ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΪΑΤΡΙΚΗ

Αναζήτηση Στόχων και Πλοήγηση Ρομπότ

Σιγάλας Ευάγγελος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υπεύθυνος«
Πλαγιανάκος Βασίλειος
Επίκουρος Καθηγητής

Λαμία, 2013

Περιεχόμενα

1 Εισαγωγή **1**
- 1.1 Τι είναι η ψηφιακή επεξεργασία εικόνας . . . 1
- 1.2 Προέλευση της ψηφιακής επεξεργασίας εικόνας . . . 2
- 1.3 Εφαρμογές της ψηφιακής επεξεργασίας εικόνας . . . 3
  - 1.3.1 Εφαρμογές στην απεικόνιση με ακτίνες γ . . . 4
  - 1.3.2 Εφαρμογές στην απεικόνιση με ακτίνες X . . . 4
  - 1.3.3 Εφαρμογές στην απεικόνιση στις ζώνες του ορατού και του υπέρυθρου . . . 4
  - 1.3.4 Εφαρμογές σε άλλες μορφές απεικόνισης . . . 5

2 OpenTLD **6**
- 2.1 Περιγραφή . . . 6
- 2.2 Ανίχνευση (Tracking) . . . 8
  - 2.2.1 Εκτίμηση Οπτικής Ροής . . . 8
  - 2.2.2 Μετρήσεις Λάθους . . . 9
  - 2.2.3 Μοντέλο Μετασχηματισμού . . . 10
  - 2.2.4 Συμπεράσματα . . . 10
- 2.3 Εντοπισμός (Detection) . . . 10
  - 2.3.1 Μέθοδος ολισθαίνοντος παραθύρου . . . 11
  - 2.3.2 Μέθοδος αφαίρεσης υπόβαθρου . . . 11
  - 2.3.3 Φίλτρο διακύμανσης . . . 13
  - 2.3.4 Ταξινομητής . . . 14
  - 2.3.5 Σύγκριση περιγράμματος . . . 14
- 2.4 Εκμάθηση (Learning) . . . 15
  - 2.4.1 Συγχώνευση και Εγκυρότητα . . . 15
  - 2.4.2 P/N-learning . . . 16
- 2.5 Συμπέρασμα . . . 17

3 Επικοινωνία OpenTLD-Peoplebot **18**
- 3.1 Περιγραφή . . . 18
- 3.2 Μοντέλο Πελάτη-Εξυπηρετητή (Client-Server model) . . . 18
- 3.3 Υποδοχές (Sockets) . . . 19
- 3.4 Διαδικασία Επικοινωνίας . . . 19

4 Αναζήτηση στόχων **21**
- 4.1 Σκοπός . . . 21
- 4.2 Προϋποθέσεις . . . 21
- 4.3 Διαδικασία αναζήτησης . . . 22

Α΄ Κώδικας **24**

Β΄ Βιβλιοθήκη OpenCV **45**

Βιβλιογραφία **48**

Περίληψη

Στην παρούσα εργασία γίνεται περιγραφή της αναζήτησης στόχων από ένα ρομπότ. Το ρομπότ που επιλέχθηκε να χρησιμοποιηθεί είναι το peoplebot το οποίο κατασκευάζεται από την Adept MobileRobots.

Απο μία κάμερα που είναι προσαρμοσμένη στο ρομπότ, και η οποία αποτελεί τα 'μάτια' του, παίρνουμε την εικόνα του περιβάλλοντός του. Με επεξεργασία της εικόνας αυτής βρίσκουμε το στίγμα του στόχου που μας ενδιαφέρει.

Για την επεξεργασία της εικόνας χρησιμοποιήθηκε το πρόγραμμα OpenTLD το οποίο αρχικά υλοποιήθηκε σε MATLAB απο τον Zdenek Kalal. Συγκεκριμένα χρησιμοποιήθηκε η έκδοσή του σε C++ , η οποία και υλοποιήθηκε από τον Georg Nebehay[3]. Σκοπός μας ήταν προσαρμόζοντας το συγκεκριμένο πρόγραμμα στο ρομπότ μας να καταφέρουμε να του δώσουμε το στίγμα του στόχου που μας ενδιαφέρει. Στη συνέχεια, και σύμφωνα με αυτές τις συντεταγμένες, να καθοδηγήσουμε το ρομπότ έτσι ώστε να αναζητήσει το στόχο.

Κεφάλαιο 1

Εισαγωγή

1.1 Τι είναι η ψηφιακή επεξεργασία εικόνας

Μία εικόνα ορίζεται ως μία διδιάστατη συνάρτηση f(x,y), όπου x και y αποτελούν τις συντεταγμένες της εικόνας στο χώρο και η ένταση της φωτεινότητας της εικόνας σε κάποιο σημείο καθορίζει το πλάτος της συνάρτησης. Μία εικόνα ονομάζεται ψηφιακή όταν οι τιμές των x και y καθώς και της συνάρτησης f είναι πεπερασμένες. Αυτά τα πεπερασμένα σε πλήθος στοιχεία της ψηφιακής εικόνας ονομάζονται εικονοστοιχεία. Έτσι όταν μιλάμε για ψηφιακή επεξεργασία εικόνας αναφερόμαστε στην επεξεργασία ψηφιακών εικόνων με την βοήθεια κάποιου υπολογιστή.

Σε αυτό το σημείο πρέπει να σημειώσουμε πως κατα γενική ομολογία τα όρια ανάμεσα στις περιοχές της επεξεργασίας εικόνας, της ανάλυσης εικόνας και της υπολογιστικής όρασης δεν είναι σαφή. Κάποιοι ορίζουν ως επεξεργασία εικόνας μια διεργασία όπου τόσο η είσοδος όσο και η έξοδος είναι εικόνες. Αυτό αναγκαστικά μας περιορίζει. Σύμφωνα με αυτόν τον ορισμό μια διαδικασία όπως ο υπολογισμός της μέσης τιμής μια εικόνας, που ως αποτέλεσμα έχει έναν απλό αριθμό, δεν θα θεωρούνταν σχετική με την επεξεργασία εικόνας. Αναφορικά με την υπολογιστική όραση, στόχος της είναι η εξομοίωση της ανθρώπινης όρασης με υπολογιστή. Αυτό γίνεται δυνατό με τη μάθηση, την εξαγωγή συμπερασμάτων και τη λήψη αποφάσεων, βάσει οπτικών ερεθισμάτων. Η ανάλυση εικόνας βρίσκεται ανάμεσα στα πεδία της επεξεργασίας εικόνας και της υπολογιστικής όρασης.

Για να διαχωρίσουμε καλύτερα τις περιοχές μπορούμε να ορίσουμε τρείς τύπους υπολογιστικών διαδικασιών:

- διαδικασίες χαμηλού επιπέδου,
- διαδικασίες ενδιάμεσου επιπέδου και
- διαδικασίες υψηλού επιπέδου.

Οι διαδικασίες χαμηλού επιπέδου περιλαμβάνουν πρωταρχικές λειτουργίες, όπως η προεπεξεργασία μίας εικόνας για την μείωση του θορύβου της. Χαρακτηρίζονται απο το ότι τόσο η είσοδος όσο και η έξοδος τους είναι εικόνες. Οι διαδικασίες ενδιάμεσου επιπέδου περιλαμβάνουν λειτουργίες όπως η κατάτμηση (ο διαχωρισμός δηλαδή της εικόνας σε μικρότερες περιοχές ή αντικείμενα), η περιγραφή αυτών των αντικειμένων καθώς και η αναγνωρισή τους. Χαρακτηρίζονται απο

το ότι η είσοδος τους είναι μία εικόνα, αλλά η έξοδος που παράγουν είναι ιδιότητες που εξάγονται απο αυτήν την εικόνα. Τέλος, οι διαδικασίες υψηλού επιπέδου προσπαθούν να δώσουν κάποιο νόημα σε ένα σύνολο απο αναγνωρισμένα αντικείμενα, όπως και στην ανάλυση εικόνας, έτσι ώστε να μπορούν να πραγματοποιηθούν νοητικές λειτουργίες που σχετίζονται με την όραση.

Σύμφωνα με τα παραπάνω βλέπουμε πως υπάρχει μια επικάλυψη ανάμεσα στην ανάλυση και την επεξεργασία εικόνας, σχετικά με την αναγνώριση των αντικειμένων που περιλαμβάνονται σε μία εικόνα. Έτσι απο εδώ και πέρα όταν θα χρησιμοποιούμε τον όρο ψηφιακή επεξεργασία εικόνας θα αναφερόμαστε σε διαδικασίες των οποίων οι είσοδοι και έξοδοι είναι εικόνες και στόχος τους είναι η εξαγωγή χαρακτηριστικών απο αυτές τις εικόνες, καθώς και η αναγνώριση των επιμέρους αντικειμένων της.

1.2 Προέλευση της ψηφιακής επεξεργασίας εικόνας

Η ιστορία της ψηφιακής επεξεργασίας εικόνας είναι άμεσα συνδεδεμένη με την ανάπτυξη των ψηφιακών υπολογιστών. Η ιδέα για την κατασκευή ενός υπολογιστή έχει ξεκινήσει ήδη στη Μικρά Ασία πριν από πέντε χιλιάδες χρόνια, με την κατασκευή του Άβακα. Επίσης τα τελευταία διακόσια χρόνια έχουν κατασκευαστεί πολλές μηχανές για τις οποίες μπορούμε να πούμε οτι αποτελούν τα θεμέλια αυτού που ονομάζουμε σήμερα υπολογιστή. Ωστόσο, η βάση του μοντέρνου ψηφιακού υπολογιστή, αναπτύχθηκε πολύ πρόσφατα, το 1940, με την εισαγωγή δύο εννοιών-κλειδιών απο τον John von Neumann :

- μίας μνήμης για την αποθήκευση ενός προγράμματος και των δεδομένων του και
- της διακλάδωσης υπό συνθήκη.

Αυτές οι δύο έννοιες αποτελούν τη βάση της κεντρικής μονάδας επεξεργασίας (CPU), η οποία είναι και η καρδιά ενός σημερινού υπολογιστή. Μετά τον John von Neumann ακολούθησαν μία σειρά απο εφευρέσεις-κλειδιά οι οποίες είχαν ως αποτέλεσμα τη δημιουργία ηλεκτρονικών υπολογιστών, τόσο ισχυρών ώστε να χρησιμοποιηθούν για την ψηφιακή επεξεργασία εικόνας. Αυτές μπορούμε να τις συνοψίσουμε στις εξής:

1. εφεύρεση του τρανζίστορ το 1948,
2. ανάπτυξη της COBOL και της FORTRAN στις δεκαετίες του 1950 και 1960, οι οποίες είναι γλώσσες προγραμματισμού υψηλού επιπέδου,
3. εφεύρεση του ολοκληρωμένου κυκλώματος το 1958,
4. ανάπτυξη των λειτουργικών συστημάτων στις αρχές του 1960,
5. ανάπτυξη του μικροεπεξεργαστή (ένα απλό chip που περιλαμβάνει την κεντρική μονάδα επεξεργασίας, τη μνήμη και τα συστήματα εισόδου-εξόδου) στις αρχές του 1970,
6. η εισαγωγή των προσωπικών υπολογιστών το 1981,

7. η προοδευτική σμίκρυνση των εξαρτημάτων ενός υπολογιστή η οποία συνεχίζεται μέχρι και σήμερα.

Ταυτόχρονα με αυτές τις εξελίξεις, μεγάλη πρόοδος έγινε και στις συσκευές μαζικής αποθήκευσης και στα συστήματα απεικόνισης, τα οποία αποτέλουν βασικα προαπαιτούμενα της ψηφιακής επεξεργασίας εικόνας.

Οι πρώτοι υπολογιστές, τόσο ισχυροί ώστε να μπορούν να κάνουν ψηφιακή επεξεργασία εικόνας, εμφανίστηκαν στις αρχές του 1960. Αυτό που ουσιαστικά γέννησε την ψηφιακή επεξεργασία εικόνας ήταν η διαθεσιμότητα αυτών των μηχανημάτων κατά την έναρξη του διαστημικού προγράμματος. Η συνύπαρξη αυτών των δύο γεγονότων έφερε στο προσκήνιο έννοιες σχετικές με την ψηφιακή επεξεργασία εικόνων. Τις εικόνες που έστελναν τα διαστημικά οχήματα πίσω στα επιστημονικά εργαστήρια τις επεξεργάζονταν έτσι ώστε να διορθώσουν παραμορφώσεις που είχαν υποστεί. Αυτή η εμπειρία που απέκτησαν οι επιστήμονες με την επεξεργασία των πρώτων εικόνων από τη Σελήνη τη χρησιμοποίησαν για να βελτιώσουν τις μεθόδους ενίσχυσης και διόρθωσης εικόνας.

Παράλληλα με τα διαστημικά προγράμματα, οι τεχνικές ψηφιακής επεξεργασίας εικόνας άρχισαν να χρησιμοποιούνται και στις ιατρικές απεικονίσεις. Η εφεύρεση της Υπολογιστικής Αξονικής Τομογραφίας είναι ένα κορυφαίο γεγονός της εφαρμογής της επεξεργασίας εικόνας στο χώρο της ιατρικής διάγνωσης.

Απο το 1960 μέχρι και σήμερα το πεδίο της επεξεργασίας εικόνας έχει γνωρίσει μεγάλη ανάπτυξη. Πέρα απο τις εφαρμογές που ήδη αναφέραμε, οι τεχνικές επεξεργασίας εικόνας χρησιμοποιούνται σε ένα ευρύ φάσμα εφαρμογών. Οι ίδιες διαδικασίες που χρησιμοποιούνται για να ενισχύσουν την αντίθεση ή να κωδικοποιήσουν τα επίπεδα έντασης σε χρώμα για την ερμηνεία των ακτίνων X, χρησιμοποιούνται και απο γεωγράφους για τη μελέτη μορφωμάτων μόλυνσης που εμφανίζονται σε εικόνες. Οι αρχαιολόγοι, επίσης, χρησιμοποιούν τεχνικές αποκατάστασης θολών εικόνων για την καταγραφή τεχνουργημάτων τα οποία έχουν καταστραφεί μετα την αποτυπωσή τους.

Όλα αυτά είναι παραδείγματα που το αποτέλεσμα της επεξεργασίας εικόνας ερμηνεύται απο τον άνθρωπο. Υπάρχει όμως, όπως έχουμε ήδη αναφέρει, και η εφαρμογή της ψηφιακής επεξεργασίας εικόνας, σύμφωνα με την οποία κάποιες διαδικασίες εξάγουν πληροφορία από μία εικόνα σε μορφή κατάλληλη για επεξεργασία από υπολογιστή. Αυτό το πεδίο της επεξεργασίας εικόνας ονομάζεται μηχανική αντίληψη. Παραδείγματα τέτοιου είδους πληροφορίας που χρησιμοποιεί η μηχανική αντίληψη είναι οι στατιστικές ροπές, οι συντελεστές μετασχηματισμού Fourier και πολυδιάστατα μέτρα αποστάσεων. Προβλήματα που παρουσιάζονται στο χώρο της μηχανικής αντίληψης είναι η αυτόματη αναγνώριση χαρακτήρων, η αυτόματη επεξεργασία δακτυλικών αποτυπωμάτων και η βιομηχανική όραση για την κατασκευή αντικειμένων.

