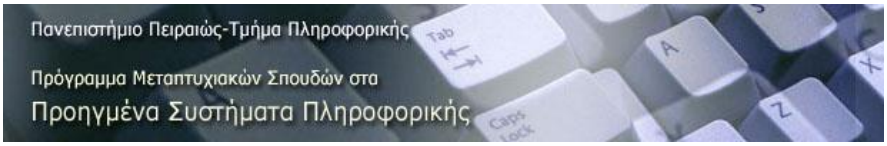
Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Χρήση ενός αλγορίθμου εκτίμησης της οπτικής ροής σε ένα ρομπότ για την αποφυγή εμποδίων**
Ονοματεπώνυμο Φοιτητή	**Γεώργιος Πέτρου**
Αριθμός Μητρώου	**ΜΠΣΠ/07045**
Κατεύθυνση	**Ευφυείς Τεχνολογίες Αλληλεπίδρασης Ανθρώπου - Υπολογιστή**
Επιβλέπων	**Γεώργιος Α. Τσιχριντζής , Καθηγητής**



Ημερομηνία Παράδοσης **28/09/2010**

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Γεώργιος Τσιχριντζής Καθηγητής	Δημήτριος Αποστόλου Επίκουρος Καθηγητής	Χαράλαμπος Κωνσταντόπουλος Λέκτορας

Περίληψη

Το θέμα της εργασίας αυτής είναι ο σχεδιασμός, η κατασκευή και ο προγραμματισμός ενός απλού ρομπότ, το οποίο χρησιμοποιώντας κάποιον αλγόριθμο εκτίμησης της οπτικής ροής θα είναι ικανό να αποφύγει τα εμπόδια που υπάρχουν στο περιβάλλον του. Για την κατασκευή του ρομπότ χρησιμοποιήθηκε ένα γνωστό σετ ρομποτικής και ένας φορητός υπολογιστής προσαρμοσμένος στην βάση του. Για την ανίχνευση των εμποδίων χρησιμοποιήθηκε ένας ευρέως διαδεδομένος αλγόριθμος υπολογισμού της οπτικής ροής όπως έχει υλοποιηθεί σε μια βιβλιοθήκη υπολογιστικής όρασης.

Το σύστημα χρησιμοποιεί την κάμερα του υπολογιστή για την ανάκτηση των εικόνων που στην συνέχεια χρησιμοποιούνται για τον υπολογισμό της οπτικής ροής για δύο διαδοχικές εικόνες. Έπειτα βάσει των αποτελεσμάτων χρησιμοποιείται μια απλή στρατηγική αποφυγής των εμποδίων. Για την κίνηση του ρομπότ οι εντολές δίνονται από τον υπολογιστή μέσω του πρωτοκόλλου επικοινωνίας Bluetooth στην υπολογιστική μονάδα του ρομπότ που με την σειρά της κινεί ανάλογα τον αριστερό και δεξιό κινητήρα. Το τελικό σύστημα μπορεί με επιτυχία να αποφύγει τα εμπόδια που συναντάει στην πορεία του.

Abstract

The subject of this dissertation is the design, construction and programming of a simple robot, that will utilize an optical flow estimation algorithm in order to avoid existing obstacles in its environment. A popular robotics kit and a netbook attached το its base were used for the construction of the robot. A well known algorithm for the estimation of optical flow was used for the detection of the obstacles, as it is implemented in a computer vision library.

The system is using the camera from the netbook to retrieve the images which are then used to calculate the optical flow between two consecutive frames. Later, these results are used by a simple strategy algorithm to void the obstacles. The commands for the movement of the robot are transferred through the Bluetooth communication protocol to the central processing unit of the robot, which in turn moves the left and right motor accordingly. The final system can successfully avoid the obstacles found in its course.

Περιεχόμενα

1 Εισαγωγή

Την σημερινή εποχή τα ρομπότ χρησιμοποιούνται σε πολλούς τομείς της καθημερινότητας, πολλοί από τους οποίους μπορεί να είναι επικίνδυνοι για τους ανθρώπους. Αρκετές από αυτές τις εφαρμογές απαιτούν από τις μηχανές αυτές να έχουν υψηλό δείκτη αυτονομίας ο οποίος επιτυγχάνεται με την χρήση μιας ποικιλίας αισθητήρων και υπολογιστικών μονάδων. Με τον όρο αυτονομία αναφερόμαστε στην ικανότητα του ρομπότ να παίρνει δεδομένα από τους αισθητήρες του και να λαμβάνει αποφάσεις για την επιτυχή πλοήγηση του στο περιβάλλον του και εκτέλεση της εργασίας του χωρίς ανθρώπινη παρέμβαση.

Για την αποστολή αυτή πρέπει να αντιμετωπίσει μια σειρά προβλημάτων που περιλαμβάνουν τον εντοπισμό, αναγνώριση και αποφυγή εμποδίων σε πραγματικό χρόνο λαμβάνοντας υπόψη τους περιορισμούς που διαθέτει σε υπολογιστική δύναμη. Κάθε ρομπότ προκειμένου να μπορεί να ανταπεξέλθει επιτυχώς σε αυτή την πρόκληση θα πρέπει να έχει τρείς κύριες λειτουργίες:

- Αντίληψη (perception)
- Σχεδιασμό (planning)
- Πράξη (action)

Στην εργασία αυτή θα ασχοληθούμε με τους αισθητήρες όρασης. Η πλοήγηση ενός ρομπότ με βάση κάποιο σύστημα τεχνητής όρασης μπορεί να χωριστεί σε τρείς κατηγορίες [1]:

- Μέθοδος βασισμένη σε χάρτη (Map-based navigation)
- Μέθοδος βασισμένη στην κατασκευή χάρτη (Map-building-based navigation)
- Μέθοδος χωρίς την χρήση χάρτη (Map-less navigation)

Συνήθως στις δύο πρώτες κατηγορίες χρησιμοποιούνται δύο κάμερες για την ανάκτηση του βάθους των αντικειμένων [2] ή ακόμα και επιπλέον αισθητήρες [3].

Τα συστήματα όρασης μπορούν να παρέχουν ένα σημαντικό εύρος πληροφοριών με βάση τον άμεσο περιβάλλον στο οποίο βρίσκεται το ρομπότ, ώστε να συμβάλουν στην αυτονομία του. Προκειμένου να αποφύγουμε την χρήση ενός στερεοσκοπικού συστήματος θα χρησιμοποιήσουμε κάποια μέθοδο ανάκτησης της οπτικής ροής για την αναγνώριση ενός ή περισσοτέρων εμποδίων. Στην βιβλιογραφία υπάρχουν μερικά παραδείγματα χρήσης αυτής της μεθόδου [4], [5], [6]. Κάποιες έρευνες έχουν δείξει ότι σε αρκετές περιπτώσεις η οπτική ροή μπορεί να έχει καλύτερες επιδόσεις από άλλους αισθητήρες όπως τα σόναρ [7]. Σε κάποιες εργασίες η επεξεργασία γίνεται εκτός του ρομπότ σε κάποιον απομακρυσμένο υπολογιστή και οι εντολές δίνονται μέσω δικτύων επικοινωνίας. Σε αυτές τις περιπτώσεις μπορεί να χρησιμοποιηθεί περισσότερη επεξεργαστική ισχύς αλλά η επικοινωνία μεταξύ του ρομπότ και του υπολογιστή μπορεί να χαθεί ευκολότερα [8].

1.1 Σκοπός της εργασίας

Ο τελικός σκοπός της εργασίας αυτής είναι η δημιουργία ενός ρομπότ το οποίο θα επεξεργάζεται τις εικόνες που θα λαμβάνει από μια απλή κάμερα ώστε να αποφύγει τα εμπόδια που θα συναντάει στην πορεία του. Σαν εμπόδια ορίζουμε τους τοίχους και τα διάφορα αντικείμενα που να υπάρχουν. Ο αλγόριθμος που θα αναγνωρίζει τα εμπόδια θα χρησιμοποιεί έναν αλγόριθμο εκτίμησης της οπτικής ροής και ανάλογα με το αποτέλεσμα θα διορθώνει την πορεία του ρομπότ.

1.2 Περίγραμμα της εργασίας

Στην ενότητα αυτή γίνεται μια σύντομη περιγραφή των περιεχομένων που βρίσκονται στα επόμενα κεφάλαια.

Στο κεφάλαιο 2 γίνεται μια σύντομη βιβλιογραφική ανασκόπηση σχετικά με το μοντέλο της κάμερας, τον ορισμό της οπτικής ροής και την χρήση της από κάποιους οργανισμούς για την αποφυγή εμποδίων. Επίσης, γίνεται αναφορά στα είδη των αλγορίθμων οπτικής ροής και μια αναλυτικότερη περιγραφή των πιο γνωστών αλγορίθμων. Τέλος, παρουσιάζονται οι αλγόριθμοι εύρεσης γωνιών, έναν από τους οποίους θα χρησιμοποιήσουμε στην υλοποίηση του συστήματος.

Το κεφάλαιο 3 παρουσιάζει την υλοποίηση του συστήματος το οποίο αποτελείται από το ρομπότ και το πρόγραμμα που υλοποιήθηκε για την λειτουργία του. Πιο συγκεκριμένα, παρουσιάζεται ο σχεδιασμός

και η κατασκευή του ρομπότ, η επικοινωνία του υπολογιστή με το ρομπότ, ο αλγόριθμος αποφυγής των εμποδίων και το τελικό πρόγραμμα.

Στο κεφάλαιο 4 παρατίθενται τα αποτελέσματα της λειτουργίας του συστήματος για διαφορετικές πειραματικές συνθήκες.

Το κεφάλαιο 5 συνοψίζει το αποτέλεσμα της παρούσας εργασίας και γίνεται αναφορά σε μελλοντικές προσθήκες που θα μπορούσαν να γίνουν.

Τέλος στο παράρτημα Α παρουσιάζεται μια σειρά εικόνων της πορείας του ρομπότ και των επεξεργασμένων εικόνων που βλέπει το σύστημα. Στο παράρτημα Β παρατίθεται ο κώδικας σε C++.NET που χρησιμοποιήθηκε στην εργασία με σχόλια στα σημαντικότερα σημεία του.

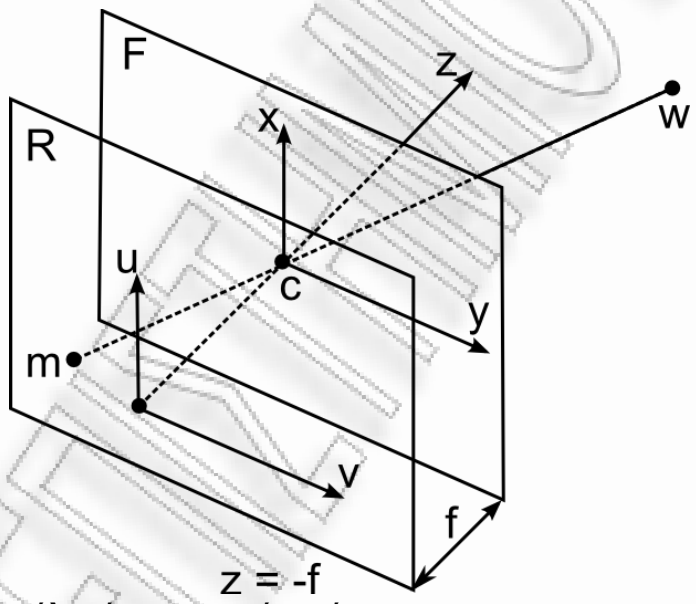
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

2 Βιβλιογραφική ανασκόπηση

Στο κεφάλαιο αυτό γίνεται μια σύντομη βιβλιογραφική ανασκόπηση σχετικά με το θέμα της εργασίας. Αρχικά γίνεται μια μικρή εισαγωγή στο απλό μοντέλο της κάμερας και στην συνέχεια ορίζεται η οπτική ροή και δίνονται κάποια παραδείγματα χρήσης της από φυσικούς οργανισμούς. Έπειτα παρουσιάζονται οι βασικές κατηγορίες αλγορίθμων οπτικής ροής και αναλύονται περεταίρω κάποιοι από τους πιο γνωστούς αλγορίθμους. Τέλος, παρουσιάζονται οι αλγόριθμοι εξεύρεσης γωνιών που μπορούν να χρησιμοποιηθούν από κάποιους αλγόριθμους εκτίμησης της οπτικής ροής.

2.1 Μοντέλο κάμερας στενής οπής

Μια συνηθισμένη κάμερα λαμβάνει μια σειρά από εικόνες από το περιβάλλον στο οποίο βρίσκεται. Κάθε μία από αυτές τις εικόνες είναι μια προβολή της τρισδιάστατης σκηνής σε έναν δισδιάστατο πίνακα από εικονοστοιχεία (pixels). Το πιο συνηθισμένο μοντέλο κάμερας που μπορεί να περιγράψει πως μπορεί να αναπαρασταθεί ο τρισδιάστατος χώρος σε μια δισδιάστατη εικόνα είναι το μοντέλο κάμερας στενής οπής (pinhole model). Η σχέση μεταξύ ενός τρισδιάστατου σημείου με συντεταγμένες (X, Y, Z) στις συντεταγμένες (u, v) της εικόνας περιγράφεται από τις εξισώσεις $u = f\frac{X}{Z}, v = f\frac{Y}{Z}$, όπου f είναι το εστιακό μήκος της κάμερας.



Εικόνα 2-1: Το μοντέλο κάμερας στενής οπής

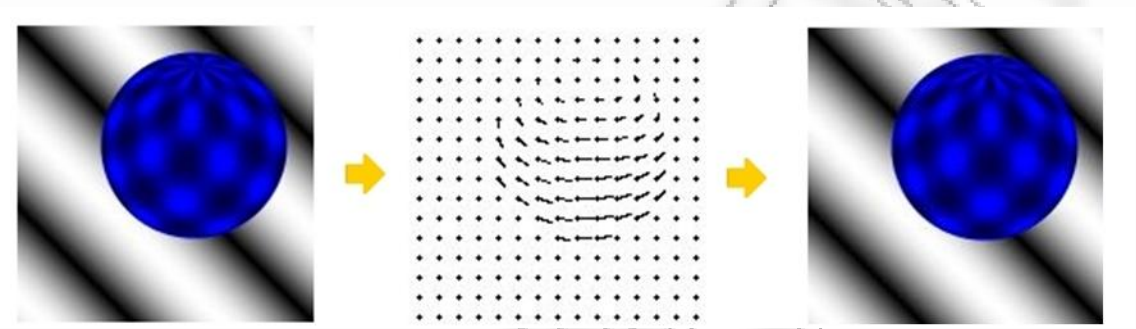
2.2 Οπτική ροή

Ένας παρατηρητής (βιολογικός οργανισμός ή ρομπότ) που χρησιμοποιεί την οπτική του αντίληψη για να κατανοήσει το περιβάλλον στο οποίο βρίσκεται, βασίζεται στον υπολογισμό και την ερμηνεία της οπτικής κίνησης των αντικειμένων που τον περιβάλλουν. Για κάθε κάμερα η οποία βρίσκεται σε σχετική κίνηση με το περιβάλλον της, δημιουργείται ένα κινούμενο πεδίο φωτεινότητας το οποίο προσπίπτει στο επίπεδο καταγραφής της κάμερας. Σαν αποτέλεσμα δημιουργείται ένα δισδιάστατο διανυσματικό επίπεδο ταχύτητας της προβολής των αντικειμένων επάνω στο επίπεδο της κάμερας το οποίο καλείται οπτική ροή (optical flow).

Σε αυτή την σκηνή εάν κινηθεί η κάμερα ή κάποιο αντικείμενο θα υπάρξει μια μετακίνηση αυτού του αντικειμένου στον πίνακα. Η μετακίνηση αυτή εξαρτάται από την απόσταση του αντικειμένου από την κάμερα. Εάν είναι μακριά από την κάμερα η μετακίνηση θα είναι μικρή. Εάν βρίσκεται κοντά στην κάμερα θα είναι μεγαλύτερη. Το διάνυσμα της μετακίνησης αυτής θα μπορεί να χρησιμοποιηθεί για την μέτρηση της απόστασης αυτού του αντικειμένου. Η εκτίμηση αυτών των διανυσμάτων μπορεί να μας

δώσει την πεδίο οπτικής ροής της εικόνας. Υπάρχουν αλγόριθμοι που υπολογίζουν αυτά τα διανύσματα για κάθε εικονοστοιχείο και αλγόριθμοι που χρησιμοποιούν συγκεκριμένα σημεία όπως γωνίες. Όπως είναι κατανοητό είναι πιο γρήγορος ο υπολογισμός της οπτικής ροής για συγκεκριμένα σημεία.

Ένα μοντέλο οπτικής ροής παίρνει σαν δεδομένα μια σειρά από εικόνες από την σκηνή στην οποία βρίσκεται η κάμερα. Έστω ότι η αρχική εικόνα αναπαριστά την σκηνή σε χρόνο t. Κάθε σημείο (u, v) στην εικόνα θα έχει ένταση $I(u, v, t)$. Εάν η χρονική διαφορά μεταξύ δύο συνεχόμενων εικόνων είναι μικρή ($^1/_{10} - ^1/_{30}$ δευτερόλεπτα) η προσέγγιση της φωτεινότητας της εικόνας είναι ακριβής. Έστω λοιπόν ότι κάποιο σημείο θα εμφανιστεί σε κάποιες άλλες συντεταγμένες στην επόμενη εικόνα. Εάν θεωρήσουμε ότι η μετακίνηση αυτή μπορεί να αναπαρασταθεί από ένα διάνυσμα $(\Delta u, \Delta v)$ η ένταση θα παραμείνει σταθερή βάση του τύπου $I(u + \Delta u, v + \Delta v, t + \Delta t) = I(u, v, t)$.



Εικόνα 2-2: Οπτική ροή μεταξύ των δύο εικόνων. Η σφαίρα γυρνάει από τα αριστερά προς τα δεξιά δημιουργώντας το πεδίο οπτικής ροής που φαίνεται στο κέντρο. Πηγή: http://of-eval.sourceforge.net/

Ο υπολογισμός της οπτικής ροής μπορει να χρησιμοποιηθεί για:

- Εκτίμηση κίνησης (Motion estimation)
- Συμπίεση βίντεο (Video compression)
- Εντοπισμός αντικειμένου (object detection)
- Εντοπισμός κίνησης (movement detection)
- Αποφυγή εμποδίων (Obstacle avoidance)
- Σταθεροποίηση εικόνας (Image stabilization)
- Υπολογισμό περιστροφών (Rotation computation)

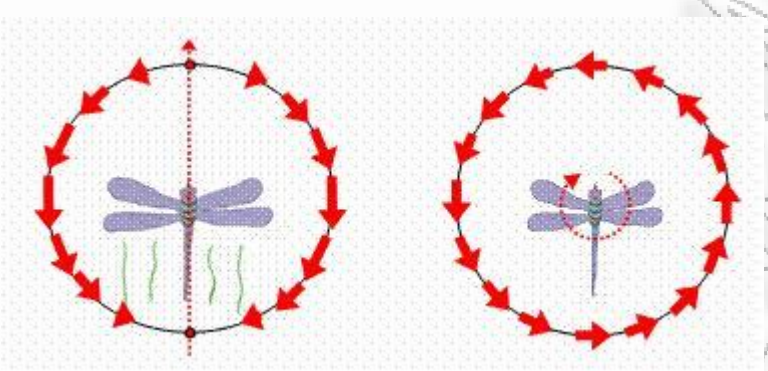
Εάν διπλασιαστεί η ταχύτητα που μετακινείται η κάμερα θα διπλασιαστεί και το μέγεθος της οπτικής ροής σε σχέση με τα αντικείμενα που βρίσκονται στην σκηνή. Εάν ένα αντικείμενο έρθει στην μισή απόσταση από αυτή που ήταν η οπτική ροή θα διπλασιαστεί επίσης. Επίσης η οπτική ροή εξαρτάται από την γωνία που κοιτάμε ένα αντικείμενο και την κατεύθυνση που μετακινείται η εικόνα. Έστω ότι μετακινούμαστε ευθεία . Η οπτική ροή θα είναι μεγαλύτερη για ένα αντικείμενο το οποίο βρίσκεται σε γωνία 90^o, παρά ένα αντικείμενο που βρίσκεται ακριβώς μπροστά μας. Παρόλα αυτά οι γωνίες του αντικειμένου αυτού δεν είναι ακριβώς ευθεία οπότε θα φαίνονται σαν να μετακινούνται οπότε και το αντικείμενο θα εμφανίζεται μεγαλύτερο.

2.2.1 Η οπτική ροή στην φύση

Αυτού του είδους η προσέγγιση απαντάται στην φύση όπου για παράδειγμα τα έντομα την χρησιμοποιούν για την αποφυγή εμποδίων ή εχθρών[9] και έχουν χρησιμοποιηθεί σε ρομπότ για αυτό τον σκοπό [10]. Επίσης οι μέλισσες χρησιμοποιούν την οπτική ροή για εκτίμηση των αποστάσεων [11].

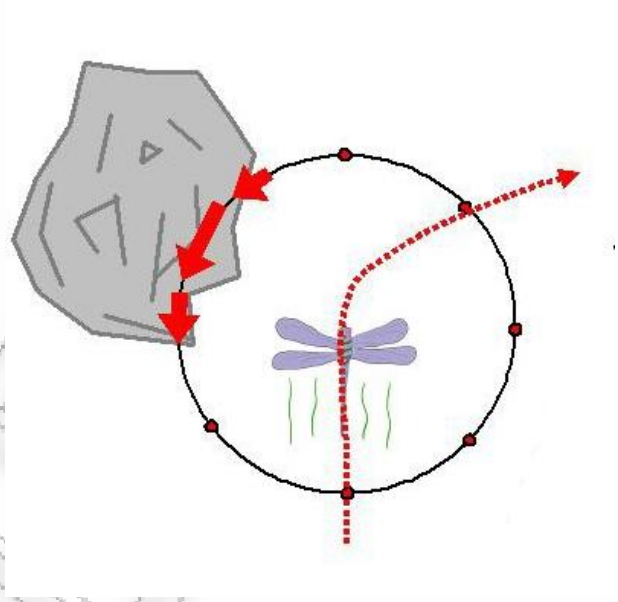
Η επόμενη εικόνα δείχνει την από πάνω εικόνα ενός εντόμου και την οπτική ροή σε κάθε μια από τις δύο περιπτώσεις. Στην αριστερή εικόνα το έντομο κινείται προς τα εμπρός. Η οπτική ροή μετακινείται από μπροστά προς τα πίσω και είναι μεγαλύτερη στα αριστερά και τα δεξιά απ' ότι μπρός και πίσω. Η εικόνα στα δεξιά δείχνει το έντομο όταν στρίβει προς τα δεξιά. Σε αυτή την περίπτωση η οπτική ροή

πηγαίνει προς τα αριστερά σε όλες τις πλευρές. Επομένως εκτός από τον εντοπισμό εμποδίων η οπτική ροή μπορεί να χρησιμοποιηθεί και σαν μέτρο εκτίμησης της κίνησης του οργανισμού.



Εικόνα 2-3: Η οπτική ροή που αντιλαμβάνεται ένα ιπτάμενο έντομο όταν προχωράει ευθεία και όταν στρίβει προς τα δεξιά. Πηγή: http://mcxperi.wordpress.com/

Στην επόμενη εικόνα παρουσιάζεται ένα παράδειγμα αποφυγής εμποδίου από ένα έντομο. Παρατηρούμε ότι η μεγάλη οπτική ροή από την αριστερή μεριά προειδοποιεί το έντομο ότι υπάρχει ένα εμπόδιο οπότε στρίβει προς τα δεξιά.



Εικόνα 2-4: Αποφυγή εμποδίων από ένα ιπτάμενο έντομο. Πηγή: http://mcxperi.wordpress.com/

2.3 Αλγόριθμοι οπτικής ροής

Υπάρχουν αρκετοί μέθοδοι υπολογισμού της οπτικής ροής. Γενικά μπορούν να καταταγούν σε τρείς κατηγορίες [12]:

- Διαφορικές μέθοδοι με βάση την ένταση (Intensity differential methods)
- Μέθοδοι φιλτραρίσματος με βάση την συχνότητα (Frequency - based methods)
- Μέθοδοι με βάση την συσχέτιση (Correlation - based methods)

Στην ενότητα αυτή παρουσιάζονται κάποιοι από τους βασικότερους αλγορίθμους υπολογισμού της οπτικής ροής σε μια ακολουθία εικόνων. Μια αναλυτικότερη ανασκόπηση των διαφορετικών προσεγγίσεων επάνω στο θέμα του υπολογισμού της οπτικής ροής, καθώς και σύγκριση μεταξύ τους γίνεται στο [13].

2.3.1 Αλγόριθμος αντιστοίχησης κομματιού (Block Matching)

Με την τεχνική αυτή η εικόνα χωρίζεται σε επικαλυπτόμενα κομμάτια [14], [15]. Για κάθε κομμάτι υπολογίζεται η μετατόπιση του με βάση την προηγούμενη εικόνα που ελαχιστοποιεί κάποια μέτρηση λάθους. Διάφορες τεχνικές μπορούν να χρησιμοποιηθούν όπως κανονικοποιημένη ετεροσυσχέτιση (normalized cross correlation), άθροισμα των απολύτων διαφορών (sum of absolute differences), άθροισμα των διαφορών των τετραγώνων (sum of squared differences), κτλ. Με την μέθοδο αυτή θα υπάρχει ένα διάνυσμα οπτικής ροής για κάθε κομμάτι.

