λέιζερ-αποστασιόμετρου

Τα λέιζερ-αποστασιόμετρα χρησιμοποιούν μία ακτίνα λέιζερ για να προσδιορίσουν την απόσταση από ένα αντικείμενο. Η πιο συνηθισμένη μέθοδος λειτουργίας, στηρίζεται στην Αρχή του Χρόνου Πτήσης *(Time of Flight principle)*. Το αποστασιόμετρο αποστέλλει έναν παλμό-λέιζερ σε στενή ακτίνα προς το αντικείμενο-στόχο. Ύστερα, μετράει τον χρόνο που χρειάζεται για να ανακλαστεί από τον στόχο πίσω στον αποστολέα. (Εικόνα 1.2.1)



Εικόνα 1.2.1: Σχηματική αναπαράσταση λειτουργίας λέιζερ-αποστασιόμετρου[1]

Η απόσταση εκφράζεται από την σχέση 1.6, που προκύπτει από τους παρακάτω υπολογισμούς:

[1]Πηγή: http://www.tankonyvtar.hu/en/tartalom/tamop425/0032_precizios_mezogazdasag/images/new/Theory%20of%20pu
of-Flight%20(TOF)%20%20laser%20scanning.jpg

Για την απόσταση L μεταξύ δύο σημείων, ισχύει η σχέση:

$$L = \frac{ct}{2} \tag{1.1}$$

Όμως: $t = \frac{\varphi}{\omega}$ (1.2) $\omega = 2\pi f$ (1.3) $\lambda = \frac{c}{f}$ (1.4)

όπου:

$L =$ απόσταση μεταξύ πομπού και στόχου

$c =$ ταχύτητα φωτός

$t =$ χρόνος επιστροφής παλμού

$\varphi =$ καθυστέρηση φάσης

$\omega =$ γωνιακή συχνότητα κύματος

$f =$ συχνότητα κύματος

$\lambda =$ μήκος κύματος

Αντικαθιστώντας τις σχέσεις 1.2 και 1.3 στην σχέση 1.1, έχουμε:

$$L = \frac{c\varphi}{2\omega} = \frac{c}{4\pi f}(N\pi + \Delta\varphi) \tag{1.5}$$

όπου:

$N =$ ακέραιος αριθμός, πλήθος των ημικύκλων του κύματος κατά την απόσταση που διένυσε

$\Delta\varphi =$ Διαφορά φάσης

Τέλος, αντικαθιστούμε την σχέση 1.4 στην σχέση 1.5 και έχουμε:

$$L = \frac{\lambda}{4}(N + \Delta N) \tag{1.6}$$

Σημαντικοί παράγοντες που επηρεάζουν την απόδοση και την λειτουργία του laser, είναι τα χαρακτηριστικά του παλμού και η εμβέλεια της ακτίνας.

1.2.2 Παλμός λέιζερ

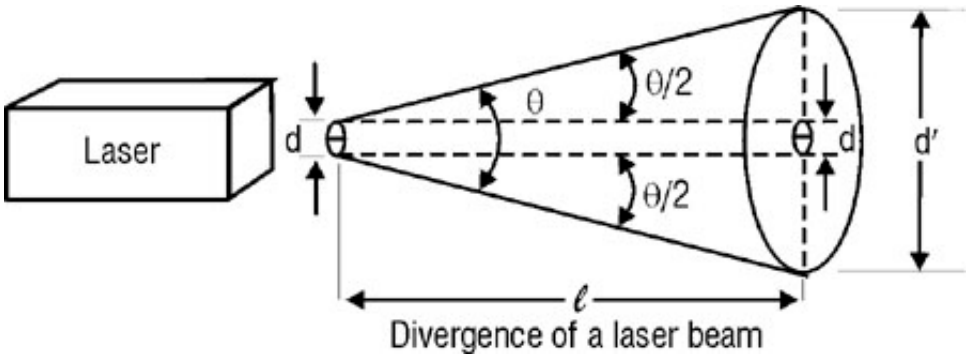
Γνωρίζοντας την συχνότητα του παλμού, μπορεί να γίνει χρήση του φαινόμενου Ντόπλερ *(Doppler effect)* ώστε να υπολογιστεί αν το αντικείμενο κινείται προς το laser ή αντίθετα, καθώς και με ποια ταχύτητα.

Επιπλέον, ο χρόνος ανύψωσης/πτώσης του παλμού επηρεάζει την ακρίβεια του laser. Ο μικρός χρόνος ανύψωσης/πτώσης, σε συνδυασμό με έναν γρήγορο δέκτη, συντελεί στην δυνατότητα εντοπισμού αντικειμένων που βρίσκονται σε απόσταση χιλιοστών.

1.2.3 Εμβέλεια ακτίνας

Όπως προαναφέρθηκε, η ακτίνα που εκπέμπει το laser είναι στενή. Ωστόσο, λόγω του φαινομένου της απόκλισης δέσμης *(beam divergence)*, η διάμετρος της ακτίνας μπορεί να αυξηθεί. Κατά συνέπεια, μεγαλώνει και το πεδίο όρασης της ακτίνας. Το φαινόμενο αυτό ενισχύεται από τον σπινθηρισμό που συμβαίνει όταν η ακτίνα διαπεράσει φυσαλίδες αέρα που υπάρχουν τριγύρω. Δηλαδή, οι φυσαλίδες λειτουργούν ως φακοί, που το μέγεθός τους κυμαίνεται από μικροσκοπικοί έως το μισό της απόστασης του laser από το έδαφος.

Σε εφαρμογές οικιακού περιβάλλοντος και γενικά κλειστών χώρων, αυτό δεν αποτελεί πρόβλημα. Τουναντίον, είναι πλεονέκτημα, καθώς μπορεί να γίνει χαρτογράφηση ταχύτερα.



Εικόνα 1.2.2: Απεικόνιση του φαινομένου απόκλισης δέσμης[2]

1.2.4 SICK LMS100 Laser Rangefinder

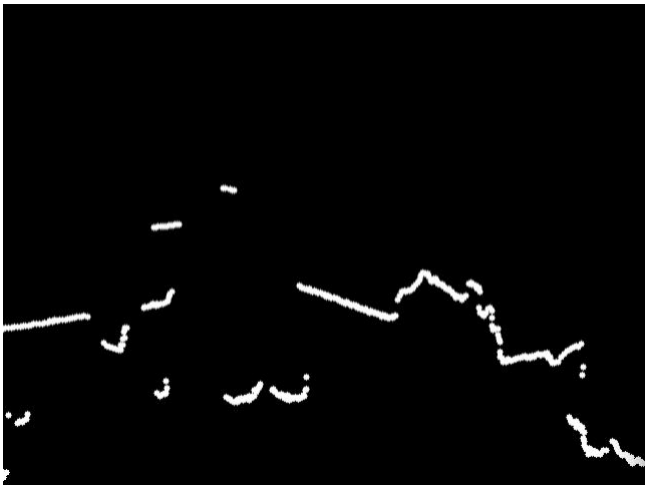
Στην συγκεκριμένη εφαρμογή, χρησιμοποιήθηκε το SICK LMS100 Laser Rangefinder (εικόνα 1.2.3). Το συγκεκριμένο μοντέλο κρίθηκε κατάλληλο, καθώς προσφέρει υψηλή απόδοση σε εφαρμογές όπου η ακρίβεια σε μεγάλες αποστάσεις και ταχύτητες, δεν αποτελεί ζητούμενο. Μερικά χαρακτηριστικά του είναι:

[2]Πηγή: http://i.stack.imgur.com/BVsdb.jpg

- Εμβέλεια: 0,5 έως 20 μέτρα
- Γωνία ανοίγματος: 270 μοίρες
- Συχνότητα σάρωσης(παλμού): 25 έως 50 Hertz
- Διαστάσεις (ΠxYxM): 102 mm x 152 mm x 105 mm

Εικόνα 1.2.3: Το λέιζερ-αποστασιόμετρο SICK LMS100[3]

Στην εικόνα 1.2.4, φαίνεται η έξοδος του laser, κατά την σάρωση ενός χώρου. Τα λευκά σημεία είναι εμπόδια στα οποία έγινε ανάκλαση της ακτίνας.



Εικόνα 1.2.4: Αποτέλεσμα σάρωσης χώρου με λέιζερ-αποστασιόμετρο

[3]Πηγή: http://www.hizook.com/files/users/3/SICK_LMS100.jpg

1.3 Κάμερα RGB

Στην ενότητα αυτή, επεξηγείται ο βασικός τρόπος λειτουργίας του μοντέλου χρωμάτων RGB, το οποίο είναι και το επικρατέστερο στις εφαρμογές μηχανικής όρασης. Επιπλέον, παρουσιάζεται η κάμερα που επιλέχθηκε.

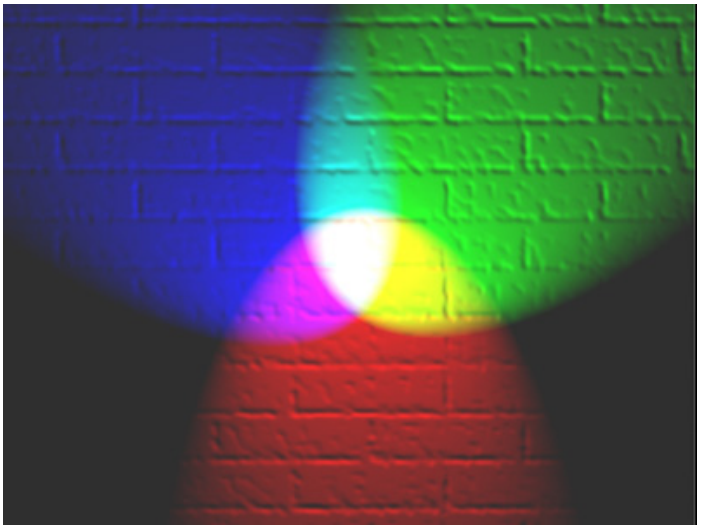
1.3.1 Μοντέλο χρωμάτων RGB

Το μοντέλο χρωμάτων RGB, χρησιμοποιεί τα χρώματα Κόκκινο, Πράσινο, Μπλε *(**R**ed, **G**reen, **B**lue)* ως πρωτεύοντα, απ' όπου προκύπτει και η ονομασία του. Αυτό σημαίνει ότι κάθε χρώμα ή απόχρωση, εκφράζεται με ένα συνδυασμό των παραπάνω χρωμάτων. Το μοντέλο χαρακτηρίζεται ως *προσθετικό*, επειδή οι τιμές κάθε πρωτεύοντος χρώματος προστίθενται για την δημιουργία του παράγωγου χρώματος (Εικόνα 1.3.1).

Όταν το μοντέλο ενσωματώνεται στους υπολογιστές, συνηθίζεται να υλοποιείται με 8-bit αριθμούς για κάθε χρώμα. Δηλαδή, η εκάστοτε τιμή του πρωτεύοντος χρώματος κυμαίνεται από το 0 έως το 255, ως ακέραιος αριθμός. Κάθε παράγωγο χρώμα, εκφράζεται με τις τιμές των επιμέρους χρωμάτων σε παρένθεση, χωρισμένες με υποδιαστολή.

Μηδενίζοντας όλες τις τιμές, δημιουργείται το μαύρο χρώμα και μεγιστοποιώντας τες, το λευκό. Προφανώς, για την επίτευξη πρωτεύοντων χρωμάτων, απλά μηδενίζουμε τις υπόλοιπες τιμές. Ενδεικτικά, παρατίθενται μερικοί χρωματικοί συνδυασμοί:

- Βαθύ μπλε: (0,0,139)
- Καφέ: (165,42,42)
- Κίτρινο: (255,255,0)
- Ασημί: (128,128,128)
- Πορτοκαλί: (255,165,0)



Εικόνα 1.3.1: Προσθετική μίξη χρωμάτων RGB με χρήση προβολέων[4]

1.3.2 RGB-D κάμερα Kinect v2

Ο αισθητήρας Kinect v2 (εικόνα 1.3.2) κυκλοφόρησε στην αγορά ως εξάρτημα της παιχνιδομηχανής Xbox One και διαδέχτηκε το Kinect για το Xbox 360. Λόγω των χαρακτηριστικών και των δυνατοτήτων του, γνώρισε (και ακόμα γνωρίζει) μεγάλη απήχηση στις εφαρμογές και στην έρευνα Ρομποτικής και Μηχανικής Όρασης.

Για την εκτέλεση εφαρμογών όρασης, επιλέχθηκε η συγκεκριμένη κάμερα διότι προσφέρει ευρύ πεδίο όρασης και εικόνες υψηλής ανάλυσης. Επιπλέον, παρέχει αισθητήρα βάθους, καθιστώντας το πιο ελκυστικό, αφού μπορεί κανείς να εκτελέσει πολλαπλές διεργασίες με μία συσκευή. Στην παρούσα μελέτη, δεν έγινε χρήση εικόνων βάθους, οπότε δεν γίνεται αναφορά σε αυτές και στην αντίστοιχη τεχνολογία. Ωστόσο, μένει ανοιχτό το ενδεχόμενο χρήσης εικόνων βάθους σε μελλοντική εργασία.

Οι (υπο)αισθητήρες που μας ενδιαφέρουν για την όραση, είναι οι παρακάτω:

- RGB κάμερα
- Τρισδιάστατος αισθητήρας βάθους

[4]Πηγή: https://en.wikipedia.org/wiki/RGB_color_model#/media/File:RGB_illumination.jpg

RGB κάμερα

Η κάμερα του Kinect v2 έχει το πλεονέκτημα της υψηλής ανάλυσης, που είναι χρήσιμη για την ανάπτυξη περίπλοκων εφαρμογών που απαιτούν υψηλή ακρίβεια στις μετρήσεις και τα δεδομένα που δέχονται. Η συγκεκριμένη έχει ανάλυση 1920x1080 pixels και ο ρυθμός προβολής κάδρων *(framerate)* είναι 30fps *(frames per second)*

Τρισδιάστατος αισθητήρας βάθους *(3D Depth Sensor)*

Για την μέτρηση αποστάσεων, χρησιμοποιεί την Αρχή του Χρόνου Πτήσης, όπως τα λέιζερ-αποστασιόμετρα που προαναφέρθηκαν. Συνοπτικά, τα κύρια χαρακτηριστικά τού:

- Ανάλυση: 512x424 pixels
- Ρυθμός προβολής κάδρων: 30 frames per second
- Εμβέλεια: 0,5 έως 4,5 μέτρα

Εικόνα 1.3.2: Η RGB-D κάμερα Kinect v2[5]

[5]Πηγή: http://image.slidesharecdn.com/kinectv2introductionandtutorial-141114042655-conversion-gate01/95/kinect-v2-introduction-and-tutorial-6-638.jpg?cb=1445117505

Κεφάλαιο 2

ΛΟΓΙΣΜΙΚΟ

Σε αυτό το κεφάλαιο γίνεται μια παρουσίαση των γενικών εργαλείων λογισμικού που αποτέλεσαν την βάση στην ανάπτυξη της παρούσας μελέτης. Δίνονται επίσης πληροφορίες όσον αφορά τις απαιτήσεις και την εγκατάσταση τους στον υπολογιστή.

2.1 Robot Operating System (ROS)

Το ROS είναι ένα λειτουργικό σύστημα που έχει ως σκοπό την ανάπτυξη λογισμικού που προσαρμόζεται εύκολα σε διαφορετικά ρομπότ. Η ιδέα είναι να δοθεί μία εργαλειοθήκη στους ερευνητές, ώστε να μην χρειάζεται να καταναλώνουν χρόνο στην δημιουργία προπαρασκευαστικών λειτουργιών μιας πλατφόρμας.

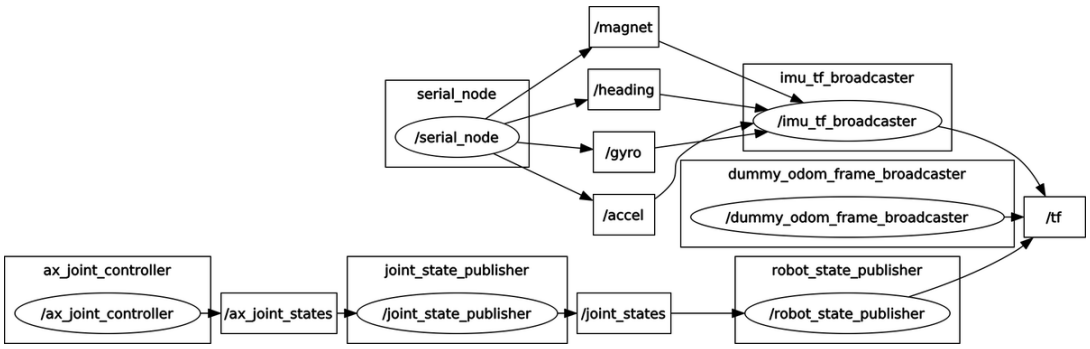
Η ανάπτυξη του ξεκίνησε το 2007 από το Εργαστήριο Τεχνητής Νοημοσύνης του Στάνφορντ (Stanford Artificial Intelligence Laboratory). Από το 2008, την περαιτέρω ανάπτυξη ανέλαβε το Willow Garage, μία ερευνητική ομάδα όπου συνεργάζονταν περισσότερα από 20 διαφορετικά - σχετικά με την ρομποτική - ιδρύματα.

Γνωρίζοντας τεράστια απήχηση, οι κατασκευαστές ρομποτικών πλατφορμών και εξαρτημάτων (αισθητήρες, ελεγκτές κτλ.), ξεκίνησαν να προσαρμόζουν τα προϊόντα τους ώστε να είναι συμβατά με το ROS. Συνεπώς, υπάρχει μία μεγάλη γκάμα επιλογών και οι ερευνητές έχουν την ευχέρεια να αντικαθιστούν πιο εύκολα εξοπλισμό, αν αυτό κριθεί απαραίτητο.

Η υψηλή προσαρμοστικότητα του ROS επιτυγχάνεται με την υιοθέτηση μίας δι-

κτυακής λογικής (εικόνα 2.1.1). Αυτό σημαίνει ότι κάθε κόμβος (node) αναλαμβάνει ένα σύνολο διεργασιών (π.χ: ρύθμιση αισθητήρων, αποστολή/λήψη δεδομένων) που εξυπηρετεί την λειτουργία του συστήματος. Συνήθως, οι κόμβοι είναι αλληλοεξαρτώμενοι. Δηλαδή, ένας τυχαίος κόμβος Α δεν μπορεί να λειτουργήσει σωστά αν δεν λάβει τα δεδομένα από τον κόμβο Β κ.ο.κ. Με αυτό τον τρόπο, εγκαθιδρύεται η επικοινωνία των επιμέρους εξαρτημάτων του ρομπότ και δίνεται η δυνατότητα στοχευμένων αλλαγών, όπου είναι επιθυμητό.

Αντιλαμβάνεται κανείς λοιπόν, ότι το ROS είναι ένα ισχυρό εργαλείο που ενδείκνυται για κάθε είδους ρομποτικών εφαρμογών και επιταχύνει την διαδικασία ανάπτυξης μεγαλεπήβολων έργων.



Εικόνα 2.1.1: Ενδεικτικό διάγραμμα δικτύου ROS[1]

Το ROS είναι πλήρως συμβατό με το λειτουργικό σύστημα Ubuntu Linux. Είναι δυνατή η ενσωμάτωση και σε άλλα λειτουργικά συστήματα (π.χ: Mac OS X, Windows, άλλες διανομές Linux), αλλά όχι στον μέγιστο βαθμό λειτουργικότητας, ο οποίος επίσης διαφέρει ανά σύστημα. Σαφώς λοιπόν, πιο διαδεδομένη είναι η χρήση στο λειτουργικό Ubuntu. Αναπτύχθηκε ταχύτερα από όλες καθώς σε αυτό ξεκίνησε η δημιουργία του ROS και εκεί στοχεύουν πάντοτε κατά την ανάπτυξη. Σημαντικοί λόγοι που επιλέχθηκαν τα Ubuntu, είναι η σταθερότητα, οι δυνατότητες και η υψηλή απόδοση που προσφέρουν, καθώς και το γεγονός ότι το λειτουργικό αυτό διανέμεται δωρεάν.

Με το πέρασμα των χρόνων, κυκλοφόρησαν πολλές εκδόσεις του ROS, με την πιο πρόσφατη (μέχρι τον χρόνο συγγραφής αυτού του βιβλίου) να είναι η "ROS Kinetic Kame", συμβατή με την έκδοση Ubuntu Linux 16.04 . Κατά την εκπόνηση αυτής της

[1]Πηγή: https://dxydas.com/2015/05/31/ros-interconnections/#jp-carousel-1327

μελέτης, χρησιμοποιήθηκε η έκδοση "ROS Indigo Igloo" ή, όπως συνηθίζεται ν' αναφέρεται, ROS Indigo. Επιπλέον, η συγκεκριμένη είναι συμβατή με Ubuntu 14.04 .

Στις επόμενες σελίδες, αποσαφηνίζονται μερικές βασικές έννοιες και εργαλεία του ROS, ώστε να είναι κατανοητή η συνέχεια της παρούσας μελέτης.

2.1.1 Πακέτα (Packages)

Πακέτο είναι το σύνολο των αρχείων που εκτελούν μία επιθυμητή διεργασία. Το σύστημα αρχείων που χρησιμοποιείται για την δημιουργία των πακέτων, ονομάζεται *catkin*. Το catkin είναι ενσωματωμένο στο ROS και εγκαθίσταται αυτόματα με αυτό. Τα πακέτα αποθηκεύονται στον φάκελο *src* (συντομογραφία του *source*) μέσα στο χώρο εργασίας του catkin (*catkin workspace*). Σημαντικό χαρακτηριστικό τους, είναι ότι δεν είναι δυνατή η ύπαρξη εμφολευμένων πακέτων.

Τα απαραίτητα στοιχεία ενός πακέτου είναι:

- Αρχείο *package.xml*

  Περιέχει πληροφορίες που περιγράφουν την φύση του πακέτου

- Αρχείο *CMakeLists.txt*

  Περιγράφει πως να "χτιστεί" το πακέτο και που να εγκατασταθεί

Τα παρακάτω είναι τα στοιχεία όπου αποθηκεύονται τα εκτελέσιμα αρχεία που πραγματοποιούν τις επιθυμητές λειτουργίες

- Φάκελος *src*

  Περιέχει εκτελέσιμα αρχεία κώδικα

- Φάκελος *launch*

  Περιέχει *launchfiles*, τα οποία θα εξηγηθούν παρακάτω.

2.1.2 Κόμβοι (Nodes)

Στην πραγματικότητα, ένας κόμβος είναι απλώς ένα εκτελέσιμο αρχείο κώδικα. Για την επικοινωνία των nodes, υπάρχει βιβλιοθήκη-πελάτης (*client library*) για κάθε γλώσσα προγραμματισμού. Οι γλώσσες με πλήρως ανεπτυγμένες βιβλιοθήκες-πελάτες,

είναι οι: C++, Python, LISP. Πλέον, βρίσκονται σε πειραματικό στάδιο και για άλλες γλώσσες, όπως Java, Ruby, Lua και πολλές ακόμα. Ενδείκνυται λοιπόν η χρήση των 3 πρώτων (συνηθίζεται όμως η χρήση των C++, Python).

Ένα πλεονέκτημα του ROS, είναι η δυνατότητα χρήσης κόμβων γραμμένων σε διαφορετικές γλώσσες, στο ίδιο πακέτο. Αυτό προσφέρει σημαντική ευελιξία στον προγραμματιστή, αφού πολλές φορές τυχαίνει να απαιτείται ένα σύνολο διεργασιών που κάποιο μέρος τους μπορεί να επιτευχθεί πιο εύκολα σε διαφορετικές γλώσσες.

Οι κόμβοι μπορούν να δημιουργήσουν θέματα (*topics*) και να εκδόσουν (*publish*) ή να "γίνουν συνδρομητές" (*subscribe*) σε αυτά. Δηλαδή, μπορούν να αποστείλουν/γράψουν δεδομένα στα θέματα αλλά και να διαβάσουν δεδομένα από αυτά.

Συνεπώς, από σκοπιά εκτέλεσης επιθυμητών λειτουργιών, μπορούμε να πούμε ότι οι κόμβοι, είναι ο εγκέφαλος του συστήματος.

2.1.3 Θέματα (Topics)

Τα θέματα είναι ο δίαυλος επικοινωνίας μεταξύ των κόμβων. Περιέχουν πληροφορίες και δεδομένα που είναι χρήσιμα για την εύρυθμη λειτουργία του κάθε κόμβου. Στην εικόνα 2.1.2, βλέπουμε ένα απλό παράδειγμα.

Εικόνα 2.1.2: Επικοινωνία ROS κόμβων μέσω θέματος[2]

Το παραπάνω είναι αποτέλεσμα του εργαλείου *rqt_graph* που παρέχει το ROS. Οι κόμβοι απεικονίζονται με ελλείψεις και το θέμα με ένα βέλος. Βλέποντας την εικόνα, συμπεραίνουμε ότι ο κόμβος */teleop_turtle* στέλνει δεδομένα στο θέμα */turtle1/command_velocity*. Με τη σειρά του, ο κόμβος */turtlesim* διαβάζει τα δεδομένα από το θέμα.