1.3 Εφαρμογές της ψηφιακής επεξεργασίας εικόνας

Τη σημερινή εποχή δεν υπάρχει καμία περιοχή τεχνικής δραστηριότητας που να μην περιλαμβάνει ως ένα βαθμό την ψηφιακή επεξεργασία εικόνας. Σε αυτήν την ενότητα θα παρουσιάσουμε κάποια από τα πεδία στα οποία η ψηφιακή επεξεργασία εικόνας βρίσκει εφαρμογή. Επειδή αυτά τα πεδία ποικίλουν, πρέπει με κάποιον τρόπο να τα ταξινομήσουμε ώστε να γίνουν κατανοητά. Ένας απλός τρόπος να

γίνει αυτό είναι να τα αναφέρουμε με βάση τον τρόπο που χρησιμοποιούν για να παράξουν τις εικόνες. Βασική πηγή ενέργειας εικόνων είναι το ηλεκτρομαγνητικό φάσμα, καθώς και τα ακουστικά, υπερηχητικά και ηλεκτρονικά κύματα.

1.3.1 Εφαρμογές στην απεικόνιση με ακτίνες γ

Η απεικόνιση εικόνων με ακτίνες γ βρίσκει σημαντική εφαρμογή στην πυρηνική ιατρική καθως και στις αστρονομικές παρατηρήσεις. Στην πυρηνική ιατρική αυτό που γίνεται είναι ο ασθενής να καταπίνει ένα ραδιενεργό ισότοπο το οποίο εκπέμπει ακτίνες γ. Στη συνέχεια ανιχνευτές συλλέγουν αυτές τις ακτίνες και δημιουργούν την εικόνα. Αυτές οι εικόνες και η ψηφιακή επεξεργασία τους μας βοηθάει στο να εντοπίσουμε προβλήματα του ασθενή σχετικά με τα οστά του, όπως όγκους ή λοιμώξεις.

1.3.2 Εφαρμογές στην απεικόνιση με ακτίνες X

Οι ακτίνες X είναι μία διαδικασία απεικόνισης που χρησιμοποιείται εδώ και πολλά χρόνια. Ένα πεδίο εφαρμογής τους είναι η ιατρική διάγνωση. Επιπλέον χρησιμοποιούνται στη βιομηχανία και στην αστρονομία. Μία κλασική εφαρμογή των ακτίνων X στην ιατρική διάγνωση είναι η χρήση τους απο τους ιατρούς για τον εντοπισμού κάποιου κατάγματος σε ασθενείς. Σημαντική εφαρμογή βρίσκουν επίσης και στην αγγειογράφηση. Με αυτήν τη διαδικασία λαμβάνονται εικόνες από τα αγγεία του ασθενή. Αποτέλεσμα της διαδικασίας αυτής είναι η μεγάλη αντίθεση των αιμοφόρων αγγείων που επιτρέπει στον ιατρό να εντοπίσει θρομβώσεις και άλλες ανωμαλίες. Μέσω της ψηφιακής επεξεργασίας εικόνας είναι δυνατόν αυτή η αντίθεση να γίνει ακόμη μεγαλύτερη με σκοπό τη διευκόλυνση του ιατρού. Μία ακόμη εφαρμογή στην ιατρική διάγνωση είναι η αξονική τομογραφία. Με την βοήθεια των ακτίνων X είναι δυνατόν να λαμβάνουμε εικόνες που περιέχουν κάθετες τομές του ασθενή και βοηθούν τους ιατρούς στην καλυτερη διάγνωση. Τέλος, μια εφαρμογή των ακτίνων X στη βιομηχανία έχει να κάνει με εικόνες που περιέχουν ηλεκτρικά κυκλώματα. Αυτές χρησιμοποιούνται για τον έλεγχο πλακετών ηλεκτρικών κυκλωμάτων και τον εντοπισμό ατελειών ή κάποιου κατασκευαστικού λάθους.

1.3.3 Εφαρμογές στην απεικόνιση στις ζώνες του ορατού και του υπέρυθρου

Η ζώνη του ορατού ηλεκτρομαγνητικού φάσματος περιλαμβάνει πολλές γνωστές δραστηριότητες μας και η απεικόνιση της είναι πολύ σπουδαία αναφορικά με το πλήθος των εφαρμογών της. Πολύ συχνά όμως στις εφαρμογές αυτές η απεικόνιση στη ζώνη του ορατού χρησιμοποιείται σε συνδυασμό με την απεικόνιση στη ζώνη του υπέρυθρου. Τέτοιες εφαρμογές συναντάμε στη μικροσκοπία φωτός, στην αστρονομία, στη βιομηχανία της τηλεπισκόπησης και στην επιβολή του νόμου. Μία εφαρμογή είναι η παρατήρηση και η πρόβλεψη του καιρού με τη βοήθεια εικόνων που λαμβάνουμε από δορυφόρους. Σημαντική εφαρμογή της απεικόνισης στη ζώνη του ορατού φάσματος είναι η αυτόματη οπτική επιθεώρηση των βιομηχανικών προϊόντων. Η ψηφιακή επεξεργασία μιας τέτοιας φωτογραφίας βοηθάει στον εντοπισμό κάποιου εξαρτήματος που λείπει από αυτό το προϊόν. Επίσης εφαρμογές μπορούμε να έχουμε και σε εικόνες δακτυλικών αποτυπωμάτων. Πάνω σε αυτές γίνεται εντοπισμός κάποιων χαρακτηριστικών και ακολουεί αναζήτησή τους σε μία

βάση δεδομένων με σκοπό το ταίριασμα. Τέλος η ψηφιακή επεξεργασία εφαρμόζεται σε εικόνες χαρτονομισμάτων με σκοπό την αυτόματη καταμέτρησή τους ή την ανάγνωση των σειριακών αριθμών τους.

1.3.4 Εφαρμογές σε άλλες μορφές απεικόνισης

Η απεικόνιση που βασίζεται στη χρήση του ήχου, βρίσκει εφαρμογές στη γεωλογία, στη βιομηχανία και στην ιατρική. Στη γεωλογία χρησιμοποιούν υπερήχους έτσι ώστε να προσδιορίζουν την εικόνα του υπεδάφους, με σκοπό την εύρεση πετρελαίου και ορυκτών. Στην ιατρική και πιο συγκεκριμένα στη μαιευτική χρησιμοποιείται η απεικόνιση με τη χρήση υπερήχων για την απεικόνιση των βρεφών. Έτσι είναι δυνατόν να διαπιστωθεί πόσο υγιής είναι η αναπτυξή τους. Επίσης μέσω του υπερήχου μπορεί να προσδιοριστεί και το φύλο του παιδιού.

Κεφάλαιο 2

OpenTLD

2.1 Περιγραφή

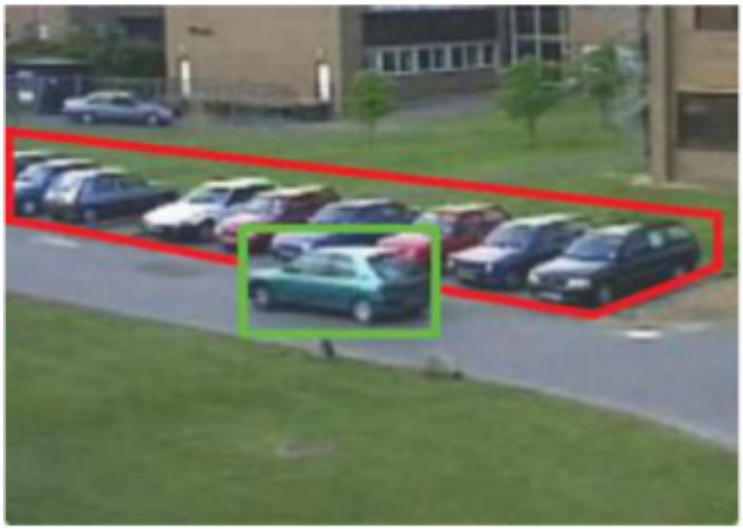
Το ανθρώπινο μάτι για να εντοπίζει αντικείμενα επεξεργάζεται την πληροφορία η οποία φτάνει στον αμφιβληστροειδή χιτώνα του ματιού και ενεργοποιεί κάποιο ερέθισμα. Σε αντίθεση με το ανθρώπινο μάτι, στην υπολογιστική όραση χρησιμοποιούμε μαθηματικές τεχνικές έτσι ώστε να εξάγουμε πληροφορία σχετική με κάποιο αντικείμενο, βασιζόμενοι σε εικόνες απο κάποια κάμερα. Αυτές οι μέθοδοι έχουν πολλές εφαρμογές, κάποιες απο τις οποίες είναι:

- αναγνώριση γραφής,
- καθοδήγηση ρομπότ,
- ανακατασκευή κάποιου σκηνικού,
- κατηγοριοποίηση αντικειμένων.

Μια ακόμη σημαντική εφαρμογή η οποία και θα μας απασχολήσει είναι αυτή της ανίχνευσης και εντοπισμού στόχων. Αυτήν την αναζήτηση των στόχων την εκτελεί το πρόγραμμα OpenTLD, [3] το οποίο και θα περιγράψουμε στο παρών κεφάλαιο.

Το συγκεκριμένο πρόγραμμα κάνει αναζήτηση ενός και μόνο στόχου. Το πρόβλημα της αναζήτησης ενός στόχου ορίζεται ώς εξής: αφού μας δωθεί μια σείρα απο εικόνες $I_1 ... I_n$, να μπορούμε να εκτιμήσουμε τη θέση x_k του στόχου για κάθε I_k. Στην περίπτωσή μας η θέση x_k είναι η πάνω αριστερή γωνία του πλαισίου οριοθέτησης. Πλαίσιο οριοθέτησης θα ονομάζουμε το κουτί που περικλείει το αντικείμενο το οποίο μας ενδιαφέρει στην είκονα, και το οποίο θα αρχικοποιείται απο το χρήστη.

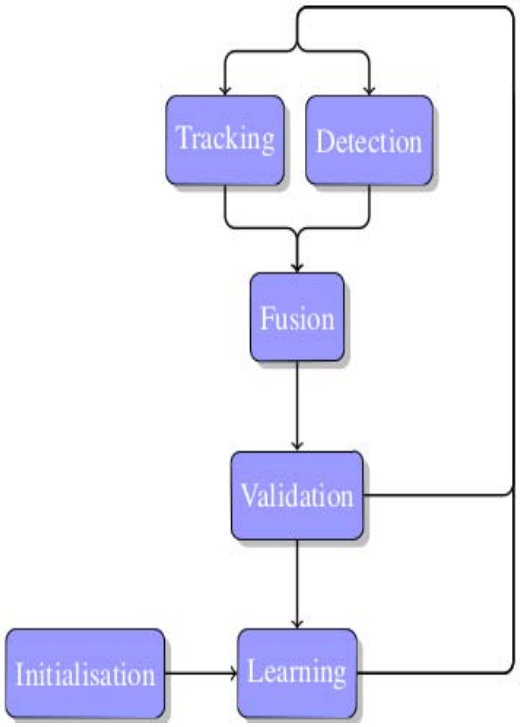
Η κύρια πρόκληση στην αναζήτηση στόχων είναι η σύγχυση. Σύγχυση είναι το φαινόμενο κατά το οποίο χαρακτηριστικά του αντικειμένου που μας ενδιαφέρει είναι δύσκολο να τα ξεχωρίσουμε από χαρακτηριστικά που εξαγουμε από άλλα αντικείμενα στην εικόνα. Ένα παράδειγμα σύγχυσης βλέπουμε στην εικόνα 2.1, όπου πολλά αντικείμενα είναι παρόμοια στο σχήμα με το αντικείμενο που μας ενδιαφέρει.



Εικόνα 2.1

Μία άλλη πρόκληση είναι αυτή της μεταβλητότητας της εμφάνισης του ίδιου του στόχου. Αυτό περιλαμβάνει αλλαγές στη στάση του αντικειμένου ή παραμόρφωση του σχήματός του ή ακόμη και αλλαγή της φωτεινότητάς του. Μέθοδοι οι οποίες χρησιμοποιούν ένα πλαίσιο για να οριοθετήσουν το αντικείμενο-στόχο αντιμετωπίζουν το πρόβλημα της ανανέωσης αυτού του πλαισίου, το οποίο έχει να κάνει με το πώς θα ενημερωθεί έτσι ώστε να παραμένει αντιπροσωπευτικό του μοντέλο του στόχου. Εάν το πρωταρχικό πλαίσιο δεν αλλάξει ποτέ, τελικά δεν θα είναι μία ακριβής αναπαράσταση του μοντέλου. Αντίθετα, εάν το πλαίσιο προσαρμόζεται σε κάθε αλλαγή στην εμφάνιση, τα λάθη θα συσσωρεύονται και το πλαίσιο σταδιακά θα απομακρύνεται απο το στόχο. Αυτό το πρόβλημα σχετίζεται με το δίλημμα σταθερότητας-προσαρμοστικότητας, το οποίο έχει να κάνει με το συμβιβασμό της σταθερότητας, για την διατήρηση της πληροφορίας, και της προσαρμοστικότητας που απαιτείται για νέα εκμάθηση. Αυτό το δίλημμα αντιμετωπίζεται από όλα τα συστήματα εκμάθησης. Τέλος, προβλήματα αντιμετωπίζονται, όταν ο στόχος καλύπτεται από άλλα αντικείμενα ή αν φύγει από το οπτικό πεδίο της κάμερας. Για την αντιμετώπιση αυτού του προβλήματος είναι απαραίτητος ένας μηχανισμός που να ανιχνεύει το στόχο ανεξάρτητα από την προηγούμενη θέση του.

Το πρόγραμμα OpenTLD ουσιαστικά αποτελείται απο τρεις διαδικασίες. Την ανίχνευση του στόχου (Tracking), τον εντοπισμό του στόχου(Detection) και την εκμάθηση του στόχου(Learning). Για την ανίχνευση χρησιμοποιείται η μέθοδος αναδρομικής ανίχνευσης του Kalal [9] , ο οποίος βασίζεται στην εκτίμηση της οπτικής ροής χρησιμοποιώντας τη μέθοδο των Lucas και Kanade. Για τον εντοπισμό διατηρούνται πλαίσια τα οποία κανονικοποιούνται ως προς τη φωτεινότητα και το μέγεθος. Επίσης κρατούνται ξεχωριστά πλαίσια με θετικά στιγμιότυπα του στόχου και με αρνητικά στιγμιότυπα από το υπόβαθρο. Αυτά αποτελούν τη βάση του εντοπιστή ο οποίος δρά ανεξάρτητα απο τον ανιχνευτή. Η αρχικοποίηση ξεκινάει με το χρήστη ο οποίος υποδεικνύει το αντικείμενο-στόχο τοποθετώντας το σε ένα πλαίσιο. Τότε έχουμε και το πρώτο βήμα εκμάθησης. Στη συνέχεια ο ανιχνευτής και ο εντοπιστής τρεχουν παράλληλα και τα αποτελέσματά τους συγχωνεύονται σε ένα τελικό αποτέλεσμα. Αν το αποτέλεσμα περάσει το στάδιο επιβεβαίωσης τότε πραγματοποιείται η εκμάθηση. Στη συνέχεια η ίδια διαδικασία επαναλαμβάνεται. Στην Εικόνα 2.2 βλέπουμε το διάγραμμα ροής αυτής της διαδικασίας



Εικόνα 2.2

2.2 Ανίχνευση (Tracking)

Σε αυτό το σημείο θα περιγράψουμε πώς δουλεύει η διαδικασία ανίχνευσης του στόχου στο πρόγραμμα OpenTLD. Αρχικά, όπως ήδη αναφέραμε, ο χρήστης πρέπει να ορίσει το αντικείμενο μέσα σε ένα πλαίσιο στην πρώτη εικόνα. Πέρα από αυτό δεν χρειάζεται άλλη πληροφορία για το αντικείμενο.