Πιο αναλυτικά έστω I(x,y,t) = I(p,t) μια χωροχρονική ακολουθία εικόνων στην οποία υπάρχει ένα κινούμενο αντικείμενο όπου p=(x,y) αντιπροσωπεύει το διάνυσμα της ταχύτητας. Η μέθοδος χωρίζει μια εικόνα $I(p, t_1)$ σε κομμάτια σταθερού μεγέθους R και προσπαθεί να ελαχιστοποιήσει την συνάρτηση: $E = \sum_{p \in R} |I(p, t_1) - I(p + d, t_2)|^2$ και μια άλλη εικόνα $I(p, t_2)$ προκειμένου να βρεθεί το διάνυσμα μετατόπισης $d = (d_x, d_y)$.

Το σημαντικότερο πρόβλημα της μεθόδου αυτής είναι οι μεγάλες απαιτήσεις σε υπολογιστική δύναμη. Αντί λοιπόν να γίνεται αναζήτηση του κάθε κομματιού σε όλη την εικόνα, η αναζήτηση περιορίζεται σε ένα κομμάτι στης εικόνας γύρω από την αρχική θέση του κομματιού. Για την περαιτέρω βελτιστοποίηση του αλγορίθμου έχουν αναπτυχθεί διάφορες μέθοδοι όπως:

- Αναζήτηση τριών βημάτων
- Προχωρημένη αναζήτηση τριών βημάτων
- Απλή και αποδοτική αναζήτηση
- Αναζήτηση τεσσάρων βημάτων
- Αναζήτηση διαμαντιού
- Προσαρμοστική αναζήτηση

Άλλα μειονεκτήματα της μεθόδου αυτής είναι τα κομμάτια της εικόνας μπορεί να έχουν παραμορφωθεί από την προβολή σε διαφορετικές χρονικές στιγμές. Επίσης οι περιοχές μπορεί να διαφέρουν και ως προς την φωτεινότητά τους. Τα πλεονεκτήματα της μεθόδου είναι ότι μπορεί να παράγει πυκνά πεδία διανυσμάτων οπτικής ροής.

2.3.2 Αλγόριθμος Horn - Schunck

Πρόκειται για μια διαφορική μέθοδο υπολογισμού της οπτικής ροής η οποία χρησιμοποιεί έναν γενικό περιορισμό απαλότητας, ο οποίος λέγεται περιορισμός σταθερότητας φωτεινότητας [16]. Ο αλγόριθμος βασίζεται στην υπόθεση ότι η φωτεινότητα ενός σημείου δεν μεταβάλλεται σημαντικά κατά την διάρκεια του χρόνου. Η μέθοδος αυτή υπολογίζει την οπτική ροή για κάθε εικονοστοιχείο, δίνει ποιο πυκνό πεδίο ροής αλλά τα διανύσματα δεν είναι τόσο ακριβή.

Η μέθοδος χρησιμοποιεί ένα καθολικό κριτήριο ομαλότητας. Προσπαθεί να ελαχιστοποιήσει την συνάρτηση ενέργειας:

$$f = \int\int\int \left((\nabla I \cdot \vec{V} + I_t)^2 + a(|\nabla \mathrm{V_x}|^2 + |\nabla \mathrm{V_y}|^2 + |\nabla \mathrm{V_z}|^2) \right) dxdydz$$

Όπου $\nabla I = \begin{bmatrix} I_x \\ I_y \\ I_z \end{bmatrix}$ είναι οι παράγωγοι της εικόνας κατά μήκος των x,y και z διευθύνσεων, I_t είναι η παράγωγος στον χρόνο, $\vec{V}$είναι το διάνυσμα οπτικής ροής με συνιστώσες V_x, V_y, V_z. Η παράμετρος α αντιπροσωπεύει μια σταθερά κανονικοποίησης. Μεγαλύτερες τιμές για την παράμετρο αυτή δημιουργούν ομαλότερο πεδίο οπτικής ροής.

Για την ελαχιστοποίηση της συνάρτησης αυτής χρησιμοποιούνται οι επόμενες εξισώσεις Euler-Lagrange:

$$\Delta V_x - \frac{1}{a} I_x \left(I_x V_x + I_y V_y + I_z V_z + I_t\right) = 0$$
$$\Delta V_y - \frac{1}{a} I_y \left(I_x V_x + I_y V_y + I_z V_z + I_t\right) = 0$$
$$\Delta V_z - \frac{1}{a} I_z \left(I_x V_x + I_y V_y + I_z V_z + I_t\right) = 0$$

Με Δ συμβολίζεται ο τελεστής Laplace, όπου $\Delta V_x = \frac{\vartheta}{\vartheta x}\frac{\vartheta V_x}{\vartheta x}$, $\Delta V_y = \frac{\vartheta}{\vartheta y}\frac{\vartheta V_y}{\vartheta y}$ και $\Delta V_z = \frac{\vartheta}{\vartheta z}\frac{\vartheta V_z}{\vartheta z}$. Χρησιμοποιώντας την μέθοδο Gauss – Seidel για την επίλυση των εξισώσεων ως προς V_x, V_y και V_z προκύπτει το επαναληπτικό σχήμα:

$$V_x^{k+1} = \frac{\Delta V_x^k - \frac{1}{a} I_x (I_y V_y^k + I_z V_z^k + I_t)}{\frac{1}{a} I_x^2}$$
$$V_y^{k+1} = \frac{\Delta V_y^k - \frac{1}{a} I_y (I_x V_x^k + I_z V_z^k + I_t)}{\frac{1}{a} I_y^2}$$
$$V_z^{k+1} = \frac{\Delta V_z^k - \frac{1}{a} I_z (I_x V_x^k + I_y V_y^k + I_t)}{\frac{1}{a} I_z^2}$$

Όπου το k αντιπροσωπεύει το πιο πρόσφατο αποτέλεσμα και το k+1 συμβολίζει την επόμενη επανάληψη. Οι τιμές ΔV_iλαμβάνονται από την εξίσωση:

$$\Delta V_i = \sum_{n(p)} V_i\left(N(p)\right) - V_i(p)$$

Ο αλγόριθμος αυτός μπορεί να δώσει μεγάλη πυκνότητα διανυσμάτων οπτικής ροής αλλά είναι πιο ευαίσθητος στον θόρυβο.

2.3.3 Αλγόριθμος Lukas – Kanade

Πρόκειται για μια διαφορική μέθοδο βασισμένη στην προσέγγιση σειρών Taylor η οποία χρησιμοποιεί τοπικούς περιορισμούς απαλότητας. Η μέθοδος αυτή υπολογίζει την οπτική ροή για κάθε εικονοστοιχείο, δίνει ένα ποιο αραιό οπτικό πεδίο από την μέθοδο Horn-Schunk αλλά τα διανύσματα που προκύπτουν είναι πιο ακριβή [17].

Έστω μια ακολουθία εικόνων και ένα εικονοστοιχείο στη θέση $((x, y, z, t)$ με φωτεινότητα $I(x, y, z, t)$ στην πρώτη εικόνα. Στην επόμενη εικόνα το εικονοστοιχείο αυτό θα έχει μετακινηθεί κατά $\delta x, \delta y, \delta z, \delta t$. Επομένως για την διατήρηση της φωτεινότητας θα έχουμε: $I(x, y, z, t) = I(x + \delta x, y + \delta y, z + \delta z, t + \delta t)$. Υποθέτοντας ότι η κίνηση είναι σχετικά μικρή η ανάπτυξη της προηγούμενης εξίσωσης κατά Taylor θα δώσει: $I(x + \delta x, y + \delta y, z + \delta z, t + \delta t) = I(x, y, z, t) + \frac{\vartheta I}{\vartheta x}\delta x + \frac{\vartheta I}{\vartheta y}\delta y + \frac{\vartheta I}{\vartheta z}\delta z + \frac{\vartheta I}{\vartheta z}\delta z + \frac{\vartheta I}{\vartheta t}\delta t + \cdots$

Για μικρές μετατοπίσεις οι όροι υψηλότερης τάξης μπορούν να αγνοηθούν οπότε βάσει της παραπάνω σχέσεως προκύπτει ότι:

$$\frac{\vartheta I}{\vartheta x}\delta x + \frac{\vartheta I}{\vartheta y}\delta y + \frac{\vartheta I}{\vartheta z}\delta z + \frac{\vartheta I}{\vartheta t}\delta t = 0$$

ή

$$\frac{\vartheta I}{\vartheta x}\frac{\delta x}{\delta t} + \frac{\vartheta I}{\vartheta y}\frac{\delta y}{\delta t} + \frac{\vartheta I}{\vartheta z}\frac{\delta z}{\delta t} + \frac{\vartheta I}{\vartheta t}\frac{\delta t}{\delta t} = 0$$

οπότε θα έχουμε:

$$\frac{\vartheta I}{\vartheta x}V_x + \frac{\vartheta I}{\vartheta y}V_y + \frac{\vartheta I}{\vartheta z}V_z + \frac{\vartheta I}{\vartheta t} = 0$$

Όπου V_x, V_y, V_z αντιπροσωπεύουν τις συνιστώσες της ταχύτητας για τις μεταβλητές x,y,z αντίστοιχα. Οι παράγωγοι από την προηγούμενη σχέση μπορούν να γραφτούν και ως εξής:

$$I_x V_x + I_y V_y + I_z V_z = -It \Rightarrow \nabla \mathrm{I} \cdot \vec{\mathrm{V}} = -I_t$$

Η παραπάνω εξίσωση έχει τρείς αγνώστους οπότε δεν μπορεί να λυθεί άμεσα. Επομένως χρειάζεται ακόμα ένας περιορισμός με την χρήση ενός συνόλου εξισώσεων. Για την μέθοδο αυτή χρησιμοποιείται μια μη επαναληπτική μέθοδος που υποθέτει ότι η οπτική ροή είναι τοπικά σταθερή.
Έστω λοιπόν ότι η ροή είναι σταθερή σε ένα παράθυρο γύρο από το εικονοστοιχείο που μελετάμε, με διαστάσεις m x m x m, όπου m > 1. Έστω ότι τα εικονοστοιχεία του παραθύρου αριθμούνται από 1 έως n, τότε εύκολα προκύπτουν οι επόμενες εξισώσεις:

$$\begin{aligned} I_{x1}V_x + I_{y1}V_y + I_{z1}V_z &= -I_{t1} \\ I_{x2}V_x + I_{y2}V_y + I_{z2}V_z &= -I_{t2} \\ &\vdots \\ I_{xn}V_x + I_{yn}V_y + I_{zn}V_z &= -I_{tn} \end{aligned}$$

Με τις εξισώσεις αυτές προκύπτει το σύστημα εξισώσεων:

$$\begin{bmatrix} I_{x1} & I_{y1} & I_{z1} \\ I_{x2} & I_{y2} & I_{z2} \\ \vdots & \vdots & \vdots \\ I_{xn} & I_{yn} & I_{zn} \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = \begin{bmatrix} -I_{t1} \\ -I_{t2} \\ \vdots \\ -I_{tn} \end{bmatrix}$$

Ή αλλιώς: $A\vec{u} = -b$

Χρησιμοποιώντας την μέθοδο των ελαχίστων τετραγώνων θα έχουμε διαδοχικά:

$$A^T A\vec{u} = A^T(-b)$$

ή

$$\text{Ή } \vec{u} = (A^T A)^{-1} A^T(-b)$$

ή

$$\begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} I_{x_i}^2 & \sum_{i=1}^{n} I_{x_i} I_{y_i} & \sum_{i=1}^{n} I_{x_i} I_{z_i} \\ \sum_{i=1}^{n} I_{x_i} I_{y_i} & \sum_{i=1}^{n} I_{y_i}^2 & \sum_{i=1}^{n} I_{y_i} I_{z_i} \\ \sum_{i=1}^{n} I_{x_i} I_{z_i} & \sum_{i=1}^{n} I_{y_i} I_{z_i} & \sum_{i=1}^{n} I_{z_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_{i=1}^{n} I_{x_i} I_{t_i} \\ -\sum_{i=1}^{n} I_{y_i} I_{t_i} \\ -\sum_{i=1}^{n} I_{z_i} I_{t_i} \end{bmatrix}$$

Σύμφωνα με την τελευταία εξίσωση η οπτική ροή μπορεί να βρεθεί βάσει του υπολογισμού της εικόνας. Επίσης υπάρχει η δυνατότητα προσθήκης κάποιας συνάρτησης βάρους W(i,j,k), όπου i,j,k ∈ [1,m]. Η συνάρτηση αυτή ενισχύει την επιρροή του κεντρικού εικονοστοιχείου στο παράθυρο. Ο αλγόριθμος αυτός δεν μπορεί να δώσει ένα ιδιαίτερα πυκνό πεδίο διανυσμάτων οπτικής ροής. Είναι αρκετά καλή μέθοδος σε εικόνες με αρκετό θόρυβο.

2.3.4 Πυραμιδοειδής αλγόριθμος Lukas - Kanade

Για την βελτίωση των αποτελεσμάτων των αλγορίθμων έχει ακολουθηθεί και η διαδικασία της κατασκευής πυραμίδας, όπου γίνεται ιεραρχική υλοποίηση των αλγορίθμων [18]. Πιο αναλυτικά γίνεται η κατασκευή μιας πυραμίδας Laplace, όπου στο χαμηλότερο επίπεδο βρίσκονται οι εικόνες στην αρχική του μορφή και στα ανώτερα επίπεδα υπάρχουν οι μικρότερες και ομαλοποιημένες εκδοχές τους. Αρχίζοντας από την κορυφή της πυραμίδας εκτελούνται οι αλγόριθμοι οπτικής ροής και τα αποτελέσματά τους χρησιμοποιούνται σαν μια αρχική εκτίμηση για την οπτική ροή του επόμενου επιπέδου.

Σε αυτή την μέθοδο [19], η οπτική ροή υπολογίζεται για μια μικρότερη έκδοση της εικόνας βάση του αλγορίθμου Lukas - Kanade. Το αποτέλεσμα είναι πιθανό να μην είναι πολύ ακριβές. Όμως τα προκύπτουσα διανύσματα χρησιμοποιούνται σαν αρχικές τιμές για τον υπολογισμό της οπτικής ροής της μεγαλύτερης εικόνας. Αυτή η μέθοδος μπορεί να χρησιμοποιηθεί για τον υπολογισμό μεγαλύτερων κινήσεων μεταξύ δυο διαδοχικών εικόνων απ' ότι με άλλες μεθόδους. Η οπτική ροή υπολογίζεται για κάθε αντικείμενο της εικόνας.

Η μέθοδος την πυραμίδας δίνει μικρότερο χρόνο εκτέλεσης αφού το πεδίο οπτικής ροής υπολογίζεται γρηγορότερα στα υψηλότερα επίπεδα και η αρχική εκτίμηση επιτρέπει την επιτάχυνση της διαδικασίας στα χαμηλότερα επίπεδα καθώς και την ποιότητα των αποτελεσμάτων. Στον πυραμιδοειδή αλγόριθμο Lukas – Kanade μπορούν να ανιχνευτούν μεγαλύτερες μετατοπίσεις σε σχέση με τον απλό αλγόριθμο,

καθώς αρχικά ανιχνεύονται στα υψηλότερα επίπεδα όπου το μέγεθός τους είναι μικρότερο και μεταφέρονται στην συνέχεια στα χαμηλότερα επίπεδα.

2.4 Εύρεση γωνιών

Προκειμένου να υπολογίσουμε την οπτική ροή σε μια εικόνα, θα πρέπει να βρούμε τα σημεία εκείνα της εικόνας που είναι διαφορετικά σε σχέση με τα περισσότερα σημεία και είναι εύκολο να ανιχνευτούν σε διαδοχικές εικόνες. Για παράδειγμα τα σημεία αυτά θα μπορούν να εμφανίζουν ένα τοπικό μέγιστο στην καμπυλότητα της συνάρτησης φωτεινότητας και περιέχουν σημαντικές πληροφορίες για την εικόνα. Τα σημεία αυτά μπορούν να αντιστοιχούν σε γωνίες των αντικειμένων όσο και σε αλλαγές στην υφή τους. Με τον όρο γωνίες δεν αναφερόμαστε μόνο στις γεωμετρικές γωνίες των αντικειμένων, αλλά γενικότερα στην διαφορά της φωτεινότητας μεταξύ των σημείων.

Υπάρχουν διάφορα είδη αλγορίθμων για τον εντοπισμό των γωνιών και χωρίζονται στις επόμενες κατηγορίες:

- **Βασισμένοι στην μορφολογία:** Οι αλγόριθμοι αυτοί αρχικά εντοπίζουν τις ακμές της εικόνας και στην συνέχεια χρησιμοποιούν μορφολογικές μεθόδους για τον εντοπισμό των γωνιών στις ακμές. Οι αλγόριθμοι αυτοί προϋποθέτουν την ύπαρξη γνώσης για τα χαρακτηριστικά των ακμών της εικόνας.
- **Βασισμένοι στα πρότυπα:** Στους αλγορίθμους αυτούς έχουν προκαθοριστεί κάποια πρότυπα εικόνων τα οποία προσπαθούμε να ταιριάξουμε τα σημεία της εικόνας
- **Βασισμένοι στο σήμα:** Οι αλγόριθμοι αυτοί χρησιμοποιούν τν εικόνα ως μια συνάρτηση φωτεινότητας στις δύο διαστάσεις και χρησιμοποιώντας τις παραγώγους της προσπαθούν να εντοπίσουν τις γωνίες στην εικόνα.

2.4.1 Αλγόριθμος Moravec

Πρόκειται για έναν από τους πρώτους αλγορίθμους εξεύρεσης γωνιών. Ο αλγόριθμος ελέγχει κάθε εικονοστοιχείο για να δει εάν πρόκειται για γωνία ή όχι με το να μετράει την ομοιότητα του κομματιού γύρω από το σημείο με τα γειτονικά σημεία του [20]. Το μέτρο ομοιότητας που χρησιμοποιείται είναι το άθροισμα των τετραγωνικών διαφορών. Μικρότερος αριθμός υποδεικνύει και μεγαλύτερη ομοιότητα.

2.4.2 Αλγόριθμος Harris

Ο αλγόριθμος Harris [21] είναι μια βελτίωση του προηγούμενου αλγορίθμου και χρησιμοποιεί τις παραγώγους δευτέρου βαθμού της φωτεινότητας της εικόνας $(\vartheta^2 x, \vartheta^2 y, \vartheta x, \vartheta y)$. Ο αλγόριθμος αυτός εντοπίζει επιτυχώς τις περισσότερες γωνίες σε μια εικόνα με ικανοποιητική ακρίβεια, ακόμα και με θόρυβο. Μπορεί επίσης να εντοπίζει σχεδόν τα ίδια σημεία εάν η εικόνα ληφθεί από μια ελαφρώς διαφορετική θέση.

2.4.3 Αλγόριθμος Shi – Tomasi

Αυτός ο αλγόριθμος [22] είναι μια βελτίωση του αλγορίθμου Harris. Οι γωνίες υπολογίζονται με βάση την ιδιοτιμή (eigenvalue). Μεγαλύτερη τιμή σημαίνει ότι το σημείο είναι πιθανόν πιο σταθερό. Έχει την μέγιστη ιδιοτιμή ανάμεσα στα γειτονικά εικονοστοιχεία και είναι μακριά από άλλα αντίστοιχα καλά σημεία.

3 Υλοποίηση

Στο κεφάλαιο αυτό γίνεται αναφορά στην υλοποίηση του συστήματος. Αρχικά περιγράφεται η δημιουργία του ρομπότ και στην συνέχεια ο προγραμματισμός της επικοινωνίας του υπολογιστή με το ρομπότ και η υλοποίηση του αλγορίθμου για την αποφυγή των εμποδίων.

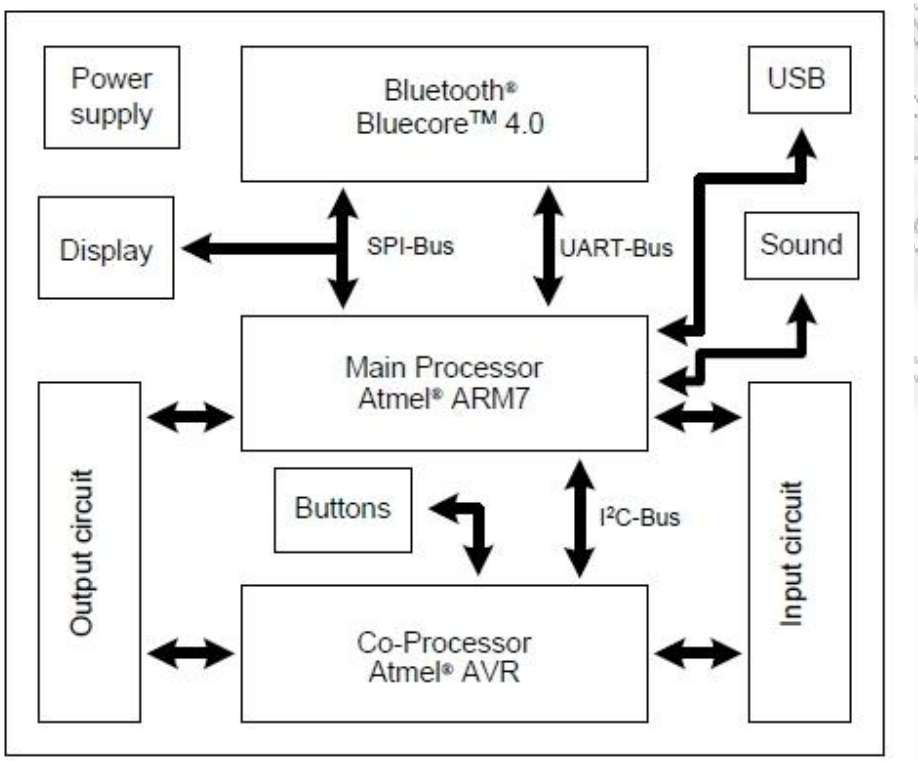
3.1 Ρομπότ

Όπως αναφέρθηκε σε προηγούμενη ενότητα το ρομπότ που θα χρησιμοποιηθεί για την εργασία κατασκευάστηκε χρησιμοποιώντας το σετ ρομποτικής Mindstorms NXT της εταιρίας LEGO. Για την κατασκευή του χρησιμοποιήθηκε η υπολογιστική μονάδα που παρέχεται από το σετ, δύο σερβοκινητήρες, διάφορα κομμάτια LEGO, καθώς και ένας φορητός ηλεκτρονικός υπολογιστής ASUS Netbook 1005 series (επεξεργαστής Intel Atom N280 1.66 GHz, μνήμη 1GB DDR-2, κάμερα 0.3M Pixels, Bluetooth V2.1 + EDR). Στις επόμενες σελίδες περιγράφονται αναλυτικά τα κομμάτια του συστήματος.

3.1.1 Υπολογιστική μονάδα

Η υπολογιστική μονάδα (Εικόνα 3-1) διαθέτει έναν επεξεργαστή 32 bit ARM7 (512 Kbytes Flash, 64 Kbytes RAM) και έναν επεξεργαστή 8 bit AVR (8Kbytes Flash, 512 bytes RAM). Μπορεί να λάβει δεδομένα από τέσσερις αισθητήρες και να συνδεθεί έως και με τρείς σερβοκινητήρες μέσω καλωδίων τύπου RJ12. Επίσης διαθέτει μια LCD οθόνη (100 X 64 εικονοστοιχεία) καθώς και τέσσερα κουμπιά για την διαχείριση του μενού επιλογών. Έχει την δυνατότητα αναπαραγωγής ήχων με ρυθμό δειγματοληψίας 8kHz. Το ρεύμα στην μονάδα δίνεται με την χρήση έξι μπαταριών AA (1.5V η καθεμία) ή μια επαναφορτιζόμενη μπαταρία λιθίου (για την εργασία αυτή χρησιμοποιήθηκε η δεύτερη επιλογή). Η επικοινωνία με κάποιον ηλεκτρονικό υπολογιστή μπορεί να γίνει μέσω ενσύρματης (USB) και ασύρματης επικοινωνίας (Bluetooth). Η επικοινωνία των κομματιών της μονάδας δίνεται στην εικόνα Εικόνα 3-2.

Εικόνα 3-1: Η υπολογιστική μονάδα του LEGO NXT.



Εικόνα 3-2: Τα μέρη από τα οποία αποτελείται η υπολογιστική μονάδα και τα μονοπάτια επικοινωνίας τους. Πηγή: http://mindstorms.lego.com

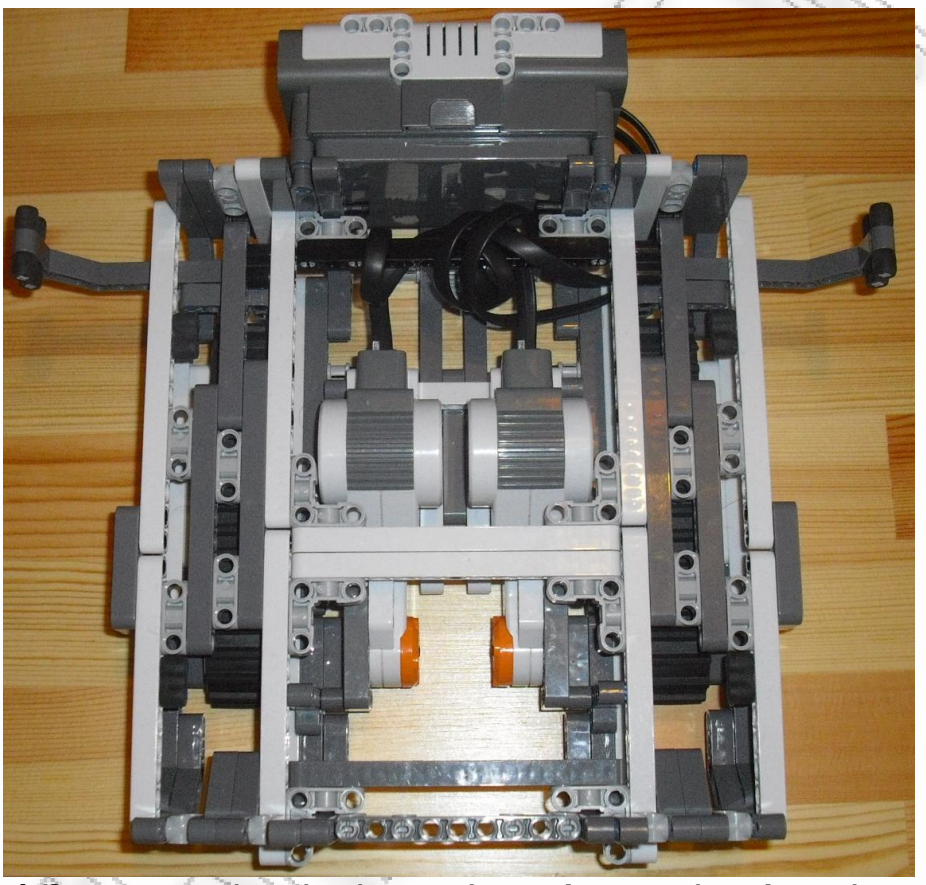
3.1.2 Σερβοκινητήρες

Για την εργασία θα χρησιμοποιηθούν δύο σερβοκινητήρες (Εικόνα 3-3), ένας για την αριστερή και ένας για την δεξιά κίνηση.