Σε αυτό το σημείο πρέπει να διευκρινηστεί μία ακόμα φορά, ότι δεν είναι δυνατή η απ' ευθείας αποστολή δεδομένων μεταξύ κόμβων. Πρέπει μέσα στον κώδικα τους, να

[2]http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics?action=AttachFile&do=get&target=rqt_graph_turtle_key.png

γίνεται η αποστολή και η εγγραφή σε ένα ή περισσότερα θέματα.

2.1.4 Εξυπηρετητής Παραμέτρων (Parameter Server)

Ο εξυπηρετητής παραμέτρων είναι ένα λεξικό προσβάσιμο από όλα τα στοιχεία του ROS. Οι κόμβοι τον χρησιμοποιούν για την αποθήκευση και την ανάκτηση τιμών κατά την διάρκεια της λειτουργίας του συστήματος. Επειδή δεν έχει σχεδιαστεί για υψηλές επιδόσεις και ταχύτητες, χρησιμοποιείται για παραμέτρους ρυθμίσεων και διαμόρφωσης. Για παράδειγμα, είναι δυνατό να ορίζεται η απόσταση της κάμερας από το έδαφος κατά την εκκίνηση του συστήματος και ύστερα να έχει πρόσβαση σε αυτή τη τιμή κάθε κόμβος.

Οι παράμετροι συνεχίζουν να υπάρχουν στον εξυπηρετητή, ακόμα και μετά την περάτωση των εργασιών. Είναι εύκολο να γίνει εκκαθάριση αν το επιθυμούμε. Φυσικά, μπορούμε να τις ανακτήσουμε εκτελώντας το πρόγραμμα που τις δημιούργησε ή χειροκίνητα. Τα ονόματα των παραμέτρων προκύπτουν από εκείνα των θεμάτων και των κόμβων απ' όπου ορίστηκαν.

2.1.5 Αρχεία μηνυμάτων - υπηρεσιών (Message - Services files)

Τα αρχεία μηνυμάτων είναι αρχεία κειμένου που περιέχουν την περιγραφή των μερών του μηνύματος που δημιουργείται από τους κόμβους. Ουσιαστικά, δίνει πληροφορίες για τον τύπο των μεταβλητών του μηνύματος (ακέραιος, πραγματικός κτλ.). Είναι εφικτό ένα μέρος του αρχείου μηνύματος, να έχει τύπο ένα άλλο μήνυμα. Η κατάληξη των αρχείων αυτών είναι *.msg* . Όταν θέλουμε να έχουμε αρχεία μηνυμάτων, δημιουργούμε τον φάκελο *msg* στο πακέτο μας και τα συλλέγουμε σε αυτόν.

Τα αρχεία υπηρεσιών είναι και αυτά αρχεία κειμένου, αλλά περιγράφουν τα 2 μέρη της υπηρεσίας: 1) αίτημα και 2) απάντηση. Η κατάληξη τους είναι *.srv* και συλλέγονται στον φάκελο *srv* του πακέτου μας.

Όταν χτίζουμε το πακέτο, τα αρχεία μετατρέπονται σε κώδικα γλωσσών που χρησιμοποιούμε στο πακέτο μας.

2.1.6 Αρχεία bag (bagfiles)

Τα αρχεία *bag* προσφέρουν την δυνατότητα αποθήκευσης των δεδομένων που περνούν από τα θέματα κατά την διάρκεια λειτουργίας του ρομπότ. Ύστερα, ο προγραμματιστής τα αναπαράγει και συνεχίζει την ανάπτυξη του κώδικα αυτόνομα. Είναι δηλαδή ένας τρόπος εξομοίωσης της κατάστασης του συστήματος. Τα αρχεία αυτά, έχουν την κατάληξη *.bag* .

Μπορούμε φυσικά να δούμε πληροφορίες σχετικά με το περιεχόμενο του αρχείου χωρίς να το αναπαράγουμε. Πληροφορίες όπως μέγεθος, διάρκεια, ημερομηνία εγγραφής, τύποι μηνυμάτων που εγγράφηκαν και ποια θέματα περιέχει.

Ωστόσο, δεν μπορούμε να περιμένουμε ακριβώς ίδια συμπεριφορά όταν αναπαράγουμε τα δεδομένα. Πολλές φορές, οι διεργασίες που εκτελούνται επηρεάζονται από τον χρονισμό του συστήματος. Τα αρχεία *bag* όμως, δεν μπορούν να συγχρονιστούν με τον χρόνο έκδοσης των μηνυμάτων και δεδομένων. Για παράδειγμα, τα δεδομένα από μία αυτόνομη πλοήγηση, θα έχουν απόκλιση από την πραγματικότητα. Συνήθως αυτό δεν αποτελεί πρόβλημα, εκτός αν η εφαρμογή απαιτεί υψηλή ακρίβεια. Σε αυτή την περίπτωση, ο προγραμματιστής πρέπει να το λάβει υπ' όψιν τού κατά τα πειράματά του.

2.1.7 Launchfiles

Τα *launchfiles* είναι αρχεία που έχουν την μορφή *XML*. Η δουλειά τους είναι η ταυτόχρονη εκκίνηση πολλαπλών διαδικασιών, εκτελώντας μόνο αυτό. Τέτοιες διαδικασίες συνήθως είναι εκτέλεση κόμβων, παραμετροποίηση και χρήση εσωτερικών εργαλείων του ROS. Επιπλέον, ένα *launchfile* μπορεί να εκτελεί πολλαπλά *launchfiles*. Για τον προγραμματισμό τους, υπάρχει μία ξεχωριστή κατηγορία εντολών.

Η κατάληξή τους είναι, κατά σύμβαση, *.launch* . Μπορούν να αποθηκευθούν οπουδήποτε μέσα στο πακέτο μας, αλλά χάριν ευκολίας, γίνεται ακόμα μία σύμβαση και τα συλλέγουμε στον φάκελο *launch*.

Στο κεφάλαιο 6 φαίνονται αναλυτικά τα περιεχόμενα του πακέτου που παράχθηκε ως αποτέλεσμα αυτής της εργασίας.

2.1.8 Εσωτερικά εργαλεία και εντολές του ROS

Το ROS παρέχει μία σειρά εργαλείων για χειρισμό των στοιχείων που το αποτελούν. Εκτελούνται μέσω εντολών που εισάγονται σε ένα τερματικό (*terminal*). Ακολουθεί ένας πίνακας με τα εν λόγω εργαλεία:

Εργαλείο	**Λειτουργία**
rosbag	Διεργασίες σχετικές με αρχεία .bag .
rosbash	Όνομα ενός συνόλου εντολών και λειτουργιών.
roscd	Αλλαγή φακέλου στο σύνολο του ROS. Μέλος του rosbash.
rosclean	Διαγραφή αρχείων που δημιουργήθηκαν από το ROS.
roscore	Ενεργοποιεί απαραίτητες διεργασίες για την λειτουργία του ROS.
rosdep	Εγκατάσταση εξαρτήσεων συστήματος για το πακέτο.
rosed	Επιτρέπει την επεξεργασία αρχείων, χρησιμοποιώντας το όνομα του πακέτου όπου εμπεριέχεται. Μέλος του roshbash.
roscreate-pkg	Δημιουργία αρχείων απαραίτητων για την λειτουργία πακέτων ROS.
roscreate-stack	Δημιουργία αρχείων απαραίτητων για την λειτουργία στοιβών ROS.
rosrun	Τρέχει εκτελέσιμα αρχεία, χωρίς να είναι απαραίτητη η πλοήγηση στην θέση του.
roslaunch	Εκτελεί launchfile, χωρίς να είναι απαραίτητη η πλοήγηση στη θέση του.
roslocate	Εντοπίζει την θέση του πακέτου. Αφορά github, svn κτλ.
rosmake	Βεβαιώνει ότι χτίζονται οι εξαρτήσεις του πακέτου, πριν χτιστεί αυτό.
rosmsg	Εμφανίζει την δομή των μηνυμάτων.
rosnode	Εμφανίζει πληροφορίες κόμβων.

rospack	Όνομα ενός συνόλου εντολών και λειτουργιών, σχετικά με το αρχειακό σύστημα των πακέτων ROS.
rosparam	Επιτρέπει την λήψη και το σετάρισμα τιμών στον εξυπηρετητή παραμέτρων.
rossrv	Εμφανίζει την δομή των υπηρεσιών.
rosservice	Εμφανίζει πληροφορίες για τις υπηρεσίες και για τα μηνύματα που στέλνονται στα θέματα.
rosstack	Όνομα ενός συνόλου εντολών και λειτουργιών, σχετικά με το αρχειακό σύστημα των πακέτων ROS.
rostopic	Εμφανίζει πληροφορίες για τα θέματα και τα μηνύματα που στέλνονται στα θέματα.

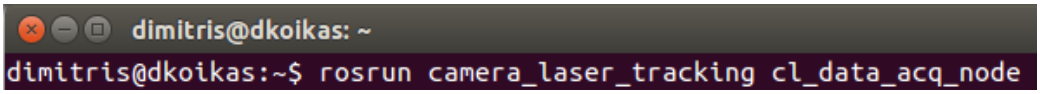
Πίνακας 2.1: Εργαλεία ROS εκτελέσιμα από το τερματικό

Τα πιο συνηθισμένα εργαλεία είναι τα rosrun και roslaunch, καθώς εκκινούν εκτελέσιμα αρχεία. Η σύνταξη των εντολών είναι:

όνομα_εργαλείου όνομα_πακέτου όνομα_αρχείου

π.χ:

- **rosrun camera_laser_tracking cl_data_acq_node**
- **roslaunch camera_laser_tracking project_track_offline.launch**



Εικόνα 2.1.3: Παράδειγμα εκκίνησης κόμβου μέσω τερματικού

2.2 Η βιβλιοθήκη OpenCV

Το όνομα της βιβλιοθήκης είναι ακρωνύμιο του *Open source Computer Vision*. Πρόκειται για ένα ανοιχτό project που βρίσκει ανταπόκριση σε ακαδημαϊκές και εμπορικές εφαρμογές. Λειτουργεί με τις γλώσσες προγραμματισμού C++, C, Python, Java και στα λειτουργικά συστήματα Windows, Linux, Mac OS, iOS και Android. Υποστηρίζεται από μία κοινότητα χρηστών και προγραμματιστών που συνεχώς μεγαλώνει. Αυτό το γεγονός προσφέρει μεγάλη ευχέρεια στην ανάπτυξη και αποσφαλμάτωση του κώδικα. Η έκδοση που χρησιμοποιήθηκε στην διάρκεια αυτής της μελέτης είναι η 3.1.0 .

Η δομή τής αποτελείται από μικρότερες βιβλιοθήκες, όπου η κάθε μία εξυπηρετεί ξεχωριστές λειτουργίες. Αυτές κυμαίνονται από σχετικά απλές διαδικασίες όπως επεξεργασία εικόνων/βίντεο έως και εφαρμογή προηγμένων αλγορίθμων π.χ: νευρωνικά δίκτυα.

Στις επόμενες ενότητες, περιγράφονται τα χαρακτηριστικά των υποβιβλιοθηκών που χρησιμοποιήθηκαν.

2.2.1 Η βιβλιοθήκη core

Περιέχει μία σειρά βασικών λειτουργιών και δομών που είναι προσβάσιμες και γνωστές σε όλες τις υπόλοιπες. Επιτρέπει την δημιουργία αντικειμένων Mat, που είναι ένας πολυδιάστατος πίνακας και χρησιμοποιείται ως η ψηφιακή περιγραφή μίας εικόνας. Συμπεριλαμβάνονται επίσης συναρτήσεις που επιτρέπουν την επεξεργασία εικόνων με πολλούς τρόπους. Ενδεικτικά αναφέρονται η αλλαγή μεμονωμένων εικονοκυττάρων *(pixels)*, η αλλαγή φωτεινότητας/αντίθεσης, η ζωγραφική πάνω στην εικόνα, οι μαθηματικές πράξεις μεταξύ εικόνων.

2.2.2 Η βιβλιοθήκη imgproc

Αυτή η βιβλιοθήκη περιέχει συναρτήσεις για πιο προχωρημένες διεργασίες. Τέτοιες διεργασίες είναι η εφαρμογή φίλτρων, οι μορφολογικές μετατροπές των αντικείμενων, η δημιουργία φίλτρων μέσω υπάρχουσων συναρτήσεων, ο εντοπισμός σχημάτων.

2.2.3 Η βιβλιοθήκη imgcodecs

Μία απλή βιβλιοθήκη που παρέχει τις συναρτήσεις για την αποθήκευση και την ανάγνωση εικόνων στον/από δίσκο.

2.2.4 Η βιβλιοθήκη highgui

Επιτρέπει την δημιουργία γραφικών διεπαφών χρήστη (*GUI - Graphical User Interface*) και τον χειρισμό αρχείων βίντεο. Επιπλέον, είναι ένας από τους τρόπους επικοινωνίας του Kinect με την OpenCV.

Κεφάλαιο 3

ΔΙΑΜΕΤΡΗΣΗ ΚΑΜΕΡΑΣ

Η διαμέτρηση των αισθητήρων και των εξαρτημάτων που υπάρχουν στο σύστημα μας, είναι μία διαδικασία απαραίτητη για την σωστή λήψη δεδομένων από το περιβάλλον, ώστε η εφαρμογή που αναπτύσσεται να είναι ακριβής και να ανταποκρίνεται στην πραγματικότητα. Στο παρόν κεφάλαιο επεξηγείται η σημασία και ο τρόπος επίτευξης της διαμέτρησης σε μία κάμερα *(camera calibration)*.

Ένα μειονέκτημα των διαδεδομένων συσκευών στις μέρες μας, είναι η παραμόρφωση που παρουσιάζεται στην έξοδο της κάμερας. Όταν η εικόνα υφίσταται παραμόρφωση, είναι εύκολο να παρατηρήσει κανείς το φαινόμενο. Οι γραμμές που στην πραγματικότητα είναι ευθείες, παρουσιάζονται καμπυλωτές στην έξοδο. (Εικόνα 3.1.2)

3.1 Παραμόρφωση εικόνας

Η παραμόρφωση εξαρτάται από δύο παράγοντες: 1) την ακτινική παραμόρφωση και 2) την εφαπτομενική παραμόρφωση *(radial and tangential distortion)*. Εξαλείφοντας αυτές, η παραμόρφωση διορθώνεται και η διαμέτρηση σχεδόν ολοκληρώνεται.

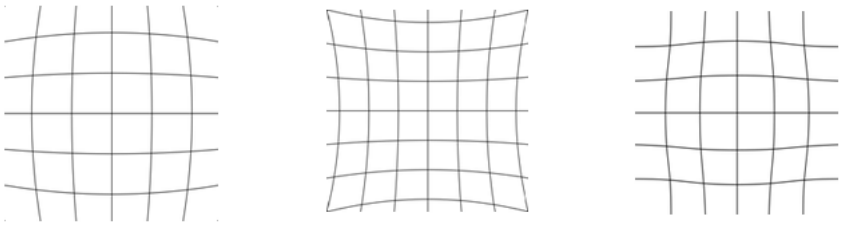
3.1.1 Ακτινική παραμόρφωση

Τα είδη της ακτινικής παραμόρφωσης είναι τα εξής:

- Παραμόρφωση - βαρέλι *(barrel distortion)*
- Παραμόρφωση - μαξιλαράκι για καρφίτσες *(pincushion distortion*)

- Παραμόρφωση - μουστάκι *(mustache distortion)*

Στην εικόνα 3.1.1 φαίνονται κατά σειρά οι αντίστοιχες παραμορφώσεις εφαρμοσμένες σε ένα πλέγμα.



Εικόνα 3.1.1: Είδη ακτινικής παραμόρφωσης[1]

Λαμβάνοντας την εικόνα, έστω ότι παρατηρούμε ένα σημείο με συντεταγμένες (x, y). Οι διορθωμένες, ως προς την ακτινική παραμόρφωση, συντεταγμένες (x_c, y_c) υπολογίζονται από τις σχέσεις:

$$x_c = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.1}$$

$$y_c = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.2}$$

με:

$$r^2 = x^2 + y^2 \tag{3.3}$$

3.1.2 Εφαπτομενική παραμόρφωση

Κατ' επέκταση, οι διορθωμένες, ως προς την εφαπτομενική παραμόρφωση, συντεταγμένες (x_c, y_c) υπολογίζονται από τις σχέσεις:

$$x_c = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{3.4}$$

$$y_c = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{3.5}$$

με:

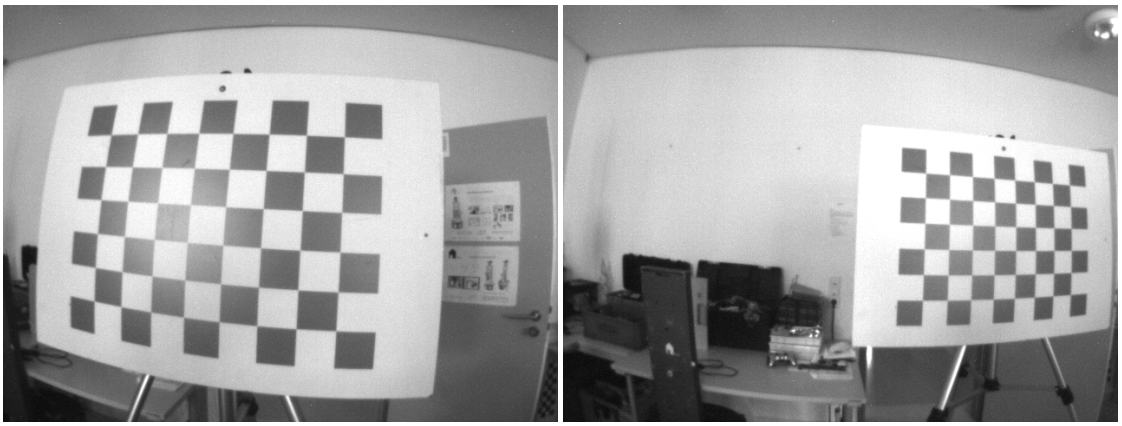
$$r^2 = x^2 + y^2 \tag{3.6}$$

[1]Πηγή: Advanced Computer Vision CH6. Feature-based Alignment Professor: Prof. Fuh Presenter: Nick Chu, page 7, cv2.csie.ntu.edu.tw/cv/_private/CV2_CH6_2015.pptx

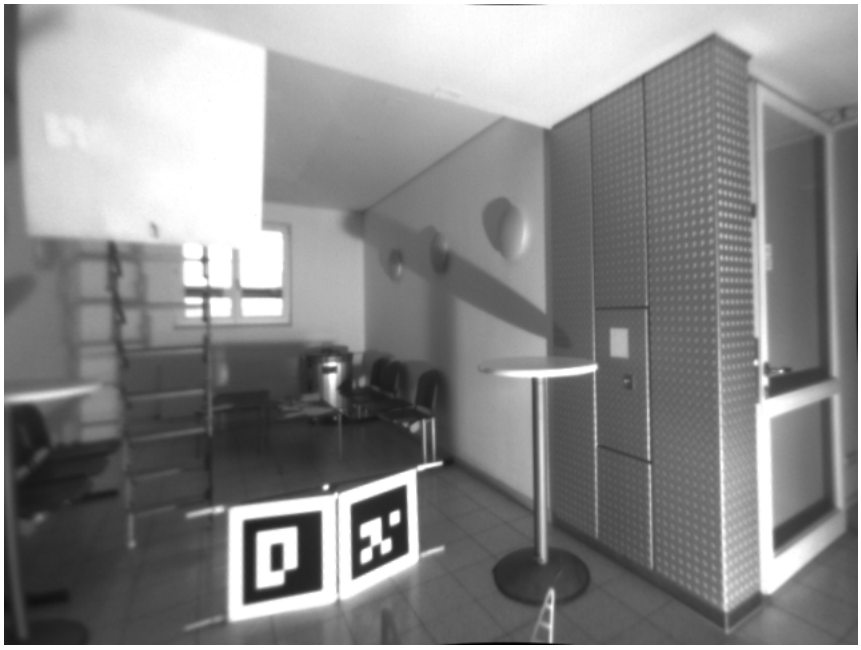
3.1.3 Συντελεστές Παραμόρφωσης

Οι τιμές των k_1, k_2, k_3, p_1, p_2 αποτελούν τους συντελεστές παραμόρφωσης *(distortion coefficients)* της κάμερας. Εκφράζονται ως ένας πίνακας $1x5$ και ο στόχος μας είναι ο υπολογισμός τους.

$$\text{Συντελεστές Παραμόρφωσης} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{3.7}$$



Εικόνα 3.1.2: Παραδείγματα εξόδου κάμερας με παραμόρφωση



Εικόνα 3.1.3: Παράδειγμα εξόδου κάμερας χωρίς παραμόρφωση

3.2 Εσωτερικές παράμετροι κάμερας

Πέρα από τον υπολογισμό των συντελεστών παραμόρφωσης, πρέπει να υπολογιστούν και οι εσωτερικές παράμετροι της κάμερας *(camera intrinsic parameters)*. Με αυτό το τρόπο ολοκληρώνεται η διαμέτρηση της κάμερας αυτής καθ' αυτής.

3.2.1 Πίνακας κάμερας

Ο πίνακας κάμερας *(camera matrix)* περιγράφει κάποια χρήσιμα χαρακτηριστικά της. Αυτό σημαίνει ότι κάθε κάμερα έχει τον δικό τής πίνακα και οι τιμές του εξαρτώνται από τα εξαρτήματα του υλικού της. Ο πίνακας έχει μέγεθος $3x3$ και η μορφή του είναι η εξής:

$$\text{Πίνακας κάμερας} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

Οι τιμές f_x, f_y εκφράζουν το εστιακό μήκος της κάμερας, ενώ οι συντεταγμένες (c_x, c_y) εκφράζουν το σημείο εστιάσης τής. Φυσικά, μόλις υπολογιστούν όλες οι παραπάνω παράμετροι, μπορούν να αποθηκευθούν σε ένα αρχείο για μελλοντική χρήση της κάμερας. Για να τεθούν σε ισχύ, πρέπει να διαβάζονται από το πρόγραμμα οδήγησης της κάμερας, κατά την εκκίνησή τής.

Σε αυτό το σημείο, πρέπει να σημειωθεί ότι ο πίνακας της κάμερας έχει στην πραγματικότητα διαστάσεις $3x4$. Η τέταρτη (4η) στήλη λαμβάνεται υπ' όψιν όταν επιθυμούμε την μετατροπή σημείων από τον τρισδιάστατο χώρο (3D) στον δισδιάστατο (2D). Δηλαδή, όταν θέλουμε να διαμετρήσουμε μία κάμερα που δέχεται και δεδομένα βάθους.

3.2.2 Εστιακό μήκος και σημείο εστίασης

Το εστιακό μήκος *(focal length)* είναι η οριζόντια απόσταση μεταξύ του φακού και του σημείου εστίασης. Η απόσταση αυτή μετριέται σε pixels και ύστερα μετατρέπεται σε μονάδες του πραγματικού κόσμου (έστω F_x) μέσω της σχέσης:

$$F_x = f_x \frac{W}{w} \tag{3.9}$$

όπου:

f_x = απόσταση σημείου εστίασης σε pixels

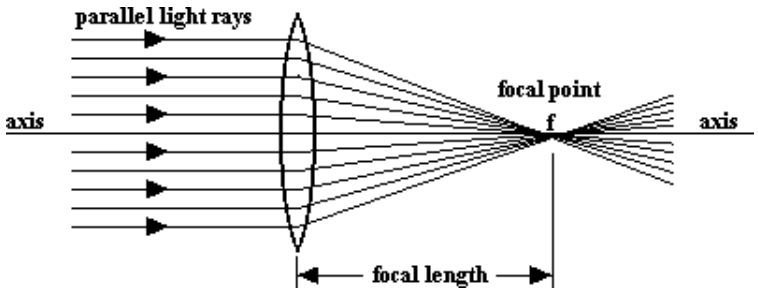
W = πλάτος αισθήτηρα κάμερας σε χιλιοστά *(mm)*

w = πλάτος εικόνας σε pixels

Από τον πίνακα παρατηρούμε ότι υπάρχουν δύο (2) εστιακά μήκη, αλλά λαμβάνουμε υπ' όψιν μας μόνο ένα (1), επειδή πρακτικά δεν είναι όλα χρήσιμα.

Το εστιακό μήκος μας βοηθάει να αποκτήσουμε διαίσθηση για το "πόσο δυνατά" συγκλίνουν ή αποκλίνουν οι ακτίνες φωτός, αφού περάσουν από τον φακό. Για παράδειγμα, ένας φακός που έχει μικρό εστιακό μήκος, έχει και μεγάλη οπτική δύναμη. Αυτό σημαίνει ότι έχει την ιδιότητα να συγκλίνει το φως προς το σημείο εστίασης με απότομο τρόπο. Για κάθε εφαρμογή, επιλέγονται και οι φακοί με το κατάλληλο σημείο εστίασης.