Το πρώτο πράγμα που γίνεται είναι να κατασκευάζεται μια σειρά από σημεία μέσα στο πλαίσιο οριοθέτησης που έχει δώσει ο χρήστης. Στη συνέχεια υπολογίζεται η οπτική ρόη σε καθένα από αυτά τα σημεία σύμφωνα με τη μέθοδο των Lucas και Kanade. Η μέθοδος δουλεύει καλύτερα για σημεία τα οποία βρίσκονται σε γωνίες και είναι αδύνατον να ανιχνεύσει σημεία τα οποία βρίσκονται σε ομογενείς περιοχές. Χρησιμοποιώντας την πληροφορία απο τη μέθοδο των Lucas και Kanade καθώς και απο δύο μεθόδους για μέτρηση λάθους, -κανονικοποιημένη συσχέτιση και τον !εμπρός-πίσω έλεγχο για λάθος(forward-backward error)-, βρίσκουμε σημεία τα οποία είναι εσφαλμένα. Εάν η μέση τιμή των μετρήσεων του εμπρός-πίσω λάθους είναι μεγαλύτερη απο κάποιο κατώφλι σταματάμε την αναδρομική ανίχνευση, καθώς είναι πολυ πιθανό το πλαίσιο οριοθέτησης να έχει παρεκλίνει. Τελικά όσα σημεία έχουν απομείνει τα χρησιμοποιούμε για να υπολογίσουμε το πλαίσιο οριοθέτησης στην επόμενη εικόνα, εφαρμόζοντας ένα μοντέλο μετασχηματισμού με κατάλληλες αλλαγές στην κλίμακα.

2.2.1 Εκτίμηση Οπτικής Ροής

Η μέθοδος των Lucas και Kanade κάνει τρεις βασικές υποθέσεις. Η πρώτη έχει να κάνει με τη σταθερότητα της φωτεινοτητάς και εκφράζεται με αυτόν τον τύπο,

$$I(X) = J(X + d)$$

ο οποίος και δηλώνει ότι ένα εικονοστοιχείο που βρίσκεται στη θέση X στην εικόνα I θα μετατοπιστεί κατα d στην εικόνα J αλλά διατηρεί τη φωτεινότητα του. Το d θα το αναφέρουμε ως διάνυσμα μετατόπισης.

Η δεύτερη υπόθεση αναφέρεται ως χρονική συνοχή, που σημαίνει ότι το διάνυσμα μετατόπισης δεν μπορεί να είναι μεγάλο. Αυτο σημαίνει οτι το J(X) είναι περίπου

$$J(X) \approx I(X) + I'(X)d$$

όπου I'(X) είναι η κλίση του I στη θέση Ξ. Έτσι μία εκτίμηση για το d είναι

$$d \approx (J(X) - I(X))/I'(X)$$

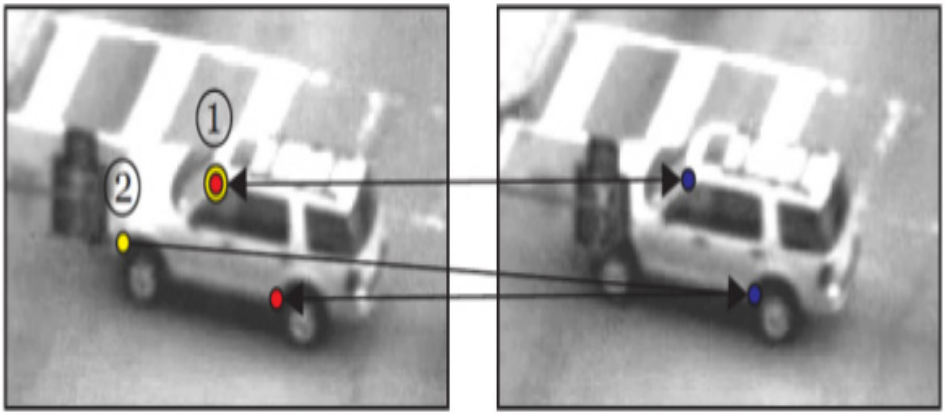
Η τρίτη υπόθεση έχει να κάνει με τη χωρική συνοχή. Αυτό σημαίνει πως όλα τα γειτονικά εικονοστοιχεία κινούνται συνεκτικά, δηλαδή όλα μαζί. Σύμφωνα με αυτά μπορούμε να βρούμε το d ελαχιστοποιώντας τον όρο

$$\sum_{(x,y)\in W} (J(X) - I(X) - I'(X)d)^2$$

Το W καθορίζει την περιοχή γύρω απο κάθε pixel

2.2.2 Μετρήσεις Λάθους

Η μέτρηση του !εμπρός-πίσω λάθους προτείνεται απο τον Kalal[9]. Θα προσπαθήσουμε να το περιγράψουμε σύμμφωνα με την εικόνα 2.3.



Εικόνα 2.3

Στην αριστερή εικόνα βλέπουμε ότι το σημείο 1 ανιχνεύεται σωστά στην αντίστοιχη θέση του στη δεξιά εικόνα. Ενώ αντίθετα το σημείο 2 ανιχνευέται σε λάθος θέση. Αυτή η μέτρηση λάθους για την οποία μιλάμε υποδεικνύει ότι η ανίχνευση όλων των σημείων πρέπει να είναι αντιστρέψιμη. Το σημείο 1 ανιχνεύεται πίσω στην κανονική του θέση, ενώ το σημείο 2 όχι. Το λάθος αυτό ορίζεται ως η Ευκλείδια Απόσταση

$$\varepsilon = \mid p - p'' \mid$$

όπου το p'' είναι

$$p'' = LK(LK(p)),$$

δηλαδή η μέθοδος Lucas-Kanade εφαρμόζεται δύο φορές στο p.

Σε συνδυασμό με την !εμπρός-πίσω μέτρηση λάθους εφαρμόζεται και μία άλλη μέτρηση που βασίζεται στην ομοιότητα της περιοχής γύρω απο το σημείο p και της περιοχής γύρω απο το p' . Αυτή η ομοιότητα ελέγχεται σύμφωνα με την κανονικοποιημένη συσχέτιση αυτών των δύο περιοχών.

$$NCC(P_1, P_2) = \frac{1}{n-1} \sum_{x=1}^{n} \frac{(P_1(x)-m_1)(P_2(x)-m_2)}{s_1 s_2}$$

2.2.3 Μοντέλο Μετασχηματισμού

Ακολουθώντας τη μέθοδο του Kalal[9] υπολογίζουμε τη μέση τιμή όλων των εμπρός-πίσω λαθών med_{FB} και τη μέση τιμή med_{NCC} όλων των μετρήσεων ομοιότητας και κρατάμε όλα τα σημεία τα οποία έχουν εμπρός-πίσω λάθος μικρότερο του med_{FB} και μέτρο ομοιότητας μεγαλύτερο του med_{FB}. Να σημειώσουμε εδώ πως στην περίπτωση όπου το med_{FB} βρεθεί μεγαλύτερο από κάποιο προκαθορισμένο κατώφλι δεν λαμβάνουμε υπόψη μας τα αποτελέσματα της ανίχνευσης και τη θεωρούμε λανθασμένη. Με τα εναπομείναντα σημεία θα φτιάξουμε το μετασχηματισμένο πλαίσιο οριοθέτησης. Υπολογίζουμε την απόσταση πριν και μετά την ανίχνευση και η οποιαδήποτε αλλαγή θεωρείται αλλαγή στην κλίμακα. Η μετατόπιση στον x-άξονα υπολογίζεται σύμφωνα με τη μέση τιμή όλων των οριζόντιων μετατοπίσεων όλων των σημείων. Ανάλογα και για τον y-άξονα.

2.2.4 Συμπεράσματα

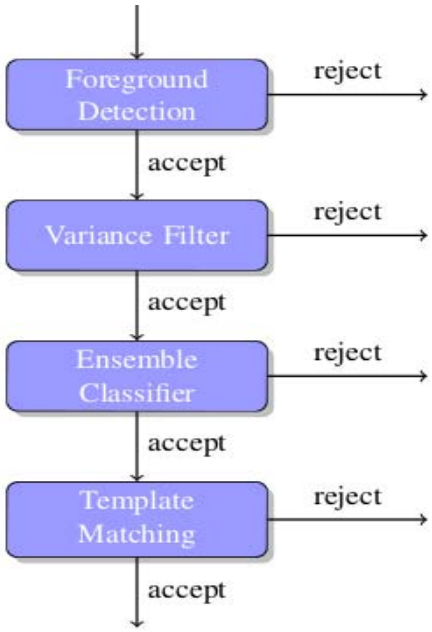
Μέχρι τώρα περιγράψαμε πως ο ανιχνευτής ακολουθεί το στόχο που μας ενδιαφέρει. Δεν χρειάζεται καμία πληροφορία εκτός από τη θέση του στην προηγούμενη εικόνα. Εξηγήσαμε τον τρόπο που γίνεται ο υπολογισμός του καινούργιου πλαισίου οριοθέτησης, καθώς και δύο μεθόδους για τη βελτίωση αυτού του υπολογισμού. Παρ΄ όλ΄ αυτά ο ανιχνευτής από τη στιγμή που θα χάσει το στόχο του θα είναι αδύνατο στη συνέχεια να τον επανεντοπίσει.

2.3 Εντοπισμός (Detection)

Σε αυτήν την ενότητα θα περιγράψουμε τον τρόπο με τον οποίο γίνεται ο εντοπισμός του στόχου. Ο εντοπιστής είναι αυτός που θα επανεκκίνησει τον ανιχνευτή την στιγμή που θα χάσει το στόχο του. Αντίθετα με τον ανιχνευτή που χρειάζεται τη θέση του στην προηγούμενη εικόνα ο εντοπιστής για να βρεί το στόχο εκτελεί μια εξαντλητική αναζήτηση, καθώς χιλιάδες υποπαράθυρα της εικόνας υπολογίζονται, και καταναλώνει τον περισσότερο χρόνο της διαδικασίας.

Ο εντοπιστής βασίζεται στη μέθοδο του ολισθαίνοντος παραθύρου. Του δίνεται σαν είσοδος μία εικόνα και αυτός στη συνέχεια υπολογίζει μία συνάρτηση ταξινόμησης βάσει συγκεκριμένων υποπαραθύρων αυτής της εικόνας. Κάθε υποπαράθυρο ελέγχεται ανεξάρτητα ως προς το αν περιέχει το στόχο. Ο έλεγχος αυτός γίνεται βάσει της ροής στην εικόνα 2.4. Ένα υποπαράθυρο προχωράει στο επόμενο στάδιο της ροής εάν γίνει αποδεκτό από το προηγούμενο. Τα διάφορα επίπεδα έχουν ως στόχο να απορρίπτουν όσα περισσότερα μη σχετικά υποπαράθυρα, με τις λιγότερες δυνατές υπολογιστικές πράξεις. Στο πρώτο επίπεδο χρησιμοποιείται η μέθοδος αφαίρεσης του υπόβαθρου έτσι ώστε να περιοριστεί η αναζήτηση στις εμπρόσθιες περιοχες της εικόνας. Αυτό απαιτεί ένα μοντέλο υπόβαθρου. Αν αυτό δεν είναι

διαθέσιμο προσπερνάμε αυτό το επίπεδο. Στο δεύτερο, όλα τα υποπαράθυρα που παρουσιάζουν μία διακύμανση μικρότερη απο κάποιο κατώφλι απορρίπτονται. Το τρίτο περιέχει έναν ταξινομητή. Το τέταρτο επίπεδο αποτελείται από μια μέθοδο σύγκρισης στιγμιότυπων που χρησιμοποιεί τον κανονικοποιημένο συνετελεστή συσχέτισης σαν μέτρο σύγκρισης.



Εικόνα 2.4

2.3.1 Μέθοδος ολισθαίνοντος παραθύρου

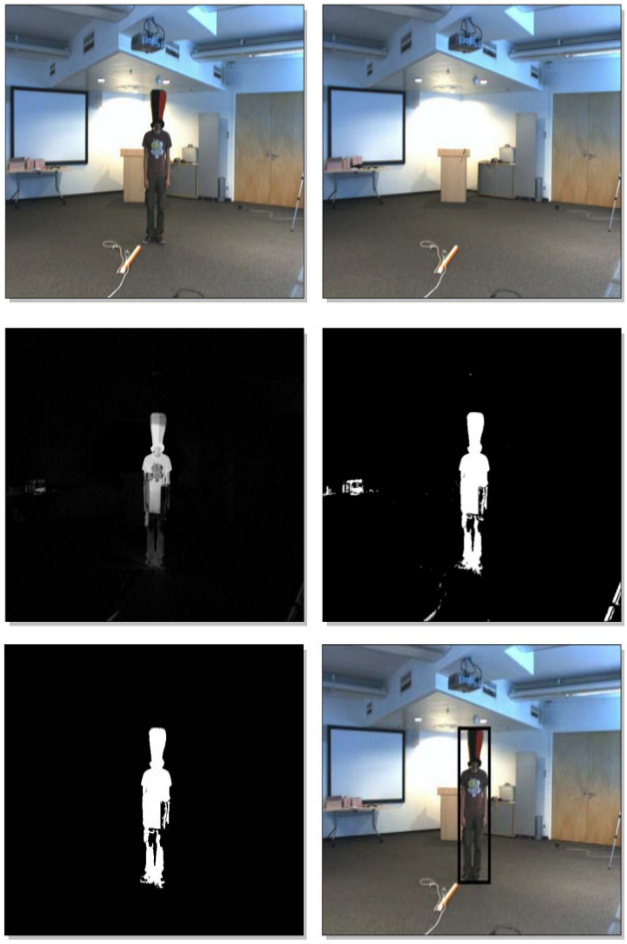
Στη μέθοδο του ολισθαίνοντος παραθύρου για την αναζήτηση του στόχου ελέγχονται όλα τα υποπαράθυρα της εικόνας για το αν τον περιέχουν ή όχι. Ωστόσο, όλα τα δυνατά υποπαράθυρα μίας εικόνας είναι πάρα πολλά, γιατί ο αριθμός τους αυξάνει εκθετικά με βάση το μέγεθος της εικόνας. Ο χώρος αναζήτησης ελλατώνεται εάν εφαρμόσουμε κάποιους περιορισμούς. Πρώτον υποθέτουμε ότι ο στόχος διατηρεί τις διαστάσεις του. Επιπλέον θέτουμε περιθώρια d_x και d_y ανάμεσα σε δύο γειτονικά υποπαράθυρα και τα θέτουμε να έχουν τιμή το $1/10$ της τιμής του αρχικού πλαίσιου οριοθέτησης. Για να γίνει αναζήτηση του στόχου σε διαφορετική κλίμακα ορίζεται ένας συντελεστής κλιμάκωσης με βάση το αρχικό πλαίσιο οριοθέτησης. Τέλος λαμβάνουμε υπόψη μας υποπαράθυρα με ελάχιστο εμβαδό εικοσί πέντε εικονοστοιχεία.

2.3.2 Μέθοδος αφαίρεσης υπόβαθρου

Για την αναγνώριση κάποιου κινούμενου στόχου σε ένα βίντεο μπορεί να εφαρμοστεί η μέθοδος αφαίρεσης υπόβαθρου, στην οποία κάθε εικόνα του βίντεο συγκρίνεται με ένα μοντέλο υπόβαθρου. Τώρα θα περιγράψουμε πως αυτή η μέθοδος επιταχύνει την διαδικασία της αναζήτησης του στόχου. Περιλαμβάνει τέσσερα βήματα όπως φαίνεται στην εικόνα 2.5. Η πάνω δεξιά εικόνα αποτελεί το υπόβαθρο I_{bg} και η επάνω αριστερά εικόνα είναι η εικόνα μας I στην οποία θα αναζητήσουμε τον στόχο μας. Πρώτα βρίσκουμε την απόλυτη διαφορά αυτών των δύο.