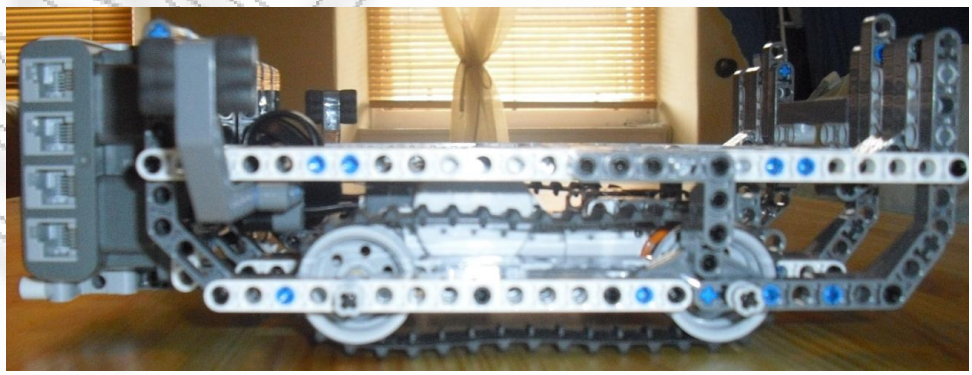
Εικόνα 3-3: Ένας τυπικός σερβοκινητήρας του σετ.

3.1.3 Κατασκευή ρομπότ

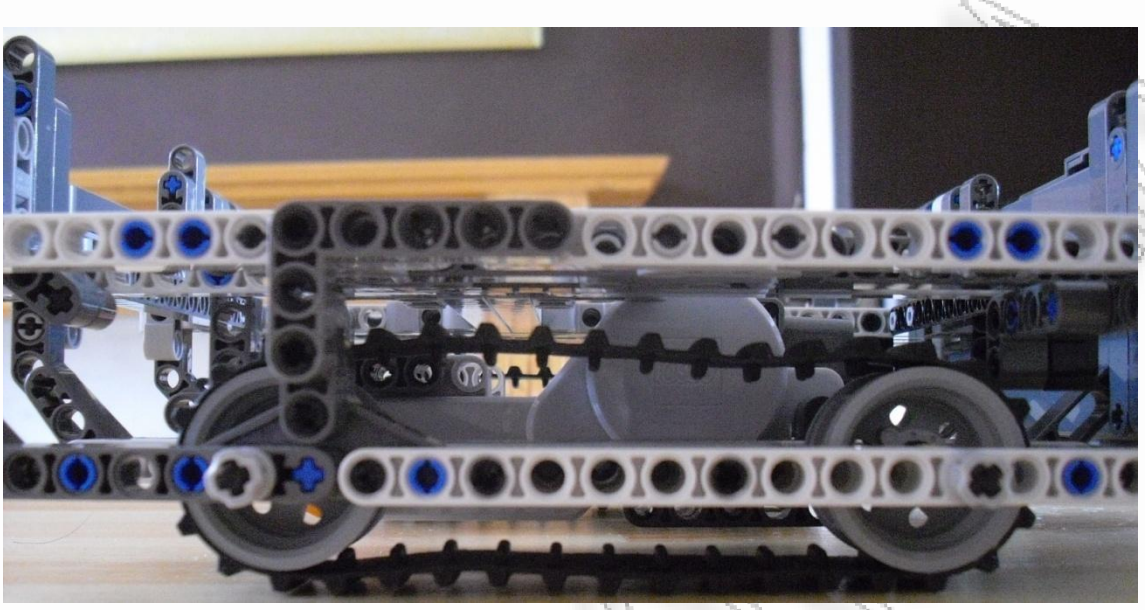
Για την κατασκευή του ρομπότ έπρεπε να ληφθούν υπόψη διάφορες παράμετροι. Πρώτα απ' όλα να είναι ελαφρύ ώστε να μπορεί να μετακινηθεί εύκολα χωρίς την υπερβολική κατανάλωση της μπαταρίας, αλλά ταυτόχρονα να είναι ισχυρό ώστε να μπορεί να κουβαλάει τον ηλεκτρονικό υπολογιστή με ευκολία. Έπειτα το κέντρο βάρους του ρομπότ θα έπρεπε να είναι κέντρο του ώστε να μην υπάρχουν κάθετες μετατοπίσεις κατά την πλοήγησή του. Τέλος, θα πρέπει να ληφθεί υπόψη το μέγεθος του ηλεκτρονικού υπολογιστή, ώστε να φτιαχτεί ένα πλαίσιο που θα τον κρατάει σταθερό, αλλά θα επιτρέπει την εύκολη μετακίνηση του συστήματος. Με βάση όλες τις προηγούμενες παραμέτρους κατασκευάστηκε το ρομπότ που απεικονίζεται στις επόμενες εικόνες.



Εικόνα 3-4: Όψη του ρομπότ από επάνω χωρίς τον ηλεκτρονικό υπολογιστή.



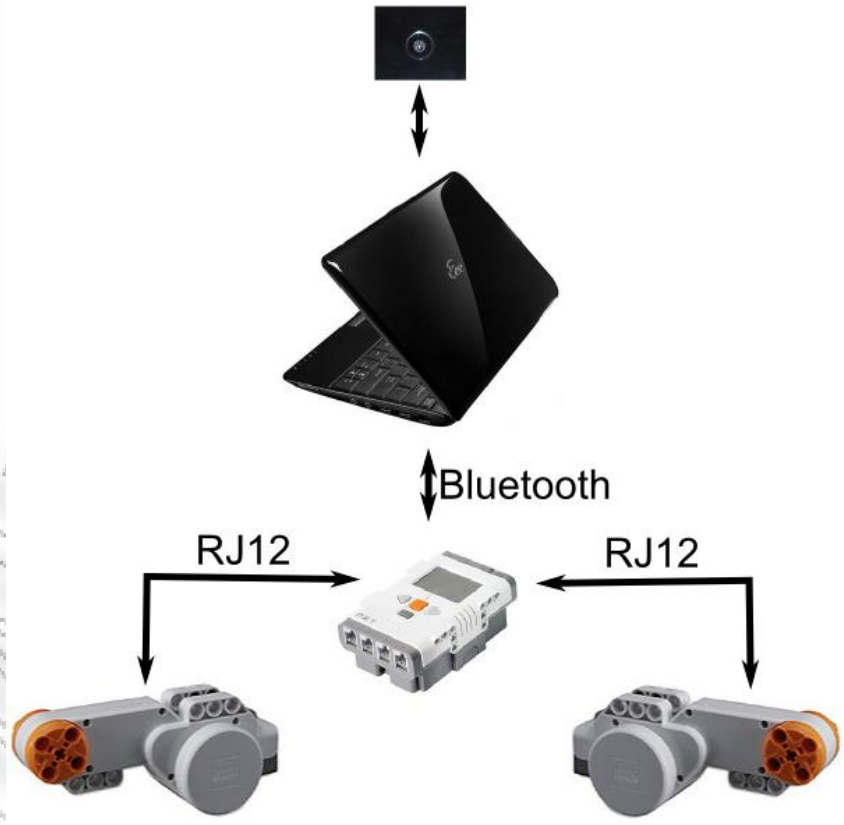
Εικόνα 3-5: Η πλαϊνή όψη του ρομπότ χωρίς τον ηλεκτρονικό υπολογιστή.



Εικόνα 3-6: Κοντινό πλάνο των ερπυστριών που χρησιμοποιεί το ρομπότ για την επαφή με το έδαφος.

Εικόνα 3-7: Η μπροστινή όψη του ρομπότ με τον υπολογιστή προσαρμοσμένο στην βάση του.

Εικόνα 3-8: Η πλαϊνή όψη του ρομπότ με τον υπολογιστή προσαρμοσμένο στην βάση του.



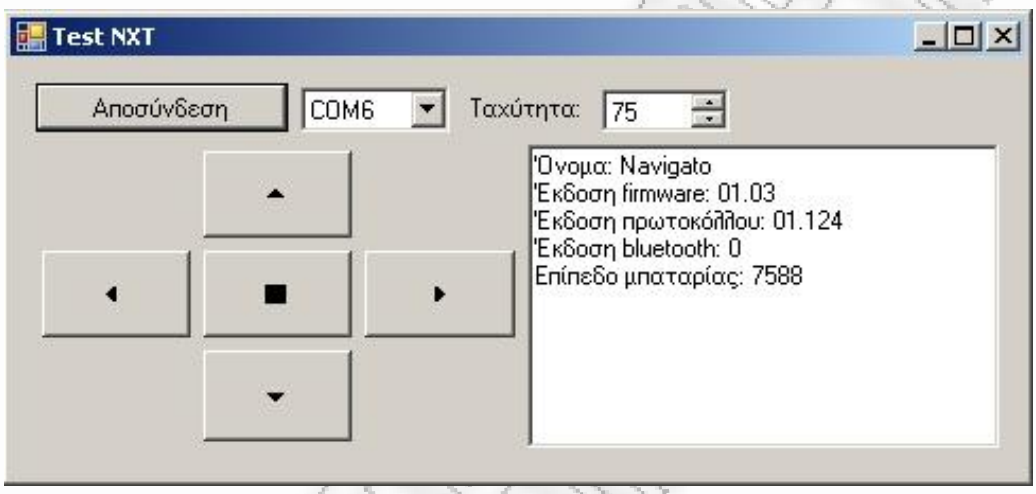
Εικόνα 3-9: Το διάγραμμα λειτουργίας του συστήματος

3.2 Λογισμικό

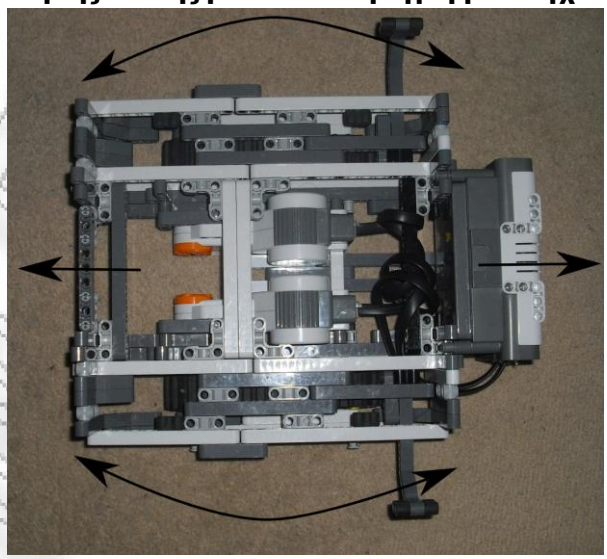
Στην ενότητα αυτή παρουσιάζεται το λογισμικό που χρησιμοποιήθηκε για την επικοινωνία με το ρομπότ. Τα σημαντικότερα κομμάτια του κώδικα παρατίθενται στο παράρτημα Β.

3.2.1 Επικοινωνία με το ρομπότ

Για την επικοινωνία με το ρομπότ χρησιμοποιήθηκαν τα επίσημα πρωτόκολλα επικοινωνίας (Bluetooth developer kit, hardware developer kit, software developer kit) που παρέχονται από την ιστοσελίδα http://mindstorms.lego.com/en-us/support/files/default.aspx. Με βάση τις οδηγίες αυτές κατασκευάστηκε η κλάση, με την χρήση της οποίας θα μετακινείται το ρομπότ.



Εικόνα 3-10: Απεικόνιση της οθόνης με το απλό πρόγραμμα ελέγχου του ρομπότ



Εικόνα 3-11: Οι κινήσεις που μπορεί να εκτελέσει το ρομπότ

3.2.2 Υπολογισμός της οπτικής ροής

Αρχικά ο προγραμματισμός σχετικά με τον έλεγχο του ρομπότ και τον υπολογισμό της οπτικής ροής ξεκίνησε να υλοποιείται στο περιβάλλον MATLAB. Η εκτέλεση των αλγορίθμων όμως έπαιρνε αρκετό χρόνο οπότε προτιμήθηκε η χρήση κάποιας βιβλιοθήκης στην γλώσσα C++.

Υπάρχουν διάφορες βιβλιοθήκες υπολογιστικής όρασης που θα μπορούσαν να χρησιμοποιηθούν στην εργασία αυτή. Ενδεικτικά παρουσιάζεται μια λίστα από τις πιο γνωστές βιβλιοθήκες.

- OpenCV (http://opencv.willowgarage.com)
- IVT (http://ivt.sourceforge.net/)
- VXL (http://vxl.sourceforge.net/)
- RAVL (http://www.ee.surrey.ac.uk/CVSSP/Ravl/)
- Camelia (http://camellia.sourceforge.net/)

Για την εργασία επιλέχτηκε η βιβλιοθήκη OpenCV λόγω του ότι έχει αρκετές έτοιμες βελτιστοποιημένες συναρτήσεις. Έτσι ο υπολογιστικός χρόνος για την εκτέλεση του αλγορίθμου θα μειωθεί σημαντικά.

Οι πιο σημαντικές συναρτήσεις που χρησιμοποιήθηκαν στην εργασία αυτή από την βιβλιοθήκη OpenCV αν φέρονται στις επόμενες γραμμές. Πιο συγκεκριμένα, αναφέρεται το όνομα της κάθε συνάρτησης και μια σύντομη περιγραφή της λειτουργίας της. Για τις παραμέτρους της κάθε συνάρτησης μπορείτε να ανατρέξετε στην σελίδα http://opencv.willowgarage.com/documentation/c/, καθώς και στο [23].

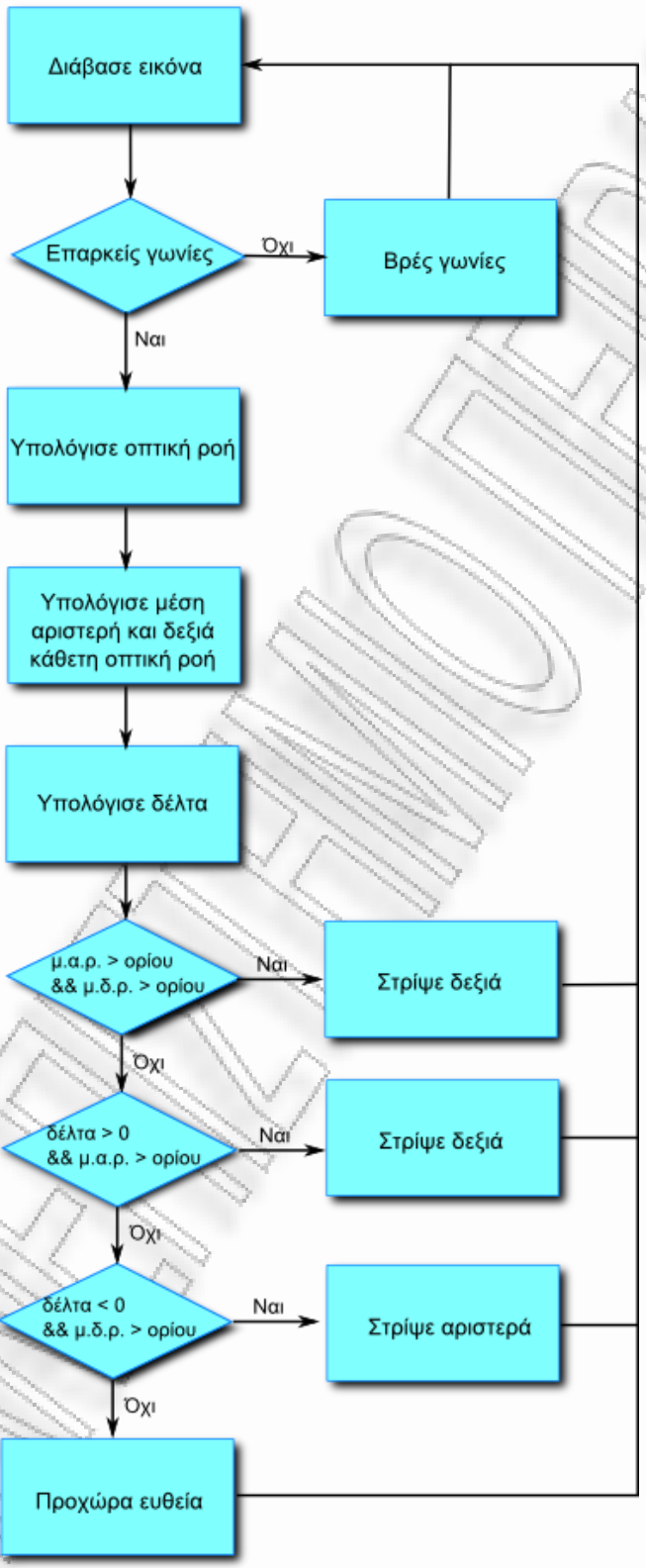
- cvGoodFeaturesToTrack: Η συνάρτηση αυτή υλοποιεί τον αλγόριθμο των Shi και Tomasi για την εύρεση σημείων στην εικόνα που είναι πιο εύκολα να εντοπιστούν.
- cvFindCornerSubPix: Η συνάρτηση αυτή βρίσκει γωνίες στην εικόνα με ακρίβεια μικρότερη του εικονοστοιχείου.
- cvCalcOpticalFlowPyrLK: Η συνάρτηση αυτή υλοποιεί τον αλγόριθμο των Lukas και Kanade με την χρήση πυραμίδας.

3.2.3 Στρατηγική αποφυγής εμποδίων

Ο αλγόριθμος λειτουργεί ως εξής: Αρχικά ανακτάται η εικόνα από την κάμερα του υπολογιστή. Στην συνέχεια εάν δεν υπάρχει επαρκής αριθμός γωνιών για την αριστερή και δεξιά πλευρά της εικόνας χρησιμοποιείται ο αλγόριθμος των Shi και Tomasi για την εύρεση των γωνιών. Εάν υπάρχει επαρκής αριθμός γωνιών αρχικά υπολογίζεται η οπτική ροή για όλα τα σημεία με την χρήση της πυραμιδωτής μορφής του αλγορίθμου των Lukas και Kanade. Έπειτα υπολογίζεται η μέση αριστερή και δεξιά κάθετη οπτική ροή, καθώς και η μεταβλητή δέλτα που δίνεται από τον τύπο:

$$δέλτα = \frac{μέση\ αριστερή\ κάθετη\ οπτική\ ροή - μέση\ δεξιά\ κάθετη\ οπτική\ ροή}{μέση\ αριστερή\ κάθετη\ οπτική\ ροή + μέση\ δεξιά\ κάθετη\ οπτική\ ροή}$$

Εικόνα 3-12: Ένα παράδειγμα εκτίμησης της οπτικής ροής όταν το ρομπότ στρίβει επί τόπου



Εικόνα 3-13: Το διάγραμμα του αλγορίθμου αποφυγής εμποδίων. Σαν μ.α.ρ. ορίζεται η μέση αριστερή κάθετη οπτική ροή και σαν μ.δ.ρ. η μέση δεξιά κάθετη οπτική ροή.

Υπολογίζουμε μόνο την κάθετη οπτική ροή δηλαδή το Δy διότι το ρομπότ μπορεί να στρίβει μόνο επί τόπου (on the spot). Αυτό σημαίνει ότι όταν υπάρχει μεγάλο Δx οφείλεται στην αλλαγή κατεύθυνσης του ρομπότ και όχι στην ύπαρξη κάποιου εμποδίου, όπως φαίνεται στην ακόλουθη εικόνα.
Αφού υπολογιστούν αυτές οι μεταβλητές ακολουθείται μια απλή στρατηγική αποφυγής των εμποδίων. Αυτή λειτουργεί ως εξής:

- Εάν και οι δύο μέσες τιμές των κάθετων οπτικών ροών υπερβαίνουν ένα όριο τότε υπάρχει κάποιο εμπόδιο και στην δεξιά και στην αριστερή μεριά, οπότε το ρομπότ θα στρίψει δεξιά. Φυσικά και το ρομπότ θα μπορούσε να στρίψει αριστερά σε αυτή την περίπτωση.
- Εάν η μεταβλητή δέλτα είναι μεγαλύτερη του μηδέν και η μέση αριστερή κάθετη οπτική ροή μεγαλύτερη του ορίου τότε υπάρχει εμπόδιο αριστερά οπότε το ρομπότ θα στρίψει δεξιά.
- Εάν η μεταβλητή δέλτα είναι μικρότερη του μηδέν και η μέση δεξιά κάθετη οπτική ροή μεγαλύτερη του ορίου τότε θα υπάρχει εμπόδιο στα δεξιά οπότε το ρομπότ θα πρέπει να στρίψει αριστερά
- Εάν τέλος, δεν συμβαίνει τίποτα από τα προηγούμενα, δεν υπάρχει κάποιο εμπόδιο οπότε το ρομπότ θα προχωρήσει ευθεία.

Στο σημείο αυτό θα πρέπει να τονιστεί ότι στο πρόγραμμα θα υπάρχουν δύο διαφορετικά νήματα. Το ένα νήμα θα ασχολείται με τον υπολογισμό της οπτικής ροής και των υπόλοιπων μεταβλητών που χρησιμοποιεί ο αλγόριθμος και ένα δεύτερο νήμα το οποίο ανάλογα με τις τιμές των μεταβλητών δίνει την αντίστοιχη εντολή στο ρομπότ για να μετακινηθεί.

Εικόνα 3-14: Παράδειγμα λειτουργίας του αλγορίθμου, όταν υπάρχει εμπόδιο στα αριστερά. Τα πράσινα σημεία είναι εκείνα που δεν παρουσίασαν σημαντική κίνηση, ενώ τα κόκκινα βέλη δείχνουν την κίνηση των σημείων σε δύο διαδοχικές εικόνες.