Το σημείο εστίασης *(focal point)* ορίζεται ως το σημείο όπου εστιάζουν οι ακτίνες φωτός που ήταν παράλληλες πριν περάσουν από τον φακό.



Εικόνα 3.2.1: Αναπαράσταση σύγκλισης ακτίνων φωτός προς το σημείο εστίασης[2]

3.3 Διαδικασία διαμέτρησης

Σε αυτή την ενότητα, περιγράφονται τα βήματα που πρέπει να γίνουν για να εκτελεστεί η διαμέτρηση. Εξηγούνται κάποιες ιδιαιτερότητες που αφορούν την φύση των εργαλείων και συμπεριλαμβάνονται μερικές οδηγίες για την χρήση τους.

[2]Πηγή: http://www.cbakken.net/obookshelf/image073.gif

3.3.1 Λήψη και αξιοποίηση δεδομένων

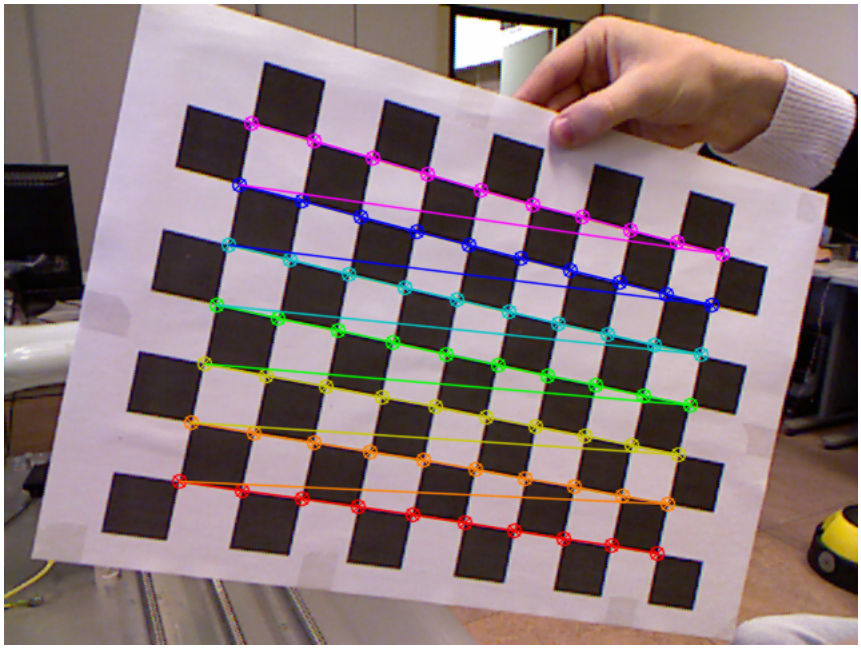
Η διαδικασία της διαμέτρησης απαιτεί ένα σύνολο δεδομένων που περιέχει φωτογραφίες ενός μοτίβου. Το πιο διαδεδομένο είναι ένας πίνακας σαν σκακιέρα. Αποτελείται από λευκά και μαύρα τετράγωνα αλλά δεν είναι απαραίτητα $8x8$. Συνηθίζεται να έχουν αυτά τα χρώματα, ώστε η αντίθεση να είναι μεγάλη και να γίνεται πιο εύκολος ο εντοπισμός τους. Στις εικόνες 3.1.2 και 3.3.1 φαίνονται τέτοια μοτίβα.

Υπάρχουν λοιπόν ποικίλλα προγράμματα που, αφού εισάγουμε τις διαστάσεις του μοτίβου, υπολογίζουν όλες τις εσωτερικές παραμέτρους της κάμερας. Όταν αναφερόμαστε στις διαστάσεις του μοτίβου, εννοούμε *(αριθμός κάθετων τετραγώνων) - 1*, *(αριθμός οριζόντιων τετραγώνων) - 1* και *μήκος πλευράς τετραγώνων*. Παρακάτω θα γίνει αντιληπτό γιατί αφαιρούμε ένα (1) τετράγωνο από κάθε πλευρά.

Ο πιο σωστός τρόπος συλλογής των δεδομένων είναι: Η κάμερα μένει σταθερή και κινούμε το μοτίβο σε διάφορες θέσεις. Προσπαθούμε να συμπεριλάβουμε πολλές διαφορετικές οπτικές γωνίες κατά την συλλογή των εικόνων (ακραίες γωνίες, κοντινή/μεγάλη απόσταση κτλ.). Θεωρητικά, το μέγεθος του συνόλου πρέπει να είναι μεγαλύτερο του δέκα (>10). Επειδή αυτό είναι πολύ αόριστο, έγιναν μερικά σύντομα πειράματα κατά την εκπόνηση αυτής της μελέτης. Η διαμέτρηση φαίνεται να είναι πιο αποτελεσματική όταν το σύνολο περιείχε από εικοσιπέντε (25) έως τριάντα (30) φωτογραφίες. Εξασφαλίζεται έτσι ότι υπάρχουν αρκετές καλές εικόνες για την διαμέτρηση. Φυσικά, αυτό δεν είναι δεσμευτικό για όλες τις περιπτώσεις.

Τα βήματα που εκτελούν τα προγράμματα αυτά είναι:

- Εντοπισμός του εσωτερικού μοτίβου (Εικόνα 3.3.1)
  Αυτό εξηγεί την αφαίρεση ενός τετραγώνου κατά την είσοδο
- Υπολογισμός συντελεστών παραμόρφωσης
- Υπολογισμός εσωτερικών παραμέτρων
- Υπολογισμός σφάλματος επαναπροβολής *(reprojection error)*
- Αποθήκευση των παραμέτρων σε αρχείο XML ή YAML



Εικόνα 3.3.1: Παράδειγμα εντοπισμού εσωτερικού μοτίβου[3]

3.3.2 Σφάλμα επαναπροβολής

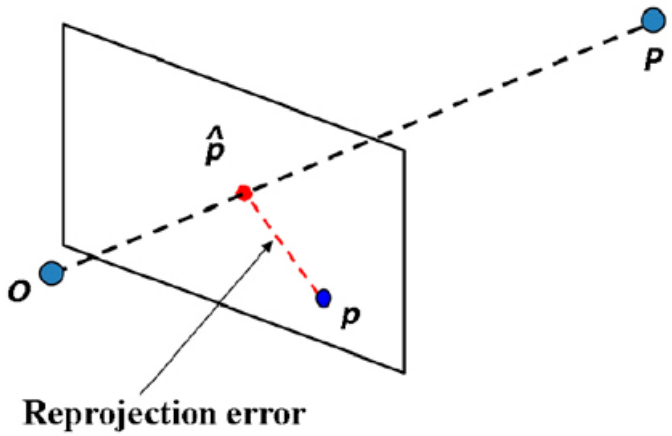
Το σφάλμα επαναπροβολής *(reprojection error)* είναι ένας δείκτης που επιβεβαιώνει την επιτυχία (ή μη) της διαμέτρησης. Για να θεωρηθεί επιτυχημένη, το σφάλμα πρέπει να είναι μικρότερο του ένα (<1). Επιπλέον, όσο πιο κοντά είναι στο μηδέν (0), τόσο πιο επιτυχημένη είναι.

Έστω ότι η διαμέτρηση ολοκληρώθηκε και έχουμε υπολογίσει τις επιθυμητές παραμέτρους. Το σύστημα μας αποτελείται από δύο (2) συστήματα συντεταγμένων: 1) δισδιάστατο σύστημα συντεταγμένων της κάμερας και 2) τρισδιάστατο σύστημα συντεταγμένων του πραγματικού κόσμου. Έχοντας κάποια σημεία στο σύστημα της κάμερας, μπορούμε να υπολογίσουμε ένα σημείο στο σύστημα του πραγματικού κόσμου. Ύστερα, αναιρώντας την εφαρμογή των παραμέτρων, επαναπροβάλλουμε το σημείο στο σύστημα της κάμερας. Επειδή δεν μπορεί να επιτευχθεί τέλεια διαμέτρηση, οι συντεταγμένες του σημείου θα διαφέρουν από αυτές του αρχικού.

Το σφάλμα επαναπροβολής λοιπόν, ορίζεται ως η ευκλείδια απόσταση μεταξύ του σημείου που έχει επαναπροβληθεί στο σύστημα συντεταγμένων της κάμερας και του

[3]Πηγή: http://nicolas.burrus.name/uploads/Research/kinect_chessboard_color.png

αρχικού σημείου.



Εικόνα 3.3.2: Σχηματική αναπαράσταση υπολογισμού σφάλματος επαναπροβολής[4]

Γενικά, η διαμέτρηση ενδέχεται να χρειαστεί πολλαπλές δοκιμές μέχρι να έχουμε επιθυμητό αποτέλεσμα. Συνήθως η δυσκολία έγκειται στην συλλογή καλών δεδομένων, οπότε απαιτείται λίγη εξοικείωση με αυτό.

[4]Πηγή: https://i.stack.imgur.com/kJOo1.jpg

Κεφάλαιο 4

ΓΕΩΓΡΑΦΙΚΟΣ ΣΥΣΧΕΤΙΣΜΟΣ ΜΕΤΑΞΥ ΚΑΜΕΡΑΣ - ΛΕΪΖΕΡ

Όπως αναφέρθηκε σε προηγούμενα κεφάλαια, είναι απαραίτητο να γνωρίζουμε την θέση μεταξύ των αισθητηρίων. Ο στόχος είναι η λήψη σωστών δεδομένων για να εισαχθούν στο σύστημα. Καθώς τα κύρια αισθητήρια είναι το laser και η κάμερα, είναι ζωτικής σημασίας η ακριβής περιγραφή του συσχετισμού τους. Σε αυτό το κεφάλαιο, παρουσιάζεται ο τρόπος υπολογισμού των συντεταγμένων και εξηγείται η λειτουργία των μετατροπών στα διάφορα συστήματα συντεταγμένων.

4.1 Δέντρο tf

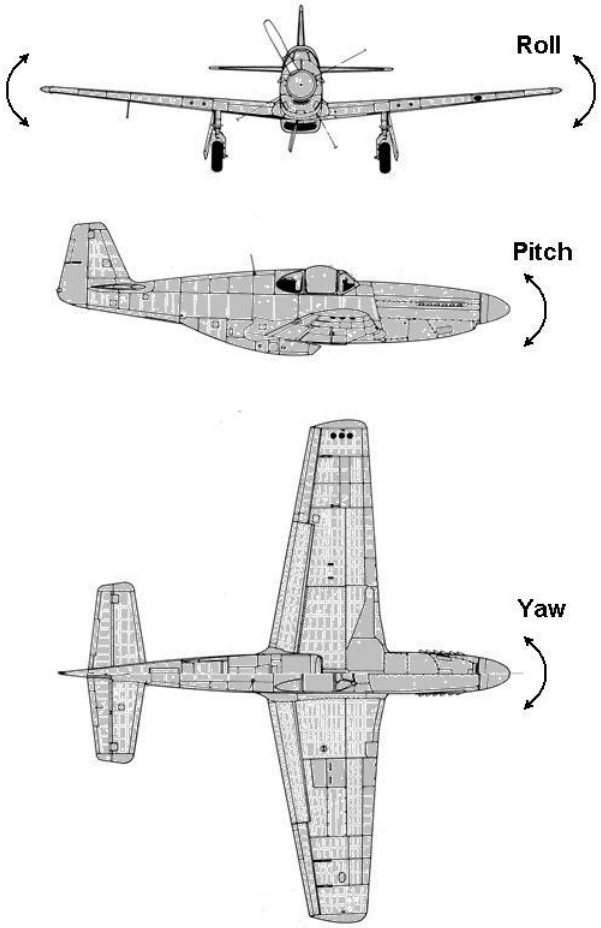
Στο ROS, κάθε κομμάτι του ρομπότ (σύνδεσμος) (αισθητήρες, ενεργοποιητές, συσκευές), έχει δικό του σύστημα συντεταγμένων. Οι θέσεις όλων αυτών, περιγράφονται στο δέντρο tf *(tf tree)*. Τα συστήματα συντεταγμένων ονομάζονται *κάδρα συντεταγμένων (coordinate frames)*.

Όταν αναφερόμαστε στα κομμάτια του ρομπότ στο δέντρο tf, τα ονομάζουμε συνδέσμους *(links)*. Τα αντιστοιχα συστήματα συντεταγμένων, τα ονομάζουμε αρθρώσεις *(joints)*. Μπορούμε να ορίσουμε καινούργιους/ες συνδέσμους/αρθρώσεις σε σχέση με οποιονδήποτε υπάρχον/ουσα σύνδεσμο/άρθρωση επιθυμούμε. Ο σύνδεσμος που χρησιμοποιούμε ως σημείο αναφοράς, ονομάζεται σύνδεσμος-γονέας *(parent link)* και ο

εξαρτώμενος, ονομάζεται σύνδεσμος-παιδί *(child link)*. Οι πληροφορίες που χρειάζονται είναι οι αποστάσεις ύψους, μήκους, πλάτους *(height, length, width)* και οι γωνίες περιστροφών. Αυτές είναι οι *roll, pitch, yaw*.

- roll: Είναι η γωνία κλίσης αριστερά ή δεξιά
- pitch: Είναι η γωνία κλίσης βύθισης ή ανύψωσης
- yaw: Είναι η γωνία στρέψης αριστερά ή δεξιά

Οι παραπάνω γωνίες μετρούνται σε ακτίνια *(radians)*.



Εικόνα 4.1.1: Απεικόνιση roll, pitch, yaw[1]

[1]Πηγή: http://licn.typepad.com/.a/6a01156f9658cc970b0192aaf0c289970d-pi

Ακολουθεί ένα παράδειγμα για να κατανοηθεί καλύτερα η χρησιμότητα και η σημασία των μετατροπών.

Έστω ένα σταθερό ρομπότ με μία κάμερα και έναν ρομποτικό βραχίονα. Ως σημείο αναφοράς του βραχίονα, λογίζεται η άκρη της αρπάγης. Η κάμερα είναι τοποθετημένη 30 εκατοστά πίσω και 40 εκατοστά πάνω από τον βραχίονα. Κάποια στιγμή, η κάμερα εντοπίζει ένα αντικείμενο σε απόσταση 50 εκατοστών. Αν ο βραχίονας κινηθεί 50 εκατοστά μπροστά, δεν θα πιάσει το αντικείμενο, αφού απέχει από τον βραχίονα 50 - 30 = 20 εκατοστά. Επιπλέον, θα κινηθεί σε λάθος ύψος.

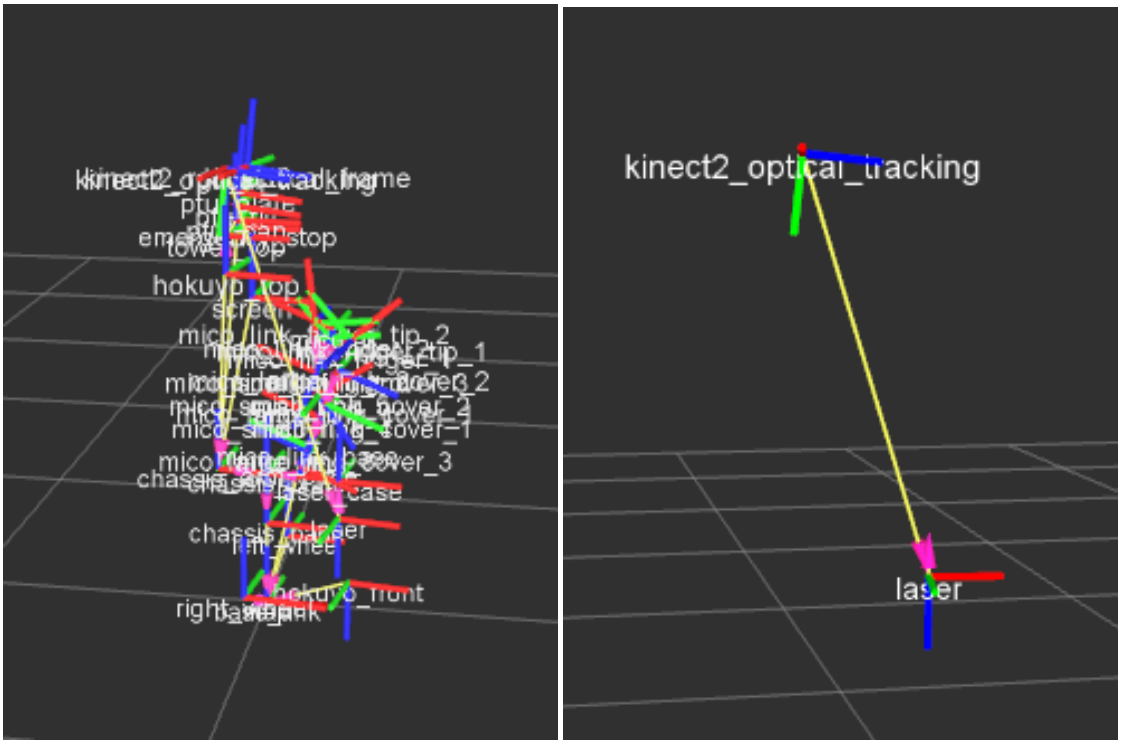
Το κομμάτι του δέντρου tf που περιγράφει τον συσχετισμό μεταξύ κάμερας και laser, είναι το παρακάτω:

```
<joint name="laser_to_kinect2_optical" type="fixed">
<parent link="laser"/>
<child link="kinect2_optical_tracking"/>
<origin xyz="-0.334739 0.0898334 -1.09609" rpy="1.45771263534
   0.0117121896908 1.58159495809"/>
</joint>
<link name="kinect2_optical_tracking"/>
```

Ο συσχετισμός μίας κάμερας με ένα σημείο αναφοράς του εξωτερικού περιβάλλοντος, ονομάζεται και εξωτερική διαμέτρηση *(extrinsics calibration)*.

Με την βοήθεια του εργαλείου RViz (το όνομα του προκύπτει από το *ROS Visualisation*), μπορούμε να δούμε αναλυτικά την μορφή του δέντρου tf. Στην εικόνα 4.1.2 φαίνεται ολόκληρο το δέντρο της Lisa, αλλά και μεμονωμένος ο συνδυασμός μεταξύ του laser και της κάμερας Kinect v2.

Σημειώνεται ότι η κάμερα τοποθετήθηκε σε γωνία 0,1 rad προς τα κάτω (περίπου 5,72 μοίρες).



Εικόνα 4.1.2: Το δέντρο tf της Lisa και μεμονωμένο κομμάτι του (laser-κάμερα)

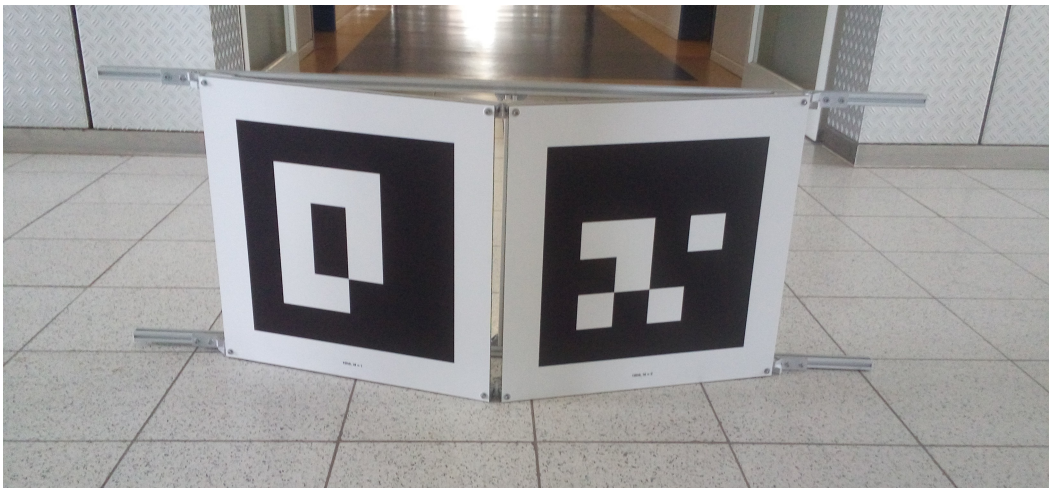
4.2 AprilTags

Το AprilTags αναπτύχθηκε από το Εργαστήριο Ρομποτικής APRIL, μέρος του τμήματος Υπολογιστικής Επιστήμης και Μηχανικής, που ανήκει στο Πανεπιστήμιο του Μίσιγκαν, Η.Π.Α (APRIL Robotics Laboratory, Computer Science and Engineering Department, University of Michigan, USA). Είναι ένα σύστημα βασισμένο σε οπτικά σημεία αναφοράς. Αυτό σημαίνει ότι το λογισμικό αναγνωρίζει κάποια ετικέτα *(tag)* και υπολογίζει τα στοιχεία της θέσης τής σε σχέση με την κάμερα. Υπολογίζει δηλαδή την θέση του στον χώρο, την κατεύθυνση και την ταυτότητα της ετικέτας.

Σημαντικό πλεονέκτημα είναι ότι δεν επηρεάζεται από την γωνία θέασης ούτε από τις συνθήκες φωτισμού που επικρατούν στον χώρο. Ο εντοπισμός των ετικετών είναι λοιπόν αξιόπιστος σε πολύ υψηλό βαθμό. Οι ετικέτες μοιάζουν πολύ με τα QR-codes, αλλά διαφέρουν στο γεγονός ότι τα AprilTags μεταφέρουν μικρότερο φορτίο δεδομένων. Αυτό τους δίνει την ιδιότητα να εντοπίζονται ακόμα και σε μακρινές αποστάσεις. Υπάρχει διαθέσιμο ένα σύνολο δεδομένων που περιέχει τις ετικέτες με τα ονόματα τους. Αφού είναι ασπρόμαυρες, μπορούν να εκτυπωθούν σε έναν απλό εκτυπωτή και είναι

έτοιμες για χρήση.

Σε αυτή την μελέτη, χρησιμοποιήθηκαν οι ετικέτες με ταυτότητα 16h6 id = 1 και id = 2. Δεν έγινε μία απλή εκτύπωση καθώς η διαμέτρηση απαιτεί εντοπισμό και από το laser. Αυτό σημαίνει ότι η ετικέτα δεν θα έπρεπε να έχει το μέγεθος χαρτιού, καθώς θα ήταν πολύ δύσκολο να επιτευχθεί ο στόχος. Στην εικόνα 4.2.1 φαίνεται ο εξοπλισμός που χρησιμοποιήθηκε.



Εικόνα 4.2.1: Πλαίσιο αλουμινίου με AprilTags

4.3 Η προσέγγιση της διαμέτρησης

Η ιδέα για την διαμέτρηση περιγράφεται στην εργασία [6] από τους Frank Neuhaus κ.ά. Σε αυτή την ενότητα, γίνεται μία συνοπτική και απλή επεξήγηση.

Η είσοδος αποτελείται από δεδομένα που έχουν συλλεχθεί από το laser και την κάμερα ταυτόχρονα. Αυτό σημαίνει ότι για κάθε φωτογραφία της κάμερας, υπάρχει ένα αντίστοιχο σύννεφο σημείων *(pointcloud)*. Μέρος του pointcloud περιέχει τις θέσεις των ετικετών την δεδομένη στιγμή. Οπότε, από τις φωτογραφίες γνωρίζουμε την σχετική θέση ετικετών-κάμερας και από τα pointclouds, την σχετική θέση ετικετών-laser. Με δεδομένες αυτές τις συνθήκες, είναι δυνατό να υπολογιστεί η σχέση μεταξύ κάμερας και laser.

4.4 Η διαδικασία της διαμέτρησης

Το εργαλείο multi_dof_kinematic_calibration αναπτύχθηκε από τους Neuhaus κ.ά για την επίτευξη των παραπάνω στόχων. Είναι διαθέσιμο στην παρακάτω ηλεκτρονική διεύθυνση: https://github.com/cfneuhaus/multi_dof_kinematic_calibration .