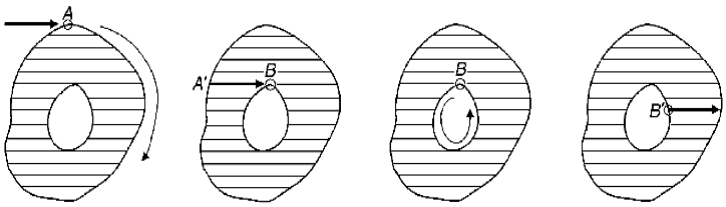
$$I_{absdiff} = \mid I_{bg} - I \mid$$

Το αποτέλεσμα αυτής της πράξης το βλεπουμε στην πρώτη εικόνα της δεύτερης γραμμής. Τώρα εφαρμόζοντας μία κατωφλίωση στο $I_{absdiff}$ δημιουργούμε τη δεύτερη εικόνα της δεύτερης γραμμής.



Εικόνα 2.5

Σε αυτό το σημείο εφαρμόζεται ένας αλγόριθμος ο οποίος βάζει ετικέτες στα εικονοστοιχεία της εικόνας. Αυτές οι ετικέτες τοποθετούνται μόνο με ένα πέρασμα από την εικόνα. Ξεκινώντας από την πρώτη γραμμή, ο αλγόριθμος σαρώνει την εικόνα από τα αριστερά προς τα δεξιά όπως φαίνεται στην εικόνα 2.6.



Εικόνα 2.6

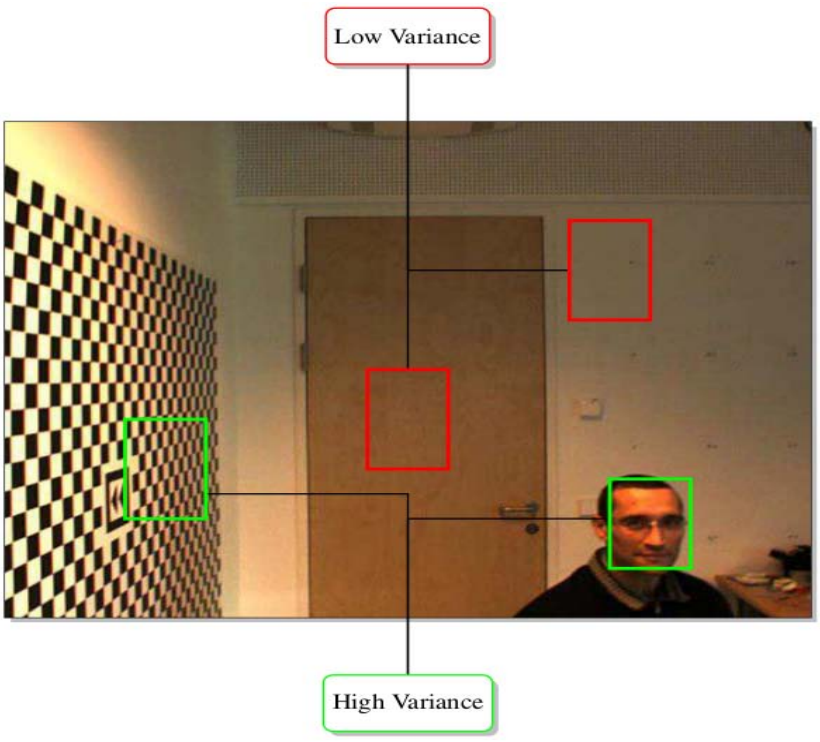
Εάν συναντήσει ένα εικονοστοιχείο A το οποίο δεν έχει ετικέτα τότε τοποθετεί σε αυτό και σε όλα τα εικονοστοιχεία του περιγράμματος του αντικειμένου μια μοναδική ετικέτα. Αυτό το περίγραμμα θεωρείται εξωτερικό περίγραμμα. Στην περίπτωσή που συναντήσει ένα εικονοστοιχείο που έχει ετικέτα τότε σε όλα τα άσπρα εικονοστοιχεία στα δεξιά του τοποθετείται η ίδια ετικέτα μέχρι να συναντήσει κάποιο άλλο περίγραμμα. Εάν αυτό έχει ήδη ετικέτα σημαίνει πως είναι εξωτερικό

περίγραμμα οπότε ο αλγόριθμος προχωράει. Όταν αυτό όμως δεν έχει ετικέτα, όπως στην περίπτωση του σημείου B της εικόνας 2.6 σημαίνει πως είναι εσωτερικό περίγραμμα και σε όλα τα εικονοστοιχεία του τοποθετείται η ίδια ετικέτα με το εικονοστοιχείο B. Αν συναντήσει ένα σημείο εσωτερικού περιγράμματος που έχει ήδη ετικέτα όπως το σημείο B' της εικόνας 2.6 τότε όλα τα άσπρα εικονοστοιχεία στα δεξιά του παίρνουν την ίδια ετικέτα με το εικονοστοιχείο A. Το μικρότερο δυνατό πλαίσιο που χωράει ένα αντικείμενο καθορίζεται από τις συντεταγμένες των εξωτερικών εικονοστοιχείων. Αυτο το εμβαδό υπολογίζεται με το άθροισμα των άσπρων εικονοστοιχείων σε ένα αντικείμενο.

Πίσω στην εικόνα 2.5. Τώρα αφαιρούμε όλα τα αντικείμενα των οποίων το εμβαδό είναι μικρότερο σε μέγεθος από το αρχικό πλαίσιο οριοθέτησης. Το αποτέλεσμα φαίνεται στην πρώτη εικόνα της τρίτης γραμμής. Όλα τα υποπαράθυρα που δεν περιέχονται ολόκληρα σε ένα απο τα μικρότερα πλαίσια οριοθέτησης γύρω απο τα εναπομείναντα αντικείμενα, απορρίπτονται. Στην περίπτωση που δεν είναι διαθέσιμο κάποιο μοντέλο υπόβαθρου τότε όλα τα υποπαράθυρα είναι αποδεκτά.

2.3.3 Φίλτρο διακύμανσης

Η διακύμανση σε ένα κομμάτι της εικόνας είναι ένα μέτρο ομοιομορφίας. Στην εικόνα 2.7 φαίνονται δυο υποπαράθυρα, με κόκκινο χρώμα πλαισίου, που βρίσκονται σε ομοιόμορφες περιοχές του υπόβαθρου. Και τα δύο αυτά υποπαράθυρα έχουν διακύμανση μικρότερη από το υποπαράθυρο που περιέχει το στόχο που αναζητούμε· αυτό είναι το πράσινο πλαίσιο στα δεξιά της εικόνας. Με το φίλτρο διακύμανσης υπολογίζεται η διακύμανση κάθε υποπαραθύρου και απορρίπτονται όλα αυτά που έχουν διακύμανση μικρότερη απο κάποιο κατώφλι. Έτσι είναι δυνατόν να ξεχωρίσουμε το στόχο από κάποιες ομοιόμορφες περιοχές του υπόβαθρου, αλλά όχι απο κάποιες οι οποίες παρουσιάζουν υψηλή διακύμανση. Όπως για παράδειγμα στο πράσινο πλαίσιο στα αριστερά της εικόνας.



Εικόνα 2.7

2.3.4 Ταξινομητής

Στο τρίτο στάδιο του εντοπισμού εφαρμόζεται μία μέθοδος ταξινόμησης γνωστή ως ταξινόμηση τυχαίου δείγματος (random fern classification). Αυτή βασίζει την απόφασή της στη σύγκριση της τιμής της έντασης πολλών εικονοστοιχείων. Για κάθε υποπαράθυρο που εξετάζεται υπολογίζεται μία πιθανότητα P_{pos}. Στην περίπτωση που αυτή η πιθανότητα είναι μικρότερη απο 0.5, το υποπαράθυρο απορρίπτεται. Αυτή η μέθοδος ταξινόμησης είναι πιο αργή από τη μέθοδο του φίλτου διακύμανσης.

2.3.5 Σύγκριση στιγμιότυπων

Στο τέταρτο στάδιο του εντοπισμού εφαρμόζεται μία μέθοδος σύγκρισης στιγμιότυπων. Πρώτα μετατρέπονται όλα τα υποπαράθυρα σε μέγεθος 15x15εικονοστοιχείων. Για τη σύγκριση δύο υποπαραθύρων χρησιμοποιείται ο κανονικοποιημένος συντελεστής συσχέτισης.

$$ncc(P_1, P_2) = \frac{1}{n-1} \sum_{x=1}^{n} \frac{(P_1(x)-m_1)(P_2(x)-m_2)}{s_1 s_2}$$

Εάν τον ερμηνεύσουμε γεωμετρικά μας δίνει το συνημίτονο της γωνίας ανάμεσα στα δύο κανονικοποιημένα διανύσματα. Αυτός ο συντελεστής παίρνει τιμές απο -1 έως 1, με τις τιμές κοντά στο 1 να σημαίνουν πως τα δύο υποπαράθυρα μοιάζουν. Με βάση τον ακόλουθο μαθηματικό τύπο η απόσταση ορίζεται να παίρνει τιμές απο 0 έως 1.

$$d(P_1, P_2) = 1 - \frac{1}{2}(ncc(P_1, P_2) + 1)$$

Διατηρούνται στιγμιότυπα μίας θετικής και μίας αρνητικής κλάσης. Στην θετική κλάση αναφερόμαστε ως P^+ και στην αρνητική ως P^-. Όταν δίνεται ένα υποπαράθυρο P που είναι άγνωστη η κλάση του τότε υπολογίζεται η απόστασή του απο τη θετική κλάση

$$d^+ = mind(P_0, P_i), P_i \in P^+$$

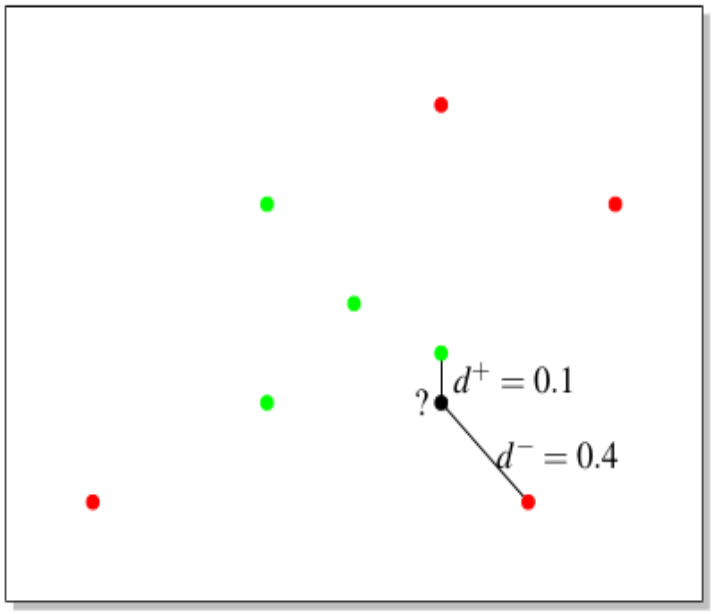
καθώς και η απόστασή του απο την αρνητική κλάση

$$d^- = mind(P_0, P_j), P_j \in P^-$$

Στην εικόνα 2.8 οι πράσινες τελείες αποτελούν στιγμιότυπα της θετικής κλάσης ενώ οι κόκκινες τελείες στιγμιότυπα της αρνητικής κλάσης. Η μαύρη τελεία είναι άγνωστης κλάσης. Η απόστασή της από το κοντινότερο θετικό στιγμιότυπο είναι $d^+ = 0.1$ και η απόσταση από το κοντινότερο αρνητικό στιγμιότυπο είναι $d^- = 0.4$. Συνδυάζουμε τις δύο αυτές αποστάσεις σε μία τιμή σύμφωνα με τον τύπο.

$$p^+ = \frac{d^-}{d^- + d^+}$$

Έτσι εκφράζεται η βεβαιότητα κατά πόσο το υποπαράθυρο ανήκει στη θετική κλάση. Έαν το p^+ είναι πάνω απο κάποιο κατώφλι j^+ τότε το παράθυρο γίνεται αποδεκτό. Το κατώφλι που χρησιμοποιείται εχει την τιμή $j^+ = 0.65$.



Εικόνα 2.8

2.4 Εκμάθηση (Learning)

Σε αυτό το κεφάλαιο θα περιγράψουμε τον τρόπο με τον οποίο συνδυάζονται τα αποτελέσματα του ανιχνευτή και του εντοπιστή, για τους οποίους μιλήσαμε στις δύο προηγούμενες ενότητες. Στη συνέχεια, αφου ελέγξουμε το αποτέλεσμα αυτής της συγχώνευσης, προχωράμε στη διαδικασία της εκμάθησης.

2.4.1 Συγχώνευση και Εγκυρότητα

Η απόφαση που θα πάρουμε για να συνδυάσουμε τα αποτελέσματα βασίζεται στον αριθμό των εντοπισμών, την τιμή σιγουριάς τους και τη σιγουριά του αποτελέσματος του ανιχνευτή. Εάν ο εντοπιστής δώσει μόνο ένα αποτέλεσμα με σιγουριά μεγαλύτερη από το αποτέλεσμα του ανιχνευτή, τότε αυτό εκχωρούμε και ως τελικό αποτέλεσμα της συγχώνευσης. Αυτό σημαίνει ότι θα γίνει επανεκκίνηση του ανιχνευτή από αυτό το σημείο. Αντίθετα αν έχουμε περισσότερα από ένα αποτελέσματα απο τον εντοπιστή ή μόνο ένα αποτέλεσμα με λιγότερη τιμή σιγουριάς από αυτό του ανιχνευτή, τότε σαν τελικό αποτέλεσμα εκχωρούμε το αποτέλεσμα του ανιχνευτή. Σε όλες τις υπόλοιπες περιπτώσεις το τελικό αποτέλεσμα παραμένει κενό, γεγονός το οποίο σημαίνει πως ο στόχος μας δεν είναι ορατός στο συγκεκριμένο στιγμιότυπο.

Για να προχωρήσουμε στο βήμα της εκμάθησης πρέπει το τελικό αποτέλεσμα να είναι και έγκυρο. Αυτό συμβαίνει στις δυο επόμενες περιπτώσεις, οι οποίες προϋποθέτουν να μην έχει γίνει επανεκκίνηση του ανιχνευτή από τον εντοπιστή.

- Το τελικό αποτέλεσμα είναι έγκυρο στην περίπτωση που ο ανιχνευτής δώσει ένα αποτέλεσμα το οποίο έχει τιμή σιγουριάς μεγαλύτερη απο θ^+
- Το τελικό αποτέλεσμα είναι επίσης έγκυρο εάν το προηγούμενο αποτέλεσμα ήταν έγκυρο και ο ανιχνευτής δώσει ένα αποτέλεσμα με τιμή σιγουριάς μεγαλύτερη απο θ^-

Σε όλες τις άλλες περιπτώσεις το τελικό αποτέλεσμα δεν είναι έγκυρο. Το κατώφλι θ^+ υποδεικνύει ότι το αποτέλεσμα ανήκει στη θετική κλάση. Το κατώφλι θ^- υποδεικνύει ότι το αποτέλεσμα ανήκει στην αρνητική κλάση. Το πρώτο πλαίσιο οριοθέτησης θεωρείται ότι είναι έγκυρο και ανήκει στη θετική κλάση.