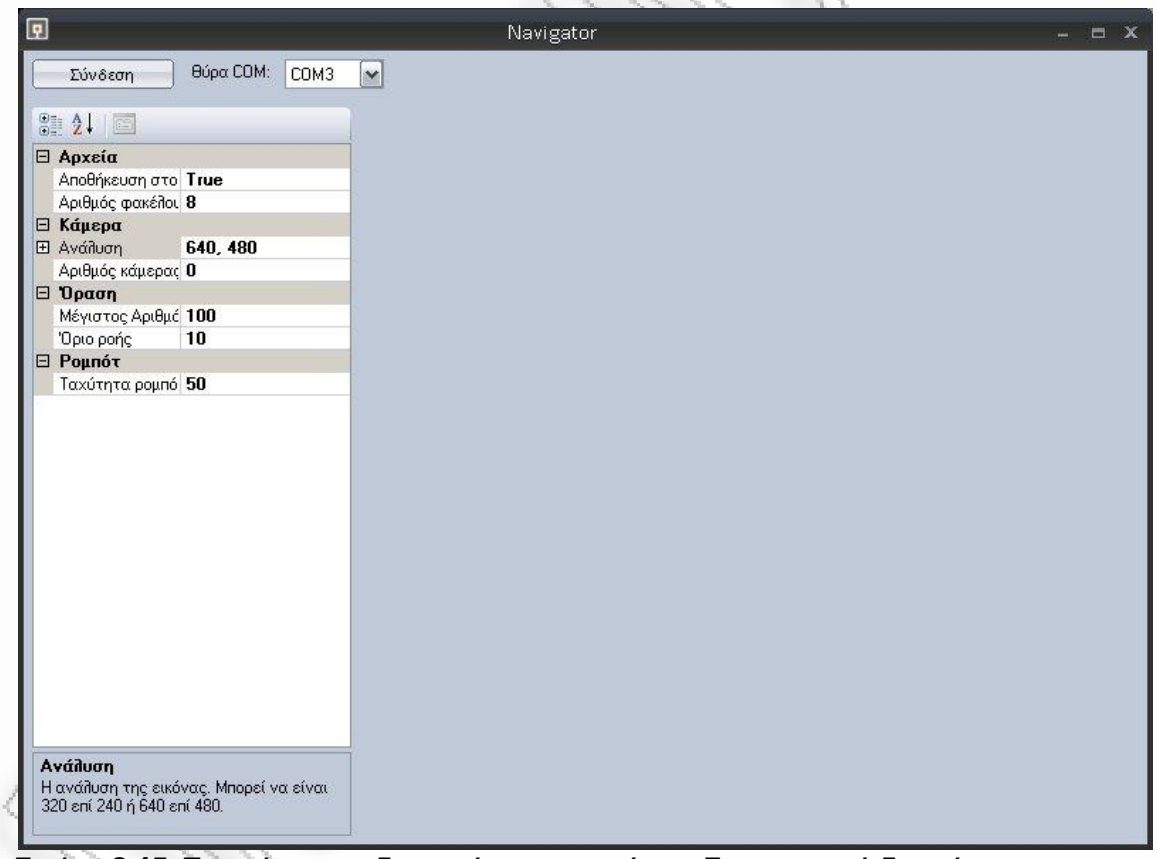
3.2.4 Διεπαφή χρήστη

Για την ευκολότερη λειτουργία του προγράμματος καθώς και την αποθήκευση δεδομένων, δημιουργήθηκε ένα πρόγραμμα σε C++.NET. Το πρόγραμμα αυτό έχει τις εξής δυνατότητες:

- Αποθήκευση των παραμέτρων σε ένα αρχείο xml.
- Εύρεση λίστας με τις σειριακές θύρες του υπολογιστή και σύνδεση με την επιλεγμένη θύρα.
- Δυνατότητα αποθήκευσης των αποτελεσμάτων των δοκιμών κατά την διάρκεια της εκτέλεσης του προγράμματος.
- Επιλογή της ανάλυσης της εικόνας. Οι δύο δυνατοί συνδυασμοί είναι 320 X 240 και 640 X 480 εικονοστοιχεία.
- Επιλογή αριθμού της κάμερας. Κάποιοι υπολογιστές διαθέτουν περισσότερες της μίας κάμερας οπότε δίνεται η δυνατότητα επιλογής της κατάλληλης.
- Επιλογή μέγιστου αριθμού γωνιών που θα προσπαθήσει να βρει ο αλγόριθμος.
- Επιλογή του ορίου οπτικής ροής που θα χρειαστεί το σύστημα για την επιλογή της κατεύθυνσης του ρομπότ.
- Επιλογή της ταχύτητας του ρομπότ. Οι τιμές της μεταβλητής αυτής είναι από 1 έως 100. Με βάση την τιμή της μεταβλητής αυτής υπολογίζεται και ο χρόνος που θα εκτελέσει μια εντολή το ρομπότ βάσει του τύπου:

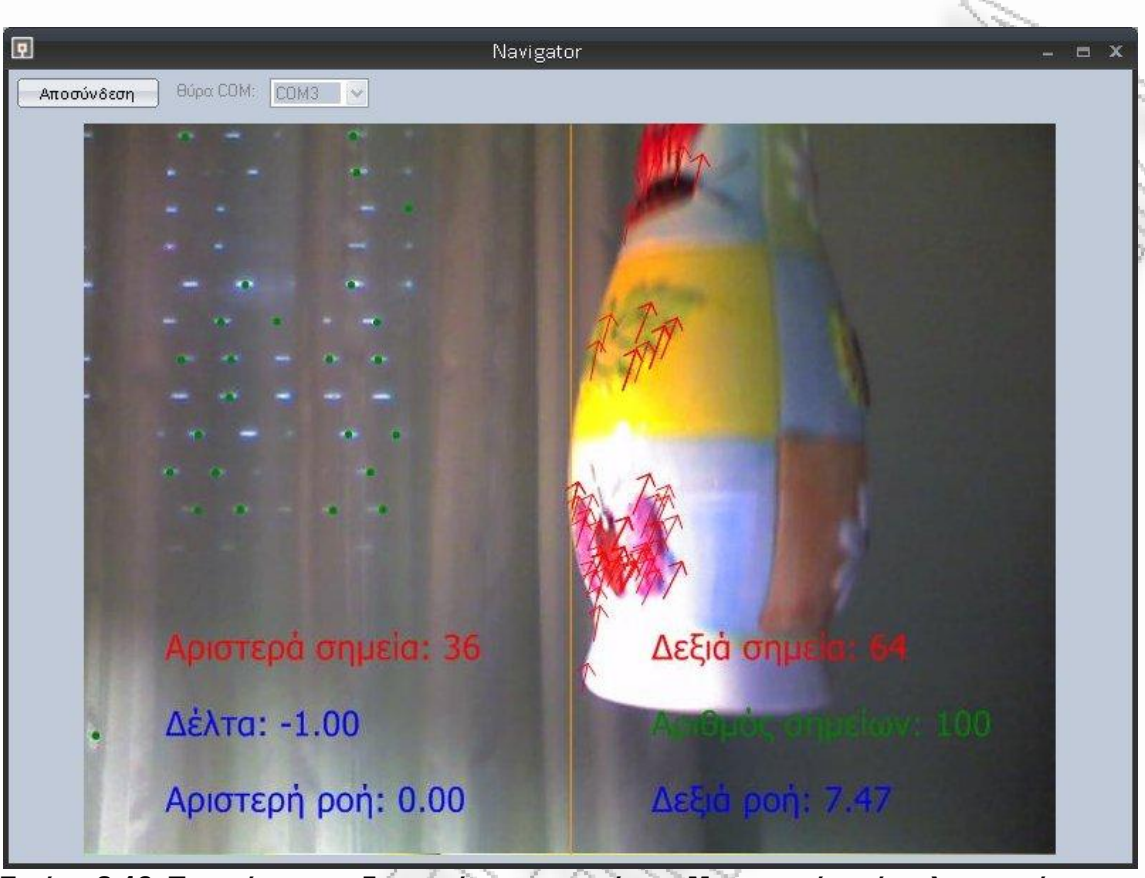
$$Χρόνος\ εκτέλεσης = -4 \cdot ταχύτητα\ ρομπότ + (1 + |δέλτα|) \cdot 400$$

Όπου η μονάδα μέτρησης του χρόνου εκτέλεσης είναι σε milliseconds.



Εικόνα 3-15: Το πρόγραμμα διεπαφής με τον χρήστη. Στα αριστερά διακρίνονται οι παράμετροι που μπορεί να λάβει το σύστημα

Στην συνέχεια παρατίθεται μια εικόνα του προγράμματος κατά την εκτέλεση του αλγορίθμου.



Εικόνα 3-16: Το πρόγραμμα διεπαφής με τον χρήστη. Μια τυπική εικόνα λειτουργίας του προγράμματος κατά την διάρκεια της λειτουργίας του αλγορίθμου.

4 Πειράματα

Στο κεφάλαιο αυτό παρουσιάζονται τα αποτελέσματα μιας σειράς δοκιμασιών στις οποίες υποβλήθηκε το ρομπότ προκειμένου να εξεταστεί η ομαλή λειτουργία του. Κάθε μια από τις δοκιμασίες πραγματοποιήθηκε δέκα φορές. Η πρώτη δοκιμασία αφορά την απλή αποφυγή ενός και δύο εμποδίων, η δεύτερη την αποφυγή σύγκρουσης με έναν τοίχο, η τρίτη την πλοήγηση σε ένα διάδρομο και η τέταρτη την πλοήγηση σε έναν μεγάλο χώρο.

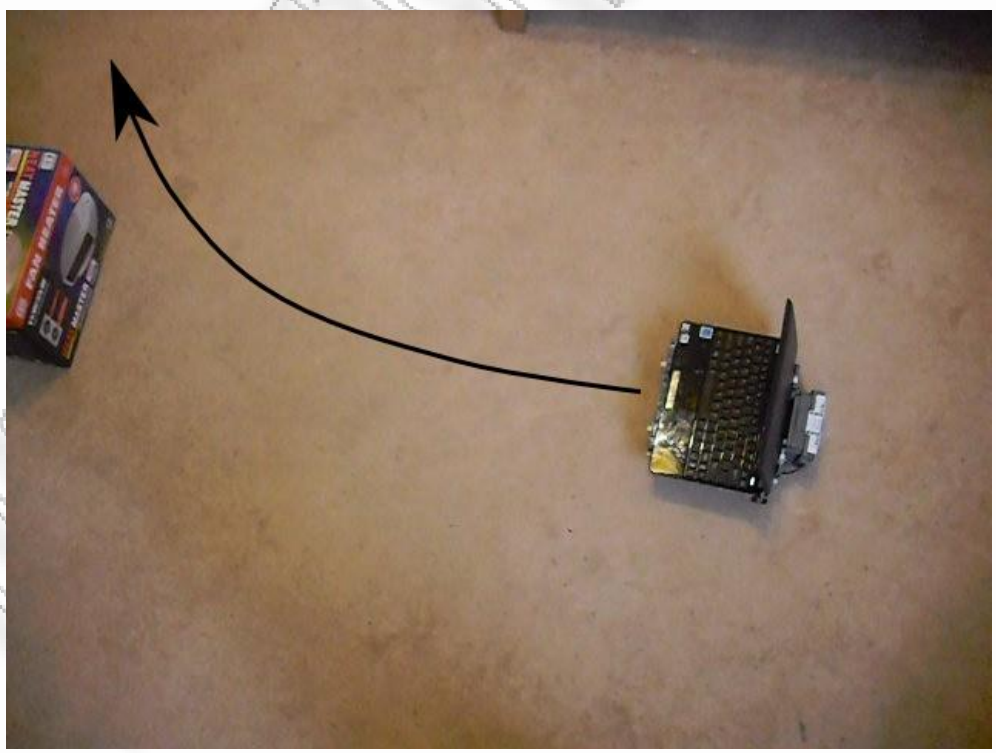
Οι τιμές που δόθηκαν στις παραμέτρους του συστήματος ήταν οι ίδιες για όλες τις δοκιμασίες και ήταν οι εξής:

- Ανάλυση εικόνας: 640 X 480
- Μέγιστος αριθμός σημείων: 100
- Όριο ροής: 10
- Ταχύτητα ρομπότ: 50

Στις επόμενες σελίδες παρατίθονται μια εικόνα που απεικονίζει την πορεία του ρομπότ, μια τυπική εικόνα που λαμβάνει και επεξεργάζεται το σύστημα, καθώς και διαγράμματα που δείχνουν τις τιμές της αριστερής και της δεξιάς μέσης ροής και της τιμής της μεταβλητής δέλτα, για μια επιτυχή δοκιμή από κάθε δοκιμασία. Στην ηλεκτρονική διεύθυνση http://www.youtube.com/watch?v=sGg_9rcbrSU μπορείτε να δείτε ένα συνοπτικό βίντεο με τις δοκιμασίες και την επίδοση του ρομπότ σε αυτές. Στο παράρτημα Α μπορείτε να δείτε μια τυπική ακολουθία εικόνων της πορείας του ρομπότ και της αντίστοιχης εικόνας που επεξεργάζεται το σύστημα.

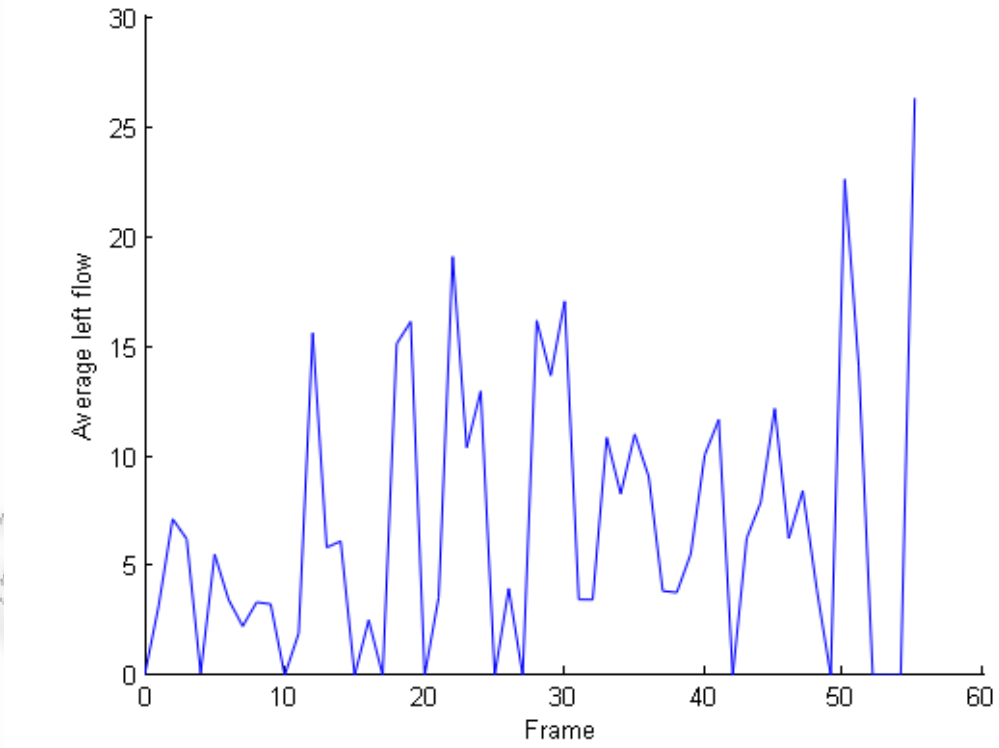
4.1 Απλή αποφυγή εμποδίων

Στην δοκιμασία αυτή το ρομπότ απέφυγε το ένα εμπόδιο και στις δέκα προσπάθειες, ενώ η αποφυγή των δύο εμποδίων πραγματοποιήθηκε τις εννιά από τις δέκα φορές. Για εμπόδια χρησιμοποιήθηκαν δύο κουτιά με αρκετά χαρακτηριστικά, ώστε να γίνεται ακριβέστερα ο υπολογισμός της οπτικής ροής.

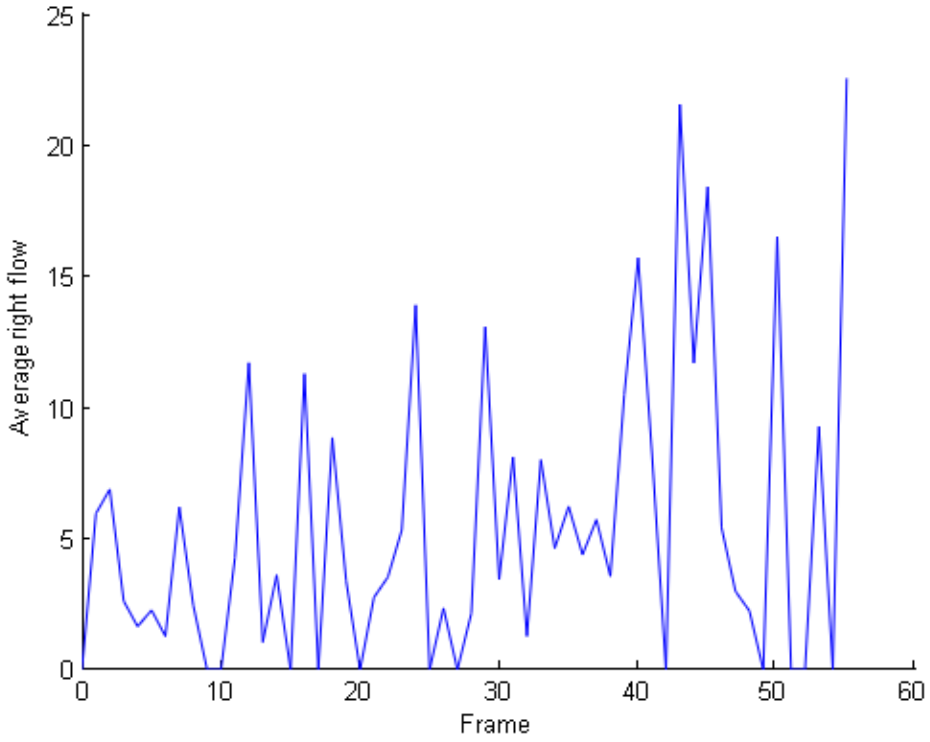


Εικόνα 4-1: Το μονοπάτι που ακολούθησε το ρομπότ για να αποφύγει το εμπόδιο

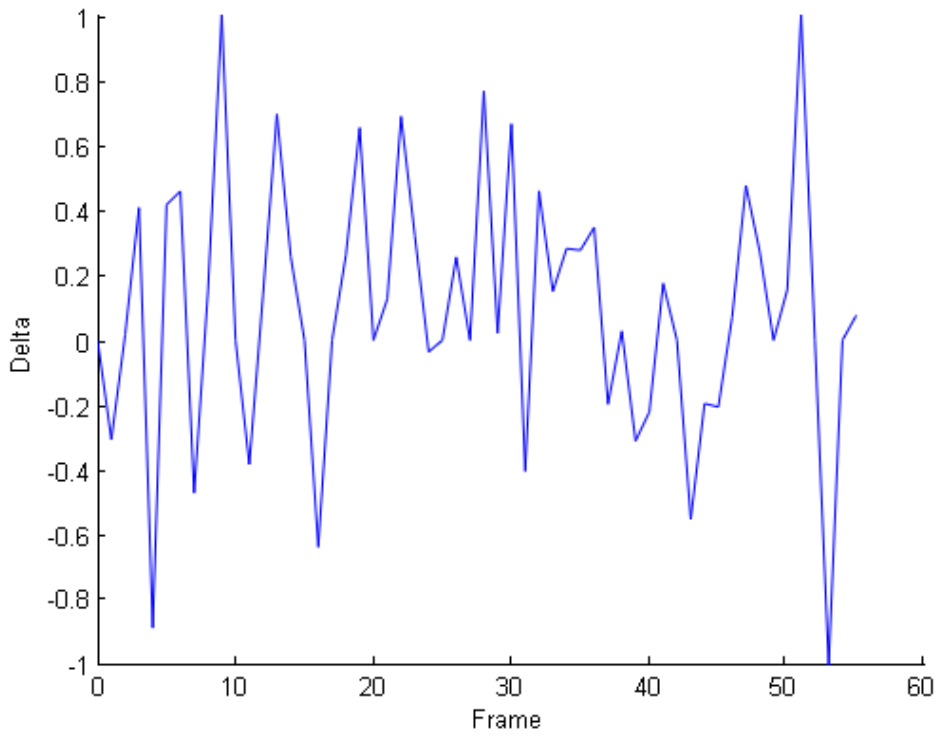
Εικόνα 4-2: Μια από τις εικόνες που βλέπει και επεξεργάζεται το ρομπότ



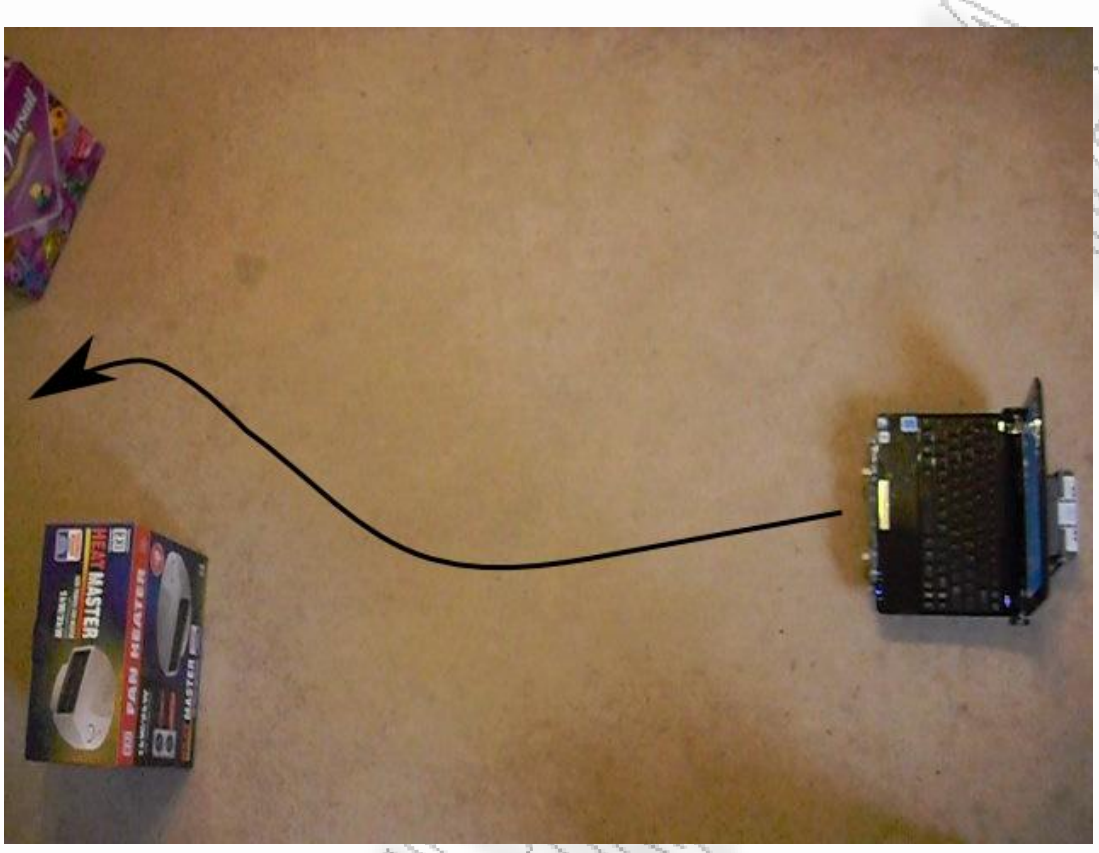
Εικόνα 4-3: Μέση τιμή της αριστερής οπτικής ροής για κάθε εικόνα



Εικόνα 4-4: Μέση τιμή της δεξιάς οπτικής ροής για κάθε εικόνα

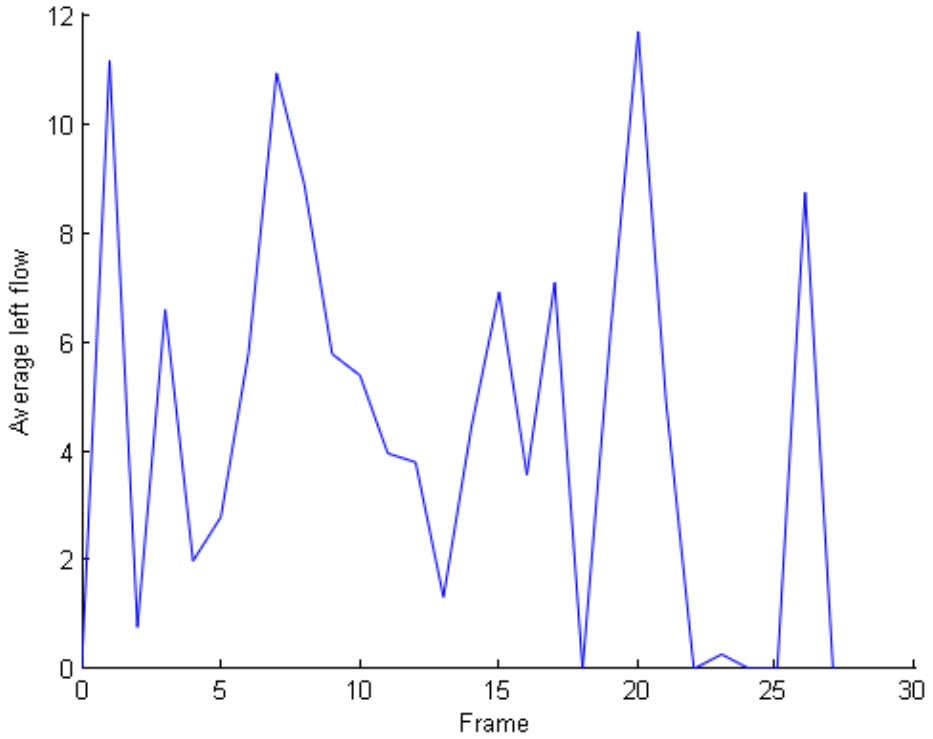


Εικόνα 4-5: Η τιμή της μεταβλητής δέλτα για κάθε εικόνα

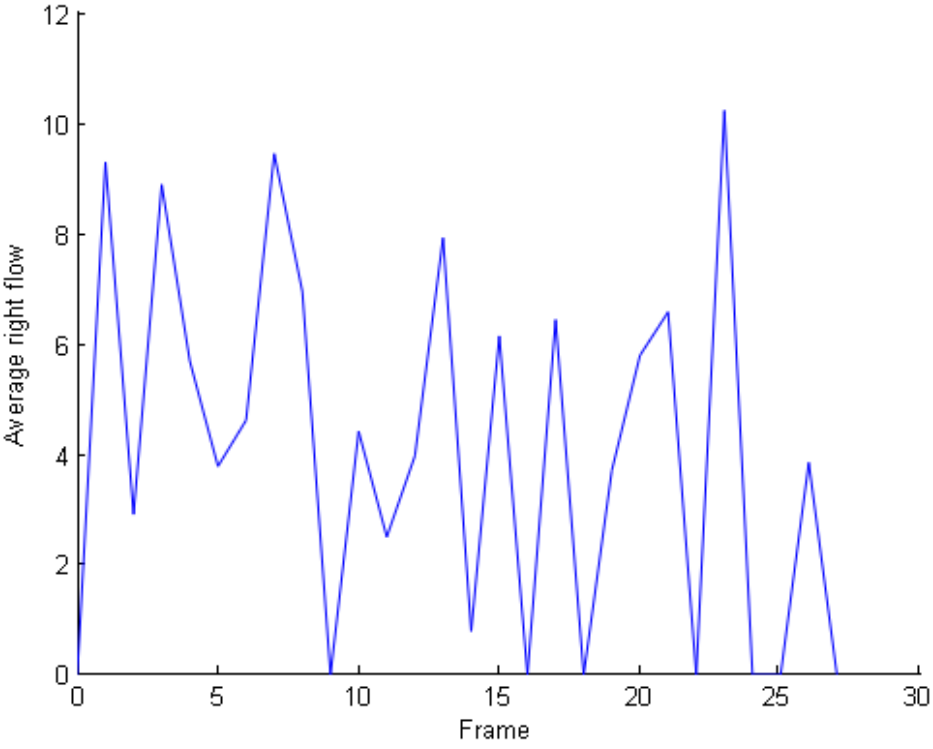


Εικόνα 4-6: Το μονοπάτι που ακολούθησε το ρομπότ για να αποφύγει τα δύο εμπόδια

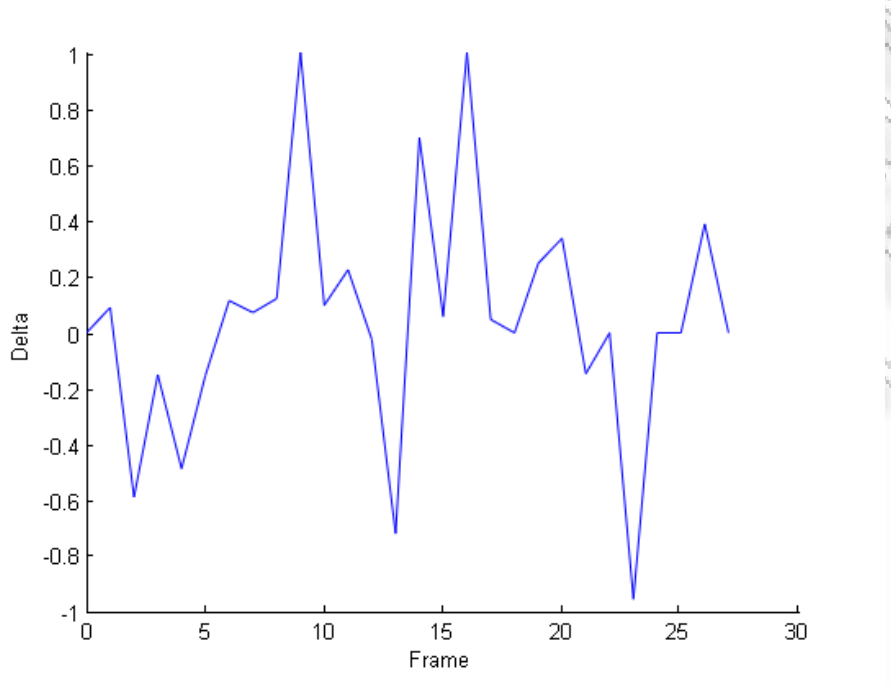
Εικόνα 4-7: Μια από τις εικόνες που βλέπει και επεξεργάζεται το ρομπότ