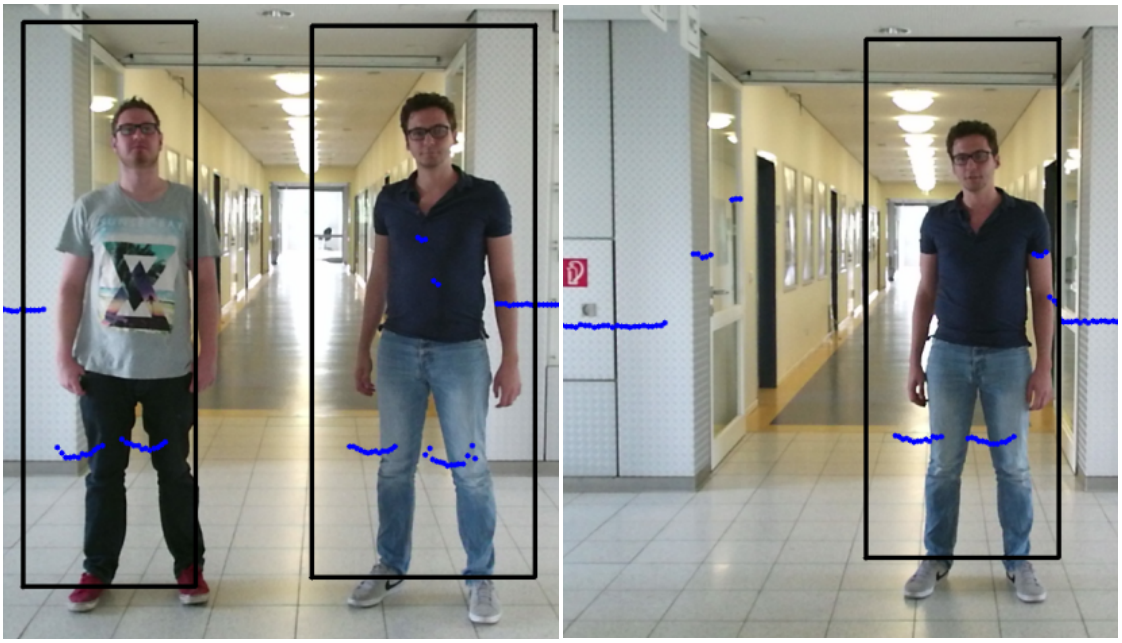
Η διαδικασία είναι παρόμοια με αυτή της εσωτερικής διαμέτρησης. Αρχικά, πρέπει να ολοκληρώσουμε την εσωτερική διαμέτρηση της κάμερας, επειδή οι παράμετροι που προκύπτουν είναι απαραίτητες για την εκτέλεση του παραπάνω εργαλείου. Παρ' όλο που η κάμερα είναι πλέον διαμετρημένη, δεν πρέπει να λάβουμε υπ' όψιν μας τις παραμέτρους όταν συλλέγουμε τις φωτογραφίες. Το ROS, δίνει την δυνατότητα πρόσβασης σε εικόνα όπου παρακάμπτεται η διαμέτρηση. Επιπλέον, όπως εξηγήθηκε νωρίτερα, η λήψη των pointclouds πρέπει να γίνεται ταυτόχρονα με την λήψη των φωτογραφιών.

Αφού εκτελέσουμε το πρόγραμμα, ελέγχουμε αν έγινε σωστή αναγνώριση των ετικετών (εικόνα 4.4.1). Ύστερα, εκφράζουμε τα έγκυρα δεδομένα σε ένα αρχείο *.json*, εισάγουμε τις εσωτερικές παραμέτρους της κάμερας και τρέχουμε το επόμενο μέρος του προγράμματος. Η έξοδος που προκύπτει είναι ο συσχετισμός στον 3-διάστατο χώρο και οι περιστροφές με ένα τετραδόνιο *(quaternion)*. Εύκολα μπορούμε να μετατρέψουμε το τετραδόνιο σε ακτίνια και να λάβουμε τις περιστροφές roll, pitch και yaw.

Τέλος, εισάγουμε τις τιμές που υπολογίστηκαν στο δέντρο tf και ελέγχουμε πειραματικά τον βαθμό επιτυχίας μάς. Στις εικόνες 4.4.2 και 4.4.3 φαίνονται ενδεικτικά παραδείγματα αποτυχημένης και επιτυχημένης διαμέτρησης αντίστοιχα. Οι εικόνες αποτελούν μέρος των bagfiles που εγγράφηκαν για την εκπόνηση αυτής της εργασίας.

Για τη λήψη των δεδομένων, προγραμματίστηκε ένας μικρός κόμβος που συμπεριλαμβάνεται στο πακέτο camera_laser_tracking. Το περιεχόμενο τού παρουσιάζεται σε επόμενο κεφάλαιο.

Εικόνα 4.4.1: Επιτυχής ταυτοποίηση των AprilTags



Εικόνα 4.4.2: Παραδείγματα μη επιτυχημένης εξωτερικής διαμέτρησης

Εικόνα 4.4.3: Παραδείγματα επιτυχημένης εξωτερικής διαμέτρησης

Τα μπλε σημεία απεικονίζουν κάθε ξεχωριστό σημείο του laser που προβάλλεται στην εικόνα της κάμερας. Παρατηρώντας τις εικόνες, βλέπει κανείς ότι αν η εξωτερική διαμέτρηση δεν είναι επιτυχής, οι κουκίδες δεν ταιριάζουν στα σωστά σημεία της εικόνας. Για παράδειγμα, τα σημεία που αντιπροσωπεύουν τα πόδια εμφανίζονται πιο αριστερά απ' ότι θα έπρεπε. Το ίδιο συμβαίνει και με τον τοίχο.

Οι λόγοι για τους οποίους μπορεί να αποτύχει η διαμέτρηση ποικίλλουν. Ένας λόγος μπορεί να είναι η ελλιπής λήψη δεδομένων, δηλαδή να εισάγουμε λίγες εικόνες. Ωστόσο, είναι δυνατό να εισαχθούν αρκετά δεδομένα αλλά να είναι "λάθος" ή συναφή μεταξύ τους. Το πλαίσιο με τα AprilTags πρέπει να βρίσκεται σε πολλά διαφορετικά σημεία του οπτικού πεδίου, ώστε να υπάρχουν πολλά σημεία αναφοράς για να υπολογιστεί η θέση. Σε περίπτωση εσφαλμένης διαμέτρησης, επαναλαμβάνουμε την διαδικασία από την αρχή.

Στην εικόνα 4.4.3, τα σημεία που αντιπροσωπεύουν τα πόδια των ατόμων είναι μεν τοποθετημένα σωστά, βρίσκονται δε σε διαφορετικό ύψος. Αυτό όμως δεν είναι σφάλμα. Όσο πιο μακριά βρίσκεται ένα σημείο στο 3-διάστατο χώρο, τόσο πιο ψηλά φαίνεται μόλις μετατραπεί στον 2-διάστατο χώρο.

Επιπλέον, πρέπει να συνυπολογίσουμε το γεγονός ότι η κάμερα βρίσκεται πιο ψηλά από το laser και με μία μικρή κλίση, οπότε η κάθετη προβολή των σημείων μπορεί να

εμφανίζεται διαφορετικά λόγω ποικίλλων εμποδίων που υπάρχουν στον χώρο. Αυτό γίνεται καλύτερα αντιληπτό κοιτώντας την εικόνα 4.4.4 παρακάτω:

Εικόνα 4.4.4: Απεικόνιση σημείων laser με εμπόδια στον χώρο

Παρατηρούμε ότι υπάρχει μία οριζόντια λευκή γραμμή από τα σημεία του laser, που απεικονίζεται στο χαρτοκιβώτιο που βρίσκεται πάνω στο τραπέζι. Αυτό συμβαίνει επειδή το laser "βλέπει" τον τοίχο κάτω από το τραπέζι, αλλά μόλις τα σημεία προβληθούν κάθετα προς τα πάνω, απεικονίζονται σε εκείνη τη θέση.

Κεφάλαιο 5

ΕΞΩΤΕΡΙΚΑ ΕΡΓΑΛΕΙΑ ΛΟΓΙΣΜΙΚΟΥ

Σε αυτό το κεφάλαιο περιγράφονται τα συμπληρωματικά εργαλεία που χρησιμοποιήθηκαν κατά την εκπόνηση αυτής της μελέτης. Πρόκειται για πακέτα λογισμικού που έχουν ως σκοπό την οδήγηση του Kinect v2 και τον εντοπισμό των ποδιών των ανθρώπων.

5.1 Το πακέτο iai_kinect2

Το λογισμικό αναπτύχθηκε από τον Thiemo Wiedemeyer, στο Ίδρυμα για την Τεχνητή Νοημοσύνη του Πανεπιστημίου της Βρέμης (Institute for Artificial Intelligence, University of Bremen).

Με αυτό το πακέτο επιτυγχάνεται η ενεργοποίηση του Kinect v2 και η λήψη δεδομένων από αυτό. Είναι βασισμένο στις βιβλιοθήκες OpenCV, PCL και libfreenect2. Επιπλέον, οι βιβλιοθήκες Eigen και OpenCL είναι χρήσιμες αλλά όχι και απαραίτητες για την λειτουργία του προγράμματος οδήγησης. Υποστηρίζεται σε ROS Hydro και Indigo.

Παρέχονται τα εξής τέσσερα (4) εργαλεία:

- Εργαλείο εξωτερικής διαμέτρησης μεταξύ κάμερας RGB και IR
- Βιβλιοθήκη προβολής εικόνας βάθους στην έγχρωμη εικόνα

- Γέφυρα μεταξύ libfreenect2 και ROS
- Πρόγραμμα προβολής εικόνας / pointcloud

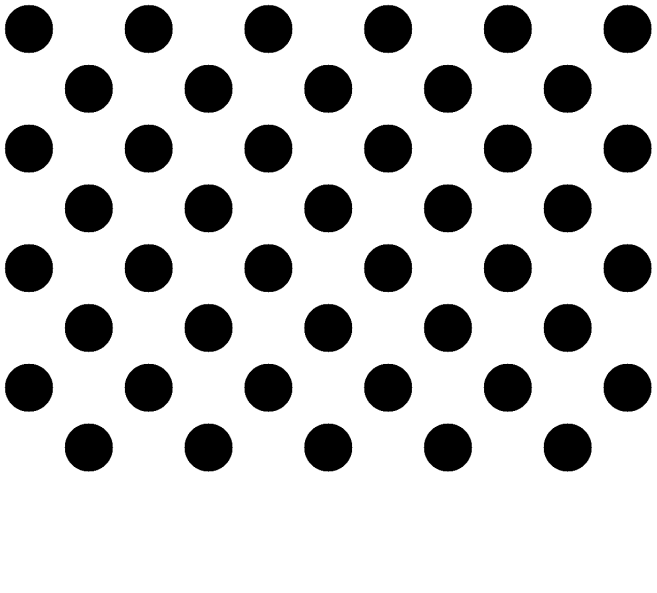
5.1.1 Εργαλείο εξωτερικής διαμέτρησης μεταξύ κάμερας RGB-IR

Η χρήση του εργαλείου δεν ήταν απαραίτητη στην παρούσα μελέτη. Παρ' όλα αυτά, γίνεται μία σύντομη αναφορά στις δυνατότητες που προσφέρει, καθώς μπορεί να είναι χρήσιμο σε μελλοντική εργασία.

Το εργαλείο λειτουργεί είτε με τη χρήση μοτίβων σκακιέρας είτε με ασύμμετρο πλέγμα κύκλων *(asymmetric circle grid)*. Τα διαθέσιμα μοτίβα σκακιέρας είναι:

- 5 x 7 x 0,03
- 7 x 9 x 0,025
- 9 x 11 x 0.02

Επειδή δεν έχει γίνει αναφορά στα ασύμμετρα πλέγματα κύκλων, στην εικόνα 5.1.1 φαίνεται ένα τέτοιο ως παράδειγμα.



Εικόνα 5.1.1: Ασύμμετρο πλέγμα κύκλων διαστάσεων 11x4

Υποθέτωντας ότι δεν έχει γίνει καμία διαμέτρηση στην κάμερα, τα βήματα που ακολουθούμε με αυτό το εργαλείο είναι τα παρακάτω:

- Συλλογή φωτογραφιών από κάμερα RGB
- Εσωτερική διαμέτρηση κάμερας RGB
- Συλλογή φωτογραφιών από κάμερα IR
- Εσωτερική διαμέτρηση κάμερας IR
- Συλλογή φωτογραφιών ταυτόχρονα και από τις δύο κάμερες
- Εξωτερική διαμέτρηση RGB-IR
- Διαμέτρηση δεδομένων βάθους

Στη συνέχεια, αποθηκεύουμε τις παραμέτρους και τις συμπεριλαμβάνουμε κάθε φορά που εκκινούμε το πρόγραμμα οδήγησης.

5.1.2 Βιβλιοθήκη προβολής εικόνας βάθους στην έγχρωμη εικόνα

Για την λειτουργία αυτού του εργαλείου, απαιτείται η εγκατάσταση της OpenCL ή της Eigen. Προτείνεται η χρήση της OpenCL καθώς έτσι επιτυγχάνεται μικρότερο φορτίο κατά την λειτουργία της CPU του υπολογιστή.

5.1.3 Γέφυρα libfreenect2 - ROS

Η εν λόγω γέφυρα συνδέει την βιβλιοθήκη libfreenect2 με το ROS και δημιουργεί πληθώρα topics με διαφορετικά χαρακτηριστικά. Επιλέγουμε αυτά που ταιριάζουν καλύτερα στην εφαρμογή που αναπτύσσουμε. Όσον αφορά την κάμερα RGB, υπάρχουν δύο (2) κατηγορίες:

- HD topics: Η εικόνα έχει ανάλυση 1920x1080 pixels
- Quarter HD topics: Η εικόνα έχει ανάλυση 960x540 pixels

Σε αυτή την εφαρμογή, χρησιμοποιήθηκαν τα topics υψηλής ανάλυσης. Προφανώς, υπάρχουν topics για τα δεδομένα βάθους. Η ανάλυση της εικόνας για αυτά είναι 512x424 pixels.

5.1.4 Πρόγραμμα προβολής εικόνας / pointcloud

Πρόκειται για ένα απλό εργαλείο που εμφανίζει έναν συνδυασμό της έγχρωμης εικόνας και της εικόνας βάθους.

Γενικά, το iai_kinect2 είναι ένα σύνολο που προσφέρει πολλά χρήσιμα εργαλεία και μπορεί κανείς να ρυθμίσει το Kinect v2 από την αρχή και να το φέρει σε πλήρως λειτουργικό επίπεδο.

5.2 Το πακέτο leg_detector

Το πακέτο *leg_detector* είναι μέρος του ευρύτερου πακέτου *people*. Το πακέτο people αποτελείται από υπο-πακέτα και βιβλιοθήκες που αποσκοπούν στην αντίληψη ανθρώπων.

Όπως υποδηλώνει το όνομα του, το πακέτο εντοπίζει τα πόδια των ανθρώπων. Χρησιμοποιεί τα δεδομένα που λαμβάνει από το laser scanner και προσπαθεί να εντοπίσει μοτίβα που ταιριάζουν με αυτά των ποδιών. Αυτό επιτυγχάνεται με την εκπαίδευση ενός ταξινομητή με χρήση Μηχανικής Μάθησης *(Machine Learning trained classifier)*. Ο ταξινομητής εκπαιδεύεται εισάγοντας ένα μεγάλο σύνολο δεδομένων από laser που λάβαμε από σκανάρισμα ποδιών. Με αυτό το τρόπο, μαθαίνει ποια είναι η μορφή των δεδομένων όταν σκανάρουν πόδια. Τελικά, κάθε φορά που εισάγονται νέες μετρήσεις, ο ταξινομητής κάνει μία εκτίμηση για το ποσοστό αντιστοιχίας αυτών με αυτές των ποδιών.

Ο ταξινομητής *(classifier)* είναι ένας αλγόριθμος που χρησιμοποιείται για την υλοποίηση ταξινόμησης δεδομένων *(data classification)*. Ουσιαστικά, πρόκειται για μία μαθηματική συνάρτηση που επιτυγχάνει αυτό ακριβώς. Οι ταξινομητές συνήθως υπάγονται στην επιβλεπόμενη μάθηση *(supervised learning)*. Η λογική πίσω από την λει-

τουργία τούς περιγράφηκε παραπάνω.

Υπάρχουν πολλές κατηγορίες ταξινομητών: Γραμμικοί ταξινομητές *(Linear classifiers)*, *Δέντρα αποφάσεων (Decision trees)*, Νευρωνικά δίκτυα *(Neural networks)* κ.ά. Σε κάθε περίπτωση, καλούμαστε να επιλέξουμε την προσέγγιση που είναι πιο ταιριαστή για το πρόβλημα μας. Κάθε τύπος ταξινομητή δουλεύει διαφορετικά και δεν αποφέρει λύση σε κάθε περίπτωση. Αυτό φυσικά απαιτεί προσεχτική μελέτη του προβλήματος και εμπειρία με αυτά τα εργαλεία.

Στην εργασία [4] από τον Kai O. Arras κ.ά. παρουσιάζεται ο τρόπος που λειτουργεί ο εν λόγω ταξινομητής. Αρχικά, βασίστηκαν στον αλγόριθμο AdaBoost [7]. Ο AdaBoost καταφέρνει επιτυχή ταξινόμηση χρησιμοποιώντας πολλούς μικρότερους και αδύναμους σε επιδόσεις ταξινομητές. Το κριτήριο ώστε να συμμετάσχει ένας αδύναμος ταξινομητής στη διαδικασία, είναι ο βαθμός ακρίβειας τού να είναι καλύτερος από μία τυχαία εκτίμηση.

Η πρόκληση λοιπόν είναι η απόσπαση χρήσιμων χαρακτηριστικών από τις μετρήσεις του laser. Οι παρατηρήσεις αποτελούνται από ένα σύνολο ακτίνων που προσκρούουν στις επιφάνειες. Κάθε ακτίνα περιγράφεται από δύο (2) αριθμούς: 1) η γωνία της ακτίνας σε σχέση με το ρομπότ και 2) το μήκος της ακτίνας. Κάθε ακτίνα χωρίζεται σε υπο-ακτίνες και κρατώντας τις οφέλιμες, ορίζεται ένα νέο σύνολο ακτίνων που περιγράφονται από 1) καρτεσιανές συντεταγμένες και 2) πολικές συντεταγμένες. Για να θεωρηθεί έγκυρη μία ακτίνα και να ληφθεί υπ' όψιν στη κατασκευή του συνόλου δεδομένων που θα χρησιμοποιηθεί για την εκπαίδευση του ταξινομητή *(training dataset)*, πρέπει να έχει δεκατέσσερα (14) χαρακτηριστικά, τα οποία αναλύονται και παρουσιάζονται στην εργασία. Μερικά από αυτά είναι το πλάτος, η γραμμικότητα και η κυκλικότητα *(width, linearity and circularity)*.

Στην εικόνα 5.2.1 φαίνονται μερικά μοτίβα που προκύπτουν από σκανάρισμα ποδιών. Παρατηρούμε ότι είναι ιδιαίτερα δύσκολο να εκπαιδευτεί ένας ταξινομητής, καθώς υπάρχουν πολλές διαφορετικές απεικονίσεις. Για παράδειγμα, η πρώτη και η τρίτη μέτρηση είναι παρόμοιες, αλλά ουδεμία σχέση έχουν με την δεύτερη. Οπότε, πρέπει να αποφασίσουμε ποιο ποσοστό ομοιότητας είναι αποδεκτό για την εφαρμογή μας. Το ποσοστό αυτό ονομάζεται και *αξιοπιστία (reliability)*.

Θα μπορούσαμε λοιπόν να επιλέξουμε ένα χαμηλό βαθμό ομοιότητας, όμως αυτό ενέχει τον κίνδυνο να οδηγηθούμε σε *λανθασμένα θετικά (false positives)*. Λανθασμένο θετικό είναι το αποτέλεσμα όπου μία κατάσταση θεωρείται παρούσα ή αληθής, ενώ στην πραγματικότητα δεν είναι έτσι. Αν έχει επιλεχθεί χαμηλός βαθμός ομοιότητας ως αποδεκτός, μπορεί να εμφανιστούν ενδείξεις εντοπισμού ποδιών εκεί που υπάρχει π.χ. κάποιο έπιπλο. Αντιθέτως, αν επιλεχθεί υψηλός βαθμός αξιοπιστίας, υπάρχει ενδεχόμενο να ληφθούν μετρήσεις από πόδια ανθρώπων αλλά να μην γίνουν δεκτές ως τέτοιες. Οπότε, αποφασίζουμε πειραματικά ποιο ποσοστό θεωρούμε αποδεκτό και πρακτικά "θυσιάζουμε" κάποιες μετρήσεις που ανταποκρίνονται στην πραγματικότητα. Αυτό φαίνεται στην εικόνα 4.4.3 στη σελίδα 44, όπου δεν εντοπίζονται και τα 3 άτομα.

Εικόνα 5.2.1: Ενδεικτικές μετρήσεις laser από σκανάρισμα ποδιών[1]

Υπάρχει μία ιδιάζουσα περίπτωση όπου μπορούμε να έχουμε λανθασμένο θετικό, ενώ ταυτόχρονα πληροίται το κριτήριο της υψηλής αξιοπιστίας. Στην εικόνα 5.2.2 βλέπουμε ότι το μοτίβο των μετρήσεων του laser μοιάζει πολύ με αυτό των ποδιών, καθώς οι αφίσες είναι τυλιγμένες και οι κύλινδροι παραπέμπουν στο σχήμα των ποδιών.

[1]Πηγή: Kai O. Arras, Oscar Martinez Mozos, Wolfram Burgard, *"Using Boosted Features for the Detection of People in 2D Range Data"*, σελ. 2

Εικόνα 5.2.2: Λανθασμένο θετικό με υψηλό βαθμό αξιοπιστίας

5.2.1 Διαχείριση δεδομένων από την έξοδο του πακέτου

Η έξοδος του πακέτου δημοσιεύεται σε μία σειρά από topics. Επιπλέον, υπάρχουν διάφορες παράμετροι που μπορούμε να ορίσουμε τις τιμές τους. Έτσι, φιλτράρουμε τα δεδομένα ώστε να εξυπηρετούν τις ανάγκες της εφαρμογής.

Τα topics αφορούν τις συντεταγμένες στις οποίες εντοπίζονται τα πόδια αλλά και στην απεικόνισή τους σε εργαλεία οπτικοποίησης (π.χ. RViz). Οι παράμετροι επηρεάζουν την κατάσταση του συστήματος και τον τρόπο διαχείρισης των δεδομένων. Για παράδειγμα, υπάρχουν πάραμετροι που ορίζουν τον αποδεκτό βαθμό αξιοπιστίας, την επιτρεπτή απόσταση των ποδιών, την επιτρεπτή απόσταση ανθρώπων ώστε να θεωρούνται ομάδα κτλ.

Οι παράμετροι επιτρέπουν στον προγραμματιστή να αλλάζει τη συμπεριφορά του συστήματος δυναμικά (δηλαδή ακόμα και ταυτόχρονα με την εκτέλεση του προγράμματος) και κατά βούληση, με τρόπο που ταιριάζει στις ανάγκες της εκάστοτε εφαρμογής. Παρακάτω φαίνονται μερικές από αυτές:

- connection_threshold: Η ελάχιστη απόσταση που πρέπει να έχουν τα σημεία του laser ώστε να θεωρηθούν ομάδα.
- min_points _per_group: Ο ελάχιστος αριθμός σημείων του laser ώστε να θεωρηθούν ομάδα.
- leg_reliability_limit: Εμφανίζονται τα πόδια που έχουν βαθμό αξιοπιστίας μεγαλύτερο ή ίσο με αυτό.
- leg_pair_separation: Η μέγιστη απόσταση που μπορούν να έχουν δύο (2) πόδια ώστε να θεωρηθούν ένα (1) άτομο.
- publish_leg_markers: Αν θα εκδίδονται (publish) οπτικοί δείκτες για τα πόδια (για χρήση σε εργαλεία οπτικοποίησης)
- publish_people_markers: Αν θα εκδίδονται (publish) οπτικοί δείκτες για τους ανθρώπους (για χρήση σε εργαλεία οπτικοποίησης)

Κεφάλαιο 6

ΤΟ ΠΑΚΕΤΟ camera_laser_tracking

Το εν λόγω πακέτο είναι το αποτέλεσμα αυτής της μελέτης. Σε αυτό το κεφάλαιο, παρουσιάζονται όλα τα μέρη του και ολόκληρος ο κώδικας που αναπτύχθηκε. Ο κώδικας συνοδεύεται με επεξηγηματικά σχόλια. Επίσης, περιλαμβάνονται μερικές σύντομες οδηγίες για την χρήση του πακέτου.

6.1 Τρόπος γραφής κώδικα

Όταν αναπτύσσουμε μακροσκελές λογισμικό, είναι χρήσιμο να ακολουθούμε ένα πρότυπο τρόπο γραφής, ώστε να είναι πιο ευανάγνωστος στον προγραμματιστή αλλά και σε όποιον θελήσει να τον μελετήσει ή επεξεργαστεί. Πριν προχωρήσουμε λοιπόν στον κώδικα, παρουσιάζεται εδώ ο σύντομος οδηγός που χρησιμοποιήθηκε. Βασίζεται στον οδηγό γραφής του ROS, αλλά έχουν γίνει μερικές μετατροπές για να ταιριάξει στις ανάγκες της ομάδας. Ο οδηγός αυτός αφορά την γλώσσα C++ .