2.4.2 P/N-learning

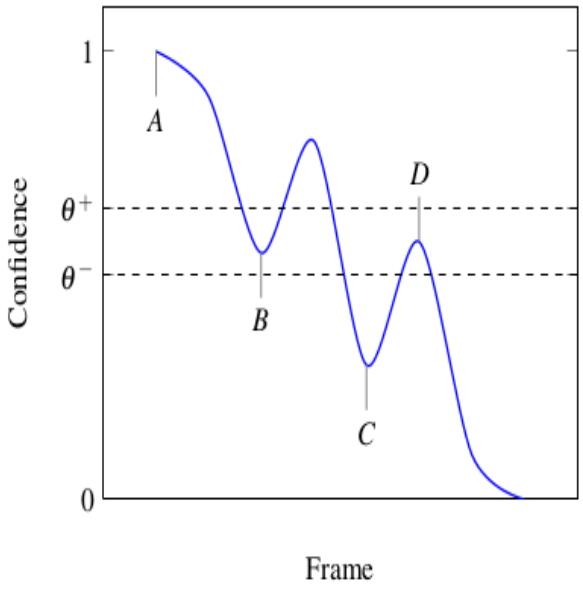
Στην μηχανική εκμάθηση υπάρχουν δύο κύριοι διαφορετικοί τύποι διεργασιών. Η επιβλεπόμενη μάθηση στην οποία τα δεδομένα εκπαίδευσης δημιουργούνται και χωρίζονται σε κλάσεις χειροκίνητα, έχοντάς τα ουσιαστικά σαν ζευγάρια (x_i, y_i) , με x_iνα είναι το παράδειγμα εκμάθησης και το y_i η αντίστοιχη ετικέτα κλάσης. Η άλλη διεργασία στη μηχανική εκμάθηση είναι η μάθηση χωρίς επίβλεψη. Εδώ τα δεδομένα μας δεν έχουν ετικέτες. Ο στόχος αυτής της διεργασίας είναι να χωρίσει αυτά τα δεδομένα σε ομάδες, με μεθόδους όπως η ομαδοποίηση και η μείωση της διαστάσης των δεδομένων.

Ανάμεσα σε αυτές τις δύο διεργασίες υπάρχει η ημι-επιβλεπόμενη μάθηση. Σε αυτήν τη μέθοδο υπάρχουν δεδομένα με ετικέτες καθώς και δεδομένα χωρίς ετικέτες. Στο πρόβλημά μας έχουμε μόνο ένα παράδειγμα με ετικέτα, το πρώτο πλαίσιο οριοθέτησης που μας δίνει ο χρήστης. Το πρόγραμμα OpenTLD για την εκμάθηση των δεδομένων χρησιμοποιεί μια μέθοδο γνωστή ως εκμάθηση-P/N(P/N learning). Σε αυτήν υπάρχουν δύο τύποι περιορισμών:

- ένας P-περιορισμός που βρίσκει τα ψευδώς αρνητικά και τα προσθέτει στα θετικά στιγμιότυπα εκπαίδευσης και
- ένας N-περιορισμός που βρίσκει τα ψευδώς θετικά και τα προσθέτει στα αρνητικά στιγμιότυπα εκπαίδευσης

Ο P-περιορισμός απαιτεί όλα τα πλαίσια τα οποία έχουν υψηλή επικάλυψη με το τελικό αποτέλεσμα να κατηγοριοποιούνται σαν θετικά στιγμιότυπα. Ο N-περιορισμός απαιτεί όλα τα πλαίσια τα οποία δεν έχουν υψηλή επικάλυψη με το τελικό αποτέλεσμα να κατηγοριοποιούνται σαν αρνητικά στιγμιότυπα. Θεωρούμε ότι ένα πλαίσιο οριοθέτησης έχει υψηλή επικάλυψη με το τελικό αποτέλεσμα, αν επικαλύπτονται τουλάχιστον κατά 60% .

Στην εικόνα 2.9 βλέπουμε πότε προστίθενται πρότυπα. Στο σημείο A, η ανίχνευση ξεκινάει με υψηλή τιμή σιγουριάς και έκγυρο αποτέλεσμα. Η εκμάθηση πραγματοποιείται, αλλά δεν προστίθενται κάποια θετικά στιγμιότυπα γιατί η τιμή σιγουριάς είναι μεγαλύτερη απο θ^+. Στη συνέχεια στο σημείο B η τιμή σιγουριάς πέφτει κάτω απο το θ^+ αλλά παραμένει πάνω απο το θ^-. Σύμφωνα με αυτά που έχουμε αναφέρει πιο πριν το αποτέλεσμα είναι έγκυρο. Σε αυτήν την περίπτωση προστίθενται θετικά πρότυπα, και αυτή η διαδικασία οδηγεί σε αύξηση της τιμής σιγουριάς. Στο σημείο C η τιμή σιγουριάς πέφτει κάτω απο το θ^-, και αυτό σημαίνει οτι το τελικό αποτέλεσμα δεν είναι πια έγκυρο. Το σημείο D βρίσκεται στο ίδιο επίπεδο με το σημείο B, παρ΄όλάυτά εκμάθηση δεν πραγματοποιείται γιατί το τελικό αποτέλεσμα δεν είναι έγκυρο



Εικόνα 2.9

2.5 Συμπέρασμα

Σε αυτό το κεφάλαιο περιγράψαμε τον τρόπο με τον οποίο το πρόγραμμα OpenTLDαναζητά και εντοπίζει έναν στόχο που έχει αρχικοποιήσει ο χρήστης. Θα χρησιμοποιήσουμε λοιπόν αυτό το πρόγραμμα με σκοπό να δίνουμε στο ρομπότ μας το στίγμα του στόχου. Τοποθετούμε λοιπόν ένα φορητό υπολογιστή στο επάνω μέρος του ρομπότ ο οποίος εκτελεί το πρόγραμμα OpenTLD και ανάλογα με το στίγμα, το ρομπότ θα εκτελεί και την κατάλληλη ενέργεια. Να σημειώσουμε πως η έξοδος του προγράμματος είναι ένα πλαίσιο που περιέχει το στόχο που μας ενδιαφέρει. Για να πάρουμε μία αντιπροσωπευτική θέση του στόχου αρκεί να διαιρέσουμε τις διαστάσεις του πλαισίου με το δύο. Έτσι καταλήγουμε με τις συντεταγμένες x,yτις οποίες στέλνουμε στο ρομπότ. Το πρόβλημα που θα μας απασχολήσει στο επόμενο κεφάλαιο είναι αυτό της επικοινωνίας του φορητού υπολογιστή και του ρομπότ.

Κεφάλαιο 3

Επικοινωνία OpenTLD-Peoplebot

3.1 Περιγραφή

Όπως ήδη περιγράψαμε το πρόγραμμα OpenTLD μέτα την επεξεργασία της εικόνας, δίνει σαν αποτέλεσμα δύο αριθμούς x,y οι οποίοι αποτελούν και τις συντεταγμένες του αντικειμένου που μας ενδιαφέρει στην εικόνα. Αυτοί οι αριθμοί όμως βρίσκονται στον φορητό υπολογιστή που είναι τοποθετημένος πάνω στο ρομπότ. Έτσι, πρέπει με κάποιο τρόπο να μεταφέρουμε αυτές τις συντεταγμένες στο ρομπότ, ώστε σύμφωνα με αυτές να εκτελέσει την κατάλληλη ενέργεια.

Για την ενδοεπικοινωνία του φορητού υπολογιστή με το ρομπότ και την μεταφορά των συντεταγμένων χρησιμοποιούμε το μοντέλο πελάτη-εξυπηρετητή (client-server model).

3.2 Μοντέλο Πελάτη-Εξυπηρετητή (Client-Server model)

Οι περισσότερες διεργασίες που επικοινωνούν μέσω δικτύου χρησιμοποιούν το μοντέλο πελάτη-εξυπηρετητή (client- server model). Οι δύο αυτοί όροι αναφέρονται στις δύο διεργασίες οι οποίες επικοινωνούν. Ο πελάτης συνδέεται στον εξυπηρετητή συνήθως για να ζητήσει μία πληροφορία.

Σε αυτό το σημείο πρέπει να σημειώσουμε ότι ο πελάτης πρέπει να γνωρίζει την ύπαρξη καθώς και την διεύθυνση του εξυπηρετητή, ενώ αντίθετα ο εξυπηρετητής δεν χρειάζεται να ξέρει τίποτα από τα δύο ούτως ώστε να είναι εφικτή η σύνδεση.

Οι διαδικασίες-βήματα που λαμβάνουν χώρα έτσι ώστε να γίνει η σύνδεση πελάτη-εξυπηρετητή είναι λίγο διαφορετικές για κάθε διεργασία, αλλά και οι δύο περιλαμβάνουν τη βασική ιδέα της υποδοχής (socket). Επιπλέον να σημειώσουμε πως μετά τη σύνδεση και οι δύο πλευρές μπορούν να στείλουν και να λάβουν δεδομένα. Τα βήματα αυτά έχουν ως εξής:

1. ο εξυπηρετητής δημιουργεί μια υποδοχή, στην οποία θα ακούει τα αιτήματα των πελατών και περιμένει πιθανές προσπάθειες σύνδεσης από τους πελάτες.

2. ο πελάτης δημιουργεί και αυτός μια υποδοχή στην πλευρά του και προσπαθεί να συνδεθεί στον εξυπηρετητή

3. ο εξυπηρετητής αποδέχεται τη σύνδεση και η μεταφορά δεδομένων μπορεί να ξεκινήσει

4. όταν τελειώσει η μεταφορά των δεδομένων, οι δύο πλευρές μπορούν να σταματήσουν τη σύνδεση

3.3 Υποδοχές (Sockets)

Μία υποδοχή (socket) είναι ουσιαστικά η αρχή ή το τέλος μίας δικτυακής σύνδεσης, όπως περιγράψαμε παραπάνω. Αν θέλουμε να αποδώσουμε μεταφορικά μία υποδοχή, θα πρέπει να φανταστούμε ότι είναι σαν την είσοδο ή την έξοδο από ένα τούνελ. Μία υποδοχή μεταφράζει όλα τα εισερχόμενα δεδομένα σε τέτοια μορφή ώστε να διαβάζονται από τον άνθρωπο και όλα τα εξερχόμενα δεδομένα τα μετατρέπει σε πακέτα δικτύου.

Η υποδοχή χαρακτηρίζεται από μια διεύθυνση IP μαζί με έναν αριθμό θύρας. Ένα ζευγάρι διεργασιών που επικοινωνούν, χρησιμοποιούν δύο υποδοχές, μια υποδοχή για κάθε διεργασία.

3.4 Διαδικασία Επικοινωνίας

Τώρα θα περιγράψουμε τη διαδικασία με την οποία επικοινωνεί το ρομπότ με τον φορητό υπολογιστή για τη μεταφορά των συνστεταγμένων του στόχου που μας ενδιαφέρει.

Το ρομπότ, στο παράδειγμά μας, λειτουργεί σαν εξυπηρετητής σύμφωνα με το μοντέλο που περιγράψαμε στην προηγούμενη ενότητα. Για αυτό το λόγο λοιπόν δημιουργεί μια υποδοχή τύπου εξυπηρετητή, δηλώνοντας δηλαδή μια θύρα στην οποία θα 'ακούει', θα περιμένει να λάβει τα δεδομένα, δηλαδή τις συντεταγμένες του στόχου, και μόλις τα λάβει θα εκτελέσει την κατάλληλη ενέργεια. Το πρόγραμμα OpenTLD το οποίο βρίσκεται στον φορητό υπολογιστή θα λειτουργήσει αντίστοιχα σαν πελάτης, δημιουργώντας μια υποδοχή τύπου πελάτη. Για να δημιουργηθεί αυτή η υποδοχή είναι απαραίτητο να γίνει γνωστή η IPδιεύθυνση του ρομπότ, καθώς και η θύρα στην οποία περιμένει τα δεδομένα. Με τη δημιουργία αυτών των υποδοχών, έχουμε ταυτόχρονα και την κατασκευή του καναλιού επικοινωνίας ρομπότ-OpenTLD.

Εφ΄ όσον εκτελεστούν όλες οι απαραίτητες ενέργειες για τον εντοπισμό του στόχου από το πρόγραμμα OpenTLD , στη συνέχεια το πρόγραμμα στέλνει τις συντεταγμένες του στόχου, μέσω του καναλιού επικοινωνίας που περιγράψαμε, στο ρομπότ

Ένα ερώτημα το οποίο μπορεί να μας απασχολήσει σε αυτό το σημείο είναι γιατί έγινε η συγκεκριμένη εφαρμογή του μοντέλου πελάτη-εξυπηρετητή, δηλαδή του ρομπότ ως εξυπηρετητή και του προγράμματος OpenTLD ως πελάτη.

Στην περίπτωση που το πρόγραμμα OpenTLD θα δρούσε ως εξυπηρετητής, απλά θα 'άκουγε' σε μία θύρα και θα περίμενε κάποιον πελάτη να συνδεθεί για να του στείλει τις συντεταγμένες του στόχου. Το ρομπότ ως πελάτης θα συνδεόταν στον εξυπηρετητή και θα απαιτούσε τις συντεταγμένες του στόχου για να εκτελέσει κάποια ενέργεια. Αυτό ίσως να φαίνεται σαν πιο λογική ανάθεση των

ρόλων πελάτη και εξυπηρετητή στο πρόγραμμα OpenTLD και το ρομπότ. Σε αυτήν την περίπτωση όμως, αντιμετωπίζουμε σοβαρά προβλήματα συγχρονισμού των δυο διεργασιών. Έστω ότι το ρομπότ χρειάζεται τις συντεταγμένες του στόχου για να εκτελέσει μία ενέργεια. Αφου λάβει τις συντεταγμένες του στόχου και αρχίσει να εκτελεί κάποια ενέργεια σύμφωνα με αυτές, υπάρχει κίνδυνος ο στόχος να αλλάξει θέση και η ενέργεια που εκτελεί το ρομπότ να μην συμφωνεί με τη νέα αυτή θέση. Το ουσιαστικό πρόβλημά μας σε αυτήν την προσέγγιση είναι ότι δεν ξέρουμε πότε να σταματήσουμε την ενέργεια που εκτελεί το ρομπότ και να ζητήσουμε νέες συντεταγμένες από το πρόγραμμα OpenTLD. Ένα ακόμη βασικό πρόβλημά μας είναι ότι σαν εξυπηρετητής το πρόγραμμα OpenTLD θα έπρεπε να σταματάει όλες τις διαδικασίες που έχουν να κάνουν με την έυρεση του στόχου και να περιμένει κάποιον πελάτη να συνδεθεί. Κατά τη διάρκεια αυτής της αναμονής η θέση του στόχου δεν θα ήταν γνωστή, κάτι το οποίο θα δυσκόλευε ακόμη περισσότερο το συγχρονισμό του ρομπότ και της θέσης του στόχου.

Λόγω όλων αυτών των προβλημάτων τα οποία περιγράψαμε, εφαρμόζουμε το μοντέλο πελάτη-εξυπηρετητή αντίστροφα. Το ρομπότ λειτουργεί σαν εξυπηρετητής και κάθε φορά περιμένει να του σταλούν οι συντεταγμένες του στόχου για να εκτελέσει την κατάλληλη ενέργεια. Έτσι δεν έχουμε πρόβλημα για το πότε θα διακόψουμε την ενέργεια του ρομπότ για να το ενημερώσουμε για τη θέση του στόχου, αφου κάθε φορά που αλλάζει η θέση του στόχου, του στέλνεται αυτόματα από το πρόγραμμα OpenTLD. Αυτό λειτουργεί σαν πελάτης και κάθε φορά δημιουργεί μία υποδοχή και στέλνει τις συντεταγμένες του στόχου, χωρίς καμία διακοπή των υπόλοιπων διαδικασιών του.