Εικόνα 4-8: Μέση τιμή της αριστερής οπτικής ροής για κάθε εικόνα



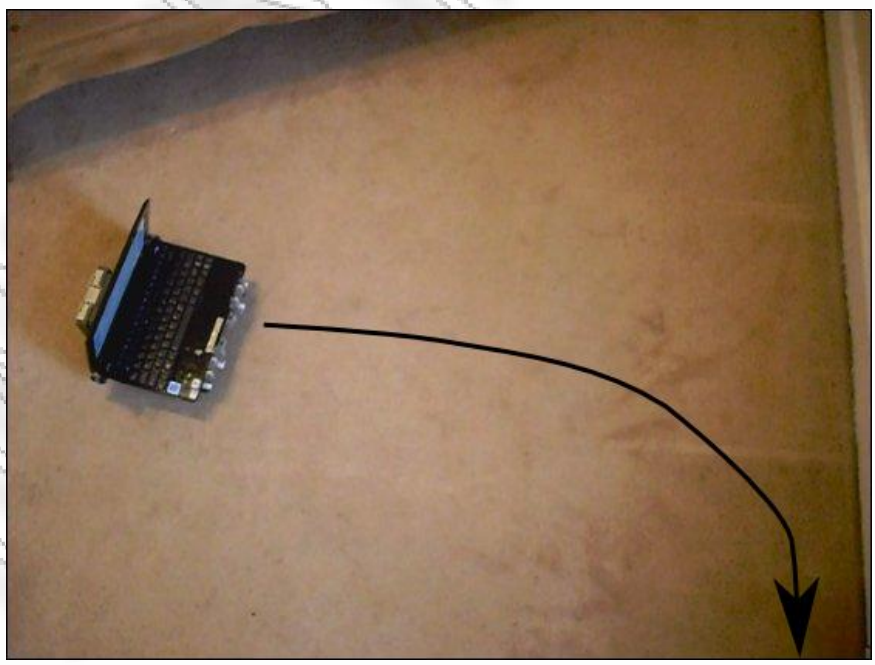
Εικόνα 4-9: Μέση τιμή της δεξιάς οπτικής ροής για κάθε εικόνα



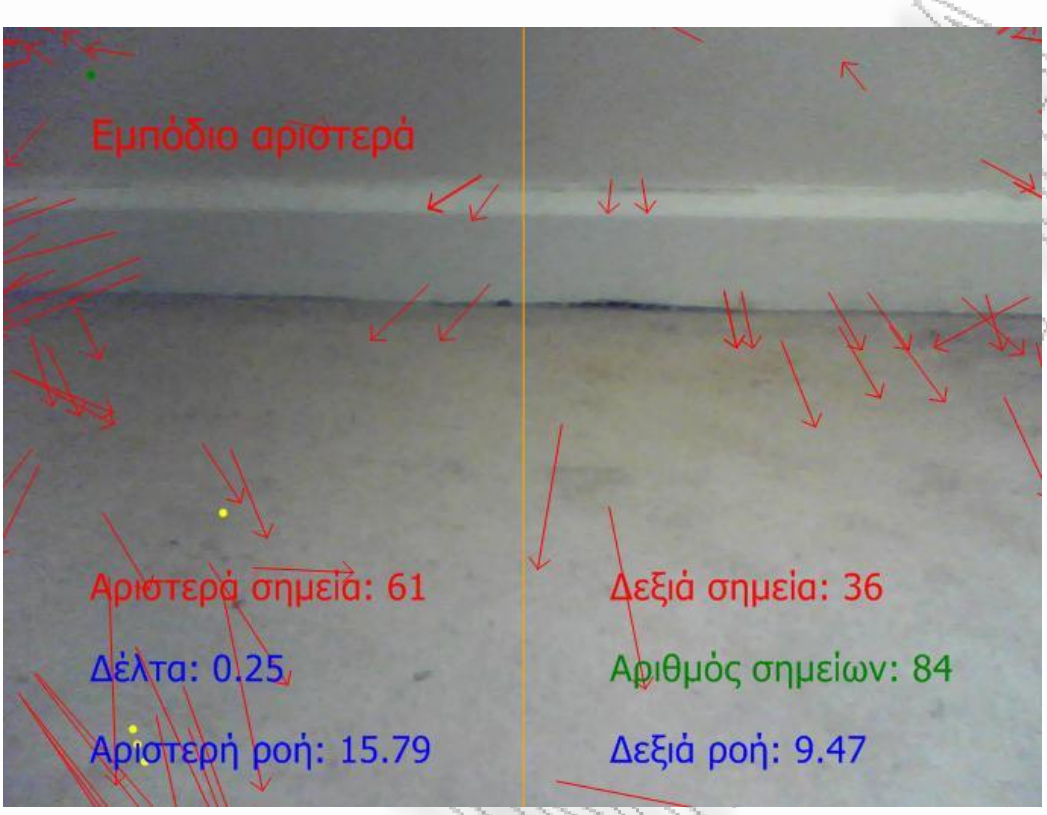
Εικόνα 4-10: Η τιμή της μεταβλητής δέλτα για κάθε εικόνα

4.2 Αποφυγή τοίχου

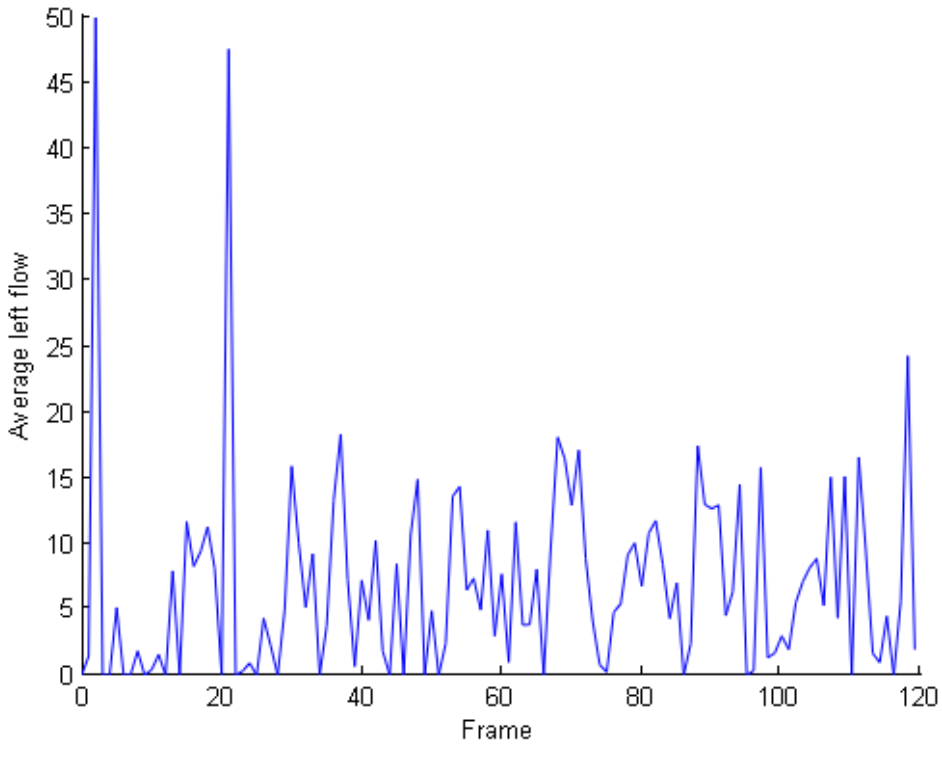
Η δοκιμασία αυτή αποδείχτηκε η πιο δύσκολη από όλες καθώς ο τοίχος δεν έχει αρκετά χαρακτηριστικά τα οποία μπορούν να χρησιμοποιηθούν επαρκώς για τον ακριβέστερο υπολογισμό της οπτικής ροής. Σαν αποτέλεσμα το ρομπότ παρότι πραγματοποίησε κάποια αλλαγή πορείας σε κάθε περίπτωση, μόνο στις πέντε από αυτές κατάφερε να αποφύγει επιτυχώς την σύγκρουση με τον τοίχο.



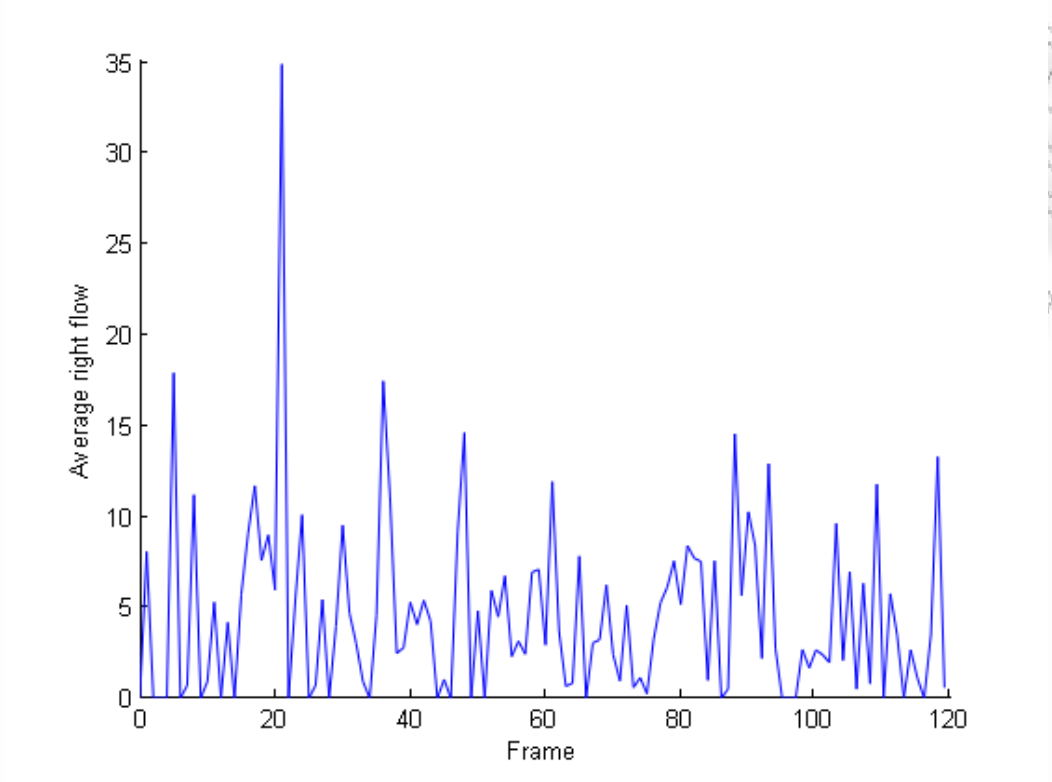
Εικόνα 4-11: Το μονοπάτι που ακολούθησε το ρομπότ για να αποφύγει τον τοίχο



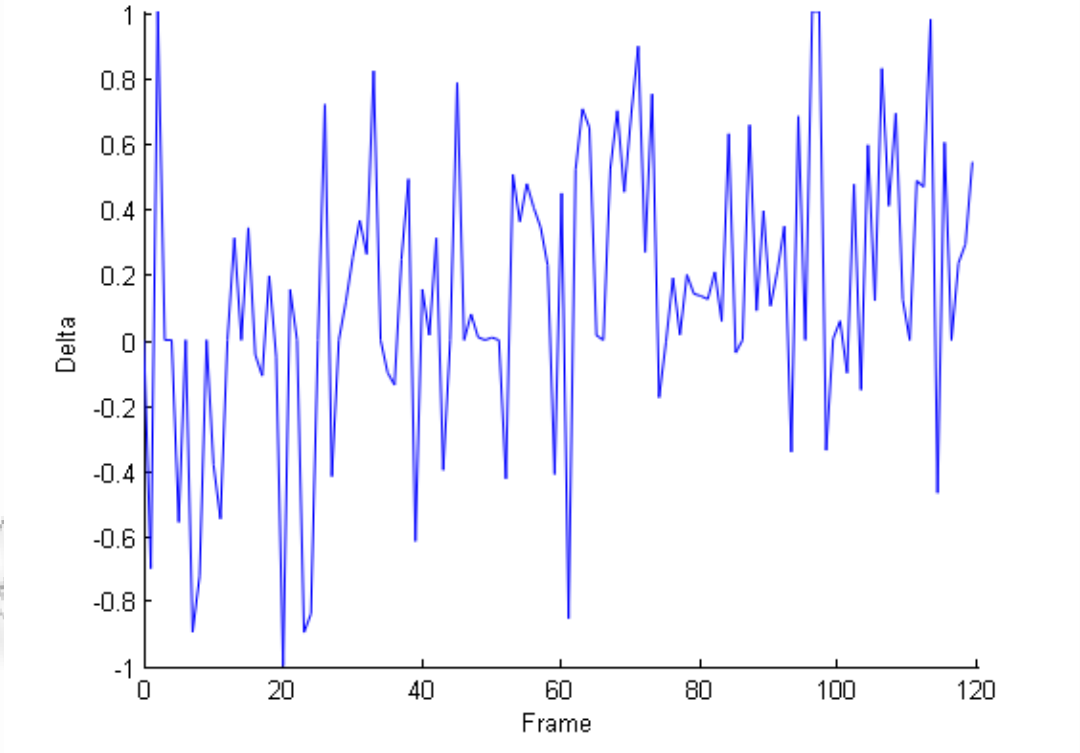
Εικόνα 4-12: Μια από τις εικόνες που βλέπει και επεξεργάζεται το ρομπότ



Εικόνα 4-13: Μέση τιμή της αριστερής οπτικής ροής για κάθε εικόνα



Εικόνα 4-14: Μέση τιμή της δεξιάς οπτικής ροής για κάθε εικόνα



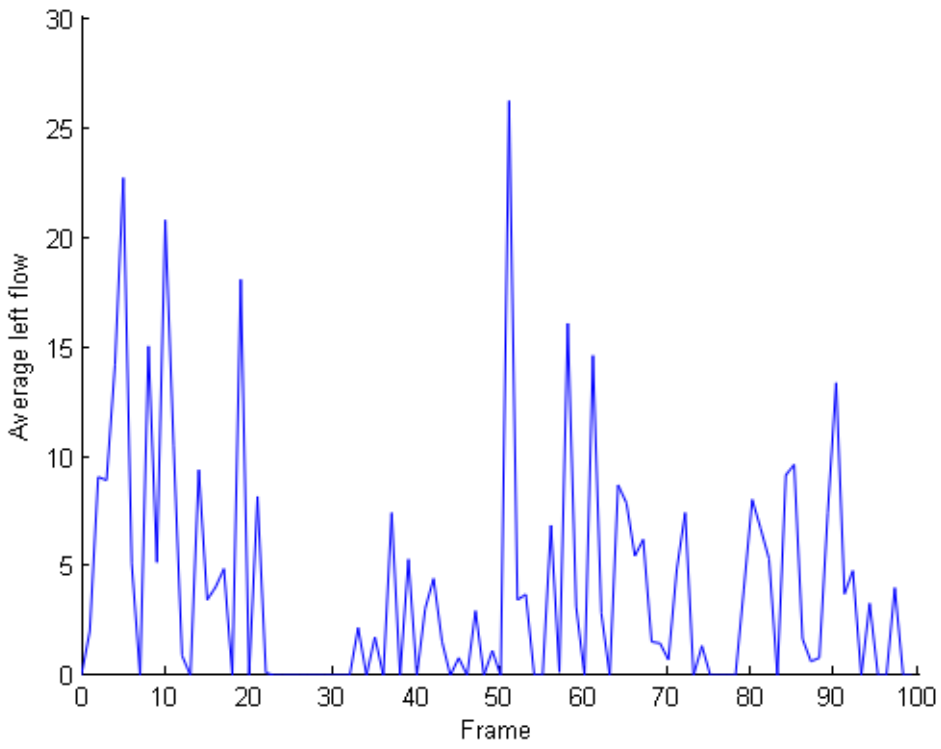
Εικόνα 4-15: Η τιμή της μεταβλητής δέλτα για κάθε εικόνα

4.3 Πλοήγηση σε ένα διάδρομο

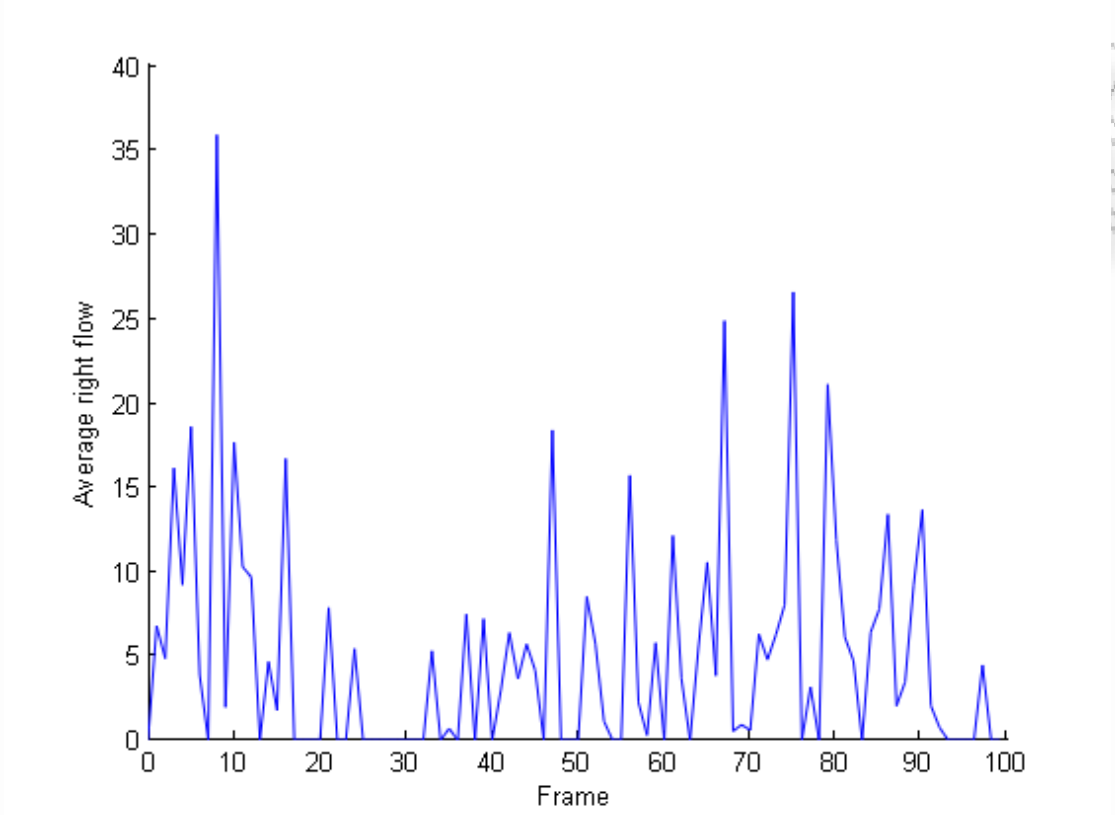
Στην δοκιμασία αυτή το ρομπότ έπρεπε να πλοηγηθεί επιτυχώς σε έναν διάδρομο χωρίς να συγκρουστεί με κάποιον τοίχο. Η δοκιμασία ήταν επιτυχής στις οκτώ από τις δέκα προσπάθειες. Το σύστημα εντόπιζε κάποια σημεία από το πάτωμα τα οποία χρησιμοποιούσε για τον υπολογισμό της οπτικής ροής με αποτέλεσμα κάποιες φορές να στρίβει προς την λάθος κατεύθυνση.

Εικόνα 4-16: Το μονοπάτι που ακολούθησε το ρομπότ για να πλοηγηθεί στον διάδρομο

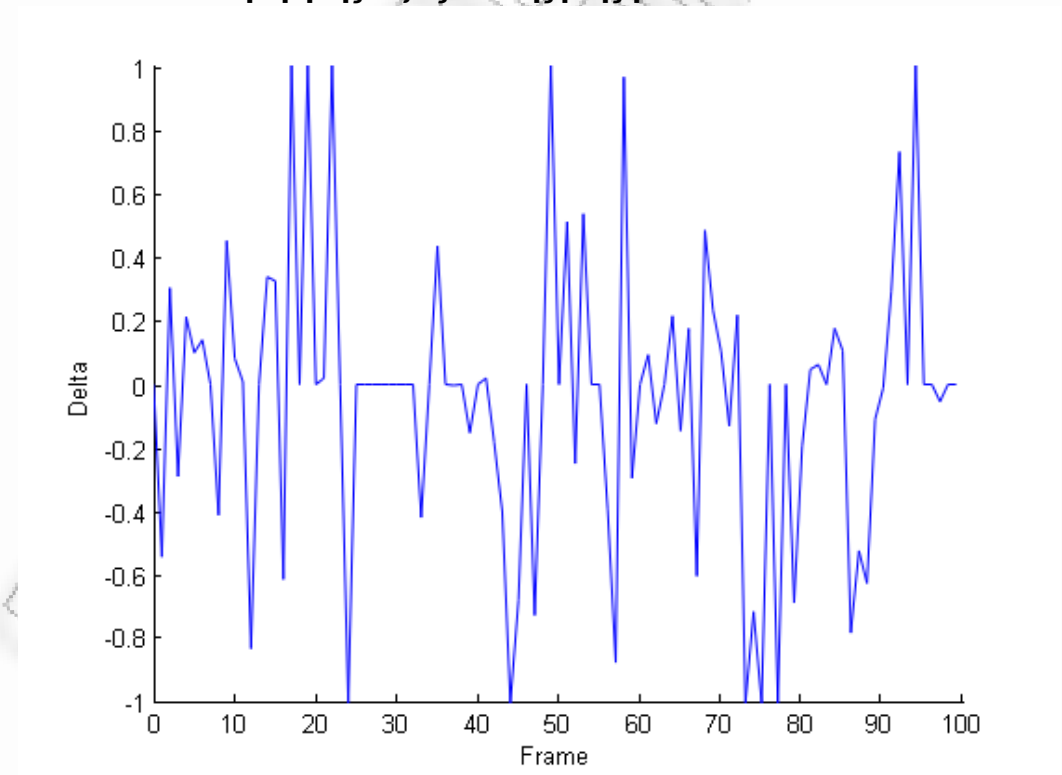
Εικόνα 4-17: Μια από τις εικόνες που βλέπει και επεξεργάζεται το ρομπότ