6.1.1 Αγκύλες

Όταν ανοίγουμε ή κλείνουμε αγκύλες, πρέπει να γίνονται σε διαφορετική γραμμή από την πιο πρόσφατη εντολή. Παρακάτω φαίνεται σε παράδειγμα ο σωστός και ο λάθος τρόπος.

```
if (condition)
{
```

```
    //code
}
if (condition) {
    //code
}
```

6.1.2 Κλάσεις

Τα ονόματα των κλάσεων είναι κεφαλαία ανά λέξη (CamelCased). Τα ακρωνύμια μπορούν να γραφτούν ολόκληρα με κεφαλαία.

```
class ExampleClass;
class HokuyoURGLaser;
```

6.1.3 Συναρτήσεις

Τα ονόματα των συναρτήσεων έχουν το γράμμα της πρώτης λέξης πεζό και τα υπόλοιπα κεφαλαία ανά λέξη (camelCased). Οι παράμετροι περιέχουν *κάτω παύλα* ανά λέξη.

```
void exampleFunction(int example_parameter);
```

6.1.4 Μεταβλητές

Τα ονόματα των μεταβλητών περιέχουν κάτω παύλα ανά λέξη. Οι μεταβλητές που είναι μέλη κλάσεων έχουν το πρόθεμα **m_** και οι σταθερές γράφονται με κεφαλαία.

```
int example_variable;
int m_member_variable;
const int CONSTANT_VARIABLE;
```

6.1.5 Namespace

Γενικά είναι καλή πρακτική να μην χρησιμοποιούμε namespaces, επειδή με αυτό το τρόπο γνωρίζουμε την πηγή κάθε συνάρτησης που καλούμε και είναι πιο εύκολες οι

διορθώσεις.

```
using namespace std; //Wrong
using namespace cv;

std::vector<int> list_of_ints; //Correct
cv::Mat depth_image;
```

6.1.6 Publishers/Subscribers/Callbacks

Έχουν το αντίστοιχο από τα παρακάτω επιθήματα : _pub, _sub, Callback .

```
ros::Publisher m_speak_pub;
ros::Subscriber m_target_reached_sub;
void moveCallback(const robot_platform::MoveRobot::ConstPtr& rosmsg
    )
```

6.1.7 Χρήση κλάσεων της βιβλιοθήκης std

Προτιμούμε την χρήση τους επειδή αποφεύγουμε δικά μας λάθη και ο κώδικας συρρικνώνεται σε αισθητό βαθμό. Για παράδειγμα, είναι καλύτερα να δώσουμε την εντολή:

```
std::map<std::string, std::string> question_answer_map;
```

παρά να κάνουμε το παρακάτω:

```
if(input.find(m_question_1_trig_1) != std::string::npos)
    q[1]++;

if(input.find(m_question_1_trig_2) != std::string::npos)
    q[1]++;

if(input.find(m_question_2_trig_1) != std::string::npos)
    q[2]++;
```

```
if(input.find(m_question_2_trig_2) != std::string::npos)
    q[2]++;

if(input.find(m_question_3_trig_1) != std::string::npos)
    q[3]++;
...
```

6.2 Ο φάκελος camera_laser_tracking

Σε αυτό το φάκελο βρίσκονται δύο (2) αρχεία: CMakeLists.txt και package.xml . Ο ρόλος τους είναι η δήλωση στοιχείων των προγραμμάτων που χρησιμοποιεί το πακέτο, ώστε να αναγνωρίζονται και να εντοπίζονται όταν εκτελούνται οι κόμβοι μας.

6.2.1 Το αρχείο CMakeLists.txt

Όταν δημιουργείται το πακέτο στο catkin, το αρχείο περιέχει ένα σύντομο χρήσιμο κώδικα και το υπόλοιπο περιεχόμενο είναι σε μορφή σχολίων. Κάθε φορά που χρειάζεται να εισάγουμε μία νέα ιδιότητα στο πακέτο, βγάζουμε το κομμάτι που μας ενδιαφέρει από το σχόλιο και συμπληρώνουμε τον κώδικα που επιθυμούμε. Εδώ λοιπόν, καταγράφεται μόνο ο ωφέλιμος κώδικας του αρχείου και όχι τα σχόλια.

```
cmake_minimum_required(VERSION 2.8.3)
project(camera_laser_tracking)
# Using c++ 2011
set(CMAKE_CXX_FLAGS "-std=c++11 ${CMAKE_CXX_FLAGS}")
set(CMAKE_CXX_FLAGS_DEBUG "${CMAKE_CXX_FLAGS_DEBUG} -Wall")

find_package(catkin REQUIRED COMPONENTS
roscpp
rospy
std_msgs
```

```
image_transport
cv_bridge
tf
laser_geometry
image_geometry
tf_conversions
people_msgs
message_generation
roslib
sensor_msgs
)
find_package (PCL REQUIRED)
```

Εδώ, δηλώσαμε ότι χρησιμοποιούμε την C++ 11 και καταγράφουμε τα στοιχεία και τις βιβλιοθήκες που πρέπει να γνωρίζει το πακέτο ώστε να λειτουργεί.

```
# Generate messages in the 'msg' folder
add_message_files(
FILES
cl_msgs.msg
)
# Generate added messages and services with any dependencies listed here
generate_messages(
DEPENDENCIES
std_msgs
sensor_msgs
people_msgs
)

catkin_package(
```

```
CATKIN_DEPENDS message_runtime sensor_msgs
)

include_directories(
${catkin_INCLUDE_DIRS}
${EIGEN3_INCLUDE_DIR}
${PCL_INCLUDE_DIRS}
)

link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})
```

Παραπάνω, δηλώθηκε η ύπαρξη μηνυμάτων που φτιάχτηκαν από τον προγραμματιστή, οι εξαρτήσεις του πακέτου από έτοιμα μηνύματα και οι φάκελοι όπου βρίσκονται τα αρχεία των βιβλιοθηκών που χρησιμοποιούμε.

```
add_executable(cl_data_acq_node src/cl_data_acq_node.cpp)
add_executable(cl_visualise_node src/cl_visualise_node.cpp)
add_executable(cl_demo src/cl_demo.cpp)

target_link_libraries(cl_data_acq_node ${catkin_LIBRARIES} ${PCL_LIBRARIES})
target_link_libraries(cl_visualise_node ${catkin_LIBRARIES} ${PCL_LIBRARIES})
target_link_libraries(cl_demo ${catkin_LIBRARIES} ${PCL_LIBRARIES})
```

Τέλος, δηλώνουμε τους κόμβους που δημιουργήσαμε και τις βιβλιοθήκες που πρέπει να συνδεθούν με αυτούς.

6.2.2 Το αρχείο package.xml

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

<name>camera_laser_tracking</name>

<version>0.0.0</version>

<description>The camera_laser_tracking package</description>

<maintainer email="dpkoikas@gmail.com">Dimitris Koikas</maintainer>

<license>BSD</license>

<buildtool_depend>catkin</buildtool_depend>

<build_depend>roscpp</build_depend>

<build_depend>rospy</build_depend>

<build_depend>std_msgs</build_depend>

<build_depend>image_transport</build_depend>

<build_depend>cv_bridge</build_depend>

<build_depend>opencv2</build_depend>

<build_depend>laser_geometry</build_depend>

<build_depend>libpcl-all-dev</build_depend>

<build_depend>tf_conversions</build_depend>

<build_depend>people_msgs</build_depend>

<build_depend>message_generation</build_depend>

<build_depend>sensor_msgs</build_depend>

<build_depend>geographic_msgs</build_depend>

<run_depend>libpcl-all</run_depend>

<run_depend>roscpp</run_depend>

<run_depend>rospy</run_depend>

<run_depend>std_msgs</run_depend>

<run_depend>image_transport</run_depend>

<run_depend>cv_bridge</run_depend>

<run_depend>opencv2</run_depend>

<run_>laser_geometry</run_depend>

<run_depend>tf_conversions</run_depend>
<run_depend>people_msgs</run_depend>
<run_depend>message_generation</run_depend>
<run_depend>message_runtime</run_depend>
<run_depend>sensor_msgs</run_depend>
<run_depend>geographic_msgs</run_depend>
<export>
</export>
</package>

Σε αυτό το αρχείο περιλαμβάνονται στοιχεία για το πακέτο και δηλώνονται όλες οι εξαρτήσεις από τις βιβλιοθήκες που χρησιμοποιούμε.

6.3 Ο φάκελος msg

Ο φάκελος περιέχει τα μηνύματα που δημιουργεί ο προγραμματιστής. Στην συγκεκριμένη περίπτωση, το πακέτο περιέχει ένα μόνο σύντομο μήνυμα.

6.3.1 Το αρχείο cl_msgs.msg

sensor_msgs/Image cl_image
people_msgs/PositionMeasurementArray cl_detected_legs
sensor_msgs/RegionOfInterest[] cl_bounding_box

Το μήνυμα περιέχει δεδομένα που θα ήταν χρήσιμα για κάποιον χρήστη του πακέτου. Δηλαδή, τις εικόνες των ατόμων που εντοπίζονται, τις συντεταγμένες των ποδιών που εντοπίζονται και τις συντεταγμένες του πλαισίου οριοθέτησης.

6.4 Ο φάκελος config

Περιέχει αρχεία με διάφορα αρχεία ρυθμίσεων, όπως αρχεία του RViz και αρχεία παραμέτρων.

6.4.1 Το αρχείο cl_parameters.yaml

/cl_visualise_node/rel_threshP: 0.7

/cl_visualise_node/base_laser_diffP: -0.2

/cl_visualise_node/person_heightP: 1.8

/cl_visualise_node/person_widthP: 0.7

/cl_visualise_node/box_marginP: 0.0

/cl_visualise_node/camera_frameP: "kinect2_optical_tracking"

/cl_visualise_node/laser_frameP: "laser"

/cl_visualise_node/base_frameP: "base_link"

/cl_visualise_node/camera_topicP: "/kinect2/hd/image_color_rect"

/cl_visualise_node/camera_info_topicP: "/kinect2/hd/camera_info"

/cl_visualise_node/laser_topicP: "/scan"

/cl_visualise_node/show_debug_imagesP: "true"

/cl_data_acq_node/desired_camera_topic_rawP: "/kinect2/hd/image_color"

/cl_data_acq_node/desired_camera_topic_rectP: "kinect2/hd/image_color_rect"

/cl_data_acq_node/desire_laser_topic_bottomP: "/scan"

/cl_data_acq_node/desire_laser_topic_topP: "/top_scan"

Οι παραπάνω παράμετροι καλούνται από τα launchfiles και αρχικοποιούνται. Για κάθε παράμετρο έχει επιλεχθεί το όνομα του κόμβου που εξυπηρετούν, ως πρόθεμα. Οι παράμετροι βοηθούν την γενίκευση και την έυκολη αλλαγή των θεμάτων που χρησιμοποιούνται. Η εύκολη αλλαγή θεμάτων και μεταβλητών επιτυγχάνεται και μέσα από τα launchfiles. Ουσιαστικά, οι παραπάνω παράμετροι φορτώνονται στον parameter server.

6.5 Ο φάκελος launch

Ο φάκελος περιέχει δύο (2) αρχεία. Το ένα εξυπηρετεί τη χρήση του πακέτου με το ρομπότ και το άλλο χωρίς το ρομπότ, με χρήση αρχείου *.bag* .

6.5.1 Το αρχείο project_track.launch

Ο driver του Kinect δεν εκκινείται σε αυτό το launchfile. Αυτό συμβαίνει επειδή συμπεριλαμβανόταν σε ένα ξεχωριστό launchfile που εκτελούσαμε κάθε φορά που χρησιμοποιούσαμε το ρομπότ. Εγκυκλοπαιδικά, η εντολή που εκκινεί τον driver είναι η:
<include file="$(find kinect2_bridge)/launch/kinect2_bridge.launch" />

Τα launchfiles ξεκινούν με το tag <launch> και κλείνουν με το tag </launch>

<launch>
<arg name="record_bag" default="false" />
<arg name="record_bag_track" default="false" />

<include file="$(find perception_people_launch)/launch/people_tracker_robot.launch"/>

Στις παραπάνω εντολές, ορίζουμε και αρχικοποιούμε arguments. Επιπλέον, εκκινούμε το πρόγραμμα εντοπισμού ποδιών. Αν κατά την εκτέλεση του αρχείου θέσουμε τιμή true σε κάποιο argument, τότε θα εκτελεστεί η αντίστοιχη ομάδα εντολών *(group)* παρακάτω.

<group if="$(arg record_bag_track)">
<node pkg="rosbag" type="record" name="dimitris_bag"
args="-o $(env HOME)/Desktop/dimitris_data/with_tracker/dimitris_bag
/scan /map /odom /tf /kinect2/hd/image_color_rect /kinect2/hd/camera_info
kinect2/hd/image_color /leg_tracker_measurements /people_tracker_measurements
/kinect2/qhd/camera_info /kinect2/qhd/points /kinect2/qhd/image_depth_rect /kinect2/qhd/image_color">

```
<param name="mkdir_tmp"
command="mkdir -m 777 -v -p $(env HOME)/Desktop/dimitris_data/with_tracker"/>
</node>
</group>
```

Εδώ, εκκινούμε τον κόμβο rosbag, ονομάζουμε το αρχείο και δίνουμε τον προορισμό αποθήκευσης του αρχείου. Ύστερα, καταγράφουμε τα topics που επιθυμούμε να εγγράψουμε. Στο tag που ονομάζεται param, δημιουργούμε τον φάκελο προορισμού που προαναφέρθηκε.

```
<group if="$(arg record_bag)">
<node pkg="rosbag" type="record" name="dimitris_bag"
args="-o $(env HOME)/Desktop/dimitris_data/without_tracker/dimitris_bag
/scan /map /odom /tf /kinect2/hd/image_color_rect /kinect2/hd/camera_info kinect2/hd/image_color">
<param name="mkdir_tmp"
command="mkdir -m 777 -v -p $(env HOME)/Desktop/dimitris_data/without_tracker"/>
</node>
</group>

</launch>
```

Σε αυτή την ομάδα εντολών, ουσιαστικά επαναλαμβάνεται η προηγούμενη, με την διαφορά ότι αυτή τη φορά, δεν εγγράφονται τα topics /leg_tracker_measurements και /people_tracker_measurements.

6.5.2 Το αρχείο project_track_offline.launch

```
<launch>
<arg name="bagfile_track" default="false" />
<arg name="bagfile" default="false" />

<rosparam command="load" file="$(find camera_laser_tracking)/config/cl_parameters.yaml"/>
```

Αρχικοποιούμε arguments για την αναπαραγωγή bagfiles και φορτώνουμε τις παραμέτρους στον parameter server. Οι τιμές που εκχωρούνται στις παραμέτρους είναι αυτές που υπάρχουν στο αρχείο.

```
<arg name="reliability" default="0.7" />
<arg name="base_to_laser" default="-0.2" />
<arg name="person_height" default="1.8" />
<arg name="person_width" default="0.7" />
<arg name="bounding_margin" default="0.0" />
<arg name="camera_frame" default="kinect2_optical_tracking" />
<arg name="laser_frame" default="laser" />
<arg name="base_frame" default="base_link" />
<arg name="camera_topic" default="/kinect2/hd/image_color_rect" />
<arg name="camera_info_topic" default="/kinect2/hd/camera_info" />
<arg name="laser_topic" default="/scan" />
<arg name="debug_images" default="true" />
```

Με τον παραπάνω κώδικα θέτουμε ως arguments τις παραμέτρους, με πιο απλά ονόματα. Αυτό εξυπηρετεί τον σκοπό της μεταχείρισης τους από το terminal κατά την εκτέλεση του launchfile. Οι τιμές που εκχωρούνται είναι όμοιες με του αρχείου που φορτώσαμε, ως δικλείδα ασφαλείας.

```
<rosparam param="cl_visualise_node/rel_threshP" subst_value="true">$(arg reliability)</rosparam>
<rosparam param="cl_visualise_node/base_laser_diffP"
subst_value="true">$(arg base_to_laser)</rosparam>
<rosparam param="cl_visualise_node/person_heightP"
subst_value="true">$(arg person_height)</rosparam>
<rosparam param="cl_visualise_node/person_widthP"
subst_value="true">$(arg person_width)</rosparam>
<rosparam param="cl_visualise_node/box_marginP"
subst_value="true">$(arg bounding_margin)</rosparam>
<rosparam param="cl_visualise_node/camera_frameP"
subst_value="true">$(arg camera_frame)</rosparam>
<rosparam param="cl_visualise_node/laser_frameP" subst_value="true">$(arg laser_frame)</rosparam>
<rosparam param="cl_visualise_node/base_frameP" subst_value="true">$(arg base_frame)</rosparam>
<rosparam param="cl_visualise_node/camera_topicP"
subst_value="true">$(arg camera_topic)</rosparam>
<rosparam param="cl_visualise_node/camera_info_topicP"
subst_value="true">$(arg camera_info_topic)</rosparam>
<rosparam param="cl_visualise_node/laser_topicP" subst_value="true">$(arg laser_topic)</rosparam>
<rosparam param="cl_visualise_node/show_debug_imagesP"
subst_value="true">$(arg debug_images)</rosparam>
```

Οι παραπάνω εντολές δίνουν στις παραμέτρους που υπάρχουν στον parameter server τις τιμές που έχουμε δώσει στα αντίστοιχα arguments.

```
<group if="$(arg bagfile_track)">
<node pkg="rosbag" type="play" name="bagfile_with_tracker" output="screen" required="true"
args="--clock /home/dimitris/bagfiles/dimitris_bag_2016-08-26-17-40-31.bag"/>
</group>
```

<group if="$(arg bagfile)">
<node pkg="rosbag" type="play" name="bagfile_without_tracker" output="screen" required="true"
args="--clock /home/dimitris/bagfiles/dimitris_bag_2016-08-04-15-02-20.bag "/>
</group>

<node name="display_projected_points" pkg="camera_laser_tracking" type="cl_visualise_node"
respawn="true" output="screen" />

Οι ομάδες εντολών των arguments bagfile_track και bagfile αναπαράγουν δεδομένα από bagfiles με ή χωρίς μετρήσεις από το πακέτο εντοπισμού ποδιών. Αυτό εξυπηρετεί την γρήγορη εκτέλεση πειραμάτων κατά την ανάπτυξη των κόμβων. Τέλος, εκτελείται ο κύριος κόμβος του πακέτου.

6.6 Ο φάκελος src

Αυτός ο φάκελος περιέχει αρχεία κεφαλίδας *(header files)* που κατασκευάζει ο προγραμματιστής, καθώς και τους κόμβους του πακέτου.

Για να γίνει κατανοητός ο κώδικας των κόμβων, πρέπει να εξηγηθεί η λειτουργία των συναρτήσεων επανάκλησης *(callback functions)*. Οι συναρτήσεις αυτές καλούνται αυτόματα κάθε φορά που συμβαίνει ένα γεγονός. Σε αυτή τη περίπτωση, τέτοια γεγονότα είναι η άφιξη νέων μετρήσεων από το laser ή/και τη κάμερα. Οπότε, χρειάζονται δύο (2) συναρτήσεις επανάκλησης, μία για κάθε αισθητήρα.

6.6.1 Το αρχείο loadRosConfig.h

Το αρχείο αυτό έχει γραφτεί από κάποιο μέλος της ομάδας στο παρελθόν και συμπεριλήφθηκε σε αυτό το πακέτο για να διευκολύνει την εκχώρηση τιμών από τον parameter server στις μεταβλητές των κόμβων.

```
1 #ifndef LOAD_ROS_CONFIG_H
2 #define LOAD_ROS_CONFIG_H
3
4 #include <ros/ros.h>
5 #include <string>
```

```
6
7  template <typename T>
8  inline bool loadConfigValue(std::string valueName, T &variable, T
       default_variable = T() )
9  {
10     if(ros::param::has(valueName))
11     {
12       ros::param::get(valueName, variable);
13       ROS_INFO_STREAM(valueName << ": " << variable);
14       return true;
15     }
16     else
17     {
18       ROS_WARN_STREAM("No Parameter: " << valueName << ".
       Defaulting to "<<default_variable<<".");
19       variable = default_variable;
20       return false;
21     }
22 }
```

Όταν στον κόμβο καλέσουμε την παραπάνω συνάρτηση, δίνουμε στην μεταβλητή &variable του περιεχόμενο της παραμέτρου valueName. Εάν η valueName δεν υπάρχει στον parameter server, τότε εκχωρείται το περιεχόμενο της default_variable. Οι εντολές εξόδου ROS_INFO_STREAM και ROS_WARN_STREAM εμφανίζουν στο terminal τί έχει συμβεί με την εξέλιξη του αιτήματος.

Για παράδειγμα, η εντολή:

loadConfigValue("/cl_visualise_node/rel_threshP", rel_thresh, 0.7);

ζητάει να εκχωρηθεί στη rel_thresh η παράμετρος /cl_visualise_node/rel_threshP. Αν η παράμετρος δεν υπάρχει, τότε παίρνει την τιμή 0.7, ώστε να μην "κρασάρει" ο κόμβος.

```
24 template <typename T>
25 inline bool loadConfigValue(std::string valueName, std::vector<T> &
       variable)
26 {
27     if(ros::param::has(valueName))
28     {
29     ros::param::get(valueName, variable);
30     ROS_INFO_STREAM(valueName << ":");
31
32     for (int i = 0; i < variable.size(); ++i)
33       ROS_INFO_STREAM("Value "<< i << " :" << variable[i]);
34
```

```
35      return true;
36      }
37      else
38      {
39        ROS_WARN_STREAM("No Parameter: " << valueName << ".
        Defaulting to empty vector.");
40        variable.clear();
41        return false;
42      }
43  }
44
45  #endif
```

Η ίδια λογική χρησιμοποιείται και σε αυτή τη συνάρτηση για την περίπτωση που επιθυμούμε να φορτώσουμε διάνυσμα σε μία παράμετρο και όχι μία απλή τιμή. Σε περίπτωση σφάλματος, εκχωρείται ένα κενό διάνυσμα.

