Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων

Πτυχιακή Εργασία

Διεπαφή Χρήστη – Υπολογιστή με την Τεχνολογία καταγραφής κίνησης οφθαλμού.

Λυδάκης Μανώλης ΑΜ:1705
Χατζάκη Χαρίκλεια ΑΜ:1744

Επιβλέπων καθηγητής : Φουντουλάκης Αντώνης

Επιτροπή Αξιολόγησης :

Ημερομηνία παρουσίασης:

Abstract

Eye-movement tracking is a method that is increasingly being employed to study usability issues in Human Computer Interaction contexts.
A good structured eye tracking system could interact with interfaces directly without the need for mouse or keyboard input.

In this paper are given informations, about the basic meaning of the term human –computer interface and how to implement HCI , how the ophthalmic system works. We developed various eye trackers, but always noting the size of radiation could be received by each user.

Finally, we introduce the reader to our eye tracking application in both hardware and software sector.

Σύνοψη

Η καταγραφή κίνησης οφθαλμού είναι μια μέθοδος που ερευνάται ολοένα και περισσότερο για την ευχρηστία που μπορεί να προσφέρει στον τομέα της διεπαφής χρήστη υπολογιστή.
Ένα άρτιο σύστημα καταγραφής οφθαλμού θα μπορούσε να αλληλεπιδρά με διεπαφές κατευθείαν χωρίς την χρήση πληκτρολογίου ή ποντικιού.

Σε αυτήν την πτυχιακή εργασία δίνονται πληροφορίες , για την βασική έννοια του όρου διεπαφή χρήστη - υπολογιστή καθώς και τρόπους υλοποίησης αυτής , για την λειτουργία του οφθαλμικού συστήματος. Αναπτύσσονται διάφορα συστήματα εντοπισμού οφθαλμικών κινήσεων καθώς και συστήματα εντοπισμού και παρακολούθησης του οφθαλμού, τονίζοντας όμως το μέγεθος της ακτινοβολίας που θα μπορούσε να δεχτεί ο εκάστου χρήστης.

Τέλος αναπτύσσεται η δική μας εφαρμογή καταγραφής κίνησης οφθαλμού σε ότι αφορά το υλικό μέρος αλλά και σε ότι αφορά το λογισμικό μέρος του συστήματος.

Πίνακας Περιεχομένων

Κεφάλαια

Πίνακας εικόνων

Πίνακας εικόνων

Εισαγωγή

Η καταγραφή κίνησης του οφθαλμού είναι μια τεχνική όπου μπορεί να μας παρέχει πληροφορίες χαρτογράφησης των κινήσεων των ματιών. Έτσι μπορούμε να γνωρίζουμε που κοιτάζει ο χρήστης σε μια δεδομένη στιγμή αλλά και με ποια συχνότητα μετατοπίζεται η προσοχή του χρήστη από μια περιοχή σε μια άλλη.
Κατά αυτόν τον τρόπο μπορούμε να εξάγουμε αποτελέσματα για την ευχρηστία μιας διεπαφής και τελικά να την εξελίξουμε.

Μια ιδιαίτερα σημαντική πτυχή της εφαρμογής αυτής είναι ότι μπορεί να επιτρέπει στον χρήστη την διεπαφή με τον υπολογιστή χωρίς την χρήση συσκευών εισόδου , το όποιο αποτελεί σημαντικό προτέρημα για ορισμένες πληθυσμιακές ομάδες χρηστών, όπως άτομα με ειδικές ανάγκες.
Ιδιαίτερα για τα άτομα με ειδικές ανάγκες θα μπορούσαμε να δημιουργήσουμε εφαρμογές ,όπως πληκτρολόγιο οθόνης, εφαρμογή ειδοποίησης βασικών αναγκών , οι οποίες θα συνεργάζονται με το σύστημα εντοπισμού και καταγραφής οφθαλμού.

Τα συστήματα που έχουν αναπτυχθεί έως σήμερα περιλαμβάνουν συσκευή κάμερας ,τοποθετημένη στο κεφάλι του χρήστη είτε κάμερα σταθερή σε κάποιο σημείο, με το ανάλογο λογισμικό μέρος. Ένας άλλος σημαντικός παράγοντας είναι η μοναδικότητα κάθε χρήστη, για αυτό θα πρέπει να επιλεγεί το κατάλληλο σύστημα ανάλογα με την περίπτωση.

Για να μπορέσει κανείς να κατανοήσει την λειτουργιά του συστήματος θα πρέπει να έχει τουλάχιστον τις βασικές γνώσεις της διεπαφής χρήστη υπολογιστή και του οφθαλμικού συστήματος . Στη πτυχιακή εργασία αναφέρονται αναλυτικά τόσο τα παραπάνω όσο και οι τρόποι με του οποίους εντοπίζεται το μάτι και παρακολουθείτε.

Τέλος, η εφαρμογή που δημιουργήσαμε ίσως θα μπορούσε να γίνει η αρχή για ένα άρτιο και εύχρηστο σύστημα εντοπισμού και παρακολούθησης οφθαλμού.