Εικόνα 4-18: Μέση τιμή της αριστερής οπτικής ροής για κάθε εικόνα



Εικόνα 4-19: Μέση τιμή της δεξιάς οπτικής ροής για κάθε εικόνα

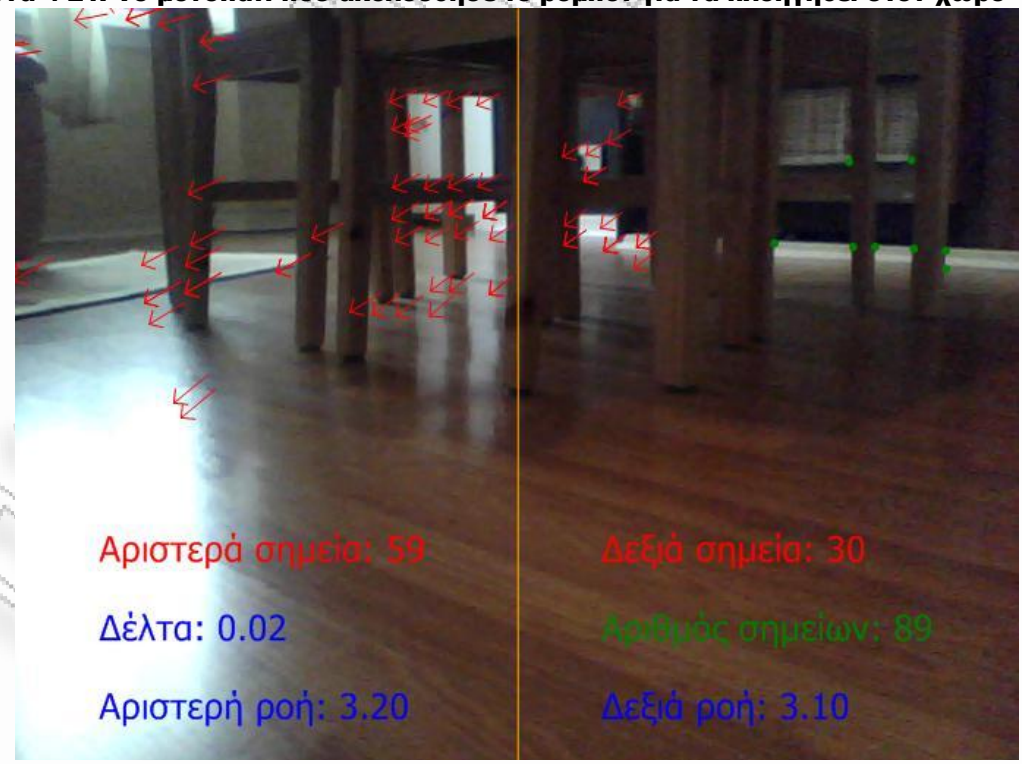


Εικόνα 4-20: Η τιμή της μεταβλητής δέλτα για κάθε εικόνα

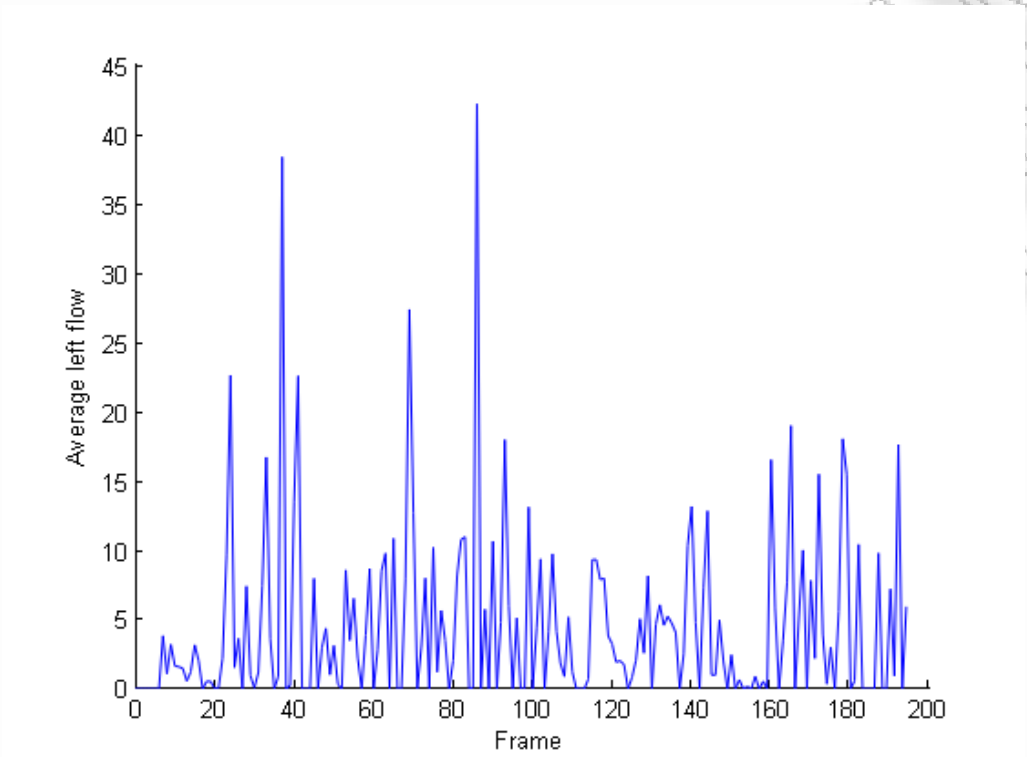
4.4 Πλοήγηση σε ένα μεγάλο χώρο

Στην τελευταία δοκιμασία το ρομπότ αφέθηκε να μετακινηθεί ελεύθερα σε έναν μεγαλύτερο χώρο, ώστε να διαπιστωθεί εάν μπορεί να μετακινηθεί με ευκολία σε αυτή την περίπτωση. Οι επτά από τις δέκα προσπάθειες ήταν επιτυχείς. Το ρομπότ δεν μπόρεσε να αποφύγει τα εμπόδια όταν δεν υπήρχαν αρκετά χαρακτηριστικά για τον υπολογισμό της οπτικής ροής.

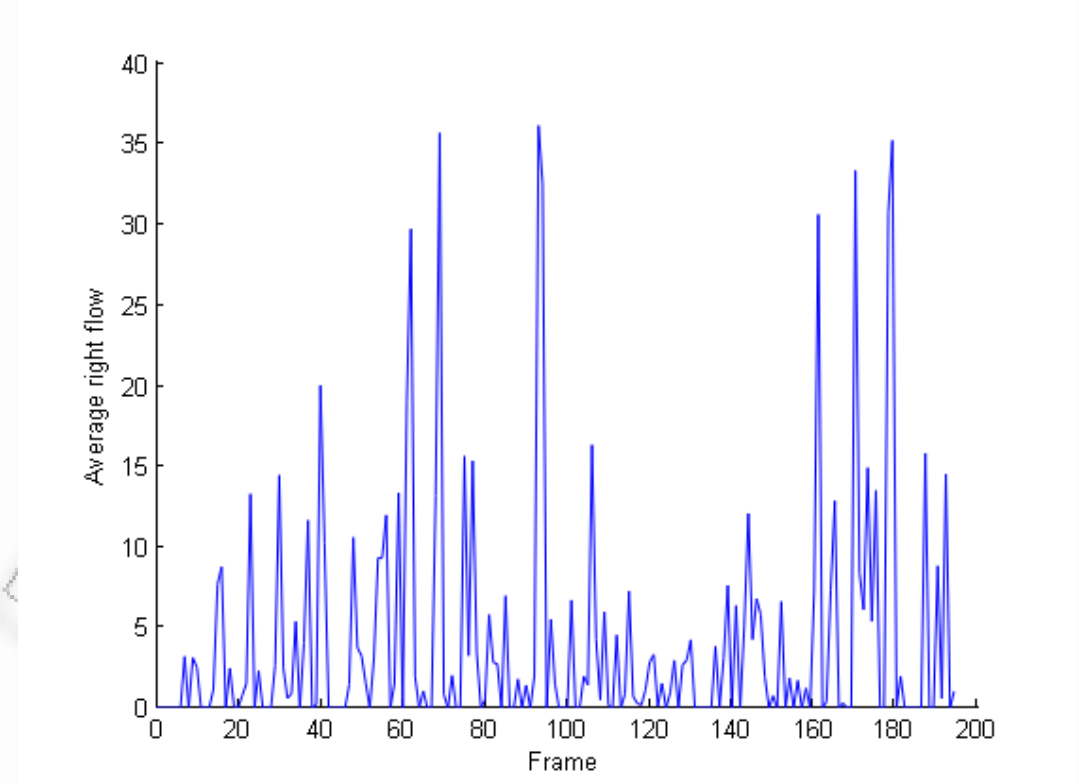
Εικόνα 4-21: Το μονοπάτι που ακολούθησε το ρομπότ για να πλοηγηθεί στον χώρο



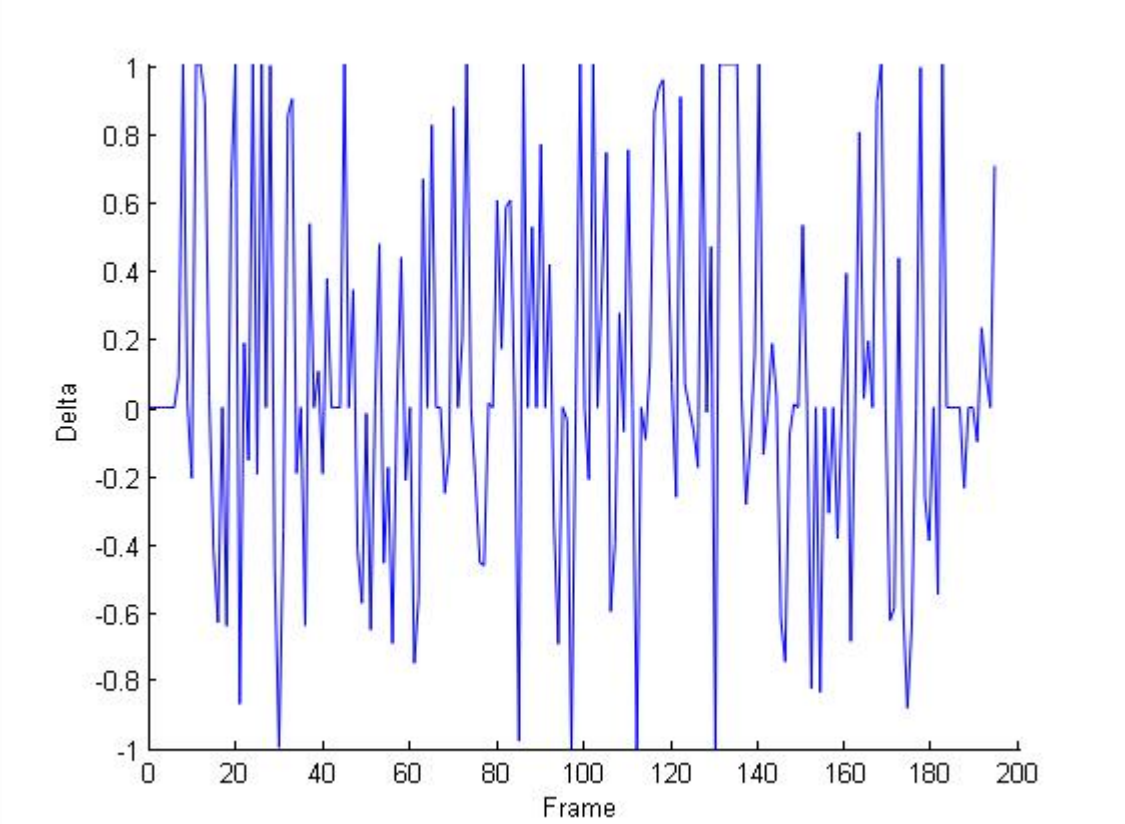
Εικόνα 4-22: Μια από τις εικόνες που βλέπει και επεξεργάζεται το ρομπότ



Εικόνα 4-23: Μέση τιμή της αριστερής οπτικής ροής για κάθε εικόνα



Εικόνα 4-24: Μέση τιμή της δεξιάς οπτικής ροής για κάθε εικόνα



Εικόνα 4-25: Η τιμή της μεταβλητής δέλτα για κάθε εικόνα

5 Συμπεράσματα - Περίληψη

Κύριος στόχος της εργασίας αυτής ήταν η δημιουργία ενός απλού ρομπότ το οποίο θα μπορούσε να αποφύγει εμπόδια χρησιμοποιώντας κάποιον αλγόριθμο βασισμένο στην υπολογιστική όραση. Ο αλγόριθμος που χρησιμοποιήθηκε υπολογίζει την οπτική ροή για τις διαδοχικές εικόνες οι οποίες λαμβάνονται από την κάμερα του υπολογιστή. Το ρομπότ με την σειρά του ακολουθεί μια απλή στρατηγική αποφυγής των εμποδίων χρησιμοποιώντας τις τιμές της μέσης αριστερής και δεξιάς οπτικής ροής και της μεταβλητής δέλτα όπως αυτή ορίστηκε στο τρίτο κεφάλαιο. Με χαρά διαπιστώσαμε πως στις περισσότερες των περιπτώσεων (39 από τις 50 προσπάθειες, ποσοστό επιτυχίας 78%) το ρομπότ κατάφερε να φέρει εις πέρας την εκάστοτε αποστολή του.

Αναλυτικά οι υπό-εργασίες που πραγματοποιήθηκαν ήταν:

- Σύντομη βιβλιογραφική ανασκόπηση σχετικά με το θέμα της εργασίας.
- Κατασκευή ενός απλού LEGO ρομπότ.
- Δημιουργία μιας κλάσης για την επικοινωνία με τον υπολογιστή του LEGO.
- Σχεδιασμός ενός αλγορίθμου αποφυγής εμποδίων με βάση την εκτίμηση της οπτικής ροής
- Πειράματα για την αποδοτικότητα του συστήματος

5.1 Μελλοντικές εργασίες

Οι εργασίες που θα μπορούσαν να πραγματοποιηθούν με βάση την εργασία αυτή είναι αρκετές. Στις επόμενες γραμμές αναφέρονται ενδεικτικά κάποιες από αυτές.

- Σύγκριση του αλγόριθμου που χρησιμοποιήθηκε στην εργασία με άλλους αλγόριθμους. Σαν μέτρο σύγκρισης μπορούν να επιλεγούν ο χρόνος που χρειάζεται για τον υπολογισμό της οπτικής ροής καθώς και η επίδοση που θα έχει το ρομπότ στην αποφυγή η όχι των εμποδίων.
- Χρήση ενός συστήματος στερεοσκοπικής όρασης ώστε να δοθεί η δυνατότητα για ανακατασκευή του τρισδιάστατου χώρου και ανίχνευση των εμποδίων που βρίσκονται στην πορεία του. Με τον τρόπο αυτό θα μπορεί να γίνει σύγκριση των δύο συστημάτων τόσο σε υπολογιστικό χρόνο όσο και στην ακρίβεια των αποτελεσμάτων.
- Χρήση κάποιων άλλων αισθητήρων όπως σόναρ το οποίο έχει χαμηλή ποιότητα αλλά μπορεί να καλύψει μεγάλη επιφάνεια ή αισθητήρα υπέρυθρων ο οποίος έχει καλύτερη ποιότητα αλλά μπορεί να καλύψει μεγαλύτερη περιοχή. Φυσικά και πάλι θα μπορεί να γίνει κάποια σύγκριση μεταξύ των δύο συστημάτων.
- Οι επιδώσεις του ρομπότ μετρήθηκαν σε συνθήκες όπου υπήρχε επαρκής φωτισμός. Ίσως με την χρήση κάποιας ειδικής κάμερα που μπορεί να λειτουργεί στο σκοτάδι να μπορεί ο ίδιος αλγόριθμος να χρησιμοποιηθεί για τον σκοπό αυτό.
- Ο υπολογιστής που χρησιμοποιήθηκε διαθέτει κάμερα με μικρή γωνία όρασης (narrow angle lens). Με την χρήση κάποιας κάμερας μεγαλύτερης γωνίας όρασης (wide angle lens) ίσως τα αποτελέσματα του συστήματος να είναι διαφορετικά. Και σε αυτή την περίπτωση μπορεί να γίνει σύγκριση ως προς την απόδοση των δύο προσεγγίσεων.

Παράρτημα Α: Τυπική ακολουθία εικόνων

Στο παράρτημα αυτό παρουσιάζεται ένα παράδειγμα αποφυγής εμποδίων από το ρομπότ. Παρατίθενται οι εικόνες από το βίντεο και οι αντίστοιχες εικόνες που βλέπει το ρομπότ.

Παράρτημα Β: Πηγαίος Κώδικας

Στο παράρτημα αυτό παρουσιάζεται ο κώδικας που χρησιμοποιήθηκε για την εφαρμογή με σχόλια στα σημαντικότερα κομμάτια του.

Nxt.h

Ο κώδικας αυτός χρησιμεύει στην επικοινωνία του ρομπότ με τον υπολογιστή.

```
namespace Navigator {
using namespace System;
using namespace System::Windows::Forms;
using namespace System::IO::Ports;
using namespace System::Text;
public ref class Nxt
{
private:
// Τα μηνύματα λάθους που μπορεί να εμφανιστούν στην επικοινωνία με το ρομπότ
enum class ErrorMessage : Byte
{
PendingCommunicationInProgress = 0x20,
SpecifiedMailBoxEmpty = 0x40,
RequestFailed = 0xBD,
UnknownCommandOpCode = 0xBE,
InsanePacket = 0xBF,
DataContainsOutOfRangeValues = 0xC0,
CommunicationBusError = 0xDD,
NoFreeMemoryInCommunicationBuffer = 0xDE,
SpecifiedChannelConnectionIsNotValid = 0xDF,
SpecifiedChannelConnectionNotConfiguredOrBusy = 0xE0,
NoActiveProgram = 0xEC,
IllegalSizeSpecified = 0xED,
IllegalMailboxQueueIDSpecified = 0xEE,
AttemptedToAccessInvalidFieldOfAStructure = 0xEF,
BadInputOrOutputSpecified = 0xF0,
InsufficientMemoryAvailable = 0xFB,
BadArguments = 0xFF
};
// Το είδος της εντολής που μπορεί να πάρει το ρομπότ
enum class Command : Byte
{
StartProgram = 0x00,
StopProgram = 0x01,
PlaySoundFile = 0x02,
PlayTone = 0x03,
SetOutputState = 0x04,
SetInputMode = 0x05,
GetOutputState = 0x06,
GetInputValues = 0x07,
MessageWrite = 0x09,
ResetMotorPosition = 0x0A,
GetBatteryLevel = 0x0B,
KeepAlive = 0x0D,
LSGetStatus = 0x0E,
LSWrite = 0x0F,
LSRead = 0x10,
MessageRead = 0x13,
OpenRead = 0x80,
OpenWrite = 0x81,
Read = 0x82,
Write = 0x83,
GetFirmwareVersion = 0x88,
SetBrickName = 0x98,
GetDeviceInfo = 0x9B
};
// Ο αριθμός του σερβοκινητήρα στον οποίο στελνεται η εντολή
enum class MotorPort : Byte
{
```

```
PortA = 0x00,
PortB = 0x01,
PortC = 0x02,
None = 0xFE,
All = 0xFF
};
// Η σειριακή πόρτα μέσω της οποίας γίνεται η επικοινωνία
SerialPort ^port;
public:
// Το είδος της ρύθμισης που έχει ο σερβοκινητήρας
[Flags()]
enum class MotorMode : Byte
{
None = 0x00,
MotorOn = 0x01,
Brake = 0x02,
Regulated = 0x04
};
// Επιπλέον ρύθμιση για τον σερβοκινητήρα
[Flags]
enum class MotorRegulationMode : Byte
{
Idle = 0x00,
MotorSpeed = 0x01,
MotorSynchronization = 0x02
};
// Η κατάσταση στην οποία βρίσκεται ο σερβοκινητήρας
[Flags]
enum class MotorRunState : Byte
{
Idle = 0x00,
RampUp = 0x10,
Running = 0x20,
Rampdown = 0x40
};
// Η εσωτερική κλάση που κρατάει τις πληροφορίες για τον σερβοκινητήρα
value class GetOutputState
{
public:
SByte Power;
MotorMode ^Mode;
MotorRegulationMode ^RegulationMode;
SByte TurnRatio;
MotorRunState RunState;
UInt32 TachoLimit;
int TachoCount;
int BlockTachoCount;
int RotationCount;
};
// Η εσωτερική κλάση που κρατάει την έκδοση του firmware του ρομπότ
value class FirmwareVersion
{
public:
Int16 MinorVersionOfProtocol;
Int16 MajorVersionOfProtocol;
Int16 MinorVersionOfFirmware;
Int16 MajorVersionOfFirmware;
};
// Η εσωτερική κλάση που κρατάει τις πληροφορίες για την έκδοση του ρομπότ
value class DeviceInfo
{
public:
String ^NxtName;
array<Byte> ^btAdress;
UInt32 bluetoothSignalStrength;
UInt32 freeUserFlash;
};
// Δείχνει εαν το ρομπότ είναι η όχι συνδεδεμένο μέσω της συριακής θύρας
bool isConnected;
private:
```

```
// Η συνάρτηση αυτή είναι υπεύθυνη για την ανταλλαγή μηνυμάτων μεταξύ του ρομπότ και του
υπολογιστή
array<Byte> ^sendMessage(array<Byte> ^message)
{
if(!isConnected)
{
throw gcnew InvalidOperationException(L"Η συσκευή δεν είναι συνδεδεμένη.");
}
else
{
System::Threading::Monitor::Enter(this);
int length = message->Length;
array<Byte> ^btMessage = gcnew array<Byte>(message->Length + 2);
btMessage[0] = Convert::ToByte(length & 0xFF);
btMessage[1] = Convert::ToByte((length & 0xFF00) >> 8);
message->CopyTo(btMessage, 2);
port->Write(btMessage, 0, btMessage->Length);
if(message[0] < 0x80)
{
int lsb = port->ReadByte();
int msb = port->ReadByte();
int size = lsb + msb * 256;
array<Byte> ^reply = gcnew array<Byte>(size);
port->Read(reply, 0, size);
if(reply[0] != 0x02)
{
throw gcnew Exception(L"Μη αναμενόμενο μήνυμα επιστροφής του τύπου " +
BitConverter::ToString(reply, 0, 1));
}
if(reply[1] != message[1])
{
throw gcnew Exception(L"Μη αναμενόμενη εντολή επιστροφής." +
BitConverter::ToString(reply, 1, 1));
}
if(reply[2] > 0)
{
ErrorMessage error = safe_cast<ErrorMessage>(reply[2]);
throw gcnew Exception(L"Σφάλμα: " + error.ToString() + L" στην εντολή " +
BitConverter::ToString(message));
}
return reply;
}
else
{
return nullptr;
}
System::Threading::Monitor::Exit(this);
}
}
// Η δομή αυτή επιστρέφει τις πληροφορίες για το firmware του ρομπότ
FirmwareVersion ^firmwareVersion()
{
FirmwareVersion ^result = gcnew FirmwareVersion();
array<Byte> ^message = gcnew array<Byte>(2);
message[0] = 0x01;
message[1] = Convert::ToByte(Command::GetFirmwareVersion);
array<Byte> ^reply = sendMessage(message);
result->MinorVersionOfProtocol = Convert::ToInt16(reply[3]);
result->MajorVersionOfProtocol = Convert::ToInt16(reply[4]);
result->MinorVersionOfFirmware = Convert::ToInt16(reply[5]);
result->MajorVersionOfFirmware = Convert::ToInt16(reply[6]);
return result;
}
// Η δομή αυτή επιστρέφει τις πληροφορίες για την έκδοση του ρομπότ
DeviceInfo ^getDeviceInfo()
{
DeviceInfo ^result = gcnew DeviceInfo();
array<Byte> ^message = gcnew array<Byte>(2);
message[0] = 0x01;
message[1] = Convert::ToByte(Command::GetDeviceInfo);
array<Byte> ^reply = sendMessage(message);
```

```
result->NxtName = Encoding::ASCII->GetString(reply, 3, 14)->TrimEnd('\0');
result->btAdress = gcnew array<Byte>(7);
Array::Copy(reply, 18, result->btAdress, 0, 7);
result->bluetoothSignalStrength = BitConverter::ToUInt32(reply, 25);
result->freeUserFlash = BitConverter::ToUInt32(reply, 29);
return result;
}
// Η δομή αυτή χρησιμεύει για την μεταφορά εντολών στους σερβοκινητήρες
System::Void setOutputState(MotorPort ^port, SByte power, MotorMode ^mode,
MotorRegulationMode ^regulationMode, SByte turnRatio, MotorRunState ^runState, UInt32
tachoLimit)
{
array<Byte> ^message = gcnew array<Byte>(12);
message[0] = 0x80;
message[1] = Convert::ToByte(Command::SetOutputState);
message[2] = Convert::ToByte(port);
message[3] = safe_cast<Byte>(power);
message[4] = Convert::ToByte(mode);
message[5] = Convert::ToByte(regulationMode);
message[6] = safe_cast<Byte>(turnRatio);
message[7] = Convert::ToByte(runState);
BitConverter::GetBytes(tachoLimit)->CopyTo(message, 8);
sendMessage(message);
}
// Η συνάρτηση αυτή δίνει την εντολή για να κινηθεί ο σερβοκινητήρας
System::Void run(MotorPort ^port, SByte power, UInt32 tachoLimit)
{
setOutputState(
port,
power,
MotorMode::MotorOn | MotorMode::Regulated,
MotorRegulationMode::MotorSpeed,
0,
MotorRunState::Running,
tachoLimit);
}
// Η συνάρτηση αυτή δίνει την εντολή για να κινηθεί ο σερβοκινητήρας
System::Void coast(MotorPort ^port)
{
setOutputState(
port,
0,
MotorMode::MotorOn | MotorMode::Regulated,
MotorRegulationMode::Idle,
0,
MotorRunState::Running,
0);
}
// Η συνάρτηση αυτή δίνει την εντολή για να σταματήσει ο σερβοκινητήρας
System::Void brake(MotorPort ^port)
{
setOutputState(
port,
0,
MotorMode::MotorOn | MotorMode::Regulated | MotorMode::Brake,
MotorRegulationMode::Idle,
0,
MotorRunState::Running,
0
);
}
// Η συνάρτηση αυτή δίνει την εντολή για να περιμένει για εντολές ο σερβοκινητήρας
System::Void idle(MotorPort ^port)
{
setOutputState(
port,
0,
MotorMode::None,
MotorRegulationMode::Idle,
0,
MotorRunState::Idle,
```

```
0);
}
// Η δομή αυτή επιστρέφει την κατάσταση του σερβοκινητήρα
GetOutputState ^getOutputState(MotorPort ^port)
{
array<Byte> ^message = gcnew array<Byte>(3);
message[0] = 0x00;
message[1] = Convert::ToByte(Command::GetOutputState);
message[2] = Convert::ToByte(port);
array<Byte> ^reply = sendMessage(message);
GetOutputState ^result = gcnew GetOutputState();
result->Power = Convert::ToByte(reply[4]);
result->Mode = safe_cast<MotorMode>(reply[5]);
result->RegulationMode = safe_cast<MotorRegulationMode>(reply[6]);
result->TurnRatio = Convert::ToSByte(reply[7]);
result->RunState = safe_cast<MotorRunState>(reply[8]);
result->TachoLimit = BitConverter::ToUInt32(reply, 9);
result->TachoCount = BitConverter::ToInt32(reply, 13);
result->BlockTachoCount = BitConverter::ToInt32(reply, 17);
result->RotationCount = BitConverter::ToInt32(reply, 21);
return result;
}
// Η δομή αυτή επαναφέρει στην αρχική του κατάσταση τον σερβοκινητήρα
System::Void resetMotorPosition(MotorPort ^port, bool relative)
{
array<Byte> ^message = gcnew array<Byte>(4);
message[0] = 0x80;
message[1] = Convert::ToByte(Command::ResetMotorPosition);
message[2] = Convert::ToByte(port);
if (relative)
message[3] = 1;
else
message[3] = 0;
sendMessage(message);
}
// Η συνάρτηση αυτή δίνει την εντολή να μετακινηθούν ταυτόχρονα οι δύο πρώτοι
σερβοκινητήρες
System::Void moveBoth(int power, UInt32 tachoLimit)
{
setOutputState(MotorPort::PortA, Convert::ToSByte(power), MotorMode::Brake |
MotorMode::MotorOn | MotorMode::Regulated, MotorRegulationMode::MotorSynchronization, 0,
MotorRunState::Running, tachoLimit);
setOutputState(MotorPort::PortB, Convert::ToSByte(power), MotorMode::Brake |
MotorMode::MotorOn | MotorMode::Regulated, MotorRegulationMode::MotorSynchronization, 0,
MotorRunState::Running, tachoLimit);
}
// Η συνάρτηση αυτή δίνει την εντολή να γυρίσουν ταυτόχρονα οι δύο πρώτοι σερβοκινητήρες
System::Void turn(int power, UInt32 tachoLimit, int turnRate)
{
setOutputState(MotorPort::PortA, Convert::ToSByte(power), MotorMode::Brake |
MotorMode::MotorOn | MotorMode::Regulated, MotorRegulationMode::MotorSynchronization,
Convert::ToSByte(turnRate), MotorRunState::Running, tachoLimit);
setOutputState(MotorPort::PortB, Convert::ToSByte(power), MotorMode::Brake |
MotorMode::MotorOn | MotorMode::Regulated, MotorRegulationMode::MotorSynchronization,
Convert::ToSByte(turnRate), MotorRunState::Running, tachoLimit);
}
// Η συνάρτηση αυτή επιστρέφει την κατάσταση ενός σερβοκινητήρα
String ^getOutput(MotorPort ^port)
{
GetOutputState ^reply = getOutputState(port);
return reply->RunState.ToString();
}
public:
// Η συνάρτηση αυτή επιστρέφει το firmware του ρομπότ
System::String ^getFirmware()
{
FirmwareVersion ^result = gcnew FirmwareVersion();
result = firmwareVersion();
return String::Format("{0:00}", result->MajorVersionOfFirmware) + "." +
String::Format("{0:00}", result->MinorVersionOfFirmware);
}
```

```
// Η συνάρτηση αυτή επιστρέφει την έκδοση του ρομπότ
System::String ^getProtocol()
{
FirmwareVersion ^result = gcnew FirmwareVersion();
result = firmwareVersion();
return String::Format("{0:00}", result->MajorVersionOfProtocol) + "." +
String::Format("{0:00}", result->MinorVersionOfProtocol);
}
// Η συνάρτηση αυτή επιστρέφει το όνομα του ρομπότ
System::String ^getName()
{
DeviceInfo ^result = gcnew DeviceInfo();
result = getDeviceInfo();
return result->NxtName;
}
// Η συνάρτηση αυτή θέτει το όνομα του ρομπότ
void setName(String ^name)
{
if(name->Length > 14)
{
name = name->Substring(0, 14);
}
array<Byte> ^nameBytes = System::Text::Encoding::ASCII->GetBytes(name);
array<Byte> ^message = gcnew array<Byte>(18);
message[0] = 0x01;
message[1] = Convert::ToByte(Command::SetBrickName);
nameBytes->CopyTo(message, 2);
sendMessage(message);
}
// Η συνάρτηση αυτή επιστρέφει την ισχύ του σήματος Bluetooth όπως λαμβάνεται από το
ρομπότ
System::UInt32 getBluetoothStrength()
{
DeviceInfo ^result = gcnew DeviceInfo();
result = getDeviceInfo();
return result->bluetoothSignalStrength;
}
// Η συνάρτηση αυτή επιστρέφει το επίπεδο της μπαταρίας του ρομπότ
System::UInt16 getBatteryLevel()
{
array<Byte> ^message = gcnew array<Byte>(2);
message[0] = 0x00;
message[1] = Convert::ToByte(Command::GetBatteryLevel);
array<Byte> ^reply = sendMessage(message);
return BitConverter::ToUInt16(reply, 3);
}
// Η συνάρτηση αυτή χρησιμεύει στο να μην κλείσει η επικοινωνία με το ρομπότ λόγω μη
χρήσης εντολών
System::Void keepAlive()
{
array<Byte> ^message = gcnew array<Byte>(2);
message[0] = 0x80;
message[1] = Convert::ToByte(Command::KeepAlive);
sendMessage(message);
}
// Η συνάρτηση αυτή δίνει εντολή στο ρομπότ να προχωρήσει προς τα εμπρός
System::Void moveForward(int power)
{
resetMotorPosition(MotorPort::PortA, true);
resetMotorPosition(MotorPort::PortB, true);
moveBoth(Math::Abs(power), 0);
}
// Η συνάρτηση αυτή δίνει εντολή στο ρομπότ να προσωρήσει προς τα πίσω
System::Void moveBack(int power)
{
resetMotorPosition(MotorPort::PortA, true);
resetMotorPosition(MotorPort::PortB, true);
moveBoth(-Math::Abs(power), 0);
}
// Η συνάρτηση αυτή δίνει εντολή στο ρομπότ να σταματήσει
System::Void stop()
```

```
{
brake(MotorPort::PortA);
brake(MotorPort::PortB);
}
// Η συνάρτηση αυτή δίνει εντολή στο ρομπότ να στρίψει αριστερά
System::Void turnLeft(int power)
{
resetMotorPosition(MotorPort::PortA, true);
resetMotorPosition(MotorPort::PortB, true);
turn(power, 0, -100);
}
// Η συνάρτηση αυτή δίνει εντολή στο ρομπότ να στρίψει δεξιά
System::Void turnRight(int power)
{
resetMotorPosition(MotorPort::PortA, true);
resetMotorPosition(MotorPort::PortB, true);
turn(power, 0, 100);
}
// Η συνάρτηση αυτή συνδέει το ρομπότ μέσω της σειριακής θύρας
System::Void Connect(String ^portName)
{
port = gcnew SerialPort(portName);
port->Open();
isConnected = true;
}
// Η συνάρτηση αυτή διακόπτει την επικοινωνία μέσω της σειριακής θύρας
System::Void Disconnect()
{
isConnected = false;
if (port->IsOpen)
{
port->Close();
}
}
Nxt()
{
isConnected = false;
}
~Nxt()
{
}
};
}
```