Σε αυτό το κεφάλαιο περιγράψαμε τον τρόπο τον οποίο επιλέξαμε για να λύσουμε το πρόβλημα της επικοινωνίας του φορητού υπολογιστή και του ρομπότ. Έτσι οι συντεταγμένες του στόχου που υπολογίζονται από το πρόγραμμα OpenTLD φτάνουν τελικά στο ρομπότ μέσω των υποδοχών. Στο επόμενο κεφάλαιο θα περιγράψουμε τις ενέργειες που εκτελεί το ρομπότ σύμφωνα με τις συντεταγμένες του στόχου.

Κεφάλαιο 4

Αναζήτηση στόχων

4.1 Σκοπός

Σε αυτό το κεφάλαιο θα περιγράψουμε τον τρόπο με τον οποίο το ρομπότ που έχουμε στη διάθεση μας θα αναζητεί κάποιο στόχο. Το ρομπότ που θα χρησιμοποιήσουμε για αυτό το σκοπό είναι το peoplebot της Adept MobileRobots (Εικόνα 4.1). Για την αναζήτηση των στόχων θα χρειαστούμε τη βοήθεια του προγράμματος OpenTLD που έχουμε περιγράψει στο Κεφάλαιο 2. Αυτό θα επικοινωνεί με το ρομπότ, σύμφωνα με τη περιγραφή του μοντέλου πελάτη-εξυπηρετητή που βρίσκεται στο Κεφάλαιο 3.

Εικόνα 4.1

4.2 Προϋποθέσεις

Πριν ξεκινήσουμε να αναζητούμε στόχους με το ρομπότ μας, θα θεωρήσουμε ότι υπάρχει υλοποιημένο πρόγραμμα το οποίο βοηθάει το ρομπότ να εντοπίζει τη θέση του, και χρησιμοποιεί την πληροφορία του προγράμματος OpenTLD για να καθοδηγήσει το ρομπότ στο να ακολουθεί το στόχο []. Επίσης θα μας ενημερώνει για το πότε το ρομπότ έχει χάσει από το οπτικό του πεδίο το στόχο.

Μία ακόμη βασική προϋπόθεση είναι η τοποθέτηση ενός φορητού υπολογιστή με μία κάμερα στο επάνω μέρος του ρομπότ. Αυτή η κάμερα ουσιαστικά θα αποτελεί και τα 'μάτια' του ρομπότ μας. Τοποθετούμε την κάμερα με τέτοιο προσανατολισμό έτσι ώστε το οπτικό της πεδίο να περιλαμβάνει το χώρο ο οποίος βρίσκεται στο μπροστινό μέρος του ρομπότ. Σε αυτόν τον φορητό υπολογιστή που βρίσκεται πάνω στο ρομπότ εγκαθιστούμε και το πρόγραμμα OpenTLD το οποίο θα αναζητά το στόχο μας χρησιμοποιώντας την ίδια κάμερα. Μέσω της συνεχώς ενδοεπικοινωνίας με το φορητό υπολογιστή το ρομπότ θα γνωρίζει ανα πάσα στιγμή τη θέση του στόχου καθώς και αν αυτός βρίσκεται στο οπτικό του πεδίο. Βασισμένοι σε αυτές τις προϋπόθεσεις θα περιγράψουμε τη διαδικασία της αναζήτησης στόχων.

4.3 Διαδικασία αναζήτησης

Σε αυτό το σημείο θα προσπαθήσουμε να περιγράψουμε τη διαδικασία την οποία ακολουθούμε έτσι ώστε το ρομπότ να αναζητήσει το στόχο του, όσο πιο αποδοτικά γίνεται κάθε φορά. Βρισκόμαστε λοιπόν στο σημείο όπου το ρομπότ ακολουθεί κάποιο στόχο που του έχει υποδείξει ο χρήστης, με τη βοήθεια του προγράμματος OpenTLD . Τη στιγμή που το πρόγραμμα πάψει να βρίσκει το στόχο στο οπτικό του πεδίο, στέλνει αντίστοιχο σήμα στο ρομπότ μας. Τη χρονική στιγμή που θα λάβουμε αυτό το σήμα θα ξέρουμε πως πρέπει να αρχίσουμε τη διαδικασία αναζήτησης του στόχου.

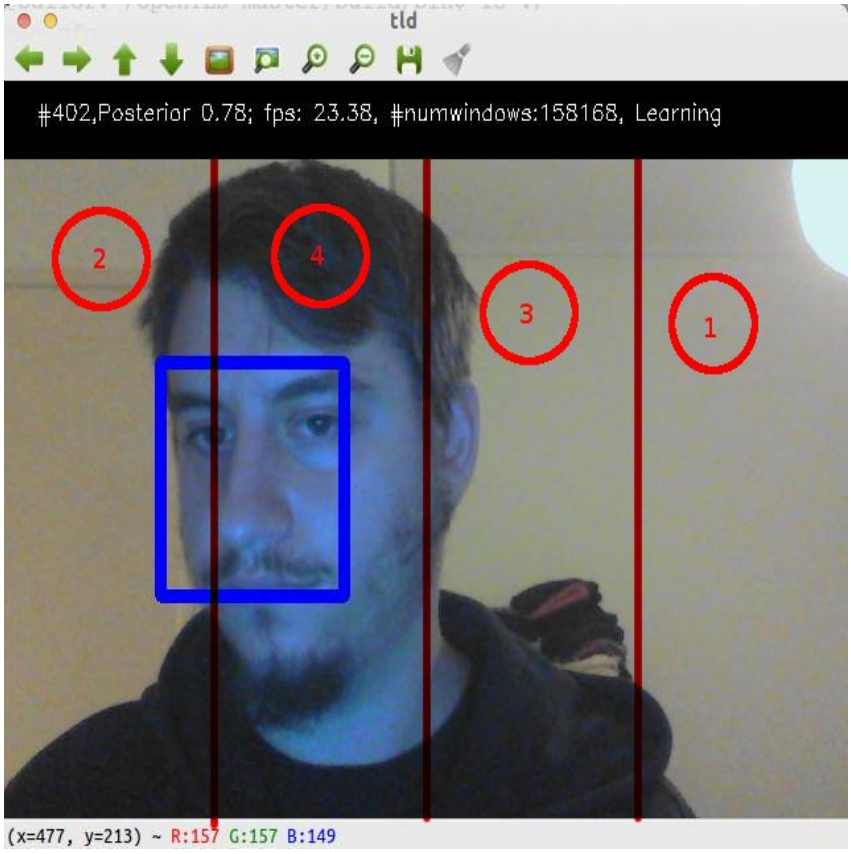
Απο τη στιγμή που ο στόχος έχει χαθεί από το οπτικό πεδίο, ένας απλός τρόπος να τον αναζητήσουμε είναι να δώσουμε εντολή στο ρομπότ να πραγματοποιήσει μία περιστροφή 360° γύρω από τον εαυτό του. Με αυτόν τον τρόπο η κάμερα που είναι προσαρμοσμένη στον φορητό υπολογιστή στο επάνω μέρος του ρομπότ θα περιστραφεί και αυτή. Έτσι, εάν ο στόχος βρίσκεται στο χώρο γύρω από το ρομπότ, τότε το πρόγραμμα OpenTLD θα τον ξαναεντοπίσει και θα ενημερώσει το ρομπότ πως η αναζήτηση πρέπει να λάβει τέλος.

Μία μικρή βελτίωση σε αυτόν τον τρόπο αναζήτησης του στόχου είναι πως αφού μας δίνεται η θέση του πριν εξαφανιστεί, τότε μπορούμε να υποθέσουμε και προς τα πού κινήθηκε. Για να το κάνουμε αυτό ελέγχουμε τις τελευταίες γνωστές συντεταγμένες του στόχου και αν αυτές βρίσκονται στα δεξιά από τη μέση του οπτικού πεδίου του ρομπότ, τότε το στρέφουμε προς τα δεξιά. Πράττουμε αναλόγως, αν οι συντεταγμένες βρίσκονται στα αριστερά του οπτικού πεδίου. Έτσι λύνουμε το πρόβλημα της αναζήτησης του στόχου μας.

Παρ΄ ότι ο τρόπος που μόλις περιγράψαμε για να προσεγγίσουμε το προβλημά μας δεν βασίζεται σε λάθος λογική, τα πειράματά μας έδειξαν πως αντιμετωπίζουμε αρκετές φορές προβλήματα στο να επανεντοπίσουμε το στόχο. Μία κύρια αιτία των προβλημάτων αυτών είναι το ίδιο το πρόγραμμα OpenTLD . Στην περίπτωση που το μοντέλο του στόχου δεν είναι καλά εκπαιδευμένο είναι δυνατόν να δημιουργηθούν προβλήματα στον εντοπισμό του. Αυτό σημαίνει πως το πρόγραμμα OpenTLD μπορεί να αναφέρει πως ο στόχος έχει χαθεί, αλλά αυτός να βρίσκεται ακόμη στο οπτικό πεδίο του ρομπότ. Αν σε αυτήν την περίπτωση το ρομπότ αρχίσει να περιστρέφεται τότε θα χάσει εντελώς το στόχο από το οπτικό του πεδίο. Αυτό δεν θέλουμε να συμβεί. Για να λύσουμε αυτό το πρόβλημα μπορούμε να σαρώσουμε ένα μέρος του οπτικού πεδίου πριν ολοκληρώσουμε μια περιστροφή. Εμπειρικά βρήκαμε ότι αρκεί να σαρώνουμε κάθε φορά ένα τόξο 140° μπροστά από το ρομπότ. Για να το επιτύχουμε αυτό τελικά, χωρίζουμε το οπτικό πεδίο του ρομπότ σε τέσσερα ίσα μέρη, όπως φαίνεται στην εικόνα 4.2 και ακολουθούμε την εξής λογική:

- εάν οι τελευταίες συντεταγμένες του στόχου βρίσκονται στο δεξιό τμήμα του οπτικού πεδίου (τμήμα 1) τότε δίνουμε εντολή στο ρομπότ να περιστραφεί προς τα δεξιά
- εάν οι τελευταίες συντεταγμένες του στόχου βρίσκονται στο αριστερό τμήμα του οπτικού πεδίου (τμήμα 2) τότε δίνουμε εντολή στο ρομπότ να περιστραφεί προς τα αριστερά
- εάν οι τελευταίες συντεταγμένες του στόχου βρίσκονται στο τμήμα 3 του οπτικού πεδίου στρέφουμε το ρομπότ πρώτα προς τα δεξιά κατά 70° και στη συνέχεια κατά 140° αριστερά
- εάν οι τελευταίες συντεταγμένες του στόχου βρίσκονται στο τμήμα 4 του οπτικού πεδίου στρέφουμε το ρομπότ πρώτα προς τα αριστερά κατά 70° και στη συνέχεια κατά 140° δεξιά

Εκτελώντας τις τελευταίες δύο ενέργειες ουσιαστικά καταφέρνουμε να σαρώνουμε τον χώρο μπροστά από το ρομπότ και να δίνουμε την ευκαιρία στο πρόγραμμα OpenTLD να επανεντοπίσει το στόχο ο οποίος βρίσκεται στο οπτικό του πεδίο. Εφαρμόζοντας αυτήν τη λογική έχουμε δημιουργήσει έναν τρόπο αναζήτησης στόχων που δουλεύει πολύ αποδοτικά, πάντα σύμφωνα με τα πειράματά μας.



Παράρτημα Α΄

Κώδικας

```
#include "Aria.h"
#include "iostream"

class Search
{
  private:
    ArRobot *myRobot;
    ArSonarDevice *mySonar;
    int width;
    int stopDistance;
    int timeSet;
  public:
    Search(ArRobot *robot, ArSonarDevice *sonar);
    int seek(int lastPositionXY[2], int currentPositionXY[2]);
};
```

source/Search.h

```
#include "Search.h"
#include "Movement.h"

Search::Search(ArRobot *robot, ArSonarDevice *sonar)
{
  myRobot = robot;
  mySonar = sonar;
  width = 640;
  stopDistance = 250;
  timeSet = 0;
  //timeSearch = 0;
}

int Search::seek(int lastPositionXY[2],int currentPositionXY[2])
{
  Movement move = Movement(myRobot);
  double range = mySonar->currentReadingPolar(-70, 70) - myRobot->
    getRobotRadius();

  ArTime start;
        //using timeSet to start the counting the first time we
    enter Search
        if (timeSet == 0)
                timeSet = start.getSec();

```

```
        //if searching for too many secs switch to goTo
        if(start.getSec() - timeSet > 24 )
        { timeSet = 0;
    return 3;
  }

        //if we have found a target
        if( currentPositionXY[0] != -1 )
        {       timeSet = 0;
                std::cout << "Found target " << currentPositionXY
  [0] << std::endl;
                return 1;
        }

        // if we are safe
        if(range > stopDistance)
        {       //if the target was last seen on the far right
                if( lastPositionXY[0] > width/2 + width/4)
                        move.turn(-20);
                //if the target was last seen on the far left
                else if( lastPositionXY[0] < width/2 - width/4)
                        move.turn(20);
                //if the target was last seen in the middle and a
  bit to the right
                else if(lastPositionXY[0] <= width/2 + width/4)
                {
          //trying to search for the target in a 140 degrees range

          //for 2 secs turning right
          if(start.getSec() - timeSet < 2 )
            move.turn(-36);
          //for 4 secs turning left
          else if(start.getSec() - timeSet < 6)
            move.turn(36);
          //if failed to find the target in these 6 secs try a full
   circle
          else
            move.turn(-20);
        }
        //if the target was last seen in the midle and a bot to the
   left
        else if(lastPositionXY[0] >= width/2 - width/4)
        {
          //trying to search for the target in a 140 degrees range

          //for 2 secs turning left
          if(start.getSec() - timeSet < 2 )
            move.turn(36);
          //for 4 secs turning right
          else if(start.getSec() - timeSet < 6)
            move.turn(-36);
          //if failed to find the target in these 6 secs try a full
   circle
          else
            move.turn(20);
        }
    }

    return 2;
  }

  //if we are not safe
```

```
81    else
      { move.stop();
83      return 3;
      }
85  }
```

source/Search.cpp

```
1   // Definition of the ServerSocket class

3   #ifndef ServerSocket_class
    #define ServerSocket_class
5
    #include "Socket.h"
7


9   class ServerSocket : private Socket
    {
11   public:

13    ServerSocket ( int port );
      ServerSocket (){};
15    virtual ~ServerSocket();

17    const ServerSocket& operator << ( const int[2] ) const;
      const ServerSocket& operator >> ( int[2] ) const;
19
      void accept ( ServerSocket& );
21
    };
23


25  #endif
```