Κεφάλαιο 1 : Διεπαφή χρήστη υπολογιστή.

1.1 Ορισμός διεπαφής χρήστη υπολογιστή

Ο Όρος διεπαφή χρήστη υπολογιστή αναφέρεται στην σχέση επικοινωνίας- αλληλεπίδρασης μεταξύ των ανθρώπων (users) και του υπολογιστή. Όπως είναι γνωστό η επικοινωνία βασίζεται στην ανταλλαγή μηνυμάτων τα οποία μεταδίδονται με ποικίλους τρόπους προκειμένου να εκφράσουν νόημα και τα οποία γίνονται αντιληπτά μέσω διαφόρων αισθητηρίων οργάνων ,όπως παραδείγματος χάριν κείμενο, εικόνα, σήμα, σύμβολο, ένδειξη.

Η αλληλεπίδραση μεταξύ των χρηστών και των υπολογιστών απαιτεί ένα περιβάλλον εργασίας (interface) το οποίο περιλαμβάνει τόσο το λογισμικό (software) όσο και το υλικό (hardware) μέρος. Για παράδειγμα οι χαρακτήρες ή τα αντικείμενα που εμφανίζονται μέσω του λογισμικού στην οθόνη ενός προσωπικού υπολογιστή όπου εισήλθαν μέσω περιφερειακών συσκευών υλικού όπως πληκτρολόγια ,ποντίκια , οθόνες κ.α.

Η Association for Computing Machinery ορίζει ως αλληλεπίδραση ανθρώπου-υπολογιστή «μια επιστήμη που ασχολείται με το σχεδιασμό, την αξιολόγηση και την υλοποίηση των διακρατικών υπολογιστικών συστημάτων για ανθρώπινη χρήση και με τη μελέτη των μεγάλων φαινομένων γύρω τους."
Μια σημαντική πτυχή της αλληλεπίδρασης μεταξύ των χρηστών και των υπολογιστών είναι η εξασφάλιση της ικανοποίησης των χρηστών

Επειδή η διεπαφή χρήστη υπολογιστή αναφέρεται στον άνθρωπο και στον υπολογιστή σαν συνδυασμό απαιτεί γνώσεις υποστήριξης και από τις δυο μεριές.
Από την πλευρά της μηχανής είναι σημαντικές γνώσεις γύρω από , τεχνικές σε γραφικά ηλεκτρονικών υπολογιστών, τα λειτουργικά συστήματα, γλώσσες προγραμματισμού και περιβάλλοντα ανάπτυξης .
Από την ανθρώπινη πλευρά έχουν σημασία, η θεωρία της επικοινωνίας, των γραφικών και των βιομηχανικών κλάδων το σχεδιασμό, τη γλωσσολογία, τις κοινωνικές επιστήμες, γνωστική ψυχολογία.
Λόγω της πολυδιάστατης φύσης της επαφής χρηστή υπολογιστή , άνθρωποι με διαφορετικό γνωσιακό επίπεδο μεταβάλλουν την επιτυχή αλληλεπίδραση.

1.2 Στόχος της επαφής χρήστη υπολογιστή

Ένας βασικός στόχος της επαφής χρήστη υπολογιστή είναι η βελτίωση των αλληλεπιδράσεων μεταξύ χρηστών και υπολογιστών, κάνοντας τους υπολογιστές πιο εύχρηστους και δεκτικούς στις ανάγκες του χρήστη. Συγκεκριμένα, διεπαφή χρήστη υπολογιστή ασχολείται με:

- Μεθοδολογίες και διαδικασίες για το σχεδιασμό διεπαφών (δηλαδή, δίνεται μια εργασία και μια κατηγορία χρηστών, σχεδιάζεται με τον καλύτερο δυνατό περιβάλλον εργασίας εντός συγκεκριμένων περιορισμών, βελτιστοποιείται για μια επιθυμητή ιδιότητα , όπως η ικανότητα μάθησης και την αποδοτικότητα της χρήσης)
- Μέθοδοι για την υλοποίηση των διεπαφών (π.χ. εργαλεία λογισμικού και βιβλιοθήκες, αποδοτικοί αλγόριθμοι)
- τεχνικές για την αξιολόγηση και τη σύγκριση των διεπαφών
- Ανάπτυξη νέων διεπαφών και τεχνικές αλληλεπίδρασης
- Ανάπτυξη περιγραφικών και προβλέψιμων μοντέλων και θεωριών αλληλεπίδρασης

Ο μακροπρόθεσμος στόχος της διεπαφής χρήστη υπολογιστή είναι να σχεδιάσει συστήματα που ελαχιστοποιούν το φράγμα μεταξύ γνωσιακού μοντέλο του ανθρώπου για το τι θέλουν να ολοκληρώσουν και την κατανόηση του υπολογιστή της αποστολής του χρήστη.