mainForm.h (testNXT)

Ο κώδικας αυτός είναι υπεύθυνος για την μεταφορά βασικών εντολών επικοινωνίας με το ρομπότ.

```
#include "Nxt.h"
namespace testNXT {
using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;
using namespace System::IO::Ports;
// Στο σημείο αυτό δεν αναφέρεται ο κώδικας δημιουργίας της φόρμας
// Η κλάση που θα χρησιμοποιηθεί για την επικοινωνία με τον υπολογιστή
Nxt ^nxt;
// Η συνάρτηση αυτή καλείται όταν ανοίγει η φόρμα, δημιουργεί την κλάση επικοινωνίας
// βρίσκει την λίστα με τις σειριακές θύρες που έχει ο υπολόγιστής
private:
System::Void mainForm_Load(System::Object^  sender, System::EventArgs^  e)
{
nxt = gcnew Nxt();
array<String^> ^serialPorts = SerialPort::GetPortNames();
for (int i = 0; i < serialPorts->Length; i++)
{
if (serialPorts[i]->Length > 4)
```

```
serialPorts[i] = serialPorts[i]->Substring(0,4);
comComboBox->Items->Add(serialPorts[i]);
}
if(comComboBox->Items->Count != 0)
comComboBox->SelectedIndex = 0;
}
// Η συνάρτηση αυτή χρησιμοποιείται για την σύνδεση / αποσύνδεση με το ρομπότ
// Όταν γίνεται η σύνδεση παίρνει τις βασικές πληροφορίες από το ρομπότ
private:
System::Void connectButton_Click(System::Object^  sender, System::EventArgs^  e)
{
try
{
if(nxt->isConnected)
{
statusRichTextBox->AppendText(L"Τερματισμός Σύνδεσης");
nxt->Disconnect();
connectButton->Text=L"Σύνδεση";
controlTableLayoutPanel->Enabled = false;
}
else
{
nxt->Connect(comComboBox->SelectedItem->ToString());
statusRichTextBox->Clear();
statusRichTextBox->AppendText(L"Όνομα: " + nxt->getName() + "\n");
statusRichTextBox->AppendText(L"Έκδοση firmware: " + nxt->getFirmware() + "\n");
statusRichTextBox->AppendText(L"Έκδοση πρωτοκόλλου: " + nxt->getProtocol() + "\n");
statusRichTextBox->AppendText(L"Έκδοση bluetooth: " + nxt-
>getBluetoothStrength().ToString() + "\n");
statusRichTextBox->AppendText(L"Επίπεδο μπαταρίας: " + nxt->getBatteryLevel().ToString()
+ "\n");
connectButton->Text=L"Αποσύνδεση";
controlTableLayoutPanel->Enabled = true;
}
}
catch (System::Exception^ ex)
{
MessageBox::Show(L"Σφάλμα: " + ex->Message, L"Σφάλμα Σύνδεσης", MessageBoxButtons::OK,
MessageBoxIcon::Error);
}
}
// Η συνάρτηση αυτή δίνει εντολή στο ρομπότ να πάει μπροστά
private:
System::Void upButton_Click(System::Object^  sender, System::EventArgs^  e)
{
try
{
nxt->moveForward(Convert::ToInt32(speedNumericUpDown->Value));
}
catch (System::Exception^ ex)
{
MessageBox::Show(L"Σφάλμα: " + ex->Message, L"Σφάλμα", MessageBoxButtons::OK,
MessageBoxIcon::Error);
}
}
// Η συνάρτηση αυτή δίνει εντολή στο ρομπότ να οπισθοχωρήσει
private:
System::Void downButton_Click(System::Object^  sender, System::EventArgs^  e)
{
try
{
nxt->moveBack(Convert::ToInt32(speedNumericUpDown->Value));
}
catch (System::Exception^ ex)
{
MessageBox::Show(L"Σφάλμα: " + ex->Message, L"Σφάλμα", MessageBoxButtons::OK,
MessageBoxIcon::Error);
}
}
// Η συνάρτηση αυτή δίνει εντολή στο ρομπότ να στρίψει προς τα αριστερά
private:
```

```
System::Void leftButton_Click(System::Object^  sender, System::EventArgs^  e)
{
try
{
nxt->turnLeft(Convert::ToInt32(speedNumericUpDown->Value));
System::Threading::Thread::Sleep(900);
nxt->stop();
}
catch (System::Exception^ ex)
{
MessageBox::Show(L"Σφάλμα: " + ex->Message, L"Σφάλμα", MessageBoxButtons::OK,
MessageBoxIcon::Error);
}
}
// Η συνάρτηση αυτή δίνει εντολή στο ρομπότ να στρίψει προς τα δεξιά
private:
System::Void rightButton_Click(System::Object^  sender, System::EventArgs^  e)
{
try
{
nxt->turnRight(Convert::ToInt32(speedNumericUpDown->Value));
}
catch (System::Exception^ ex)
{
MessageBox::Show(L"Σφάλμα: " + ex->Message, L"Σφάλμα", MessageBoxButtons::OK,
MessageBoxIcon::Error);
}
}
// Η συνάρτηση αυτή δίνει εντολή στο ρομπότ να σταματήσει
private:
System::Void stopButton_Click(System::Object^  sender, System::EventArgs^  e)
{
try
{
nxt->stop();
}
catch (System::Exception^ ex)
{
MessageBox::Show(L"Σφάλμα: " + ex->Message, L"Σφάλμα", MessageBoxButtons::OK,
MessageBoxIcon::Error);
}
}
// Η συνάρτηση αυτή καλείται όταν κλείνει η φόρμα
private:
System::Void mainForm_FormClosing(System::Object^  sender,
System::Windows::Forms::FormClosingEventArgs^  e)
{
try
{
if(nxt->isConnected)
{
nxt->Disconnect();
}
}
catch (System::Exception^ ex)
{
MessageBox::Show(L"Σφάλμα: " + ex->Message, L"Σφάλμα Σύνδεσης", MessageBoxButtons::OK,
MessageBoxIcon::Error);
}
}
};
}
```

ImageUtility.h

Ο κώδικας αυτός είναι υπεύθυνος για την διαχείριση των εικόνων.

```
#pragma once
namespace Navigator
{
```

```
using namespace System;
using namespace System::ComponentModel;
using namespace System::Windows::Forms;
using namespace System::Drawing;
public ref class ImageUtility
{
public:
// Η μεταβλητή αυτή αντιπροσωπεύει την κλίμακα της εικόνας
float scale;
// Η συνάρτηση αυτή χρησιμοποιείται για την επικοινωνία μεταξύ διαφορετικών νημάτων
delegate void showImageDelegate(PictureBox ^pictureBox, Bitmap ^bitmap, bool saveImage,
int imageNumber, int folderNumber);
// Η συνάρτηση αυτή δείχνει την εικόνα στο PictureBox
System::Void showImage(PictureBox ^pictureBox, Bitmap ^bitmap, bool saveImage, int
imageNumber, int folderNumber)
{
pictureBox->Image = bitmap;
if(saveImage)
pictureBox->Image->Save(Application::StartupPath + "\\Data\\" + folderNumber.ToString()
+ "\\Images\\"+imageNumber+".jpg",System::Drawing::Imaging::ImageFormat::Jpeg);
}
// Η συνάρτηση αυτή μετατρέπει την εικόνα από IplImage σε Bitmap
Bitmap ^iplImageToBitmap(IplImage *image)
{
Bitmap ^bitmap = gcnew Bitmap( image->width, image->height,
System::Drawing::Imaging::PixelFormat::Format24bppRgb );
System::Drawing::Imaging::BitmapData ^data = bitmap-
>LockBits(System::Drawing::Rectangle(0,0,bitmap->Width,bitmap-
>Height),System::Drawing::Imaging::ImageLockMode::ReadWrite, bitmap->PixelFormat );
memcpy( data->Scan0.ToPointer(), image->imageData, image->imageSize );
bitmap->UnlockBits( data );
return bitmap;
}
// Η συνάρτηση αυτή ζωγραφίζει μια γραμμή στην εικόνα
System::Void drawLine(Bitmap ^bitmap, Color color, PointF ^point1, PointF ^point2)
{
Graphics ^g = Graphics::FromImage(safe_cast<Image^>(bitmap));
g->SmoothingMode = System::Drawing::Drawing2D::SmoothingMode::AntiAlias;
g->DrawLine(gcnew Pen(color), point1->X * scale, point1->Y * scale, point2->X * scale,
point2->Y * scale);
}
// Η συνάρτηση αυτή γράφει κάποιο κείμενο στην εικόνα
System::Void drawText(Bitmap ^bitmap, Font ^font, String ^text, Color color, PointF
^location)
{
Graphics ^g = Graphics::FromImage(safe_cast<Image^>(bitmap));
g->TextRenderingHint = System::Drawing::Text::TextRenderingHint::AntiAlias;
g->DrawString(text, font, gcnew SolidBrush(color), scale * location->X, scale *
location->Y);
}
// Η συνάρτηση αυτή ζωγραφίζει έναν κύκλο στην εικόνα
System::Void drawCircle(Bitmap ^bitmap, Color color, PointF ^location, int size)
{
Graphics ^g = Graphics::FromImage(safe_cast<Image^>(bitmap));
g->SmoothingMode = System::Drawing::Drawing2D::SmoothingMode::AntiAlias;
g->FillEllipse(gcnew SolidBrush(color), (location->X - size /2) * scale, (location->Y -
size/2) * scale, size * scale, size * scale);
}
// Η συνάρτηση αυτή αλλάζει το μέγεθος της εικόνας
Bitmap ^getResizedBitmap (PictureBox ^pictureBox, IplImage *image)
{
Bitmap ^bitmap = iplImageToBitmap(image);
int widthDifference = pictureBox->Width - bitmap->Width;
int heightDifference = pictureBox->Height - bitmap->Height;
scale = (float)pictureBox->Width / bitmap->Width;
if (heightDifference < widthDifference)
scale = (float)pictureBox->Height / bitmap->Height;
float newWidth = bitmap->Width * scale;
float newHeight = bitmap->Height * scale;
Bitmap ^resizedBitmap = gcnew Bitmap((int)newWidth, (int)newHeight);
Graphics ^g = Graphics::FromImage(safe_cast<Image^>(resizedBitmap));
```

```
g->DrawImage(bitmap, 0.0f, 0.0f, newWidth, newHeight);
return resizedBitmap;
}
ImageUtility()
{
scale = 1.0f;
}
~ImageUtility()
{
}
protected:
!ImageUtility()
{
}
};
}
```

Settings.h

Ο κώδικας αυτός είναι υπεύθυνος για την διαχείριση των επιλογών του χρήστη.

```
#pragma once
namespace Navigator
{
using namespace System;
using namespace System::ComponentModel;
using namespace System::Windows::Forms;
using namespace System::Collections;
using namespace System::IO::Ports;
using namespace System::Drawing;
using namespace System::Xml::Serialization;
using namespace System::Drawing::Design;
public ref class Settings
{
private:
UInt32 cameraNumber;
Size resolution;
UInt32 maxPoints;
UInt32 robotSpeed;
bool saveData;
UInt32 folderNumber;
float flowThreshold;
public:
[DisplayName(L"Αριθμός κάμερας"), CategoryAttribute(L"Κάμερα"), DescriptionAttribute(L"Ο
αριθμός της κάμερας."), DefaultValueAttribute(0)]
property UInt32 CameraNumber
{
UInt32 get()
{
return cameraNumber;
}
void set(UInt32 value)
{
cameraNumber = value;
}
}
[DisplayName(L"Ανάλυση"), CategoryAttribute(L"Κάμερα"), DescriptionAttribute(L"Η ανάλυση
της εικόνας. Μπορεί να είναι 320 επί 240 ή 640 επί 480."),
DefaultValueAttribute(Size::typeid, "320,240")]
property Size Resolution
{
Size get()
{
return resolution;
}
void set(Size value)
{
if ( (value.Width != 320 && value.Width != 640) ||
(value.Height != 240 && value.Height != 480) )
```

```
throw gcnew Exception(L"Η ανάλυση της εικόνας μπορεί να είναι 320 επί 240 ή 640 επί
480.");
else
resolution = value;
}
}
[DisplayName(L"Μέγιστος Αριθμός σημείων"), CategoryAttribute(L"Όραση"),
DescriptionAttribute(L"Ο μέγιστος αριθμός σημείων για τον αλγόριθμο Shi and Tomasi."),
DefaultValueAttribute(100)]
property UInt32 MaxPoints
{
UInt32 get()
{
return maxPoints;
}
void set(UInt32 value)
{
maxPoints = value;
}
}
[DisplayName(L"Ταχύτητα ρομπότ"), CategoryAttribute(L"Ρομπότ"), DescriptionAttribute(L"Η
ταχύτητα του ρομπότ. Μπορεί να είναι από 1 εώς 100."), DefaultValueAttribute(75)]
property UInt32 RobotSpeed
{
UInt32 get()
{
return robotSpeed;
}
void set(UInt32 value)
{
if(value <1 || value > 100)
throw gcnew Exception(L"Η ταχύτητα θα πρέπει να είναι μεταξύ 1 και 100.");
else
robotSpeed = value;
}
}
[DisplayName(L"Αποθήκευση στοιχείων"), CategoryAttribute(L"Αρχεία"),
DescriptionAttribute(L"Αποθήκευση ή όχι των εικόνων της κάμερας και των στατιστικών της
οπτικής ροής."), DefaultValueAttribute(false)]
property bool SaveData
{
bool get()
{
return saveData;
}
void set(bool value)
{
saveData = value;
}
}
[DisplayName(L"Αριθμός φακέλου"), CategoryAttribute(L"Αρχεία"), DescriptionAttribute(L"Ο
αριθμός του φακέλου που θα αποθηκευτούν τα αρχεία."), DefaultValueAttribute(0)]
property UInt32 FolderNumber
{
UInt32 get()
{
return folderNumber;
}
void set(UInt32 value)
{
folderNumber = value;
}
}
[DisplayName(L"Όριο ροής"), CategoryAttribute(L"Όραση"), DescriptionAttribute(L"Το όριο
της οπτικής ροής βάσει του οποίου θα στρίψει το ρομπότ."), DefaultValueAttribute(10)]
property float FlowThreshold
{
float get()
{
return flowThreshold;
}
```

```
void set(float value)
{
flowThreshold = value;
}
}
Settings()
{
cameraNumber = 0;
resolution = Size(320,240);
maxPoints = 100;
robotSpeed = 75;
saveData = false;
folderNumber = 0;
flowThreshold = 10;
}
~Settings()
{
}
protected:
!Settings()
{
}
};
}
```

XmlUtility.h

Ο κώδικας αυτός είναι υπεύθυνος για την διαχείριση των αρχείων xml.

```
#pragma once
namespace Navigator
{
using namespace System;
using namespace System::ComponentModel;
using namespace System::Windows::Forms;
using namespace System::Xml;
public ref class XmlUtility
{
private:
String ^settingsFileName;
public:
// Φόρτωση ρύθμισης από το xml αρχείο
System::String ^loadSetting(System::String^ settingName)
{
// Φόρτωσε το xml αρχείο και ψάξε την ρύθμιση με το συγκεκριμένο όνομα
XmlDocument^ docXML = gcnew XmlDocument();
docXML->Load(settingsFileName);
XmlNodeList^ settingsList = docXML->GetElementsByTagName("Setting");
for (int i = 0; i < settingsList->Count; i++)
{
if (settingsList[i]->Attributes["Name"]->InnerText == settingName)
{
// Επέστρεψε την τιμή της ρύθμισης
return settingsList[i]->Attributes["Value"]->Value;
}
}
throw gcnew System::Exception(L"Αδυναμία φόρτωσης της ρύθμισης: "+settingName);
}

// Αποθήκευση ρύθμισης στο xml αρχείο
System::Void saveSetting(System::String^ settingName, System::String^ settingValue)
{
// Φόρτωσε το xml αρχείο και ψάξε την ρύθμιση με το συγκεκριμένο όνομα
XmlDocument^ docXML = gcnew XmlDocument();
docXML->Load(settingsFileName);
XmlNodeList^ settingsList = docXML->GetElementsByTagName("Setting");
for (int i = 0; i < settingsList->Count; i++)
{
if (settingsList[i]->Attributes["Name"]->InnerText == settingName)
{
```

```
// Αποθήκευσε την καινούργια τιμή της ρύθμισης
settingsList[i]->Attributes["Value"]->Value = settingValue;
docXML->Save(settingsFileName);
return;
}
}
throw gcnew System::Exception(L"Αδυναμία αποθήκευσης της ρύθμισης: "+settingName);
}
XmlUtility(String ^settingsFileName)
{
this->settingsFileName = settingsFileName;
}
~XmlUtility()
{
}
protected:
!XmlUtility()
{
}
};
}
```

mainForm.h (Navigator)