source/ServerSocket.h

```
1   // Implementation of the ServerSocket class

3   #include "ServerSocket.h"
    #include "SocketException.h"
5


7   ServerSocket::ServerSocket ( int port )
    {
9     if ( ! Socket::create() )
        {
11        throw SocketException ( "Could not create server socket." );
        }
13
      if ( ! Socket::bind ( port ) )
15      {
          throw SocketException ( "Could not bind to port." );
17      }

19    if ( ! Socket::listen() )
        {
21        throw SocketException ( "Could not listen to socket." );
        }
23
    }
25
    ServerSocket::~ServerSocket()
```

```
27 {
   }
29
31 const ServerSocket& ServerSocket::operator << ( const int s[2] )
       const
   {
33   if ( ! Socket::send ( s ) )
       {
35       throw SocketException ( "Could not write to socket." );
       }
37
     return *this;
39
   }
41
43 const ServerSocket& ServerSocket::operator >> ( int s[2] ) const
   {
45   if ( ! Socket::recv ( s ) )
       {
47       throw SocketException ( "Could not read from socket." );
       }
49
     return *this;
51 }
53 void ServerSocket::accept ( ServerSocket& sock )
   {
55   if ( ! Socket::accept ( sock ) )
       {
57       throw SocketException ( "Could not accept socket." );
       }
59 }
```

source/ServerSocket.cpp

```
1  // Definition of the Socket class
3  #ifndef Socket_class
   #define Socket_class
5
7  #include <sys/types.h>
   #include <sys/socket.h>
9  #include <netinet/in.h>
   #include <netdb.h>
11 #include <unistd.h>
   #include <string>
13 #include <arpa/inet.h>
15
   const int MAXHOSTNAME = 200;
17 const int MAXCONNECTIONS = 5;
   const int MAXRECV = 500;
19
   class Socket
21 {
    public:
23   Socket();
     virtual ~Socket();
25
```

```
  // Server initialization
  bool create();
  bool bind ( const int port );
  bool listen() const;
  bool accept ( Socket& ) const;

  // Client initialization
  bool connect ( const std::string host, const int port );

  // Data Transimission
  bool send ( const int[2] ) const;
  int recv ( int[2] ) const;


  void set_non_blocking ( const bool );

  bool is_valid() const { return m_sock != -1; }

 private:

  int m_sock;
  sockaddr_in m_addr;


};

#endif
```

source/Socket.h

```
// Implementation of the Socket class.


#include "Socket.h"
#include "string.h"
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <iostream>

Socket::Socket() :
  m_sock ( -1 )
{

  memset ( &m_addr,
     0,
     sizeof ( m_addr ) );

}

Socket::~Socket()
{
  if ( is_valid() )
    ::close ( m_sock );
}

bool Socket::create()
{
  m_sock = socket ( AF_INET,
        SOCK_STREAM,
        0 );
```

```
33    if ( ! is_valid() )
        return false;
35
37    // TIME_WAIT - argh
      int on = 1;
39    if ( setsockopt ( m_sock, SOL_SOCKET, SO_REUSEADDR, ( const char*
         ) &on, sizeof ( on ) ) == -1 )
        return false;
41
43    return true;
45  }
47
49  bool Socket::bind ( const int port )
    {
51
      if ( ! is_valid() )
53      {
          return false;
55      }
57
59    m_addr.sin_family = AF_INET;
      m_addr.sin_addr.s_addr = INADDR_ANY;
61    m_addr.sin_port = htons ( port );
63    int bind_return = ::bind ( m_sock,
              ( struct sockaddr * ) &m_addr,
65            sizeof ( m_addr ) );
67
      if ( bind_return == -1 )
69      {
          return false;
71      }
73    return true;
    }
75
77  bool Socket::listen() const
    {
79    if ( ! is_valid() )
        {
81        return false;
        }
83
      int listen_return = ::listen ( m_sock, MAXCONNECTIONS );
85
87    if ( listen_return == -1 )
        {
89        return false;
        }
91
      return true;
```

```
93 }
95
bool Socket::accept ( Socket& new_socket ) const
97 {
  int addr_length = sizeof ( m_addr );
99   new_socket.m_sock = ::accept ( m_sock, ( sockaddr * ) &m_addr, (
    socklen_t * ) &addr_length );

101   if ( new_socket.m_sock <= 0 )
    return false;
103   else
    return true;
105 }
107
bool Socket::send ( const int s[2] ) const
109 {
  int status = ::send ( m_sock, s, sizeof(int[2]), MSG_NOSIGNAL );
111   if ( status == -1 )
    {
113       return false;
    }
115   else
    {
117       return true;
    }
119 }
121
int Socket::recv ( int s[2] ) const
123 {
  //char buf [ MAXRECV + 1 ];
125   int buf[2];

127   for(int i=0; i<2; i++){
  buf[i] = 0;
129     s[i] = 0;
  }
131
  //memset ( buf, 0, MAXRECV + 1 );
133
  int status = ::recv ( m_sock, buf, sizeof(int[2]), 0 );
135
  if ( status == -1 )
137     {
      std::cout << "status == -1   errno == " << errno << "  in
    Socket::recv\n";
139       return 0;
    }
141   else if ( status == 0 )
    {
143       return 0;
    }
145   else
    {
147       for(int i=0; i<2; i++){
  s[i] = buf[i];
149     }
      return status;
151     }
}
```

```
153
155
bool Socket::connect ( const std::string host, const int port )
157 {
  if ( ! is_valid() ) return false;
159
  m_addr.sin_family = AF_INET;
161   m_addr.sin_port = htons ( port );

163   int status = inet_pton ( AF_INET, host.c_str(), &m_addr.sin_addr
    );

165   if ( errno == EAFNOSUPPORT ) return false;

167   status = ::connect ( m_sock, ( sockaddr * ) &m_addr, sizeof (
    m_addr ) );

169   if ( status == 0 )
    return true;
171   else
    return false;
173 }

175 void Socket::set_non_blocking ( const bool b )
{
177
  int opts;
179
  opts = fcntl ( m_sock,
181     F_GETFL );

183   if ( opts < 0 )
    {
185       return;
    }
187
  if ( b )
189     opts = ( opts | O_NONBLOCK );
  else
191     opts = ( opts & ~O_NONBLOCK );

193   fcntl ( m_sock,
    F_SETFL,opts );
195
}
```

source/Socket.cpp

```
/*  Copyright 2011 AIT Austrian Institute of Technology
2 *
*   This file is part of OpenTLD.
4 *
*   OpenTLD is free software: you can redistribute it and/or modify
6 *   it under the terms of the GNU General Public License as
    published by
*    the Free Software Foundation, either version 3 of the License,
     or
8 *   (at your option) any later version.
*
10 *   OpenTLD is distributed in the hope that it will be useful,
*   but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
12 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   *   GNU General Public License for more details.
14 *
   *   You should have received a copy of the GNU General Public
       License
16 *   along with OpenTLD.  If not, see <http://www.gnu.org/licenses
       />.
   *
18 */
   /*
20  * MainX.cpp
    *
22  *  Created on: Nov 17, 2011
    *      Author: Georg Nebehay
24  *  Extended on: Feb, 2013
    *  Author: Evangelos Sigalas
26  *  Implementation of client-Socket communication with a server(
       robot)
    */
28
   #include "Main.h"
30
   #include "Config.h"
32 #include "ImAcq.h"
   #include "Gui.h"
34 #include "TLDUtil.h"
   #include "Trajectory.h"
36
   using namespace tld;
38 using namespace cv;

40 void Main::doWork()
   {
42   Trajectory trajectory;
       IplImage *img = imAcqGetImg(imAcq);
44     Mat grey(img->height, img->width, CV_8UC1);
       cvtColor(cvarrToMat(img), grey, CV_BGR2GRAY);
46
       tld->detectorCascade->imgWidth = grey.cols;
48     tld->detectorCascade->imgHeight = grey.rows;
       tld->detectorCascade->imgWidthStep = grey.step;
50
     if(showTrajectory)
52   {
       trajectory.init(trajectoryLength);
54   }

56     /***EXTENDED CODE***/
       //Opening socket to communicate with robot server
58     ClientSocket client_socket ( "localhost", 30000 );

60     //flag to start the communication
       bool communicate = false;
62
       /***END OF EXTENDED CODE***/
64
       if(selectManually)
66     {

68         CvRect box;

70         if(getBBFromUser(img, box, gui) == PROGRAM_EXIT)
```

```
        {
            return;
        }

        if(initialBB == NULL)
        {
            initialBB = new int[4];
        }

        initialBB[0] = box.x;
        initialBB[1] = box.y;
        initialBB[2] = box.width;
        initialBB[3] = box.height;
    }

    FILE *resultsFile = NULL;

    if(printResults != NULL)
    {
        resultsFile = fopen(printResults, "w");
        if(!resultsFile)
        {
            fprintf(stderr, "Error: Unable to create results-file \"%s\"\n", printResults);
            exit(-1);
        }
    }

    bool reuseFrameOnce = false;
    bool skipProcessingOnce = false;

    if(loadModel && modelPath != NULL)
    {
        tld->readFromFile(modelPath);
        reuseFrameOnce = true;
    }
    else if(initialBB != NULL)
    {
        Rect bb = tldArrayToRect(initialBB);

        printf("Starting at %d %d %d %d\n", bb.x, bb.y, bb.width, bb.height);

        tld->selectObject(grey, &bb);
        skipProcessingOnce = true;
        reuseFrameOnce = true;
    }

    while(imAcqHasMoreFrames(imAcq))
    {
        double tic = cvGetTickCount();

        if(!reuseFrameOnce)
        {
            cvReleaseImage(&img);
            img = imAcqGetImg(imAcq);

            if(img == NULL)
            {
                printf("current image is NULL, assuming end of input.\n");
                break;
```

```
130             }
132             cvtColor(cvarrToMat(img), grey, CV_BGR2GRAY);
        }
134
        if(!skipProcessingOnce)
136     {
            tld->processImage(cvarrToMat(img));
138     }
        else
140     {
            skipProcessingOnce = false;
142     }

144 /***EXTENDED CODE***/
    //if communicate flag is true, start communication
146 if(communicate){
        if(tld->currBB != NULL){
148     try{
          int cor[2];
150       cor[0] = tld->currBB->x + tld->currBB->width/2;
          cor[1] = tld->currBB->y + tld->currBB->height/2;
152       client_socket << cor;
        }
154         catch ( SocketException& ) {}
    }
156 else{
    try{
158   int cor[2];
      cor[0] = -1;
160   cor[1] = -1;
      client_socket << cor;
162 }
        catch ( SocketException& ) {}
164 }
    }
166 /*** END OF EXTENDED CODE***/

168     if(printResults != NULL)
        {
170         if(tld->currBB != NULL)
            {
172             fprintf(resultsFile, "%d %.2d %.2d %.2d %.2d %f\n",
   imAcq->currentFrame - 1, tld->currBB->x, tld->currBB->y, tld->
  currBB->width, tld->currBB->height, tld->currConf);
            }
174         else
            {
176             fprintf(resultsFile, "%d NaN NaN NaN NaN NaN\n",
  imAcq->currentFrame - 1);
            }
178     }

180     double toc = (cvGetTickCount() - tic) / cvGetTickFrequency
    ();

182     toc = toc / 1000000;

184     float fps = 1 / toc;

186     int confident = (tld->currConf >= threshold) ? 1 : 0;
```

```
188         if(showOutput || saveDir != NULL)
            {
190             char string[128];

192             char learningString[10] = "";

194             if(tld->learning)
                {
196                 strcpy(learningString, "Learning");
                }
198
                sprintf(string, "#%d,Posterior %.2f; fps: %.2f, #
    numwindows:%d, %s", imAcq->currentFrame - 1, tld->currConf, fps
    , tld->detectorCascade->numWindows, learningString);
200             CvScalar yellow = CV_RGB(255, 255, 0);
                CvScalar blue = CV_RGB(0, 0, 255);
202             CvScalar black = CV_RGB(0, 0, 0);
                CvScalar white = CV_RGB(255, 255, 255);
204
                if(tld->currBB != NULL)
206             {
                    CvScalar rectangleColor = (confident) ? blue :
    yellow;
208                 cvRectangle(img, tld->currBB->tl(), tld->currBB->br
    (), rectangleColor, 8, 8, 0);

210         if(showTrajectory)
            {
212           CvPoint center = cvPoint(tld->currBB->x+tld->currBB->
    width/2, tld->currBB->y+tld->currBB->height/2);
              cvLine(img, cvPoint(center.x-2, center.y-2), cvPoint(
    center.x+2, center.y+2), rectangleColor, 2);
214           cvLine(img, cvPoint(center.x-2, center.y+2), cvPoint(
    center.x+2, center.y-2), rectangleColor, 2);
              trajectory.addPoint(center, rectangleColor);
216         }
            }
218       else if(showTrajectory)
          {
220         trajectory.addPoint(cvPoint(-1, -1), cvScalar(-1, -1, -1));
          }
222
          if(showTrajectory)
224       {
            trajectory.drawTrajectory(img);
226       }

228         CvFont font;
            cvInitFont(&font, CV_FONT_HERSHEY_SIMPLEX, .5, .5, 0,
    1, 8);
230         cvRectangle(img, cvPoint(0, 0), cvPoint(img->width, 50)
    , black, CV_FILLED, 8, 0);
            cvPutText(img, string, cvPoint(25, 25), &font, white);
232
            if(showForeground)
234         {

236             for(size_t i = 0; i < tld->detectorCascade->
    detectionResult->fgList->size(); i++)
                {
238                 Rect r = tld->detectorCascade->detectionResult
    ->fgList->at(i);
```

```
                    cvRectangle(img, r.tl(), r.br(), white, 1);
240             }

242         }


244
            if(showOutput)
246         {
                gui->showImage(img);
248             char key = gui->getKey();

250             if(key == 'q') break;

252             if(key == 'b')
                {
254
                    ForegroundDetector *fg = tld->detectorCascade->
    foregroundDetector;
256
                    if(fg->bgImg.empty())
258                 {
                        fg->bgImg = grey.clone();
260                 }
                    else
262                 {
                        fg->bgImg.release();
264                 }
                }
266
                if(key == 'c')
268             {
                    //clear everything
270                 tld->release();
                }
272
                if(key == 'l')
274             {
                    tld->learningEnabled = !tld->learningEnabled;
276                 printf("LearningEnabled: %d\n", tld->
    learningEnabled);
                }
278
                if(key == 'a')
280             {
                    tld->alternating = !tld->alternating;
282                 printf("alternating: %d\n", tld->alternating);
                }
284
                if(key == 'e')
286             {
                    tld->writeToFile(modelExportFile);
288             }

290             if(key == 'i')
                {
292                 tld->readFromFile(modelPath);
                }
294
                if(key == 'r')
296             {
                    CvRect box;
298
```

```
                    if(getBBFromUser(img, box, gui) == PROGRAM_EXIT
    )
300                 {
                        break;
302                 }

304                 Rect r = Rect(box);

306                 tld->selectObject(grey, &r);
                }
308             /***EXTENDED CODE***/

310     //by pressing 'f' key you start the communication with robot
    server
                if(key == 'f'){
312                 communicate = true;
                }
314     /*** END OF EXTENDED CODE***/
            }
316
            if(saveDir != NULL)
318         {
                char fileName[256];
320             sprintf(fileName, "%s/%.5d.png", saveDir, imAcq->
    currentFrame - 1);

322             cvSaveImage(fileName, img);
            }
324     }

326     if(reuseFrameOnce)
        {
328         reuseFrameOnce = false;
        }
330 }

332 cvReleaseImage(&img);
    img = NULL;
334
    if(exportModelAfterRun)
336 {
        tld->writeToFile(modelExportFile);
338 }

340 if(resultsFile)
    {
342     fclose(resultsFile);
    }
344 }
    /*** EXTENDED CODE***/
346 /*** SOCKETS IMPLEMANTATION ***/
    #include <string.h>
348 #include <errno.h>
    #include <fcntl.h>
350 #include <iostream>

352 Socket::Socket() :
      m_sock ( -1 ){
354
      memset ( &m_addr,
356       0,
          sizeof ( m_addr ) );
```

```
358
}
360
Socket::~Socket(){
362   if ( is_valid() )
        ::close ( m_sock );
364 }

366 bool Socket::create(){
      m_sock = socket ( AF_INET,
368           SOCK_STREAM,
              0 );
370
      if ( ! is_valid() )
372     return false;


374
      // TIME_WAIT - argh
376   int on = 1;
      if ( setsockopt ( m_sock, SOL_SOCKET, SO_REUSEADDR, ( const char*
         ) &on, sizeof ( on ) ) == -1 )
378     return false;


380
      return true;
382
}
384



386
bool Socket::bind ( const int port ){
388
      if ( ! is_valid() ){
390       return false;
      }
392



394
      m_addr.sin_family = AF_INET;
396   m_addr.sin_addr.s_addr = INADDR_ANY;
      m_addr.sin_port = htons ( port );
398
      int bind_return = ::bind ( m_sock,
400             ( struct sockaddr * ) &m_addr,
              sizeof ( m_addr ) );
402


404   if ( bind_return == -1 ){
        return false;
406   }

408   return true;
}
410


412 bool Socket::listen() const{
      if ( ! is_valid() ){
414       return false;
      }
416
      int listen_return = ::listen ( m_sock, MAXCONNECTIONS );
418
```

```
420   if ( listen_return == -1 ){
          return false;
422   }

424   return true;
}
426

428 bool Socket::accept ( Socket& new_socket ) const{
      int addr_length = sizeof ( m_addr );
430   new_socket.m_sock = ::accept ( m_sock, ( sockaddr * ) &m_addr, (
        socklen_t * ) &addr_length );

432   if ( new_socket.m_sock <= 0 )
        return false;
434   else
        return true;
436 }

438
    bool Socket::send ( const int s[2] ) const{
440   int status = ::send ( m_sock, s, sizeof(int[2]), MSG_NOSIGNAL );
      if ( status == -1 ){
442       return false;
      }
444   else{
          return true;
446   }
    }
448

450 int Socket::recv ( int s[2] ) const{
      //char buf [ MAXRECV + 1 ];
452   int buf[2];

454   for(int i=0; i<2; i++){
      buf[i] = 0;
456     s[i] = 0;
      }
458
      //memset ( buf, 0, MAXRECV + 1 );
460
      int status = ::recv ( m_sock, buf, sizeof(int[2]), 0 );
462
      if ( status == -1 ){
464       std::cout << "status == -1   errno == " << errno << "  in
        Socket::recv\n";
          return 0;
466   }
      else if ( status == 0 ){
468       return 0;
      }
470   else{
          for(int i=0; i<2; i++){
472         s[i] = buf[i];
          }
474       return status;
      }
476 }

478
```

```
480 bool Socket::connect ( const std::string host, const int port ){
    if ( ! is_valid() ) return false;
482
    m_addr.sin_family = AF_INET;
484 m_addr.sin_port = htons ( port );

486 int status = inet_pton ( AF_INET, host.c_str(), &m_addr.sin_addr
      );

488 if ( errno == EAFNOSUPPORT ) return false;

490 status = ::connect ( m_sock, ( sockaddr * ) &m_addr, sizeof (
      m_addr ) );

492 if ( status == 0 )
      return true;
494 else
      return false;
496 }

498 void Socket::set_non_blocking ( const bool b ){

500 int opts;

502 opts = fcntl ( m_sock,
       F_GETFL );
504
    if ( opts < 0 ){
506     return;
    }
508
    if ( b )
510   opts = ( opts | O_NONBLOCK );
    else
512   opts = ( opts & ~O_NONBLOCK );

514 fcntl ( m_sock,
      F_SETFL,opts );
516
518 }

520 ClientSocket::ClientSocket ( std::string host, int port ){
    if ( ! Socket::create() ){
        throw SocketException ( "Could not create client socket." );
522 }

524 if ( ! Socket::connect ( host, port ) ){
        throw SocketException ( "Could not bind to port." );
526 }

528 }

530

532 const ClientSocket& ClientSocket::operator << ( const int s[2] )
      const{
    if ( ! Socket::send ( s ) ){
        throw SocketException ( "Could not write to socket." );
534 }

536 return *this;
```

```
538 }
540
const ClientSocket& ClientSocket::operator >> ( int s[2] ) const{
542   if ( ! Socket::recv ( s ) ){
        throw SocketException ( "Could not read from socket." );
544   }
546   return *this;
}
548 /*** END OF EXTENDED CODE***/
```