Οι Επαγγελματίες στον κλάδο της διεπαφής χρήστη υπολογιστή είναι συνήθως σχεδιαστές που ασχολούνται με την πρακτική εφαρμογή των μεθόδων σχεδιασμού για πραγματικά προβλήματα. Η εργασία τους περιστρέφεται γύρω από τον σχεδιασμό συχνά γραφικών διεπαφών και διεπαφών διαδικτύου.

Οι ερευνητές στον τομέα αλληλεπίδρασης χρήστη υπολογιστή ασχολούνται με την ανάπτυξη νέων μεθόδων σχεδιασμού, με νέες συσκευές υλικού, κατασκευή πρωτοτύπων

νέων συστημάτων λογισμικού, αναζήτηση νέων παραδειγμάτων για την αλληλεπίδραση, και την ανάπτυξη μοντέλων και θεωριών αλληλεπίδρασης.

1.3 Σχεδιαστικές αρχές διεπαφής χρήστη

Κατά την αξιολόγηση μιας τρέχουσας διεπαφής χρήστη, είτε κατά την σχεδίαση ενός νέο περιβάλλον διεπαφής χρήστη είναι σημαντικό να έχουμε κατά νου τις ακόλουθες πειραματικές σχεδιαστικές αρχές:

- Άμεση επικέντρωση στον/ους χρήστη (ες) και στην/ις εργασία (-ες):

διευκρίνιση του πλήθους των χρηστών όπου θα διατελέσουν τις εργασίες καθώς και τα κριτήρια όπου θα τους κρίνουν ικανούς. Ποιες εργασίες πρέπει να γίνουν από τους χρήστες και ποια είναι η συχνότητα τους.

- Η εμπειρική μέτρηση:

Ελέγξτε το περιβάλλον από νωρίς με τους πραγματικούς χρήστες που έρχονται σε επαφή με το περιβάλλον σε καθημερινή βάση. Λάβετε υπόψη ότι τα αποτελέσματα μπορεί να τροποποιούνται εάν το επίπεδο των επιδόσεων του χρήστη, δεν είναι μια ακριβής απεικόνιση της πραγματικής αλληλεπίδρασης ανθρώπου-υπολογιστή. Ιδιαίτερη σημασία χρειάζεται να δοθεί τις λεπτομέρειες χρηστικότητας, όπως: ο αριθμός των χρηστών που εκτελεί την εργασία (-ες), ο χρόνος για να ολοκληρωθεί η εργασία (-ες), και ο αριθμός των λαθών που έγιναν κατά την εργασία (ες).

- Επαναληπτικός σχεδιασμός:

Μετά τον προσδιορισμό των χρηστών, τις εργασίες και εμπειρικές μετρήσεις έχουμε την επαναληπτική διαδικασία σχεδιασμού:

1. Σχεδιασμός της διεπαφής χρήστη
2. Δοκιμής
3. Αναλύστε τα αποτελέσματα
4. Επαναλαμβάνω

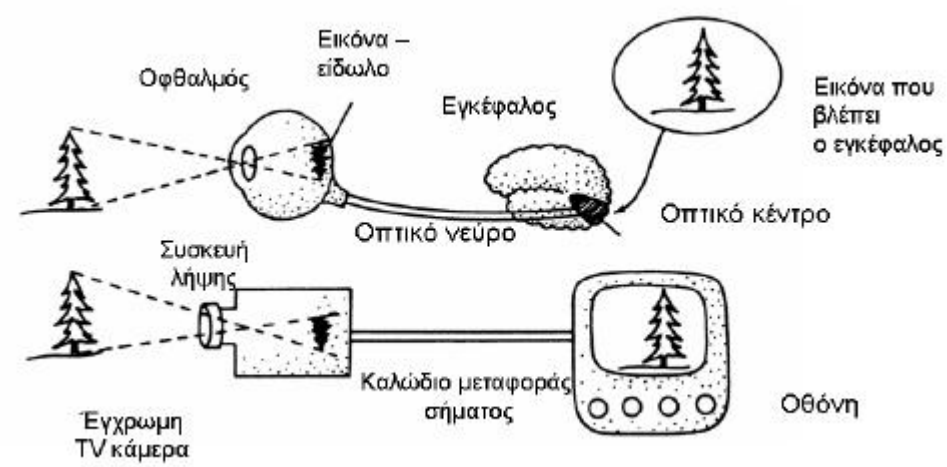
Επαναλάβετε την επαναληπτική διαδικασία σχεδιασμού μέχρι να δημιουργηθεί μια λογική, φιλική προς το χρήστη διεπαφή.

Σε αυτό το σημείο, κρίνεται αναγκαίο, και λαμβάνοντας υπ' όψιν όλα τα παραπάνω, να αναφερθούμε στην ανατομία του ματιού αλλά και σε, ανατομικές αλλά και λειτουργικές λεπτομέρειες, που αφορούν τον ανθρώπινο οφθαλμό.