6.6.2 Το αρχείο camera_laser.h

Πρόκειται για ένα header file που περιέχει άλλα header files, ώστε όταν δημιουργείται ένας νέος κόμβος, να είναι πιο εύκολη η συμπερίληψη όλων. Οπότε, απλά δηλώνουμε αυτό το αρχείο και προσθέτουμε δευτερόντα header files, εφ' όσον αυτά είναι απαραίτητα.

```
1   #ifndef CAMERA_LASER_H
2   #define CAMERA_LASER_H
3
4   #include <iostream>
5   #include <sstream>
6   #include <cstdlib>
7   #include <vector>
```

Δήλωση μηνυμάτων εικόνας συμβατών με το ROS

```
9   #include <ros/ros.h>
10  #include <signal.h>
11  #include <sensor_msgs/Image.h>
12  #include <sensor_msgs/image_encodings.h>
13  #include <image_transport/image_transport.h>
```

Δήλωση αρχείων απαραίτητων για τη λήψη και τη διαχείριση δεδομένων από το laser

```
15  #include <tf/transform_listener.h>
16  #include <sensor_msgs/LaserScan.h>
17  #include <sensor_msgs/PointCloud2.h>
```

```
18 #include <tf/message_filter.h>
19 #include <tf/transform_listener.h>
20 #include <tf/transform_broadcaster.h>
21 #include <message_filters/subscriber.h>
22 #include <laser_geometry/laser_geometry.h>
23 #include <pcl/io/pcd_io.h>
24 #include <pcl/point_cloud.h>
25 #include <pcl_conversions/pcl_conversions.h>
26 #include <pcl/PCLPointCloud2.h>
27 #include <pcl/point_types.h>
28 #include <pcl/conversions.h>
29 #include <pcl_ros/transforms.h>
```

Δήλωση αρχείων απαραίτητων για την λήψη και την διαχείρηση δεδομένων από την κάμερα

```
31 #include <opencv2/opencv.hpp>
32 #include <opencv2/core/core.hpp>
33 #include <cv_bridge/cv_bridge.h>
34 #include <opencv2/highgui/highgui.hpp>
35 #include <opencv2/imgproc/imgproc.hpp>
36
37 #include <image_geometry/pinhole_camera_model.h>
38 #include <boost/foreach.hpp>
39
40 #endif
```

6.6.3 Το αρχείο cl_data_acq_node.cpp

Πρόκειται για τον κόμβο που αναπτύχθηκε για τη λήψη δεδομένων που χρειάζονται για την εξωτερική διαμέτρηση, όπως αναφέρεται στην ενότητα 4.4 . Ο χρήστης έχει τη δυνατότητα να επιλέξει μεταξύ διαμετρημένων ή μη, δεδομένων από τη κάμερα καθώς και πιο laser θέλει να χρησιμοποιηθεί. (Υπενθυμίζεται ότι η Lisa έχει δύο (2) lasers, όπως αναφέρεται στην ενότητα 1.1). Ύστερα, πατώντας το λατινικό "P" αποθηκεύει ταυτόχρονα δεδομένα από τα αντίστοιχα topics. Τα ονόματα των εικόνων είναι 1.png, 2.png κ.ο.κ. ενώ των pointclouds είναι 1.pcl, 2.pcl κ.ο.κ. Όταν συλλεχθεί ικανοποιητικός αριθμός ζεύγων δεδομένων, τότε ο χρήστης κλείνει το πρόγραμμα πατώντας το λατινικό "E".

```
1  #include "camera_laser.h"
2  #include "loadRosConfig.h"
3
4  std::string key_press;
5  const std::string MESSAGE="Press P to collect data or E to exit: ";
6
7  int counter = 1;
8  std::stringstream sstream;
9  const std::string HOME_DIR = getenv("HOMER_DIR");
10 const std::string IM_TYPE = ".png";
11 const std::string SCAN_TYPE=".pcl";
12 const std::string IM_DIRECTORY = "/catkin_ws/src/
       camera_laser_tracking/data/camera_images/";
13 const std::string SCAN_DIRECTORY = "/catkin_ws/src/
       camera_laser_tracking/data/laser_scans/";
14 const char* converted_scan_save_directory;
15 sensor_msgs::PointCloud2 global_cloud;
16 pcl::PCLPointCloud2 converted_cloud;
17
18 bool flag = true;
19 int camera_image_selection = 0;
20 int laser_selection = 0;
21 std::string desired_camera_topic = "";
22 std::string desired_laser_topic = "";
```

Το πρώτο σκέλος του κώδικα αποτελεί προπαρασκευαστικό στάδιο. Συμπεριλαμβάνουμε τα header files που προαναφέρθηκαν και ύστερα ορίζουμε μεταβλητές ώστε να πληρούνται τις προϋποθέσεις για τη σωστή συλλογή και αποθήκευση των δεδομένων. Έχουμε έναν μετρητή, τις καταλήξεις των αρχείων, τη διαδρομή αποθήκευσης και μεταβλητές που μας επιτρέπουν να χειριζόμαστε δεδομένα εκτός των callback συναρτήσεων όπου τα λαμβάνουμε για πρώτη φορά. Επίσης, ετοιμάζουμε μεταβλητές που θα λάβουν τιμές αφού ο χρήστης επιλέξει αισθητήρες κατά την εκκίνηση του προγράμματος.

```
26 class LaserScanToPointCloud
27 {
28   public:
29     void ConfigFunction();
30     void scanCallback(const sensor_msgs::LaserScan::ConstPtr&
       scan_input);
31   private:
32     ros::NodeHandle m_laser_nh;
33     laser_geometry::LaserProjection m_projector;
34     tf::TransformListener m_tfListener;
35     ros::Publisher m_point_cloud_pub;
```

```
36      ros::Subscriber m_scan_sub;
37  };
38
39  void LaserScanToPointCloud::ConfigFunction()
40  {
41    m_scan_sub = m_laser_nh.subscribe<sensor_msgs::LaserScan>(
       desired_laser_topic, 100, &LaserScanToPointCloud::scanCallback,
       this); //
42    m_point_cloud_pub = m_laser_nh.advertise<sensor_msgs::PointCloud2
       > ("/cloud", 100, false); //
43  }
```

Εδώ κατασκευάζουμε μία κλάση επειδή διευκολύνεται η λήψη των μετρήσεων από το laser. Το δημόσιο μέρος έχει συνάρτηση για να ρυθμίσουμε από πιο topic θα λαμβάνουμε δεδομένα και σε ποιο θα αποστέλλεται το pointcloud. Επιπλέον, δηλώνεται η συνάρτηση επανάκλησης δίνοντας τον τύπο του μηνύματος που θα την πυροδοτεί.
Στο ιδιωτικό μέρος, εισάγουμε publishers και subscribers σε μεταβλητές για να είναι πιο εύκολη η χρήση τους. Επίσης, υπάρχει μεταβλητή για τη μετατροπή των μετρήσεων του laser σε pointcloud και για τη μετατροπή μεταξύ συστημάτων συντεταγμένων. Ο *node handler* επιτρέπει στη κλάση να αναγνωριστεί από το κόμβο και διαχειρίζεται τα δεδομένα τής.

```
46  void LaserScanToPointCloud::scanCallback(const sensor_msgs::
       LaserScan::ConstPtr& scan_input)
47  {
48    sensor_msgs::PointCloud2 cloud;
49    try
50    {
51      m_projector.transformLaserScanToPointCloud(scan_input->header.
       frame_id, *scan_input, cloud, m_tfListener);
52    }
53    catch (tf::TransformException& e)
54    {
55      ROS_ERROR("%s",e.what());
56      return;
57    }
58
59    m_point_cloud_pub.publish(cloud);
60    global_cloud = cloud;
61  }
```

Στη συνάρτηση επανάκλησης, ορίζουμε τη μεταβλητή cloud που δέχεται pointclouds. Ύστερα, επιχειρούμε να μετατρέψουμε τις μετρήσεις του laser σε pointcloud. Αν αυτό δεν επιτευχθεί, τότε ενημερωνόμαστε με μήνυμα σφάλματος από το terminal. Το pointcloud (ή η κενή μεταβλητή) μεταφέρεται στο topic */cloud* και αποδίδεται επίσης στη μεταβλητή παγκόσμιας εμβέλειας.

```
63  void convertCloud(const sensor_msgs::PointCloud2 cloud_to_convert)
64  {
65    pcl::PCLPointCloud2 pcl_pc;
66    pcl_conversions::toPCL(cloud_to_convert, pcl_pc);
67    converted_cloud = pcl_pc;
68  }
```

Αυτή η συνάρτηση μετατρέπει το pointcloud σε μορφή εγγράψιμη σε αρχείο.

```
70  bool writePCL(const char* path, const pcl::PCLPointCloud2& p)
71  {
72    pcl::PointCloud<pcl::PointXYZI> cloud;
73    pcl::fromPCLPointCloud2(p, cloud);
74    if (cloud.isOrganized())
75    {
76      ROS_ERROR("Received point cloud is not contiguous!");
77      return false;
78    }
79
80    FILE *fp = fopen(path, "w");
81    if (fp != NULL)
82    {
83      uint32_t num_points = cloud.points.size();
84      fwrite(&num_points, sizeof(int), 1, fp);
85      for (size_t i=0; i<num_points; ++i)
86      {
87        auto& p = cloud.points[i];
88        const Eigen::Vector4d point(p.x, p.y, p.z, p.intensity);
89        fwrite(&point[0], 4*sizeof(double), 1, fp);
90      }
91      fclose(fp);
92      return true;
93    }
94    return false;
95  }
```

Εδώ γίνεται η εγγραφή στο αρχείο. Θέλουμε τέσσερις (4) ιδιότητες του pointcloud. Τις συντεταγμένες στο 3-διάστατο χώρο και την ένταση του σημείου. Η ένταση του σημείου δίνεται από το laser και εξαρτάται από την ανακλαστικότητα της επιφάνειας που συνάντησε. Ουσιαστικά, το αρχείο αποτελείται από πολλές γραμμές με τέσσερις (4) στήλες που αποτελούνται από αριθμούς.

```
98  void imageCallback(const sensor_msgs::ImageConstPtr& msg)
99  {
100   try
101   {
102     cv::imshow("OpenCV_view", cv_bridge::toCvShare(msg)->image);
103   }
104   catch (cv::Exception& cv_e)
105   {
106     ROS_ERROR("%s", cv_e.what());
107     return;
108   }
109
110   key_press = cv::waitKey(100);
111   if (key_press == "P" || key_press == "p")
112   {
```

Αυτή είναι η συνάρτηση επανάκλησης της κάμερας. Όταν λαμβάνεται νέα εικόνα, τότε αυτή εμφανίζεται σε ένα παράθυρο. Η αναμονή 100 ms απαιτείται ώστε η ροή των εικόνων να είναι συνεχής. Πρόκειται λοιπόν για ζωντανή αναμετάδοση της εικόνας στο παράθυρο. Εκμεταλλευόμενοι την απαιτούμενη καθυστέρηση, αναμένουμε και πάτημα πλήκτρου. Αν λοιπόν πατηθεί κάποιο από τα απαιτούμενα, τότε ξεκινάει η διαδικασία ονομασίας και αποθήκευσης των δεδομένων όπως φαίνεται παρακάτω:

```
113     // Create the image filename
114     sstream<<(counter)<<IM_TYPE;
115     std::string im_filename = sstream.str();
116     sstream.str("");
117
118     //Create the image save directory path
119     sstream<<HOME_DIR<<IM_DIRECTORY<<im_filename;
120     std::string im_save_dir = sstream.str();
121     sstream.str("");
122
123     // Create the scan filename
124     sstream<<(counter)<<SCAN_TYPE;
```

```
125       std::string scan_filename = sstream.str();
126       sstream.str("");
127
128       //Create the scan save directory path
129       sstream<<HOME_DIR<<SCAN_DIRECTORY<<scan_filename;
130       std::string scan_save_dir = sstream.str();
131       sstream.str("");
132       converted_scan_save_directory = scan_save_dir.c_str(); //
        Conversion to char*
133
134       cv::imwrite(im_save_dir, cv_bridge::toCvShare(msg)->image);
135
136       convertCloud(global_cloud);
137       writePCL(converted_scan_save_directory, converted_cloud);
138
139       ROS_INFO_STREAM("Saved image " + im_filename + " and scan " +
        scan_filename);
140       ROS_INFO_STREAM(MESSAGE);
141       counter++;
142     }
143     else if (key_press == "E" || key_press == "e")
144     {
145       ROS_INFO_STREAM("Exiting...");
146       cv::destroyAllWindows();
147       ros::shutdown();
148     }
149   }
```

Τα ονόματα και η διαδρομές αποθήκευσης δημιουργούνται ενώνοντας κομμάτια λέξεων που ορίσαμε στην αρχή. Για παράδειγμα, για την ονομασία των εικόνων ενώνουμε τον μετρητή με τη κατάληξη που υπάρχει. Ύστερα, δίνουμε το όνομα σε μία μεταβλητή και καθαρίζουμε την *sstream.str* . Με τον ίδιο τρόπο δημιουργούμε τη διαδρομή και ύστερα αποθηκεύουμε. Για τα pointclouds, χρειάζεται να εκτελεστούν και οι συναρτήσεις μετατροπής και εγγραφής που γράψαμε νωρίτερα.

Στο terminal εμφανίζεται και μήνυμα που ενημερώνει κάθε φορά που αποθηκεύονται δεδομένα. Τέλος, υπάρχει η συνθήκη για να κλείνουμε το πρόγραμμα δίνοντας κατάλληλη είσοδο.

```
151   void mySigintHandler(int sig)
152   {
153     cv::destroyAllWindows();
154     ros::shutdown();
```

```
155 }
156
157 int main(int argc, char** argv)
158 {
159   ros::init(argc, argv, "cl_data_acq_node", ros::init_options::
       NoSigintHandler);
160   ros::NodeHandle nh;
161
162   while (flag == true)
163   {
164     while ( (camera_image_selection!=1) && (camera_image_selection
      !=2) )
165     {
166       ROS_INFO_STREAM("Select 1 or 2");
167       ROS_INFO_STREAM("1) Raw camera image");
168       ROS_INFO_STREAM("2) Rectified camera image");
169       ROS_INFO_STREAM("Your selection: ");
170       std::cin>>camera_image_selection;
171       if ( std::cin.fail() )
172       {
173         std::cin.clear();
174         std::cin.ignore(256,'\n');
175       }
176     }
177     while ( (laser_selection!=1) && (laser_selection!=2) )
178     {
179       ROS_INFO_STREAM("Select 1 or 2");
180       ROS_INFO_STREAM("1) Base laser");
181       ROS_INFO_STREAM("2) Top laser");
182       ROS_INFO_STREAM("Your selection: ");
183       std::cin>>laser_selection;
184       if ( std::cin.fail() )
185       {
186         std::cin.clear();
187         std::cin.ignore(256,'\n');
188       }
189     }
190     flag = false;
191   }
```

Η συνάρτηση mySigintHandler() δεν χρησιμοποιείται αλλά έχει διατηρηθεί για πιθανή μελλοντική χρήση.

Ως γνωστόν, η εκτέλεση του προγράμματος ξεκινά από τη main(). Εδώ αρχικοποιούμε και εκκινούμε τον κόμβο με τις πρώτες δύο (2) εντολές. Στο επόμενο στάδιο, ο χρήστης πρέπει να επιλέξει είδος εικόνας και πιο laser θα χρησιμοποιηθεί.

```
194   ROS_INFO_STREAM("---------------------- Getting Parameters
       ------------------------");
195   if ( (camera_image_selection == 1) && (laser_selection == 1) )
196   {
197     loadConfigValue("/cl_data_acq_node/desired_camera_topic_rawP",
       desired_camera_topic, std::string("/kinect2/hd/image_color") );
198     loadConfigValue("/cl_data_acq_node/desire_laser_topic_bottomP",
       desired_laser_topic, std::string("/scan") );
199   }
200   else if ( (camera_image_selection == 1) && (laser_selection == 2)
       )
201   {
202     loadConfigValue("/cl_data_acq_node/desired_camera_topic_rawP",
       desired_camera_topic, std::string("/kinect2/hd/image_color") );
203     loadConfigValue("/cl_data_acq_node/desire_laser_topic_topP",
       desired_laser_topic, std::string("/top_scan") );
204   }
205   else if ( (camera_image_selection == 2) && (laser_selection == 1)
       )
206   {
207     loadConfigValue("/cl_data_acq_node/desired_camera_topic_rectP",
       desired_camera_topic, std::string("/kinect2/hd/image_color_rect
      ") );
208     loadConfigValue("/cl_data_acq_node/desire_laser_topic_bottomP",
       desired_laser_topic, std::string("/scan") );
209   }
210   else if ( (camera_image_selection == 2) && (laser_selection == 2)
       )
211   {
212     loadConfigValue("/cl_data_acq_node/desired_camera_topic_rectP",
       desired_camera_topic, std::string("/kinect2/hd/image_color_rect
      ") );
213     loadConfigValue("/cl_data_acq_node/desire_laser_topic_topP",
       desired_laser_topic, std::string("/top_scan") );
214   }
215   ROS_INFO_STREAM("---------------------- Done
       ------------------------");
```

Αναλόγα με την επιλογή που έγινε προηγουμένως, φορτώνεται η τιμή της παραμέτρου από τον parameter server κατά τα γνωστά.

```
218   ROS_INFO_STREAM(MESSAGE);
219   cv::namedWindow("OpenCV_view");
220
221   image_transport::ImageTransport it(nh);
```

```
222     image_transport::Subscriber sub = it.subscribe(
         desired_camera_topic, 1, imageCallback); //
223
224     LaserScanToPointCloud laserS2PCL;
225     laserS2PCL.ConfigFunction();
226
227     signal(SIGINT, mySigintHandler);
228
229     ros::spin();
230     return 0;
231   }
```

Σε αυτό το κομμάτι, εμφανίζεται το σύντομο μήνυμα με τις οδηγίες προς τον χρήστη και δημιουργείται το παράθυρο προβολής του βίντεο. Επίσης, γίνεται εγγραφή στο topic που φέρει την εικόνα της κάμερας, δημιουργείται αντικείμενο της κλάσης και καλείται και η συνάρτηση που ρυθμίζει τα topics για το laser.

6.6.4 Το αρχείο cl_visualise_node.cpp

Αυτός ο κόμβος είναι ο εγκέφαλος του πακέτου, καθώς εδώ εκτελείται η διαδικασία της συλλογής δεδομένων και της επαναταυτοποίησης των ανθρώπων. Η διαδικασία που ακολουθείται θα επεξηγηθεί παρακάτω, παράλληλα με τον κώδικα.

```
1   #include "camera_laser.h"
2   #include "loadRosConfig.h"
3   #include "camera_laser_tracking/cl_msgs.h"
4   #include <tf_conversions/tf_eigen.h>
5   #include <people_msgs/PositionMeasurementArray.h>
6   #include <time.h>
7   #include <fstream>
8   #include <algorithm>
```

Όπως πάντα, αρχικά δηλώνονται τα αρχεία κεφαλίδας που είναι απαραίτητα για τη λειτουργία του κόμβου. Παρατηρούμε ότι αυτή τη φορά δηλώνεται και το μήνυμα που δημιουργήθηκε για τις ανάγκες των δεδομένων εξόδου του πακέτου.

```
10  class LaserScanner
11  {
12    public:
13      void ConfigFunction();
14      void scanCallback(const sensor_msgs::LaserScan::ConstPtr&
       scan_input);
```

```
15      void projectImageCallback(const sensor_msgs::Image::ConstPtr&
        msg);
16      void cameraInfoCallback(const sensor_msgs::CameraInfo::ConstPtr
        & info_msg);
17      void peopleTrackCallback(const people_msgs::
        PositionMeasurementArray::ConstPtr& position_measurement);
18    private:
19      ros::NodeHandle m_laser_nh;
20      tf::TransformListener m_tfListener;
21      tf::StampedTransform m_transform_laser, m_transform_baselink;
22      //---------------- Publishers ----------------------
23      ros::Publisher m_point_cloud_pub;
24      ros::Publisher m_img2clmsg_pub;
25      ros::Publisher m_legs2clmsg_pub;
26      ros::Publisher m_data2clmsg_pub;
27
28      //---------------- Subscribers ----------------------
29      ros::Subscriber m_scan_sub;
30      ros::Subscriber m_project2image_sub;
31      ros::Subscriber m_camera_info_sub;
32      ros::Subscriber m_people_measurements_sub;
33
34      //---------------- Message Assignments ----------------------
35      sensor_msgs::LaserScan::ConstPtr m_laserscan;
36      people_msgs::PositionMeasurementArray::ConstPtr m_people_track;
37
38  };
```

Δημιουργούμε την κλάση *LaserScanner* που περιέχει συναρτήσεις επανάκλησης για τη λήψη δεδομένων από το laser, για τη προβολή των σημείων στην εικόνα, για την ανάγνωση των εσωτερικών παραμέτρων της κάμερας και για κάθε φορά που εντοπίζονται πόδια ανθρώπων. Στο ιδιωτικό μέρος της κλάσης, υπάρχουν μεταβλητές για τη μετατροπή μεταξύ των συστημάτων συντεταγμένων καθώς και publishers/subscribers. Τέλος, δηλώνονται οι μεταβλητές που θα περιέχουν τα δεδομένα από τα μηνύματα των εξωτερικών πακέτων.

```
40  //------------- Variables to assign values from Parameter Server
        --------------------
41  double rel_thresh, base_laser_diff, person_height, person_width,
        box_margin;
42  std::string camera_frame, laser_frame, base_frame, camera_topic,
        camera_info_topic, laser_topic;
43
44  bool show_debug_images, match;
```

```
45  bool reIDprocess = false;
46  int global_Pos;
47  int flag = 0;
48  std::string lastName;
49
50  std::string key_press;
51  image_geometry::PinholeCameraModel cam_model_;
```

Στο παγκόσμιο *(global)* μέρος του προγράμματος δηλώνονται αρχικά οι μεταβλητές που αργότερα θα λάβουν τιμές από τον parameter server με τη χρήση της γνωστής συνάρτησης. Επίσης δηλώνονται μεταβλητές που θα φανούν χρήσιμες κατά τις διαδικασίες που πλαισιώνουν την επαναταυτοποίηση αλλά και σε αυτή καθ' αυτή.

```
53  // Create vectors containing 100 vector<cv::Mat> (persons)
54  std::vector<std::vector<cv::Mat>> person_images( 100, std::vector<
       cv::Mat>() );
55  std::vector<std::vector<cv::Mat>> person_reprImage( 100, std::
       vector<cv::Mat>() ); // contains 1 random picture of each person
56  std::vector<std::vector<cv::MatND>> person_reprHist(100, std::
       vector<cv::MatND>() ); // contains the histogram of this picture
57  std::vector<std::vector<cv::MatND>> person_reprHistImg(100, std::
       vector<cv::MatND>() );
58  std::vector<std::vector<int>> IDsCorrespondance(100, std::vector<
       int>() );
59  cv::Mat tempHSV, tempHist, tempRGB, reprHistImg;
60  std::vector<int> personIDs; // Contains the IDs of tracked persons
61  std::vector<std::string> personNames; // Contains the names of
       tracked persons
62
63  // Arguments for calculating Histograms
64  int h_bins = 50; int s_bins = 60;
65  int histSize[] = { h_bins, s_bins };
66
67  float h_ranges[] = { 0, 180 };
68  float s_ranges[] = { 0, 256 };
69
70  const float* RANGES[] = { h_ranges, s_ranges };
71
72  int channels[] = { 0, 1 };
```

Αρχικά, πρέπει να εξηγηθεί η έννοια του ιστογράμματος *(histogram)*. Το ιστόγραμμα έχει 2 διαστάσεις. Στον άξονα x φαίνονται οι χρωματικές τιμές των pixels και στον άξονα y η συχνότητα αυτών, δηλαδή πόσα τέτοια pixels υπάρχουν. Όμως, μπορούμε να κατασκευάσουμε ιστογράμματα με διαφορετικές παραμέτρους όπως απόχρωση, κορε-

σμός, αξία (ή ελαφρότητα) *(Hue, Saturation, Value (or lightness))* . Αυτό είναι πιο βοηθητικό επειδή η απόχρωση περιλαμβάνει ένα εύρος των pixels, οπότε είναι πιο εύκολο να κατανοηθεί πιο είναι το επικρατέστερο χρώμα και η σύγκριση μεταξύ δύο ιστογραμμάτων ως προς την ομοιότητα γίνεται ταχύτερα και πιο αποτελεσματικά.

Κάθε εικόνα που λαμβάνουμε είναι ένα αντικείμενο *cv::Mat* από τη βιβλιοθήκη OpenCV. Εδώ δημιουργούμε διανύσματα τα οποία περιέχουν διανύσματα με τις εικόνες κάθε ατόμου, αλλά και με την αντιπροσωπευτική εικόνα κάθε ανθρώπου. Το ίδιο συμβαίνει και με τα ιστογράμματα των εικόνων. Επιπλέον, παρακολουθούμε πόσους ανθρώπους έχουμε εντοπίσει, τις ταυτότητες τους και τα ονόματα τους. Τέλος, δίνουμε τιμές για τη παραμετροποίηση των ιστογραμμάτων.

Κάθε φορά που εντοπίζονται πόδια ανθρώπου, αυτόματα ξεκινούν και αποθηκεύονται εικόνες του ατόμου (θα φανεί αργότερα με ποιο τρόπο γίνεται αυτό). Μία από αυτές επιλέγεται τυχαία ως αντιπροσωπευτική του ανθρώπου και υπολογίζεται το ιστόγραμμα αυτής και αποθηκεύεται. Τα ονόματα είναι της μορφής **Person1, Person2 ... Person10 κ.ο.κ** και δίνονται αυτόματα από το πακέτο εντοπισμού.

```
77  // For when a new person is tracked
78  bool newId = false;
79  std::vector<double> similarities;
80  std::stringstream personIDsStream;
81
82  // For naming and saving the collected data
83  int frameIdCounter = 1;
84  std::stringstream general_stream;
85  const std::string HOME_DIR = getenv("HOME");
86  const std::string IM_TYPE = ".png";
87  const std::string IM_DIRECTORY = "/Desktop/cl_data/
       detectedPersonImages/";
88  const std::string IM_HIST_DIRECTORY = "/Desktop/cl_data/
       images_histograms/";
89
90  Eigen::MatrixXd K1, P;
91  Eigen::Vector3d x;
```

Στη συνέχεια ορίζονται μεταβλητές που θα χρησιμεύσουν όταν εντοπίζεται ένα νέο άτομο και, όπως και στο προηγούμενο κόμβο, ετοιμάζουμε μεταβλητές για να αποθηκεύσουμε και να ονομάσουμε τα δεδομένα που συλλέγουμε. Στους πίνακες *K1* και *P* θα

αποθηκευθούν οι εσωτερικές παράμετροι της κάμερας για να εισαχθούν στον κώδικα.

```
99  void LaserScanner::ConfigFunction()
100 {
101   m_scan_sub = m_laser_nh.subscribe<sensor_msgs::LaserScan>(
        laser_topic, 100, &LaserScanner::scanCallback, this);
102   m_project2image_sub = m_laser_nh.subscribe<sensor_msgs::Image>(
        camera_topic, 100, &LaserScanner::projectImageCallback,this);
103   m_camera_info_sub = m_laser_nh.subscribe<sensor_msgs::CameraInfo
        >(camera_info_topic, 100, &LaserScanner::cameraInfoCallback,this
        );
104   m_people_measurements_sub = m_laser_nh.subscribe<people_msgs::
        PositionMeasurementArray>("/people_tracker_measurements", 100, &
        LaserScanner::peopleTrackCallback,this);
105   m_data2clmsg_pub = m_laser_nh.advertise<camera_laser_tracking::
        cl_msgs> ("/cl_msg_data", 100, false);
106   ROS_INFO_STREAM("[cl_visualise_node] Callbacks configured"<<std::
        endl);
107
108 }
```

Η συνάρτηση ρυθμίζει σε ποια topics στέλνουν πληροφορίες οι publishers και από ποια δέχονται πληροφορίες οι subscribers. Εμφανίζεται ένα μήνυμα στην οθόνη του terminal ως επιβεβαίωση.

```
112 void LaserScanner::scanCallback(const sensor_msgs::LaserScan::
      ConstPtr& scan_input)
113 {
114   m_laserscan = scan_input;
115 }
116
117 void LaserScanner::cameraInfoCallback(const sensor_msgs::CameraInfo
      ::ConstPtr& info_msg)
118 {
119   cam_model_.fromCameraInfo(info_msg);
120   K1 = Eigen::Map<const Eigen::Matrix<double, 3, 3, Eigen::RowMajor
        > >(&info_msg->K[0]);
121   P = Eigen::Map<const Eigen::Matrix<double, 3, 4, Eigen::RowMajor>
        >(&info_msg->P[0]);
122 }
123
124 void LaserScanner::peopleTrackCallback(const people_msgs::
      PositionMeasurementArray::ConstPtr& position_measurement)
125 {
126   m_people_track = position_measurement;
```

```
127 }
```

Οι συναρτήσεις επανάκλησης *scanCallback* και *peopleTrackCallback* απλώς εισάγουν στο πρόγραμμα τις πληροφορίες από τα topics. Η *cameraInfoCallback* εισάγει τις εσωτερικές παραμέτρους της κάμερας.