Ο κώδικας αυτός είναι υπεύθυνος για την λειτουργία του κύριου προγράμματος.

```
#pragma once
#pragma unmanaged
#include <windows.h>
#include "cv.h"
#include "highgui.h"
#pragma managed
#include "Utilities/ImageUtility.h"
#include "Utilities/Settings.h"
#include "Utilities/XmlUtility.h"
#include "Utilities/Nxt.h"
namespace Navigator {
using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;
using namespace System::Threading;
using namespace System::IO;
// Στο σημείο αυτό δεν αναφέρεται ο κώδικας δημιουργίας της φόρμας
private:
// Η κλαση που θα χρησιμοποιηθεί για την διαχείριση των εικόνων
ImageUtility ^imageUtility;
// Η κλαση που θα χρησιμοποιηθεί για την διαχείριση των ρυθμίσεων
Settings ^settings;
// Η κλάση που θα χρησιμοποιηθεί για την αποθήκευση των αρχείων xml
XmlUtility ^xmlUtility;
// Η κλάση που θα χρησιμοποιηθεί για την επικοινωνία με το ρομπότ
Nxt ^nxt;
// Το νήμα που διαχειρίζεται την επεξεργασία των εικόνων
Thread ^flowThread;
// Το νήμα που διαχειρίζεται την επικοινωνία με το ρομπότ
Thread ^robotThread;
// Η μεταβλητή δέλτα
float delta;
// Η μέση οπτική ροή για το αριστερό και το δεξί μέρος της κάμερας
double leftAverage;
double rightAverage;
// Η μεταβλητή αυτή χρησιμεύει στην μέτρηση της διάρκειας μιας εντολής στο ρομπότ
int sleepTime;
// Η μεταβλητή αυτή ελέγχει εαν τρέχουν τα νήματα
static volatile bool isRunning = false;
// Οι συναρτήσεις αυτές χρησιμεύουν για την επικοινωνία μεταξύ των νημάτων
delegate void enableDelegate();
```

```
delegate void enableStartButtonDelegate();
#pragma region load save settings
private:
// Η συνάρτηση αύτή καλείται όταν ανοίγει η φόρμα και αρχικοποιεί τις κλάσεις, βρίσκει
τις διαθέσιμες σειριακές θύρες και φορτώνει τις επιλογές του χρήστη
System::Void mainForm_Load(System::Object^  sender, System::EventArgs^  e)
{
imageUtility = gcnew ImageUtility();
settings = gcnew Settings();
xmlUtility = gcnew XmlUtility(Application::StartupPath + "\\Settings\\settings.xml");
nxt = gcnew Nxt();
array<String^> ^serialPorts = SerialPort::GetPortNames();
for (int i = 0; i < serialPorts->Length; i++)
{
if (serialPorts[i]->Length > 4)
serialPorts[i] = serialPorts[i]->Substring(0,4);
comComboBox->Items->Add(serialPorts[i]);
}
if(comComboBox->Items->Count != 0)
comComboBox->SelectedIndex = 0;
try
{
comComboBox->SelectedItem = xmlUtility->loadSetting("ComPort");
if (Convert::ToBoolean(xmlUtility->loadSetting("MaximizedForm")) == true)
this->WindowState = FormWindowState::Maximized;
loadSettings(Application::StartupPath + "\\Settings\\navigator.xml");
}
catch (System::Exception^ ex)
{
MessageBox::Show("Error:" + ex->Message, "Load Setting Error", MessageBoxButtons::OK,
MessageBoxIcon::Error);
}
}
private:
// Η συνάρτηση αυτή φορτώνει τις επιλογές του χρήστη
System::Void loadSettings(String ^fileName)
{
XmlSerializer ^serializer = gcnew XmlSerializer(Settings::typeid);
XmlTextReader ^reader = gcnew XmlTextReader(fileName);
settings = safe_cast<Settings^>(serializer->Deserialize(reader));
reader->Close();
settingsPropertyGrid->SelectedObject = settings;
}
private:
// Η συνάρτηση αυτή αποθηκεύει τις επιλογές του χρήστη
System::Void saveSettings(String ^fileName)
{
XmlSerializer ^serializer = gcnew XmlSerializer(Settings::typeid);
XmlTextWriter ^writer = gcnew XmlTextWriter(fileName, gcnew
System::Text::UTF8Encoding());
serializer->Serialize(writer, settings);
writer->Close();
}
private:
// Η συνάρτηση αυτή αποθηκεύει το όνομα της σειριακής θύρας που επέλεξε ο χρήστης
System::Void comComboBox_SelectionChangeCommitted(System::Object^  sender,
System::EventArgs^  e)
{
xmlUtility->saveSetting("ComPort",comComboBox->SelectedItem->ToString());
}
private:
// Η συνάρτηση αυτή καλείται όταν κλείνει η φόρμα, κλείνει τα νήματα που λειτουργούν και
αποθηκεύει τις επιλογές του χρήστη
System::Void mainForm_FormClosing(System::Object^  sender,
System::Windows::Forms::FormClosingEventArgs^  e)
{
if (isRunning)
stop();
try
{
if (this->WindowState == FormWindowState::Maximized)
```

```
xmlUtility->saveSetting("MaximizedForm","True");
else
xmlUtility->saveSetting("MaximizedForm","False");
xmlUtility->saveSetting("ComPort",comComboBox->SelectedItem->ToString());
saveSettings(Application::StartupPath + "\\Settings\\navigator.xml");
}
catch (System::Exception^ ex)
{
MessageBox::Show("Error:" + ex->Message, "Save Setting Error", MessageBoxButtons::OK,
MessageBoxIcon::Error);
}
}
#pragma endregion
#pragma region enable Disable controls
private:
// Η συνάρτηση αυτή απενεργοποιεί τα controls της φόρμας
System::Void disable()
{
settingsPropertyGrid->Visible = false;
startButton->Enabled = false;
startButton->Text = L"Αποσύνδεση";
comLabel->Enabled=false;
comComboBox->Enabled=false;
}
private:
// Η συνάρτηση αυτή ενεργοποιεί τα controls της φόρμας
System::Void enable()
{
settingsPropertyGrid->Visible = true;
startButton->Enabled = true;
mainPictureBox->Image = nullptr;
startButton->Text = L"Σύνδεση";
comLabel->Enabled=true;
comComboBox->Enabled=true;
}
private:
// Η συνάρτηση αυτή ενεργοποιεί το κουμπί έναρξης
System::Void enableStartButton()
{
startButton->Enabled = true;
}
#pragma endregion
#pragma region optical flow
private:
// Η συναρτηση αυτή ενεργοποιεί τα δύο νήματα για την επικοινωνία με το ρομπότ και την
επεξεργασία των εικόνων
System::Void startButton_Click(System::Object^  sender, System::EventArgs^  e)
{
if (isRunning)
stop();
else
{
disable();
isRunning = true;
flowThread = gcnew Thread(gcnew ThreadStart(this,&Navigator::mainForm::opticalFlow));
flowThread->Start();
robotThread = gcnew Thread(gcnew
ParameterizedThreadStart(this,&Navigator::mainForm::robotControl));
array<String^> ^ parameters = gcnew array<String^> {comComboBox->SelectedItem-
>ToString()};
robotThread->Start(parameters);
}
}
private:
// Η συνάρτηση αυτή σταματάει τα δύο νήματα
System::Void stop()
{
isRunning = false;
}
private:
```

```
// Η συνάρτηση αυτή ελέγχει την κίνηση του ρομπότ ανάλογα με τις τιμές της μεταβλητής
δέλτα και της δεξιάς και αριστερής οπτικής ροής
System::Void robotControl(Object ^data)
{
array<String^> ^ parameters = (array<String^> ^)data;
String ^comPort = parameters[0];
try
{
nxt->Connect(comPort);
while (isRunning)
{
nxt->keepAlive();
sleepTime = (int)(-4 * settings->RobotSpeed + (1 + Math::Abs(delta)) * 400);
if(leftAverage > settings->FlowThreshold && rightAverage > settings->FlowThreshold)
{
nxt->turnRight(Convert::ToInt32(settings->RobotSpeed));
Thread::Sleep(sleepTime);
}
else if(delta > 0 && leftAverage > settings->FlowThreshold)
{
nxt->turnRight(Convert::ToInt32(settings->RobotSpeed));
Thread::Sleep(sleepTime);
}
else if (delta < 0 && rightAverage > settings->FlowThreshold)
{
nxt->turnLeft(Convert::ToInt32(settings->RobotSpeed));
Thread::Sleep(sleepTime);
}
else
{
nxt->moveForward(Convert::ToInt32(settings->RobotSpeed));
Thread::Sleep(100);
}
}
}
catch(const std::exception &e)
{
System::String ^errorMessage = System::String::Format(L"{0}", gcnew
System::String(e.what()));
throw gcnew ApplicationException(errorMessage);
}
catch (ThreadAbortException ^)
{
}
catch (System::Exception^ ex)
{
MessageBox::Show("Error:" + ex->Message, "Error", MessageBoxButtons::OK,
MessageBoxIcon::Error);
}
finally
{
nxt->stop();
nxt->Disconnect();
}
}
private:
// Η συνάρτηση αυτή υπολογίζει την οπτική ροή για τις δύο πλευρές της εικόνας και
υπολογ´ζιει την τιμή της μεταβλητής δέλτα
System::Void opticalFlow()
{
int cameraNumber = Convert::ToInt32(settings->CameraNumber);
int cameraWidth = Convert::ToInt32(settings->Resolution.Width);
int cameraHeight = Convert::ToInt32(settings->Resolution.Height);
CvCapture* capture = NULL;
IplImage *frame = NULL;
IplImage *opticalFlowImage = NULL;
IplImage *currentGrayImage = NULL;
IplImage *previousGrayImage = NULL;
IplImage *currentPyramidImage = NULL;
IplImage *previousPyramidImage = NULL;
IplImage *tempSwapImage = NULL;
```

```
CvPoint2D32f* points[2] = {0,0};
CvPoint2D32f* swapPoints;
points[0] = (CvPoint2D32f*)cvAlloc(settings->MaxPoints * sizeof(points[0][0]));
points[1] = (CvPoint2D32f*)cvAlloc(settings->MaxPoints * sizeof(points[0][0]));
char *status = (char*)cvAlloc(settings->MaxPoints);

int pointsCount = 0;
int flags = 0;
int frameCounter = 0;
TextWriter ^csvWriter;
try
{
// Σύνδεση με την κάμερα
capture = cvCaptureFromCAM(cameraNumber);
if (!capture)
{
MessageBox::Show("Could not find
camera","Error",MessageBoxButtons::OK,MessageBoxIcon::Error);
releaseMemory(capture, opticalFlowImage, currentGrayImage, previousGrayImage,
currentPyramidImage, previousPyramidImage, tempSwapImage);
if(this->InvokeRequired)
this->Invoke(gcnew
Navigator::mainForm::enableDelegate(this,&Navigator::mainForm::enable));
else
enable();
return;
}
cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, cameraWidth);
cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, cameraHeight);
if(this->InvokeRequired)
this->Invoke(gcnew
Navigator::mainForm::enableStartButtonDelegate(this,&Navigator::mainForm::enableStartBut
ton));
else
enableStartButton();
// Διάβασε την πρώτη εικόνα
frame = cvQueryFrame(capture);
// Αρχικοποίησε όλες τις εικόνες
opticalFlowImage = cvCreateImage(cvSize(frame->width, frame->height), frame->depth,
frame->nChannels);
currentGrayImage = cvCreateImage(cvSize(frame->width, frame->height), IPL_DEPTH_8U, 1 );
previousGrayImage = cvCreateImage(cvSize(frame->width, frame->height), IPL_DEPTH_8U, 1
);
currentPyramidImage = cvCreateImage(cvSize(frame->width, frame->height), IPL_DEPTH_8U, 1
);
previousPyramidImage = cvCreateImage(cvSize(frame->width, frame->height), IPL_DEPTH_8U,
1 );
int middleWidth = cameraWidth / 2;
int middleHeight = cameraHeight / 2;
leftAverage = 0.0;
rightAverage = 0.0;
int leftCount = 0;
int rightCount = 0;
if(settings->SaveData)
{
if(!Directory::Exists(Application::StartupPath + "\\Data\\" + settings-
>FolderNumber.ToString()))
{
Directory::CreateDirectory(Application::StartupPath + "\\Data\\" + settings-
>FolderNumber.ToString());
Directory::CreateDirectory(Application::StartupPath + "\\Data\\" + settings-
>FolderNumber.ToString() + "\\Images");
}
csvWriter = gcnew StreamWriter(Application::StartupPath + "\\Data\\" + settings-
>FolderNumber.ToString() + "\\" + settings->FolderNumber.ToString() + ".csv");
}
while (isRunning)
{
// Διάβασε την επόμενη εικόνα
frame = cvQueryFrame(capture);
cvCopy( frame, opticalFlowImage, 0 );
```

```
Bitmap ^bitmap = imageUtility->getResizedBitmap(mainPictureBox, opticalFlowImage);
cvCvtColor( opticalFlowImage, currentGrayImage, CV_BGR2GRAY );
imageUtility->drawText(bitmap, gcnew System::Drawing::Font("Tahoma",18), L"Αριστερά
σημεία: " + leftCount.ToString(), Color::Red, gcnew PointF(50.0f, (float)cameraHeight -
150.0f));
imageUtility->drawText(bitmap, gcnew System::Drawing::Font("Tahoma",18), L"Δεξιά σημεία:
" + rightCount.ToString(), Color::Red, gcnew PointF(50.0f + (float)middleWidth,
(float)cameraHeight - 150.0f));
// Εαν δεν υπάρχουν αρκετά σημεία, χρησιμοποίησε τον αλγόριθμο των Shi και Tomasi
if( leftCount < 20 || rightCount < 20)
{
IplImage* eigenImage = cvCreateImage( cvGetSize(currentGrayImage), 32, 1 );
IplImage* tempImage = cvCreateImage( cvGetSize(currentGrayImage), 32, 1 );
pointsCount = settings->MaxPoints;
cvGoodFeaturesToTrack( currentGrayImage, eigenImage , tempImage, points[1],
&pointsCount, 0.05, 5, 0, 3, 0, 0.04 );
cvFindCornerSubPix( currentGrayImage, points[1], pointsCount, cvSize(10,10), cvSize(-1,-
1), cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,0.03));
cvReleaseImage( &eigenImage );
cvReleaseImage( &tempImage );
flags = 0;
for(int i = 0; i < pointsCount; i++ )
{
if( !status[i] )
continue;
if(     points[1][i].x < middleWidth)
leftCount++;
else
rightCount++;
}
if(settings->SaveData)
csvWriter->WriteLine(frameCounter.ToString()+",0,0,0");
}
else // Αλλιώς βρές την οπτική ροή και υπολόγισε τις μεταβλήτες
{
cvCvtColor( frame, currentGrayImage, CV_BGR2GRAY );
cvCalcOpticalFlowPyrLK( previousGrayImage, currentGrayImage, previousPyramidImage,
currentPyramidImage, points[0], points[1], pointsCount, cvSize(10,10), 3, status, 0,
cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,0.03), flags );
flags |= CV_LKFLOW_PYR_A_READY;
int i, k;
leftCount = 0;
rightCount = 0;
leftAverage = 0.0f;
rightAverage = 0.0f;
int leftGoodPoints = 0;
int rightGoodPoints = 0;
for( i = k = 0; i < pointsCount; i++ )
{
if( !status[i] )
continue;
points[1][k++] = points[1][i];
PointF ^previousPoint = gcnew PointF(points[0][i].x, points[0][i].y);
PointF ^currentPoint = gcnew PointF(points[1][i].x, points[1][i].y);
double angle = Math::Atan2( (double) previousPoint->Y - currentPoint->Y, (double)
previousPoint->X - currentPoint->X );
double hypotenuse = Math::Sqrt( Math::Pow(previousPoint->Y - currentPoint->Y,2) +
Math::Pow(previousPoint->X - currentPoint->X,2) );
if (currentPoint->X < middleWidth)
leftCount++;
else
rightCount++;
if (hypotenuse < 5 ) // Αφαίρεσε τα σημεία που δεν έχουν μετακινηθεί αρκετά
{
imageUtility->drawCircle(bitmap, Color::Green, currentPoint, 5);
}
else if (hypotenuse > 60) // Αφαίρεσε τα σημεία που έχουν μετακινηθεί πολύ
{
imageUtility->drawCircle(bitmap, Color::Yellow, currentPoint, 5);
}
else
```

```
{
if (currentPoint->X < middleWidth)
{
leftAverage += currentPoint->Y - previousPoint->Y;
leftGoodPoints++;
}
else
{
rightAverage += currentPoint->Y - previousPoint->Y;
rightGoodPoints++;
}
currentPoint->X = (float)(previousPoint->X - 3 * hypotenuse * Math::Cos(angle));
currentPoint->Y = (float)(previousPoint->Y - 3 * hypotenuse * Math::Sin(angle));
imageUtility->drawLine(bitmap,Color::Red,previousPoint,currentPoint);
previousPoint->X = (float)(currentPoint->X + 9 * Math::Cos(angle + Math::PI / 4));
previousPoint->Y = (float)(currentPoint->Y + 9 * Math::Sin(angle + Math::PI / 4));
imageUtility->drawLine(bitmap,Color::Red,previousPoint,currentPoint);
previousPoint->X = (float)(currentPoint->X + 9 * Math::Cos(angle - Math::PI / 4));
previousPoint->Y = (float)(currentPoint->Y + 9 * Math::Sin(angle - Math::PI / 4));
imageUtility->drawLine(bitmap,Color::Red,previousPoint,currentPoint);
}
}
pointsCount = k;
// Υπολόγισε την μέση αριστερή και δεξιά οπτική ροή
if(leftGoodPoints>0)
leftAverage = Math::Abs(leftAverage) / leftGoodPoints;
if(rightGoodPoints>0)
rightAverage = Math::Abs(rightAverage) / rightGoodPoints;
// Υπολόγισε την μεταβλήτή δέλτα
delta = 0;
if(leftGoodPoints>0 || rightGoodPoints >0)
delta = (float)((leftAverage - rightAverage) / (leftAverage + rightAverage));
imageUtility->drawText(bitmap, gcnew System::Drawing::Font("Tahoma",18), L"Δέλτα:
"+delta.ToString("0.00"), Color::Blue, gcnew PointF(50.0f, (float)cameraHeight -
100.0f));
imageUtility->drawLine(bitmap, Color::Orange,gcnew PointF((float)middleWidth,0),gcnew
PointF((float)middleWidth,(float)cameraHeight));
imageUtility->drawText(bitmap, gcnew System::Drawing::Font("Tahoma",18), L"Αριστερή ροή:
" + leftAverage.ToString("0.00"), Color::Blue, gcnew PointF(50.0f, (float)cameraHeight -
50.0f));
imageUtility->drawText(bitmap, gcnew System::Drawing::Font("Tahoma",18), L"Δεξιά ροή: "
+ rightAverage.ToString("0.00"), Color::Blue, gcnew PointF(50.0f + (float)middleWidth,
(float)cameraHeight - 50.0f));
imageUtility->drawText(bitmap, gcnew System::Drawing::Font("Tahoma",18), L"Αριθμός
σημείων: " + pointsCount.ToString(), Color::Green, gcnew PointF(50.0f +
(float)middleWidth, (float)cameraHeight - 100.0f));
if (leftAverage > settings->FlowThreshold && rightAverage > settings->FlowThreshold)
{
imageUtility->drawText(bitmap, gcnew System::Drawing::Font("Tahoma",20), L"Εμπόδιο
αριστερά", Color::Red, gcnew PointF(50.0f, 5.0f));
imageUtility->drawText(bitmap, gcnew System::Drawing::Font("Tahoma",20), L"Εμπόδιο
δεξιά", Color::Red, gcnew PointF(50.0f + (float)middleWidth, 50.0f));
}
else if (delta > 0 && leftAverage > settings->FlowThreshold)
{
imageUtility->drawText(bitmap, gcnew System::Drawing::Font("Tahoma",20), L"Εμπόδιο
αριστερά", Color::Red, gcnew PointF(50.0f, 50.0f));
}
else if (delta < 0 && rightAverage > settings->FlowThreshold)
{
imageUtility->drawText(bitmap, gcnew System::Drawing::Font("Tahoma",20), L"Εμπόδιο
δεξιά", Color::Red, gcnew PointF(50.0f + (float)middleWidth, 50.0f));
}
if(settings->SaveData)
csvWriter-
>WriteLine(frameCounter.ToString()+","+leftAverage.ToString()+","+rightAverage.ToString(
)+","+delta.ToString());
}
// Ανταλλαγή εικόνων και σημείων
CV_SWAP( previousGrayImage, currentGrayImage, tempSwapImage );
CV_SWAP( previousPyramidImage, currentPyramidImage, tempSwapImage );
```

```
CV_SWAP( points[0], points[1], swapPoints );
if(this->InvokeRequired)
this->Invoke(gcnew
Navigator::ImageUtility::showImageDelegate(imageUtility,&Navigator::ImageUtility::showIm
age),mainPictureBox, bitmap, settings->SaveData, frameCounter,
Convert::ToInt32(settings->FolderNumber));
else
imageUtility->showImage(mainPictureBox, bitmap, settings->SaveData, frameCounter,
Convert::ToInt32(settings->FolderNumber));
frameCounter++;
}
}
catch(const std::exception &e)
{
System::String ^errorMessage = System::String::Format(L"{0}", gcnew
System::String(e.what()));
throw gcnew ApplicationException(errorMessage);
}
catch (ThreadAbortException ^)
{
}
catch (System::Exception^ ex)
{
MessageBox::Show("Error:" + ex->Message, "Error", MessageBoxButtons::OK,
MessageBoxIcon::Error);
}
finally
{
releaseMemory(capture, opticalFlowImage, currentGrayImage, previousGrayImage,
currentPyramidImage, previousPyramidImage, tempSwapImage);
if(csvWriter != nullptr)
csvWriter->Close();
if(this->InvokeRequired)
this->Invoke(gcnew
Navigator::mainForm::enableDelegate(this,&Navigator::mainForm::enable));
else
enable();
}
}
private:
// Η συνάρτηση αυτή απελευθερώνει την μνήμη που χρησιμοποιήθηκε για τις εικόνες
System::Void releaseMemory(CvCapture *capture, IplImage *opticalFlowImage, IplImage
*currentGrayImage, IplImage *previousGrayImage, IplImage *currentPyramidImage, IplImage
*previousPyramidImage, IplImage *tempSwapImage)
{
if(capture)
cvReleaseCapture(&capture);
if(opticalFlowImage)
cvReleaseImage(&opticalFlowImage);
if(currentGrayImage)
cvReleaseImage(&currentGrayImage);
if(previousGrayImage)
cvReleaseImage(&previousGrayImage);
if(currentPyramidImage)
cvReleaseImage(&currentPyramidImage);
if(previousPyramidImage)
cvReleaseImage(&previousPyramidImage);
}
#pragma endregion
};
}
```

Navigator.cpp

Ο κώδικας αυτός είναι υπεύθυνος για την εκκίνηση της εφαρμογής.

```
#include "mainForm.h"
using namespace Navigator;
[STAThreadAttribute]
int main(array<System::String ^> ^args)
```

```
{
        // Ενεργοποίση των οπτικών εφέ των windows XP πριν την δημιουργία των controls
        Application::EnableVisualStyles();
        Application::SetCompatibleTextRenderingDefault(false);

        // Δημιουργία της κύριας φόρμας και εκτέλεση της εφαρμογής
        Application::Run(gcnew mainForm());
        return 0;
}
```

ΔΕΝ ΕΠΙΤΡΕΠΕΤΑΙ

Βιβλιογραφία

[1] G. N. DeSouza and A. C. Kak, "Vision for mobile robot navigation: A survey," *IEEE transactions on pattern analysis and machine intelligence*, pp. 237–267, 2002.

[2] L. M. Paz, P. Piniés, J. D. Tardós, and J. Neira, "Large-scale 6-dof slam with stereo-in-hand."

[3] D. Ribas, P. Ridao, J. Neira, and J. D. Tardós, "Slam using an imaging sonar for partially structured underwater environments," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5040–5045, 2006.

[4] M. Sarcinelli-Filho, H. J. A. Schneebeli, E. M. O. Caldeira, and C. M. Soria, "Optical Flow-Based Obstacle Detection and Avoidance in Mobile Robot Navigation," in *Proceedings of the*, vol. 10, 2002.

[5] N. X. Dao, B. J. You, and S. R. Oh, "Visual navigation for indoor mobile robots using a single camera."

[6] K. Souhila and A. Karim, "Optical Flow based robot obstacle avoidance," *International Journal of Advanced Robotic Systems*, vol. 4, no. 1, pp. 13–16, 2007.

[7] T. Low and G. Wyeth, "Obstacle detection using optical flow," in *Australasian Conference on Robotics and Automation*, pp. 1–10, 2005.

[8] R. Carelli, H. Secchi, V. Mut, and O. Nasisi, "Stable algorithms for the navigation of mobile robots in corridors using optical flow," in *Proceedings of the 8 th Workshop on Information Processing and Control*, pp. 79–7.

[9] M. Srinivasan, S. Zhang, M. Lehrer, and T. Collett, "Honeybee navigation en route to the goal: visual flight control and odometry," *Journal of Experimental Biology*, vol. 199, no. 1, p. 237, 1996.

[10] D. Coombs and K. Roberts, "'Bee-bot': using peripheral optical flow to avoid obstacles," in *Proceedings of SPIE*, vol. 714, 1825.

[11] J. Tautz et al., "Honeybee odometry: performance in varying natural terrain," *PLoS Biology*, vol. 2, pp. 915–923, 2004.

[12] S. S. Beauchemin and J. L. Barron, "The computation of optical flow," *ACM Computing Surveys (CSUR)*, vol. 27, no. 3, p. 466, 1995.

[13] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *International journal of computer vision*, vol. 12, no. 1, pp. 43–77, 1994.

[14] Y. Huang and X. Zhuang, "Motion-partitioned adaptive block matching for video compression," in *Proceedings of the 1995 International Conference on Image Processing (Vol. 1)-Volume 1-Volume 1*, p. 554, 1995.

[15] A. Barjatya, "Block matching algorithms for motion estimation," in *IEEE Conference on Computational Intelligence and Multimedia Applications*, vol. 3, pp. 89–96, 2007.

[16] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial intelligence*, vol. 17, no. 1, pp. 185–203, 1981.

[17] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *International joint conference on artificial intelligence*, vol. 3, pp. 674–679, 1981.

[18] J. Bergen, P. Anandan, K. Hanna, and R. Hingorani, "Hierarchical model-based motion estimation," in *Computer Vision—ECCV'92*, pp. 237–252, 1992.

[19] J. Y. Bouguet and others, "Pyramidal implementation of the lucas kanade feature tracker description of the algorithm," *Intel Corporation, Microprocessor Research Labs, OpenCV Documents*, 1999.

[20] H. P. Morevec, "Towards automatic visual obstacle avoidance," in *Proceedings of the 5th international joint conference on Artificial intelligence-Volume 2*, p. 584, 1977.

[21] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey vision conference*, vol. 15, p. 50, 1988.

[22] J. Shi and C. Tomasi, "Good features to track," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 593–593, 1994.

[23] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., 2008.