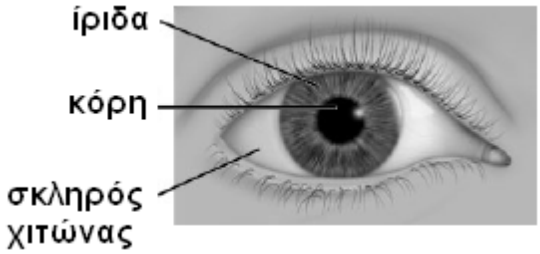
Κεφάλαιο 2 : Σύστημα όρασης

2.1 Ο οφθαλμός

Ο οφθαλμός είναι το αισθητήριο όργανο μέσω του οποίου πραγματοποιείται η λειτουργία της όρασης.. Ο οφθαλμός ανιχνεύει ηλεκτρομαγνητική ακτινοβολία, με μήκη κύματος που ανήκουν στην ορατή περιοχή του φάσματος του φωτός. Στη συνέχεια το συλλεγόμενο φως μετατρέπεται σε ηλεκτρικά σήματα τα οποία μεταδίδονται στον εγκέφαλο, ο οποίος κατόπιν τα μεταφράζει σε οπτικές εικόνες.



Εικόνα 1 : Αναπαράσταση λειτουργιάς όρασης



Εικόνα 2: Εξωτερικά, ο οφθαλμός αποτελείται από τα ακόλουθα βασικά μέρη:
την κόρη (pupil), την ίριδα (iris) και τον σκληρό χιτώνα (sclera).

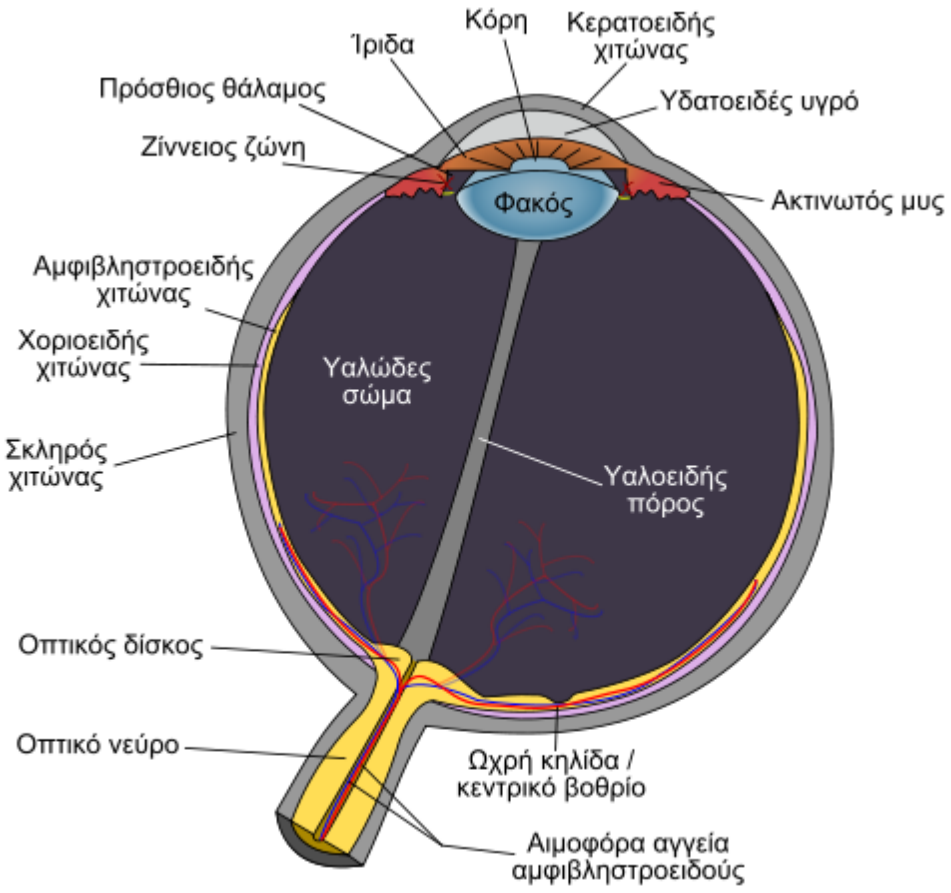
2.1.1 Ανατομία οφθαλμού

Ο οφθαλμός αποτελείται από τον οφθαλμικό βολβό ,που είναι το κυρίως όργανο της όρασης και από τα επικουρικά όργανα όπως βλέφαρα, μύες κ.τ.λ. Τα επικουρικά όργανα είναι απαραίτητα για την προστασία και την λειτουργία του οφθαλμικού βολβού.
Ο οφθαλμός έχει τρεις τύπους αδένων τους δακρυϊκούς αδένες , τους σμηγματογόνους αδένες και τους βλεφαρικούς αδένες.

Οι δακρυϊκοί αδένες βρίσκονται στο βλέφαρο, ακριβώς επάνω από κάθε οφθαλμικό βολβό του ματιού. Οι δακρυϊκοί αδένες εκκρίνουν συνεχώς μικρή ποσότητα δακρύων που διανέμεται στην επιφάνεια του ματιού με το βλεφαρισμό, αποχετεύεται στο δακρυϊκό ασκό μέσω των δακρυϊκών πόρων και κατόπιν, στη μύτη, μέσω του ρινοδακρυϊκού πόρου. Τα δάκρυα σχηματίζουν ένα προστατευτικό λεπτό υμένιο, που εφυγραίνει το μάτι και παρασύρει δυνητικώς επιβλαβή σωματίδια, όπως σκόνη και χημικές ουσίες.