Ακολουθεί η συνάρτηση *projectImageCallback* όπου συμβαίνει όλη η διαδικασία μέχρι την εμφάνιση του επιθυμητού αποτελέσματος στην οθόνη. Το κεντρικό παράθυρο πρέπει να δείχνει τη ζωντανή εικόνα από τη κάμερα, τα σημεία του laser που έχουν προβληθεί σε αυτή, ένα ορθογώνιο πλαίσιο γύρω από τους ανθρώπους που εντοπίζονται και την ταυτότητα τους.

```
129 void LaserScanner::projectImageCallback(const sensor_msgs::Image::
        ConstPtr& msg)
130 {
131
132   camera_laser_tracking::cl_msgs clMsg;
133
134   if (!m_laserscan.get() || !m_people_track.get())
135     return;
136   try
137   {
138     m_tfListener.waitForTransform(camera_frame, laser_frame, ros::
        Time(0), ros::Duration(0.01));
139     m_tfListener.lookupTransform(camera_frame, laser_frame, ros::
        Time(0), m_transform_laser);
140
141
142     Eigen::Affine3d eigenTransform_laser;
143     tf::transformTFToEigen(m_transform_laser, eigenTransform_laser)
        ;
144
145     cv::Mat live_image = cv_bridge::toCvShare(msg)->image;
146
147     // Publish leg position data to custom message
148     sensor_msgs::ImagePtr im_msg = cv_bridge::CvImage(std_msgs::
        Header(), "bgr8", live_image).toImageMsg();
149     clMsg.cl_image = *im_msg;
150
151     clMsg.cl_detected_legs.header = m_people_track->header;
```

Αν δεν έχουν ληφθεί δεδομένα από το laser ή την κάμερα, τότε το πρόγραμμα αναμένει μέχρι να μην ισχύει αυτό. Για τις μετατροπές στο δέντρο tf, πρέπει να βεβαιωθούμε ότι η μετατροπή είναι εφικτή με τη μέθοδο *waitForTransform* και ύστερα τη πραγματοποιούμε με τη μέθοδο *lookupTransfrom*. Ο μετασχηματισμός αυτός αποθηκεύεται σε ένα διάνυσμα. Έπειτα, δημιουργούμε ένα αντικείμενο cv::Mat και του εκχωρούμε την ζωντανή εικόνα της κάμερας. Τέλος, δίνουμε στο ιδιοκατασκευασμένο μήνυμα τις συντεταγμένες των ποδιών που εντοπίστηκαν, καθώς και ολόκληρη την εικόνα όταν συνέβη αυτό.

```
155     if (show_debug_images)
156       cv::imshow("OpenCV_Rectified_Image", live_image);
157
158     cv::Mat projected = (cv_bridge::toCvShare(msg)->image).clone();
159     cv::Mat projectedWithText;
160     projected.copyTo(projectedWithText);
161
162     cv::Mat black_image(480, 640, CV_8U, cv::Scalar(0));
163     cv::Rect roi(100,150,100,150);
164     cv::Mat displayROI= black_image(roi);
```

Κατά την ανάπτυξη και την αποσφαλμάτωση του κώδικα, είναι χρήσιμη η προβολή εικόνων. Θέτωντας τιμή true κατά την εκτέλεση του launchfile, ενεργοποιούνται αυτές οι εικόνες. Σε αυτές τις εικόνες συμπεριλαμβάνονται όλες, επειδή κάποιες φορές έπρεπε να γίνουν άλλες δοκιμές όσο ο κόμβος αυτός ήταν ενεργός στο προσκήνιο.
Στη συνέχεια αντιγράφουμε τη ζωντανή εικόνα για να τοποθετήσουμε αργότερα τα σημεία του laser. Αντιγράφουμε επίσης την εικόνα ως έχει, χωρίς κείμενο, ώστε να μπορούμε να κάνουμε αλλαγές όταν επαναταυτοποιούνται άτομα. Η εικόνα *projectedWithText* είναι αυτή που θα προβάλλουμε στο τέλος. Για τη περίπτωση που δεν εντοπίζεται κανείς άνθρωπος, τότε στο παράθυρο που προβάλλουμε τα άτομα εμφανίζεται μία μαύρη εικόνα.

```
166     for (int e = 0; e < m_laserscan->ranges.size(); e++)
167     {
168       float r = m_laserscan->ranges[e];
169       if (r > m_laserscan->range_max || r < m_laserscan->range_min)
170       {
171         continue;
```

```
172         }
173         if (std::isnan(r))
174           continue;
175         float angle = m_laserscan->angle_min + e * m_laserscan->
        angle_increment;
176 
177         Eigen::Vector3d x(r*cos(angle), r*sin(angle), 0);
178         Eigen::Vector3d cameraLaserCoordinates = eigenTransform_laser
         * x;
179 
180         cameraLaserCoordinates = P.block < 3,3 > (0,0) *
        cameraLaserCoordinates; // Pass Projection Matrix in Coordinates
181 
182         if (std::abs(cameraLaserCoordinates(2)) < 1e-5)
183           continue;
184         cameraLaserCoordinates /= cameraLaserCoordinates(2);
185 
186         int ix = cameraLaserCoordinates(0);
187         int iy = cameraLaserCoordinates(1);
188         if (ix < 0 || ix >= projected.cols || iy < 0 || iy >=
        projected.rows)
189           continue;
190         cv::circle(projected, cv::Point(ix, iy), 3, cv::Scalar(255),
        -1);
191       }
192 
193       m_tfListener.waitForTransform(camera_frame, base_frame, ros::
        Time(0), ros::Duration(0.01));
194       m_tfListener.lookupTransform(camera_frame, base_frame, ros::
        Time(0), m_transform_baselink);
195 
196       Eigen::Affine3d eigenTransform_baselink;
197       tf::transformTFToEigen(m_transform_baselink,
        eigenTransform_baselink);
```

Για κάθε σημείο που έχουμε λάβει από το laser, ελέγχουμε αν είναι εντός αποδεκτού εύρους και αν η φύση της πληροφορίας είναι αποδεκτή. Αν όλα είναι εντάξει, τότε εισάγουμε το σημείο και το πολλαπλασιάζουμε με τις εσωτερικές παραμέτρους της κάμερας. Έτσι επιτυγχάνεται η μετατροπή και αφού βεβαιωθούμε ότι η θέση του σημείου είναι μέσα στα όρια της εικόνας, τότε τοποθετούμε ένα σημείο στις συντεταγμένες που λάβαμε από τη μετατροπή, με τη συνάρτηση *cv::circle*.

Με τον ίδιο τρόπο που πράξαμε νωρίτερα, παίρνουμε τη μετατροπή από τη βάση του ρομπότ προς τη κάμερα. Αυτό αργότερα θα εξυπηρετήσει τη σωστή τοποθέτηση του

πλαισίου εντοπισμού.

```
199      for (int i = 0; i < m_people_track->people.size(); i++)
200      {
201        if (m_people_track->people[i].reliability < rel_thresh )
202          continue;
203        std::string tracked_person_name = m_people_track->people[i].
       name;
204
205        Eigen::Vector3d trackedPersonPosition(m_people_track->people[
       i].pos.x, m_people_track->people[i].pos.y, 0.0);
206        Eigen::Vector3d v1(trackedPersonPosition.x(),
       trackedPersonPosition.y(), base_laser_diff); //for bottom point
207        Eigen::Vector3d v2(trackedPersonPosition.x(),
       trackedPersonPosition.y(), person_height); //for upper point
208
209
210        // ------------------- Calculate perpendicular vectors on
       personPositionVector -------------------------
211
212        Eigen::Vector3d trackedPersonPositionNormalized =
       trackedPersonPosition.normalized(); // trackedPersonPosition
       Length
213
214        Eigen::Vector3d perpPersonPositionVector(-
       trackedPersonPositionNormalized.y() ,
       trackedPersonPositionNormalized.x(), 0.0);
215
216        v1 += perpPersonPositionVector * person_width/2.0;
217        v2 -= perpPersonPositionVector * person_width/2.0;
218
219        std::vector<Eigen::Vector3d> perpVectors;
220        perpVectors.push_back(v1); //perpVectors[0]
221        perpVectors.push_back(v2); //perpVectors[1]
```

Τα παρακάτω συμβαίνουν κάθε φορά που ένας νέος εντοπισμός συμβαίνει. Πρώτα ελέγχουμε τον βαθμό αξιοπιστίας της μέτρησης και αν είναι αποδεκτός αποθηκεύουμε το όνομα του για μελλοντική χρήση. Ύστερα, φτιάχνουμε τα πλαίσια εντοπισμού ώστε να αλλάζουν οι διαστάσεις τους δυναμικά.

Η εικόνα 6.5.1 βοηθά στη κατανόηση των παρακάτω. Έστω το σημείο αναφοράς με συντεταγμένες (0,0) και σημείο P όπου έχει εντοπιστεί ένα άτομο. Το διάνυσμα W ορίζεται ως η διαφορά των συντεταγμένων του σημείου με τις συντεταγμένες του σημείου αναφοράς. Το επόμενο βήμα είναι η κανονικοποίηση *(normalization)* του διανύσματος.

Αυτό σημαίνει ότι το μήκος του διανύσματος θα είναι 1 . Υπολογίζεται από τη παρακάτω σχέση:

$$W' = \frac{W}{|W|} \tag{6.1}$$

Το διάνυσμα λοιπόν έχει τη μορφή:

$$\mathbf{W'} = \begin{bmatrix} W'_x & W'_y & 0 \end{bmatrix} \tag{6.2}$$

Το κάθετο διανύσμα έχει συντεταγμένες:

$$\mathbf{w} = \begin{bmatrix} -W'_y & W'_x & 0 \end{bmatrix} \tag{6.3}$$

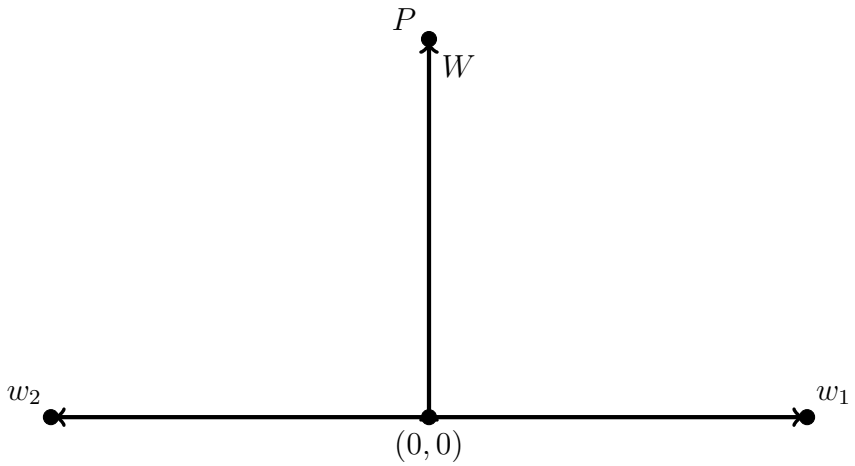
Συνεπώς, τα δύο (2) κάθετα διανύσματα υπολογίζονται από τις σχέσεις:

$$w_1 = w + Wa \tag{6.4}$$

$$w_2 = w - Wa \tag{6.5}$$

όπου:

$a =$ υποτιθέμενο πλάτος ατόμου



Εικόνα 6.6.1: Βοηθητική απεικόνιση υπολογισμού κάθετων διανυσμάτων

```
225       std::vector<Eigen::Vector3d> cameraBaselinkCoordinates;
226       // 0 index for bottom point, 1 index for upper point
227       for (int i = 0; i < 2; ++i)
228       {
229         cameraBaselinkCoordinates.push_back(eigenTransform_baselink
    * perpVectors[i]);
```

```
230         cameraBaselinkCoordinates[i] = P.block < 3,3 > (0,0) *
    cameraBaselinkCoordinates[i];
231         if (std::abs(cameraBaselinkCoordinates[i](2)) < 1e-5 )
232           continue;
233         cameraBaselinkCoordinates[i] /= cameraBaselinkCoordinates[i
    ](2);
234
235         cv::circle(projected, cv::Point(cameraBaselinkCoordinates[i
    ].x(), cameraBaselinkCoordinates[i].y() ), 3, cv::Scalar(0), -1)
    ;
236
237         if (cameraBaselinkCoordinates[i].x()<0)
    cameraBaselinkCoordinates[i].x()=0;
238         if (cameraBaselinkCoordinates[i].y()<0)
    cameraBaselinkCoordinates[i].y()=0;
239
240         if (cameraBaselinkCoordinates[i].x()>projected.cols-1)
    cameraBaselinkCoordinates[i].x()=projected.cols-1;
241         if (cameraBaselinkCoordinates[i].y()>projected.rows-1)
    cameraBaselinkCoordinates[i].y()=projected.rows-1;
242       }
243
244       float left_x = std::min(cameraBaselinkCoordinates[1].x(),
    cameraBaselinkCoordinates[0].x())-box_margin;
245       float right_x = std::max(cameraBaselinkCoordinates[1].x(),
    cameraBaselinkCoordinates[0].x())+box_margin;
246       float top_y = std::min(cameraBaselinkCoordinates[1].y(),
    cameraBaselinkCoordinates[0].y())-box_margin;
247       float bottom_y = std::max(cameraBaselinkCoordinates[1].y(),
    cameraBaselinkCoordinates[0].y())+box_margin;
248
249       // Publish bounding box information to custom message
250       sensor_msgs::RegionOfInterest box_msg;
251       box_msg.x_offset = left_x;
252       box_msg.y_offset = top_y;
253       box_msg.height = bottom_y - top_y;
254       box_msg.width = right_x - left_x;
255       clMsg.cl_bounding_box.push_back(box_msg);
256
257       clMsg.cl_detected_legs.people.push_back(m_people_track->
    people[i]);
```

Εδώ γίνεται η μετατροπή στις συντεταγμένες της κάμερας και υπολογίζουμε δύο (2) σημεία, ένα στο δάπεδο και άλλο ένα στο ύψος που υποθέτουμε ότι έχουν τα άτομα. Αν κάποια τετμημένη ή τεταγμένη είναι εκτός ορίων της εικόνας, τότε την τοποθετούμε αναγκαστικά στην αρχή των ορίων της. Σύμφωνα με αυτά τα δύο σημεία, υπολογίζουμε

τις συντεταγμένες τού πάνω αριστερά σημείου και τού κάτω δεξιά σημείου. Γνωρίζοντας αυτά τα νέα σημεία, μπορούμε να σχεδιάσουμε το πλαίσιο εντοπισμού. Επιπλέον, στέλνουμε πληροφορίες στο μήνυμα που έχουμε δημιουργήσει.

```
259         // Draw Bounding Box and write name of identified person
260         cv::rectangle(projected, cv::Point( left_x, top_y ), cv::
        Point(right_x, bottom_y), cv::Scalar(0), 3);
261
262         projected.copyTo(projectedWithText);
263         if (reIDprocess == false)
264         {
265             projected.copyTo(projectedWithText);
266             cv::putText( projectedWithText, tracked_person_name, cv::
        Point(left_x - 5, bottom_y + 25), CV_FONT_NORMAL, 1, cv::Scalar
        (0), 2, 8 );
267         }
268         else
269         {
270           // do nothing
271         }
272         roi = cv::Rect( left_x, top_y, right_x - left_x, bottom_y -
        top_y);
273
274         if (0 >= roi.x || 0 >= roi.width || (roi.x + roi.width >=
        projected.cols || 0 >= roi.y || 0 >= roi.height || (roi.y + roi.
        height) >= projected.rows ) )
275           continue;
276         displayROI = live_image(roi);
```

Σχεδιάζουμε λοιπόν το πλαίσιο γύρω από το άτομο και γράφουμε κάτω αριστερά την ταυτότητα του π.χ: Person1 . Επίσης, αποκόπτουμε την Περιοχή Ενδιαφέροντος *(Region of Interest - ROI)* και την προβάλλουμε σε ένα ξεχωριστό παράθυρο.

```
285         // --- create the number by picking up each digit
286         int numberPosStart = 6;
287         int name_length = tracked_person_name.length();
288         char charId = tracked_person_name[numberPosStart];
289         int personIdNumber = charId - '0';
290
291         if (name_length > 7)
292         {
293           int numberPosEnd = name_length - 1;
294           std::stringstream stream;
295           std::string stringId;
```

```
296           for (int k = numberPosStart; k <= numberPosEnd; k++)
297           {
298             stream<<tracked_person_name[k];
299             stringId = stream.str();
300           }
301           stream.str("");
302           personIdNumber = std::stoi( stringId );
303         }
```

Δεδομένου ότι τα ονόματα είναι της μορφής *PersonX* και η αρίθμηση ξεκινά από το μηδέν (0) στη C++, ξέρουμε ότι ο αριθμός είναι στην έκτη (6η) θέση του ονόματος. Αν όμως έχουμε υπερβεί τα 10 άτομα, τότε το μέγεθος γίνεται μεγαλύτερο του εφτά (7). Οπότε βάζουμε σε σειρά κάθε χαρακτήρα από τον 6ο και μετά για να αποκομίσουμε τον αριθμό του ατόμου.

```
306         if( std::find(personIDs.begin(), personIDs.end(),
    personIdNumber) != personIDs.end() )
307         {
308           // personIDs contains personIdNumber, do nothing for now
309         }
310         else
311         {
312           newId = true;
313           personIDs.push_back(personIdNumber);
314           personNames.push_back(tracked_person_name); // Save ID -
    name correspondance
315           ROS_INFO_STREAM("Persons tracked: "<<personIDs.size());
316           personIDsStream<<personIdNumber<<" ";
317           ROS_INFO_STREAM("IDs: "<<personIDsStream.str());
318
319         }
```

Ελέγχουμε αν πρόκειται για άτομο που έχει ήδη εντοπιστεί. Αν όχι, συμπληρώνουμε τα διανύσματα με τα στοιχεία του ατόμου και εμφανίζουμε στην οθόνη το σύνολο αυτών για να έχουμε αντίληψη των εξελίξεων έως τώρα.

```
321         general_stream<<HOME_DIR<<IM_DIRECTORY<<(frameIdCounter)<<
    IM_TYPE;
322         std::string save_path = general_stream.str();
323         general_stream.str("");
324         cv::imwrite(save_path, displayROI);
325         if (frameIdCounter == 1)
326         {
```

```
327         ROS_INFO_STREAM("\033[32m"<<"Saving images to: "<<HOME_DIR
    <<IM_DIRECTORY<<"\033[0m");
328         ROS_INFO_STREAM("\033[32m"<<"Saving representative data to:
     "<<HOME_DIR<<IM_HIST_DIRECTORY<<"\033[0m"<<std::endl);
329       }
330       frameIdCounter++;
```

Εδώ αποθηκεύονται τα δεδομένα δημιουργώντας τα ονόματα και τις διευθύνσεις με τον συνηθισμένο τρόπο. Ο πρώτος φάκελος θα περιέχει τις εικόνες όλων των ατόμων. Στον δεύτερο θα υπάρχουν οι αντιπροσωπευτικές εικόνες όλων των ατόμων και τα αντίστοιχα ιστογράμματα.

```
336       if ( (person_images[personIdNumber].empty() == true) && (
    person_reprImage[personIdNumber].empty() == true ) )
337       {
338         person_images[personIdNumber].push_back(displayROI.clone())
    ;
339         ROS_INFO_STREAM("Captured image for person with ID"<<" "<<
    personIdNumber);
340       }
341       if ( person_images[personIdNumber].empty() == false )
342       {
343         for (std::vector<int>::iterator k = personIDs.begin(); k !=
     personIDs.end(); k++) // Iterate over the IDs (1st way to do it
    )
344         {
345           if ( person_reprImage[*k].empty() ) // check if final
    dataset of this person is empty
346           {
347
348             person_reprImage[*k].push_back(person_images[*k][0]);
    // ...assign it to the final Image dataset
349             cv::cvtColor(person_reprImage[*k][0], tempRGB,
    CV_BGR2RGB);
350
351             ROS_INFO_STREAM("Calculating and saving Histogram...");
352             cv::calcHist( &tempRGB, 1, channels, cv::Mat(),
    tempHist, 2, histSize, RANGES, true, false ); // Calculate
    Histogram
353             cv::normalize( tempHist, tempHist, 0, 1, 32, -1, cv::
    Mat() );
354             double maxVal=0;
355             minMaxLoc(tempHist, 0, &maxVal, 0, 0);
356
```

```
357             person_reprHist[*k].push_back(tempHist.clone()); //
    Assign it to the final Histogram dataset
```

Όταν δεν έχουμε κρατήσει στη μνήμη αντιπροσωπευτικές φωτογραφίες του ατόμου που εντοπίζεται, αποθηκεύεται μία τυχαία και υπολογίζουμε το ιστόγραμμα της, το οποίο επίσης αποθηκεύεται σε αντίστοιχο διάνυσμα.

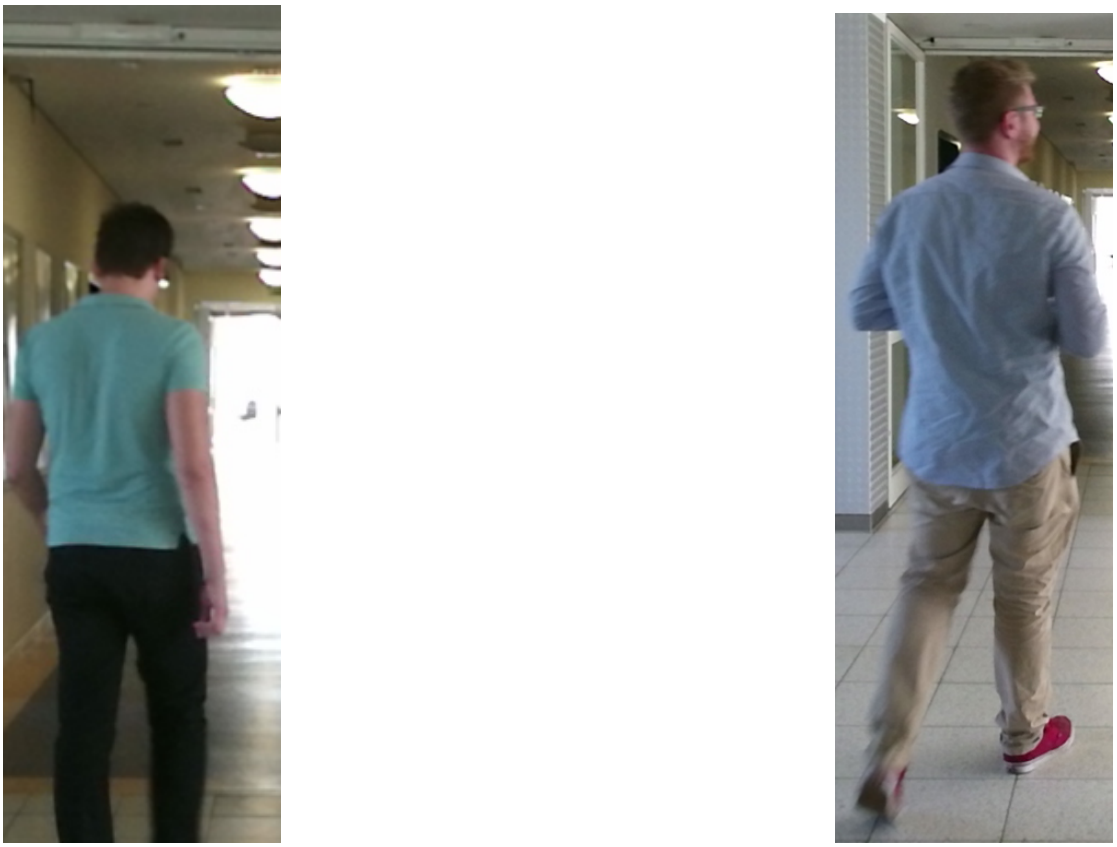
```
360             int scale = 5;
361             reprHistImg = cv::Mat::zeros(s_bins*scale, h_bins*scale
    , CV_8UC3) ;
362
363             for( int h = 0; h < h_bins; h++ )
364             {
365               for( int s = 0; s < s_bins; s++ )
366               {
367                 float binVal = person_reprHist[*k][0].at<float>(h,
    s);
368                 int intensity = cvRound(binVal*255/maxVal);
369                 cv::rectangle( reprHistImg, cv::Point(h*scale, s*
    scale), cv::Point( (h+1)*scale - 1, (s+1)*scale - 1), cv::Scalar
    ::all(intensity), CV_FILLED );
370               }
371             }
372             person_reprHistImg[*k].push_back(reprHistImg.clone()) ;
```

Εδώ οπτικοποιείται το ιστόγραμμα και μεγενθύνεται για να είναι πιο ευανάγνωστο.