source/TLD/Main.cpp

```
/*  Copyright 2011 AIT Austrian Institute of Technology
2 *
*   This file is part of OpenTLD.
4 *
*   OpenTLD is free software: you can redistribute it and/or modify
6 *   it under the terms of the GNU General Public License as
    published by
*    the Free Software Foundation, either version 3 of the License,
     or
8 *   (at your option) any later version.
*
10 *   OpenTLD is distributed in the hope that it will be useful,
*   but WITHOUT ANY WARRANTY; without even the implied warranty of
12 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*   GNU General Public License for more details.
14 *
*   You should have received a copy of the GNU General Public
    License
16 *   along with OpenTLD.  If not, see <http://www.gnu.org/licenses
    />.
*
18 */

20 /*
 * main.h
22 *
 *  Created on: Nov 18, 2011
24 *      Author: Georg Nebehay
 *  Extended on: Feb, 2013
26 *  Author: Evangelos Sigalas
 *  Implementation of client-Socket communication with a server(
    robot)
28 */

30 #ifndef MAIN_H_
#define MAIN_H_
32
#include "TLD.h"
34 #include "ImAcq.h"
#include "Gui.h"
36
enum Retval
38 {
    PROGRAM_EXIT = 0,
40     SUCCESS = 1
};
42
class Main
```

```
{
public:
    tld::TLD *tld;
    ImAcq *imAcq;
    tld::Gui *gui;
  bool showOutput;
  bool showTrajectory;
    int trajectoryLength;
    const char *printResults;
    const char *saveDir;
    double threshold;
    bool showForeground;
    bool showNotConfident;
    bool selectManually;
    int *initialBB;
    bool reinit;
    bool exportModelAfterRun;
    bool loadModel;
    const char *modelPath;
    const char *modelExportFile;
    int seed;
    int myCor[2];//coordinates of target
    bool communicate;//flag to start the communication

    Main()
    {
        tld = new tld::TLD();
        showOutput = 1;
        printResults = NULL;
        saveDir = ".";
        threshold = 0.5;
        showForeground = 0;

    showTrajectory = false;
    trajectoryLength = 0;

        selectManually = 0;

        initialBB = NULL;
        showNotConfident = true;

        reinit = 0;

        loadModel = false;

        exportModelAfterRun = false;
        modelExportFile = "model";
        seed = 0;

        gui = NULL;
        modelPath = NULL;
        imAcq = NULL;
    }

    ~Main()
    {
        delete tld;
        imAcqFree(imAcq);
    }

    void doWork();
};
```

```
106 /***EXTENDED CODE***/
    /*** SOCKETS ***/
108 #include <string>
    #include <sys/types.h>
110 #include <sys/socket.h>
    #include <netinet/in.h>
112 #include <netdb.h>
    #include <unistd.h>
114 #include <arpa/inet.h>

116 class SocketException{
     public:
118   SocketException ( std::string s ) : m_s ( s ) {};
      ~SocketException (){};
120
      std::string description() { return m_s; }
122
     private:
124
      std::string m_s;
126
    };
128
    const int MAXHOSTNAME = 200;
130 const int MAXCONNECTIONS = 5;
    const int MAXRECV = 500;
132
    class Socket{
134  public:
      Socket();
136   virtual ~Socket();

138   // Server initialization
      bool create();
140   bool bind ( const int port );
      bool listen() const;
142   bool accept ( Socket& ) const;

144   // Client initialization
      bool connect ( const std::string host, const int port );
146
      // Data Transimission
148   bool send ( const int[2] ) const;
      int recv ( int[2] ) const;
150


152   void set_non_blocking ( const bool );

154   bool is_valid() const { return m_sock != -1; }

156  private:

158   int m_sock;
      sockaddr_in m_addr;
160


162 };

164 class ClientSocket : private Socket{
     public:
166
      ClientSocket ( std::string host, int port );
```

```
168     virtual ~ClientSocket(){};

170     const ClientSocket& operator << ( const int[2] ) const;
        const ClientSocket& operator >> ( int[2] ) const;
172
};
174
/*** END OF SOCKETS ***/
176 /*** END OF EXTENDED CODE ***/

178 #endif /* MAIN_H_ */
```

source/TLD/Main.h

Παράρτημα Β΄

Βιβλιοθήκη OpenCV

Η OpenCV (Open Source Computer and Vision Library) είναι μια βιβλιοθήκη ανοιχτού κώδικα η οποία χρησιμοποιείται σε εφαρμογές υπολογιστικής όρασης και μηχανικής εκμάθησης. Ουσιαστικά φτιάχτηκε για να δώσει μια κοινή βάση σε εφαρμογές σχετικές με υπολογιστική όραση. Είναι εύκολο να χρησιμοποιηθεί καθώς και να τροποποιηθεί ο κωδικάς της, καθώς κυκλοφορεί με την BSD άδεια προιόντος η οποία έχει ελάχιστους περιορισμούς σχετικά με την αναδιανομή του κώδικα.

Περιέχει πάνω από δυόμιση χιλιάδες βελτιστοποιημένους αλγορίθμους, οι οποίοι περιλαμβάνουν κλασικούς καθώς και συγχρονούς αλγορίθμους υπολογιστικής όρασης και μηχανικής μάθησης. Αυτοί μπορούν να χρησιμοποιηθούν για αναγνώριση αντικειμένων, προσώπων, ομαδοποίηση ανθρώπινων ενεργειών απο κάμερα, ανίχνευση κινούμενων αντικειμένων στο χώρο, δημιουργία 3-D μοντέλων, εύρεση όμοιων εικόνων απο κάποια βάση δεδομένων. Η κοινότητα της OpenCV έχει πάνω απο σαράντα εφτά χιλιάδες χρήστες και υπολογίζεται οτι έχει κατεβαστεί πάνω απο εφτά εκατομμύρια φορές. Χρησιμοποιείται ευρέως σε εταιρείες και ερευνητικές ομάδες.

Διαθέτει περιβάλλον για όλες τις γνωστές γλώσσες προγραμματισμού όπως C, C++, Java, Python και Matlab και υποστηρίζεται απο όλα τα γνωστά λειτουργικά συστήματα όπως Linux, Mac OS, Android, Windows.

Η OpenCV ξεκίνησε το 1999 στην Intel απο τον Gary Bradski με σκοπό να επιταχύνει την έρευνα σε εμπορικές εφαρμογές υπολογιστικής όρασης, για την Intel , δημιουργώντας έτσι την απαίτηση για πιο δυνατούς υπολογιστές απο αυτές τις εφαρμογές. Ο Vadim Pisarevsky ακολούθησε τον Gary και ανέλαβε να διευθύνει την Ρώσικη ομάδα OpenCVτης Intel.Με το πέρασμα του χρόνου η ομάδα της OpenCV προσχώρησε και σε άλλες εταιρείες. Πολλά απο τα πρωταρχικά μέλη της κατέληξαν να δουλεύουν πάνω στη ρομποτική και για τον Willow Garage. Το 2008, ο Willow Garage είδε την ανάγκη για μια ραγδαία αύξηση των ικανοτήτων της μηχανικής αντίληψης σε ανοιχτό επίπεδο, για αυτό το λόγο άρχισε να υποστηρίζει δυναμικά την OpenCV μαζί με τους Gary και Vadim.

Το πρόγραμμα OpenTLDπου χρησιμοποιούμε για την αναζήτηση και τον εντοπισμό του στόχου μας χρησιμοποιεί συναρτήσεις αυτής της βιβλιοθήκης. Για αυτό το λόγο η OpenCV αποτελεί βασική προϋπόθεση για να λειτουργήσουν όλα αυτά που περιγράψαμε στα προηγούμενα κεφάλαια. Στη σνέχεια παραθέτουμε μια σειρά απο εντολές οι οποίες εγκαθιστούν την OpenCV στο λειτουργικό σύστημα

Ubuntu-Linux.

```
# Ubuntu 12.04 provides a package of OpenCV 2.3.1 that you can
    easily install by typing:

  sudo apt-get install libopencv-dev

### Install Dependencies ###
#
# Essentials
# These are libraries and tools required by OpenCV.

  sudo apt-get install build-essential checkinstall cmake pkg-
    config yasm

# Image I/O
# Libraries for reading and writing various image types.
# If you do not install then the versions supplied by OpenCV will
    be used.

  sudo apt-get install libtiff4-dev libjpeg-dev libjasper-dev

# Video I/O
# You need some or all of these packages to add video
#capturing/encoding/decoding capabilities to the highgui module.

  sudo apt-get install libavcodec-dev libavformat-dev libswscale-
    dev libdc1394-22-dev libxine-dev libgstreamer0.10-dev
    libgstreamer-plugins-base0.10-dev libv4l-dev

# Python
# Packages needed to build the Python wrappers.

  sudo apt-get install python-dev python-numpy

# Other third-party libraries
# Install Intel TBB to enable parallel code in OpenCV.

  sudo apt-get install libtbb-dev

# GUI
# The default back-end for highgui in Linux is GTK. You can
    optionally install QT instead of GTK
# and later enable it in the configuration (see next section).

  sudo apt-get install libqt4-dev libgtk2.0-dev



### Compile and Install ###
#
# Get a copy of the source code, extract and create a build
    directory

  tar -xvf OpenCV-2.4.0.tar.bz2
  cd OpenCV-2.4.0/
  mkdir build
  cd build

# Configure using CMake. You have a lot of options in this step.
    This is what I use:

  cmake -D WITH_QT=ON -D WITH_XINE=ON -D WITH_OPENGL=ON -D WITH_TBB
```

```
    =ON -D BUILD_EXAMPLES=ON ..

# Start compiling:

  make

# If compilation finishes without errors, you can install by saying
    :

  sudo make install

# Finally, make sure that your programs can link to the OpenCV
    library in run-time by adding
# the following line at the end of your /etc/ld.so.conf:
#
# /usr/local/lib
#
# And then configure dynamic linker run-time bindings:

  sudo ldconfig

### Testing ###
#
# An easy way to test that the compilation went well is to use the
    OpenCV test utilities.
# For example, to test the core module go to OpenCV-2.4.0/build/bin
     and run:

  ./opencv_test_core
```

source/install.readme

Βιβλιογραφία

[1] Rafael C. Gonzalez, Richard E. Woods *Ψηφιακή επεξεργασία εικόνας*. Εκδόσεις Τζιόλα 2011.

[2] Silberschatz Galvin-Gagne *Λειτουργικά Συστήματα*. Εκδόσεις Ίων.

[3] Georg Nebehay. Robust Object Tracking Based on Tracking-Learning-Detection. Technische Universität Wien.

[4] *http://tldp.org/LDP/LG/issue74/tougher.html*

[5] *http://codebase.eu/tutorial/linux-socket-programming-c/*

[6] *http://opencv.org/about.html*

[7] *https://www.willowgarage.com/pages/software/opencv*

[8] *http://www.linuxhowtos.org/C_C++/socket.htm*

[9] Z. Kalal, K. Mikolajczyk, and J. Matas. Forward-Backward Error: Automatic Detection of Tracking Failures. In *In International Conference on Pattern Recognition,* pages 23–26, 2010.