Ακόμη, περιέχουν μία φυσική αντισηπτική ουσία, τη λυσοζύμη, που συντελεί στη προστασία των ματιών από τις μολύνσεις.

Οι σμηγματογόνοι αδένες εμφανίζονται κατά μήκος των ακρών βλέφαρων.



Εικόνα 3 : Ανατομία οφθαλμού

2.1.2 Βλέφαρα και βλεφαρίδες

Τρία σημαντικά προστατευτικά μέρη του ματιού είναι τα βλέφαρα, οι βλεφαρίδες και ο επιπεφυκότας, μια βλεννώδη εκκρίνουσα μεμβράνη. Τα άνω και κάτω βλέφαρα παρέχουν σημαντική προστασία στα μάτια, καθώς έχουν την ικανότητα να κλείνουν με μία αντανακλαστική κίνηση για να τα προστατεύουν από την είσοδο τυχόν έντονου φωτός αλλά και επιβλαβών υλικών.

Επιπλέον, το κάθε βλέφαρο έχει δύο ή τρεις σειρές βλεφαρίδων, γεγονός που ενισχύει περισσότερο την προστασία του ματιού. Ο επιπεφυκότας βρίσκεται σε δύο θέσεις: την εσωτερική επένδυση των βλέφαρων και το μπροστινό μέρος του βολβού του ματιού. Είναι ένας λεπτός διαφανής υμένας που καλύπτει το άσπρο του ματιού και επενδύει εσωτερικά τα βλέφαρα, όπως και το βολβό του ματιού, κρατώντας τα σε μία υγρή κατάσταση.

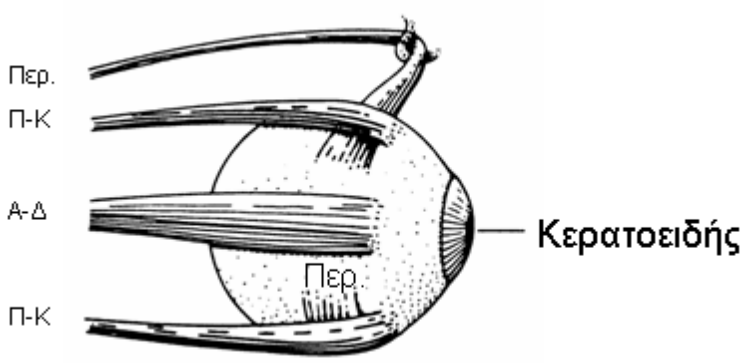
2.1.3 Εσωτερική δομή οφθαλμού

Κάθε οφθαλμικός βολβός είναι χονδρικά σφαιρικός, με διάμετρο περίπου 2,5 εκατ. και βρίσκεται προφυλαγμένος μέσα στον οφθαλμικό κόγχο, που σχηματίζουν τα οστά του κρανίου. Η ρύθμιση των κινήσεων του οφθαλμικού βολβού πραγματοποιείται με την βοήθεια των εξωβολβικών ή αλλιώς οφθαλμοκινητικών μυών (άνω ορθός, κάτω ορθός, έσω ορθός, έξω ορθός, άνω λοξός και κάτω λοξός). Οι έξι μύες λειτουργούν ανά ζεύγη: ένα ζεύγος είναι αρμόδιο για την προς τα πάνω και προς τα κάτω κίνηση, ένα ζεύγος για την αριστερή και δεξιά κίνηση και ένα ζεύγος ελέγχει την περιστροφή (εικόνα 4).

Ο βολβός του ματιού αποτελείται από τρία στρώματα: τον σκληρό χιτώνα, τον χοριοειδή χιτώνα και τον αμφιβληστροειδή χιτώνα. Ο σκληρός χιτώνας (ή το λευκό του ματιού) είναι ένα ανθεκτικό περίβλημα που περιβάλλει τον οφθαλμικό βολβό και διατηρεί το σχήμα του. χοριοειδής χιτώνας, ο οποίος τροφοδοτεί το μάτι με τις απαραίτητες θρεπτικές ουσίες, καλύπτει εσωτερικά το σκληρό. Ο χιτώνας αυτός περιλαμβάνει τον κερατοειδή χιτώνα, τον κρυσταλλοειδή φακό και την ίριδα.

Οι εισερχόμενες στο μάτι φωτεινές ακτίνες διαθλώνται αρχικά από τον κερατοειδή χιτώνα. Ο δακτύλιος των μυών της ίριδας ρυθμίζει το εύρος της κόρης για να επιτρέψει την είσοδο λιγότερων ή περισσότερων φωτεινών ακτινών. Ο κρυσταλλοειδής φακός με την ελαστικότητα που τον διακρίνει μπορεί ν' αλλάζει σχήμα ώστε να εστιάζει τις φωτεινές ακτίνες και από κοντινά και από μακρινά αντικείμενα.