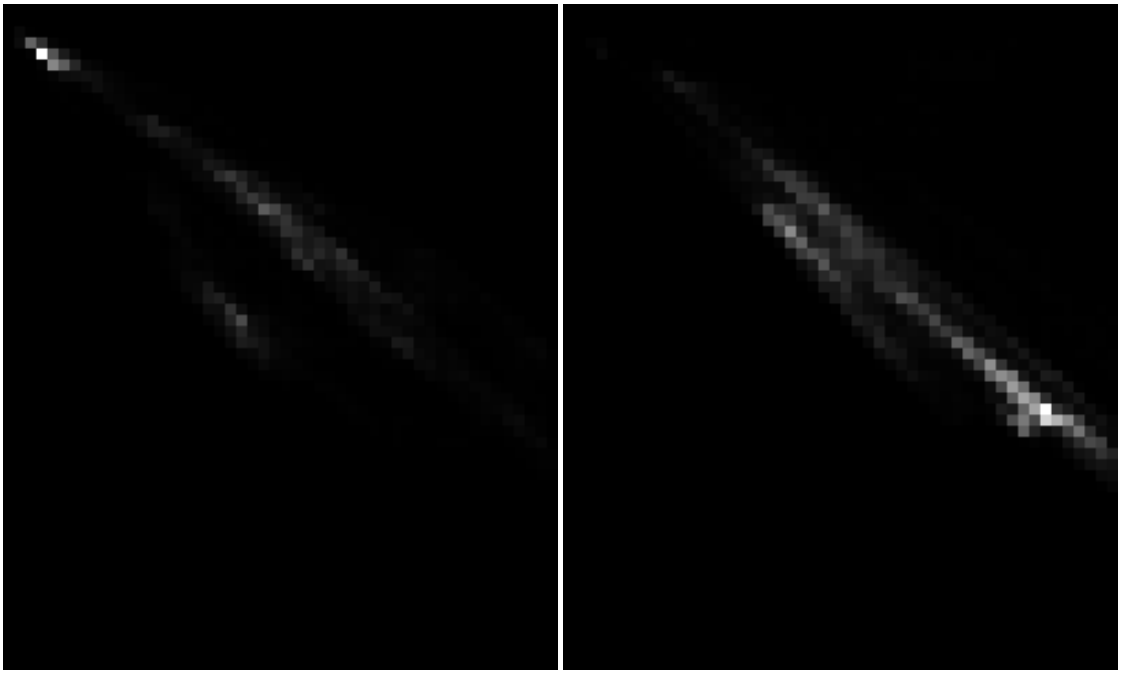
```
375             if (show_debug_images)
376             {
377               std::string ri, h;
378
379               std::stringstream windowNaming;
380               windowNaming<<"Representative_"<<*k;
381               ri = windowNaming.str();
382               windowNaming.str("");
383               windowNaming<<"Histogram_"<<*k;
384               h = windowNaming.str();
385               windowNaming.str("");
386
387               cv::imshow("Representative", person_reprImage[*k][0])
    ;
388               cv::imshow("Histogram", person_reprHistImg[*k][0]);
389
390               general_stream<<HOME_DIR<<IM_HIST_DIRECTORY<<ri<<
    IM_TYPE;
```

```
391                 save_path = general_stream.str();
392                 cv::imwrite(save_path, person_reprImage[*k][0]);
393                 general_stream.str("");
394 
395                 general_stream<<HOME_DIR<<IM_HIST_DIRECTORY<<h<<
        IM_TYPE;
396                 save_path = general_stream.str();
397                 cv::imwrite(save_path, person_reprHistImg[*k][0]);
398                 general_stream.str("");
399 
400               }
401 
402               ROS_INFO_STREAM("Done"<<std::endl<<" ");
403             }
404           }
```

Οι φωτογραφίες και τα ιστογράμματα προβάλλονται σε παράθυρα. Κατασκευάζουμε πρώτα τα ονόματα που θα έχουν τα αρχεία και τα αποθηκεύουμε στους προαναφερόμενους φακέλους. Παρακάτω ακολουθούν ενδεικτικά παραδείγματα.



Εικόνα 6.6.2: Αντιπροσωπευτικές εικόνες ατόμων



Εικόνα 6.6.3: Ιστογράμματα αντιπροσωπευτικών εικόνων

```
408         if (newId == true)
409         {
410           //Histogram comparison
411           for (auto p : personIDs) // Iterate over the IDs (2nd way
       to do it)
412           {
413             if (p == personIdNumber)
414               continue;
415             // 0 --> Correlation method
416             // 1 --> Chi-Square method
417             // 2 --> Intersection method
418             // 3 --> Bhattacharyya Distance method (Hellinger
      distance)
419             double similarity_check = compareHist( person_reprHist[
      p][0], person_reprHist[personIdNumber][0], 0 );
420             similarities.push_back( similarity_check );
421             ROS_INFO_STREAM("Similarity between person "<<p<<" and
      person "<<personIdNumber<<" is: "<<similarities[similarities.
      size()-1]);
422           }
423           std::cout<<" "<<std::endl;
```

Αν εντοπίστεί άτομο για πρώτη φορά (1η), τότε συγκρίνουμε το ιστόγραμμα του με όλα τα υπόλοιπα, ως προς την ομοιότητα τους. Τα αποτελέσματα εμφανίζονται στην οθόνη.

```
427           std::vector<double>::iterator max;
```

```
428         max = std::max_element(similarities.begin(), similarities
    .end());
429         std::cout << "max element at: " << std::distance(
    similarities.begin(), max) << '\n';
430         int maxPos = std::distance(similarities.begin(), max);
431         int Pos = -1;
432         if (personIDs.size()>1)
433         {
434           if ( similarities[maxPos] >= 0.5 )
435           {
436             reIDprocess = true;
437             ROS_INFO_STREAM("Re-identified person "<<
    personIdNumber<<" as person "<<personIDs[maxPos]);
438             projected.copyTo(projectedWithText);
439             cv::putText( projectedWithText, personNames[maxPos],
    cv::Point(left_x - 5, bottom_y + 25), CV_FONT_NORMAL, 1, cv::
    Scalar(0), 2, 8 );
440             IDsCorrespondance[personIDs[maxPos]].push_back(
    personIdNumber );
441             match = false;
442             for (auto p : personIDs)
443             {
444               Pos++;
445               global_Pos = Pos;
446               if(std::find(IDsCorrespondance[p].begin(),
    IDsCorrespondance[p].end(), personIDs[maxPos]) !=
    IDsCorrespondance[p].end())
447                 {
448                   ROS_INFO_STREAM("Re-identified person "<<
    personIDs[maxPos]<<" as person "<<p);
449                   match = true;
450                   break;
451                 }
452             }
453             if (match == false)
454             {
455               Pos = maxPos;
456               global_Pos = Pos;
457             }
458             std::cout<<"person name "<<personNames[Pos]<<std::
    endl;
459
460             projected.copyTo(projectedWithText);
461             cv::putText( projectedWithText, personNames[Pos], cv
    ::Point(left_x - 5, bottom_y + 25), CV_FONT_NORMAL, 1, cv::
    Scalar(0), 2, 8 );
462           }
463         }
```

```
464
465             newId = false;
466             similarities.clear();
467           }
468         }
469       }
470
471       cv::imshow("Projected_Points", projectedWithText);
472       if (show_debug_images)
473       {
474         cv::imshow("ROI", displayROI);
475       }
476       m_data2clmsg_pub.publish(clMsg);
477
478
479     }
```

Βρίσκουμε σε ποιο άτομο αντιστοιχεί η μεγαλύτερη ομοιότητα και ελέγχουμε αν αυτή είναι μεγαλύτερη του 0.5 . Τότε λέμε ότι το άτομο αυτό στη πραγματικότητα έχει παρουσιαστεί στο παρελθόν. Γράφουμε λοιπόν το όνομα του στην καθαρή εικόνα και την αντιγράφουμε ώστε να εμφανιστεί το νέο κείμενο. Τέλος, στέλνουμε τις πληροφορίες στο μήνυμα.

```
485
486     catch (cv::Exception& cv_e)
487     {
488       ROS_ERROR("[cl_visualise_node] %s", cv_e.what());
489       return;
490     }
491
492     key_press = cv::waitKey(10);
493
494     if (key_press == "E" || key_press == "e" || key_press == "Q" ||
        key_press == "q")
495     {
496       ROS_INFO_STREAM("[cl_visualise_node] Exiting..." << std::endl;)
        ;
497       cv::destroyAllWindows();
498       ros::shutdown();
499     }
500   }
```

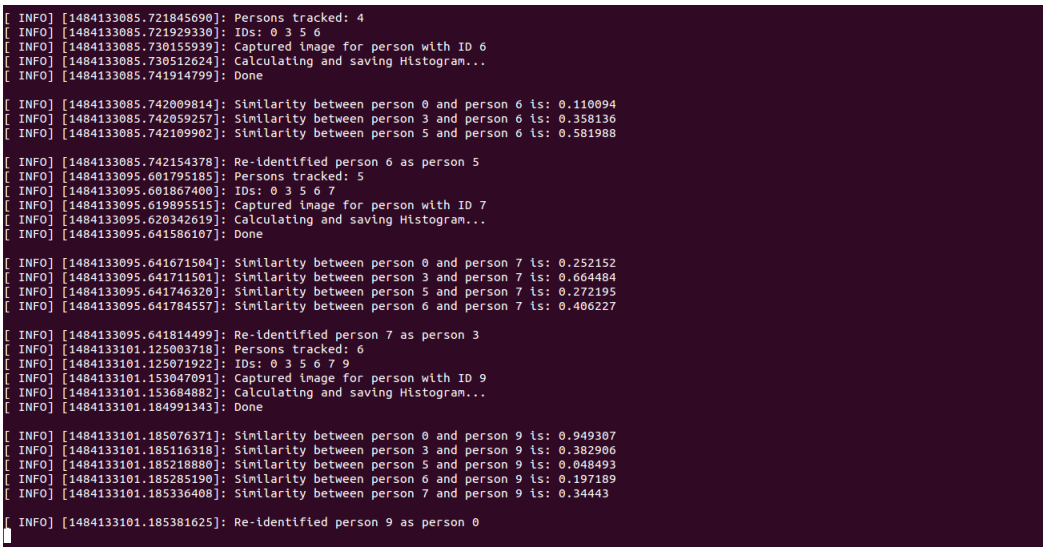
Εδώ έρχεται το πρόγραμμα αν κάτι δεν πάει καλά κατά την επικοινωνία με τους αισθητήρες. Επιπλέον, ο κόμβος μπορεί να κλείσει ανά πάσα στιγμή εισάγοντας λατινικό E

ή Q, είτε κεφαλαίο είτε πεζό.

```
502 int main(int argc, char** argv)
503 {
504   ros::init(argc, argv, "cl_visualise_node");
505   ROS_INFO_STREAM("[cl_visualise_node] cl_visualise_node
       initialised");
506   srand (time(NULL));
507
508   ROS_INFO_STREAM("---------------------- Getting Parameters
       ------------------------");
509   loadConfigValue("/cl_visualise_node/rel_threshP", rel_thresh,
       0.7);
510   loadConfigValue("/cl_visualise_node/base_laser_diffP",
       base_laser_diff, -0.2);
511   loadConfigValue("/cl_visualise_node/person_heightP",
       person_height, 1.8);
512   loadConfigValue("/cl_visualise_node/person_widthP", person_width,
        0.7);
513   loadConfigValue("/cl_visualise_node/box_marginP", box_margin,
       0.0);
514   loadConfigValue("/cl_visualise_node/camera_frameP", camera_frame,
        std::string("kinect2_optical_tracking") );
515   loadConfigValue("/cl_visualise_node/laser_frameP", laser_frame,
       std::string("laser") );
516   loadConfigValue("/cl_visualise_node/base_frameP", base_frame, std
       ::string("base_link") );
517   loadConfigValue("/cl_visualise_node/camera_topicP", camera_topic,
        std::string("/kinect2/hd/image_color_rect") );
518   loadConfigValue("/cl_visualise_node/camera_info_topicP",
       camera_info_topic, std::string("/kinect2/hd/camera_info") );
519   loadConfigValue("/cl_visualise_node/laser_topicP", laser_topic,
       std::string("/scan") );
520   loadConfigValue("/cl_visualise_node/show_debug_imagesP",
       show_debug_images, true );
521   ROS_INFO_STREAM("---------------------- Done
       ------------------------");
522
523   cv::namedWindow("Projected_Points", CV_WINDOW_NORMAL);
524   cv::moveWindow("Projected_Points", 710, 120);
525   cv::resizeWindow("Projected_Points", 1366, 768);
526   if (show_debug_images)
527   {
528
529     cv::namedWindow("ROI", CV_WINDOW_AUTOSIZE);
530     cv::namedWindow("OpenCV_Rectified_Image", CV_WINDOW_NORMAL);
531     cv::resizeWindow("OpenCV_Rectified_Image", 1366, 768);
```

```
532
533       cv::moveWindow("OpenCV_Rectified_Image", 20, 120);
534       cv::moveWindow("ROI", 20, 520);
535
536       cv::namedWindow("Representative", CV_WINDOW_AUTOSIZE);
537       cv::namedWindow("Histogram", CV_WINDOW_AUTOSIZE);
538     }
539
540     LaserScanner LS;
541     LS.ConfigFunction();
542
543     ros::spin();
544     return 0;
545   }
```

Στην αρχή του προγράμματος αρχικοποιούμε και εκκινούμε τον κόμβο. Επίσης, παίρνουμε τιμές από τον parameter server και αρχικοποιούνται τα παράθυρα που θέλουμε να χρησιμοποιήσουμε στο μέλλον. Τέλος, δημιουργείται ένα αντικείμενο της κλάσης LaserScanner και καλείται η συνάρτηση-μέλος που ρυθμίζει όλα τα topics.



Εικόνα 6.6.4: Ενδεικτικά αποτελέσματα στην οθόνη του terminal

ΣΥΜΠΕΡΑΣΜΑΤΑ

Όπως παρατηρείται από τα πειράματα που διενεργήθηκαν, η μέθοδος είναι αρκετά αποτελεσματική. Εντοπίζονται όλα τα άτομα που εισέρχονται στο οπτικό πεδίο και η επαναταυτοποίηση τους είναι επιτυχής σε μεγάλο βαθμό.

Ωστόσο, αποδεικνύεται ότι είναι εύκολο να συγχέονται ταυτότητες ατόμων που έχουν ίδια ή παρόμοια χρώματα στον ρουχισμό τους. Καθώς τα ιστογράμματα περιγράφουν τα χρώματα που υπάρχουν στην εικόνα, η ομοιότητα είναι μεγάλη κατά τη σύγκριση.

Ένας άλλος παράγοντας ευαισθησίας είναι ο φωτισμός του χώρου. Εάν ξαφνικά μεταβληθεί, αλλάζουν οι μετρήσεις που δέχεται η κάμερα για τα ίδια άτομα. Σε ελαφρώς σκοτεινές συνθήκες, το σύστημα δεν αποδίδει ικανοποιητικά αποτελέσματα, καθώς όλα τα χρώματα λαμβάνονται σε σκούρες αποχρώσεις, που είναι κοντινές σε τιμές RGB.

ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Οι επιδόσεις του συστήματος μπορούν να βελτιωθούν με τους τρόπους που αναφέρονται παρακάτω. Για να αυξηθεί η αξιοπιστία του, σχεδιάζεται η χρήση δεδομένων βάθους *(depth data)*, ώστε να μπορούμε να αποκόψουμε από την εικόνα αποκλειστικά και μόνο το άτομο. Αυτό σημαίνει ότι δεν θα υπάρχουν τα χρώματα του φόντου που προκαλούν διαταραχές στη μέθοδο σύγκρισης που έχει επιλεχθεί.

Οι ψευδώς-θετικές μετρήσεις μπορούν να περιοριστούν αν χρησιμοποιηθεί παράλληλα ένα πακέτο εντοπισμού κεφαλιών και προσώπου. Η ιδέα είναι ότι αν εντοπίζονται πόδια ανθρώπων αλλά από πάνω δεν υπάρχει κεφάλι ή πρόσωπο, τότε οι μετρήσεις απορρίπτονται. Αυτό μπορεί να δουλέψει και αντιστρόφως βεβαίως. Το παραπάνω μπορεί επίσης να προσφέρει μία δικλείδα για την ταυτοποίηση, καθώς μπορεί να υπάρχει υψηλή ομοιότητα στα ιστογράμματα, αλλά τα πρόσωπα να είναι διαφορετικά.

Μερικές φορές, ψευδώς-θετικές μετρήσεις υπάρχουν και σε περιπτώσεις όπου αλληλοκαλύπτονται τα πλαίσια εντοπισμού ανθρώπων που βρίσκονται σε πολύ κοντινή απόσταση. Είναι πιο ασφαλές λοιπόν να απορρίπτονται αυτές οι μετρήσεις. Μία επιπλέον βελτίωση θα ήταν να ελέγχεται η γεωγραφική κατανομή των χρωμάτων των pixels ή να υπολογίζεται το επικρατέστερο χρώμα στην εικόνα. Έτσι, κάθε άτομο έχει μία πληθώρα χαρακτηριστικών που το περιγράφουν και η διασταύρωσή τους κάνει τον εντοπισμό πιο εύρωστο.

Λιγότερο σημαντικές (και εύκολες) βελτιώσεις μπορούν να γίνουν στον κώδικα, όπου θα ελέγχεται η ύπαρξη των φακέλων αποθήκευσης και θα δημιουργούνται αυτόματα εάν αυτό δεν ισχύει.

Τέλος, η εφαρμογή μπορεί να πάει ένα βήμα παραπέρα δίνοντας τη δυνατότητα στο ρομπότ να ακολουθεί κάποιο συγκεκριμένο άτομο ενώ παράλληλα εκτελεί τις γνωστές διεργασίες.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Aaron Martinez, Enrique Fernández, *"Learning ROS for Robotics Programming"*, Packt Publishing Ltd., Birmingham UK, Σεπτέμβριος 2013

[2] R. Patrick Goebel, *"ROS by Example Vol. 1"*, Αυτοέκδοση, Ιανουάριος 2014

[3] Edwin Olson, "AprilTag: A robust and flexible visual fiducial system", *Proceedings of the IEEE International Conference on Robotics and Automation ICRA*, σελ. 3400-3407, Μάιος 2011

[4] Kai O. Arras, Oscar Martinez Mozos, Wolfram Burgard, "Using Boosted Features for the Detection of People in 2D Range Data", *Proceedings of the IEEE International Conference on Robotics and Automation*, σελ. 3402-3407, 2007

[5] Tully Foote, "tf: The transform library", *Technologies for Practical Robot Applications (TePRA), IEEE International Conference on Open-Source Software workshop*, σελ. 1-6, Απρίλιος 2013

[6] Frank Neuhaus, Stephan Manthe, Dietrich Paulus, "Practical Calibration of Actuated Multi-DOF Camera Systems", *Ανέκδοτη εργασία*

[7] Yoav Freund and Robert E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting", 1997

[8] Raphael Memmesheimer, Viktor Seib, Gregor Heuer, Patrik Schmidt, Darius Thies, Ivanna Mykhalchyshyna, Johannes Klöckner, Martin Schmitz, Niklas Yann Wettengel, Nils Geilen, Richard Schütz, Florian Polster, Dietrich Paulus, "RoboCup 2016 - homer@UniKoblenz (Germany)", *Robocup 2016 @home League Team Description Papers*, 2016

[9] http://www.hizook.com/files/publications/SICK_LMS100.pdf, προσπέλαση στις 25.10.2016

[10] http://www.rapidtables.com/web/color/RGB_Color.htm, προσπέλαση στις 25.10.2016

[11] http://wiki.ros.org/indigo, προσπέλαση στις 30.10.2016

[12] http://wiki.ros.org/catkin/conceptual_overview, προσπέλαση στις 30.10.2016

[13] http://wiki.ros.org/catkin/CMakeLists.txt, προσπέλαση στις 30.10.2016

[14] http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes, προσπέλαση στις 30.10.2016

[15] http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics, προσπέλαση στις 30.10.2016

[16] http://wiki.ros.org/Parameter% 20Server, προσπέλαση στις 30.10.2016

[17] http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv, προσπέλαση στις 30.10.2016

[18] http://wiki.ros.org/ROS/Tutorials/Recording%20and%20playing%20back%20data, προσπέλαση στις 30.10.2016

[19] http://wiki.ros.org/ROS/CommandLineTools, προσπέλαση στις 30.10.2016

[20] http://wiki.ros.org/rosmake, προσπέλαση στις 30.10.2016

[21] http://wiki.ros.org/rospack, προσπέλαση στις 30.10.2016

[22] http://wiki.ros.org/rosstack, προσπέλαση στις 30.10.2016

[23] http://opencv.org/, προσπέλαση στις 06.11.2016

[24] http://docs.opencv.org/3.1.0/de/d7a/tutorial_table_of_content_core.html, προσπέλαση στις 06.11.2016

[25] http://docs.opencv.org/3.1.0/d7/da8/tutorial_table_of_content_imgproc.html, προσπέλαση στις 06.11.2016

[26] http://docs.opencv.org/3.1.0/d0/de2/tutorial_table_of_content_highgui.html, προσπέλαση στις 06.11.2016

[27] http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html, προσπέλαση στις 15.11.2016

[28] http://docs.opencv.org/3.1.0/dc/dbb/tutorial_py_calibration.html, προσπέλαση στις 15.11.2016

[29] https://en.wikipedia.org/wiki/Focal_length, προσπέλαση στις 18.11.2016

[30] https://en.wikipedia.org/wiki/Focus_(optics), προσπέλαση στις 18.11.2016

[31] http://www.cambridgeincolour.com/forums/thread1371.htm, προσπέλαση στις 18.11.2016

[32] http://ksimek.github.io/2013/08/13/intrinsic/, προσπέλαση στις 20.11.2016

[33] http://stackoverflow.com/questions/29095349/how-is-the-reprojection-error-calculated-in-matlabs-triangulate-function-sadly, προσπέλαση στις 20.11.2016

[34] https://april.eecs.umich.edu/wiki/AprilTags, προσπέλαση στις 13.12.2016

[35] Thiemo Wiedemeyer, *IAI Kinect2*, https://github.com/code-iai/iai_kinect2, προσπέλαση στις 18.12.2016

[36] https://github.com/code-iai/iai_kinect2/tree/master/kinect2_calibration, προσπέλαση στις 18.12.2016

[37] https://github.com/code-iai/iai_kinect2/tree/master/kinect2_registration, προσπέλαση στις 18.12.2016

[38] https://github.com/code-iai/iai_kinect2/tree/master/kinect2_bridge, προσπέλαση στις 18.12.2016

[39] https://github.com/code-iai/iai_kinect2/tree/master/kinect2_viewer, προσπέλαση στις 18.12.2016

[40] http://wiki.ros.org/leg_detector?distro=indigo, προσπέλαση στις 15.12.2016

[41] http://wiki.ros.org/people?distro=indigo, προσπέλαση στις 18.12.2016

[42] http://wiki.ros.org/CppStyleGuide, προσπέλαση στις 8.01.2017

[43] http://docs.opencv.org/2.4/modules/imgproc/doc/histograms.html?highlight=calchist#calchist, προσπέλαση στις 19.09.2016

[44] http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_calculation/histogram_calculation.html, προσπέλαση στις 15.09.2016

[45] http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_comparison/histogram_comparison.html, προσπέλαση στις 06.09.2016

[46] https://en.wikipedia.org/wiki/Artificial_intelligence, προσπέλαση στις 11.01.2017

[47] https://en.wikipedia.org/wiki/Computer_vision, προσπέλαση στις 11.01.2017

[48] https://en.wikipedia.org/wiki/Statistical_classification, προσπέλαση στις 03.02.2017