Ο αμφιβληστροειδής χιτώνας καλύπτει εσωτερικά τον χοριοειδή και περιέχει δύο ειδών φωτοευαίσθητα κύτταρα: τα ραβδία, τα οποία είναι υπεύθυνα για την όραση σε αμυδρά φωτιζόμενους χώρους και τα κωνία, τα οποία χρησιμεύουν για την όραση στο φως και την αντίληψη των χρωμάτων.



Εικόνα 4: Η περιστροφική κίνηση του οφθαλμού μέσω των οφθαλμικών μυών.

2.1 .4 Αμφιβληστροειδής χιτώνας

Τοποθετημένος στο πίσω μέρος του ματιού, ο αμφιβληστροειδής είναι ο εσώτερος χιτώνας, ο οποίος περιέχει νευρικές ίνες και φωτοευαίσθητα κύτταρα. Στον αμφιβληστροειδή υπάρχουν δύο είδη φωτοευαίσθητων κυττάρων: τα ραβδία και τα κωνία.(Εικόνα 5)

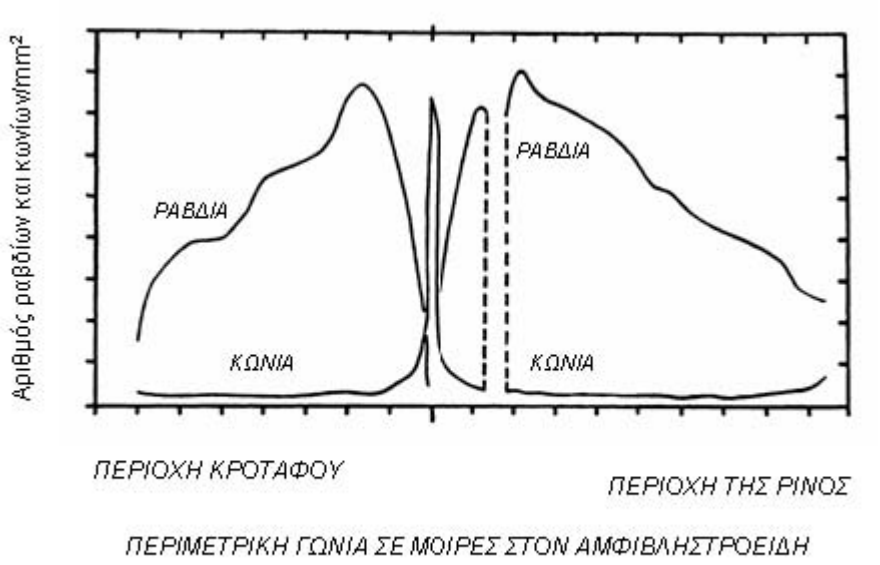
Τα ραβδία είναι υπεύθυνα για την όραση σε αμυδρό φως, αν και είναι ευαίσθητα σε όλες τις ορατές ακτινοβολίες, περιέχουν μία μόνοχρωστική και δε μπορούν να διακρίνουν τα χρώματα .

Τα κωνία είναι υπεύθυνα για την έγχρωμη και την υψηλής ευκρίνειας όραση. Κάθε κωνίο είναι ευαίσθητο στην ακτινοβολία ενός απ' τα τρία πρωταρχικά χρώματα, κόκκινο, πράσινο ή μπλε σε συνθήκες έντονου φωτισμού.

Τα κωνία διαχωρίζονται σε τρεις τύπους ή αλλιώς φασματικές κατηγορίες . Αυτά που αντιλαμβάνονται την κόκκινη ακτινοβολία (570nm), αυτά που λειτουργούν με την επίδραση της πράσινης ακτινοβολίας (530 nm) και αυτά που ερεθίζονται από την μπλε (400nm).

Όταν οι φωτεινές ακτίνες προσπέσουν στον αμφιβληστροειδή, τα κωνία και τα ραβδία διεγείρονται και παράγουν ηλεκτρικές ώσεις που αποτελούν το έναυσμα για τη δημιουργία περαιτέρω νευρικών ώσεων στα νευρικά κύτταρα, των οποίων αποφυάδες.

Οι νευρικές ώσεις μεταδίδονται στον εγκέφαλο μέσω του οπτικού νεύρου. Τα χρωμοφόρα κύτταρα πίσω από τα ραβδία και τα κωνία απορροφούν τις φωτεινές ακτίνες και αποτρέπουν την ανάκλασή τους μέσα στο μάτι.

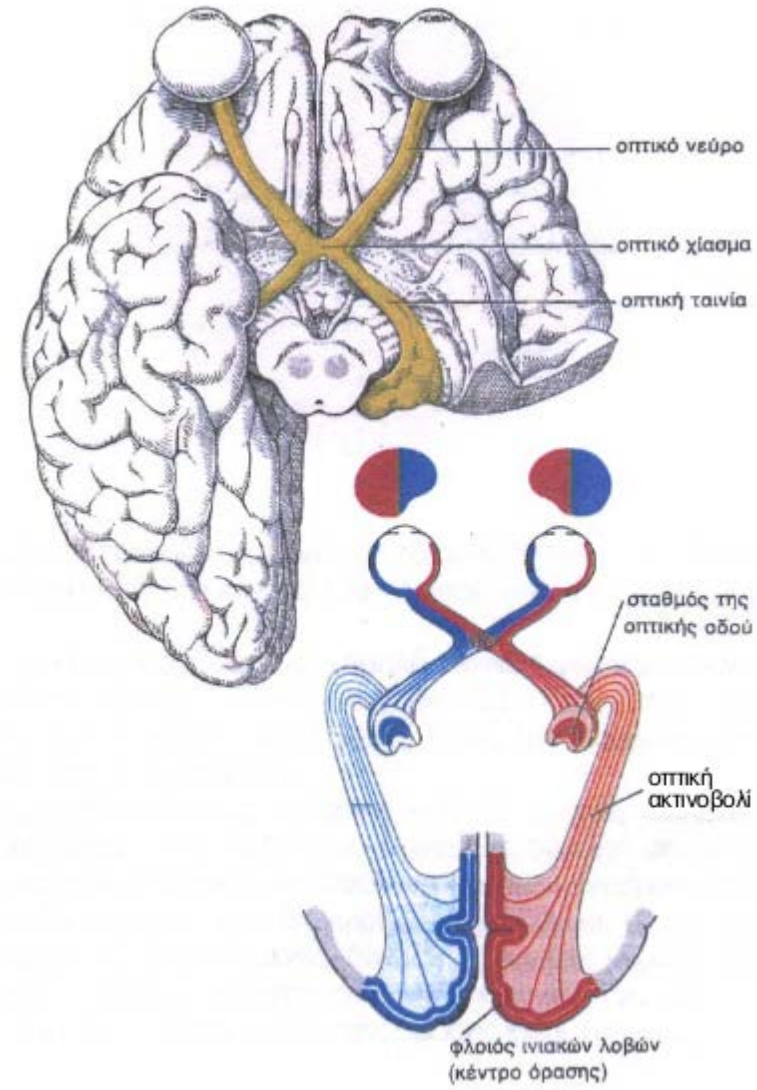


Εικόνα 5 : Κατανομή ραβδίων και κωνιών

2.1.5 Οπτικό νεύρο και οπτικός φλοιός

Οι νευρικές ώσεις από όλα τα κύτταρα (φωτοϋποδοχείς) του κάθε αμφιβληστροειδή μεταδίδονται μέσω των οπτικών νεύρων. Οι νευρικές ώσεις και από τα δύο μάτια φθάνουν σε μία περιοχή στο οπίσθιο τμήμα του εγκεφάλου, τον ινιακό λοβό , όπου εντοπίζεται το κέντρο της όρασης. Εκεί συνενώνονται, γίνεται η ανόρθωση της εικόνας και επιτυγχάνεται έτσι η πλήρης αντίληψη του οπτικού πεδίου.

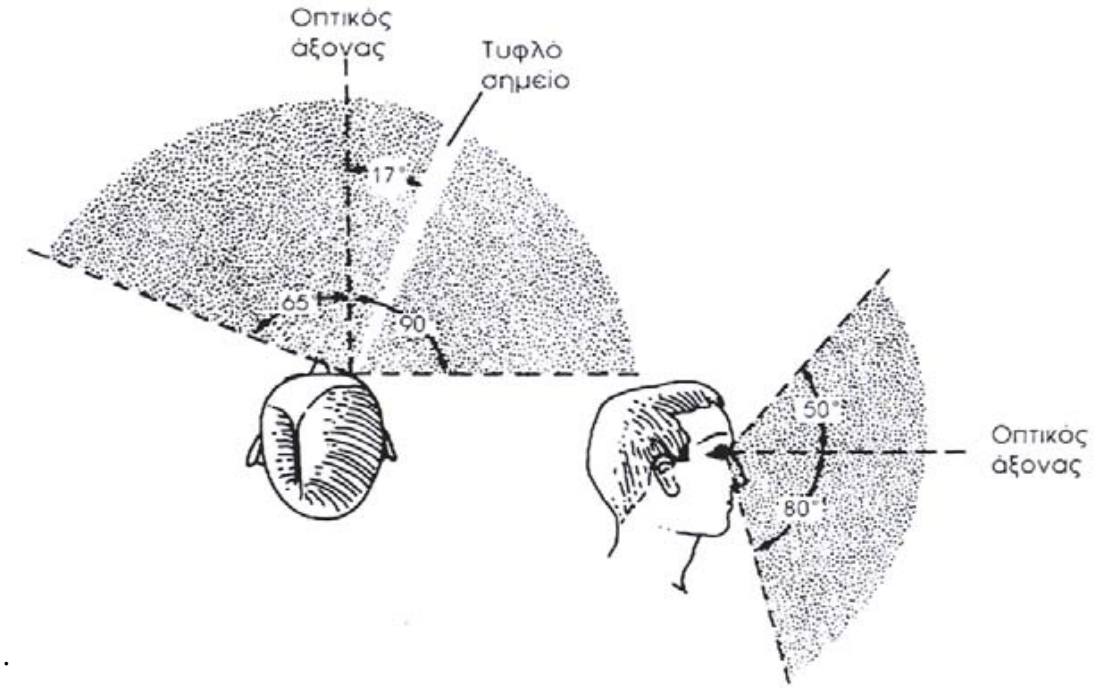
Ο οπτικός φλοιός, επομένως, που βρίσκεται στον εγκέφαλο, λαμβάνει τα νευρικά ερεθίσματα, εκεί όπου τα μηνύματα ενοποιούνται από τα δύο μάτια σε μια πλήρη εικόνα και μεταφράζονται σε μεγέθη, μορφές, αντικείμενα και χρώματα.



Εικόνα 6 : Η οπτική οδός

Κάθε ένας από τους δύο οφθαλμούς έχει τη δυνατότητα παρατήρησης αντικειμένων που βρίσκονται σε ένα αρκετά μεγάλο οπτικό πεδίο (βλ. Εικόνα 6).

Ο συνδυασμός και των δύο οφθαλμών του οπτικού συστήματος, μας παρέχει μια καλή αντίληψη του βάθους και βοηθάει στην ανάπλαση της τρισδιάστατης απεικόνισης. Βέβαια, ακόμα και στην περίπτωση της μονό-οφθαλμικής παρατήρησης, η όραση είναι επαρκής για τις περισσότερες από τις ανθρώπινες ανάγκες.



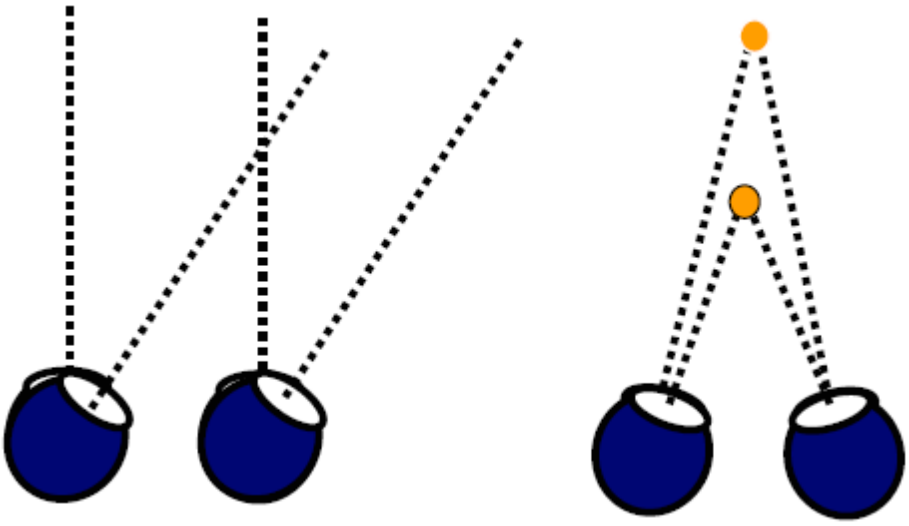
Εικόνα 7 : Εύρος οπτικής γωνίας

2.2 Τύποι οφθαλμικών κινήσεων

Οι οφθαλμικές κινήσεις μπορεί να είναι συζυγείς (conjugate) και μη συζυγείς (non − conjugate) Εικόνα 8 .

Κατά τις συζυγείς κινήσεις οι δύο οφθαλμοί κινούνται στην ίδια κατεύθυνση και περίπου κατά το ίδιο πλάτος και την ίδια γωνιακή ταχύτητα, όπως π.χ. κατά την διάρκεια μιας σακκαδικής κίνησης προς τα αριστερά όπου ο δεξιός οφθαλμός κινείται ρινικά και ο αριστερός κροταφικά.

Κατά τις μη συζυγείς κινήσεις οι δύο οφθαλμοί κινούνται σε αντίθετες κατευθύνσεις όπως συμβαίνει στην σύγκλιση των οφθαλμών κατά την προσαρμογή



Συζυγής κίνηση Μη-Συζυγής κίνηση

Εικόνα 8 : Οφθαλμικές κινήσεις

2.2.1 Σύστημα παρατήρησης (gaze system)

Η διαδικασία της προσήλωσης σε ένα αντικείμενο ελέγχεται από ένα συνδυασμό τριών υποσυστημάτων οφθαλμοκινητικού ελέγχου τα οποία συνθέτουν ένα ολοκληρωμένο σύστημα παρατήρησης (gaze system).

Οι βασικές λειτουργίες του είναι να συλλαμβάνει τις εικόνες και να τις κρατάει σταθερές στην περιοχή του κεντρικού βοθρίου, να εμποδίζει δηλαδή την απομάκρυνση τους από εκεί και να σταθεροποιεί την εικόνα ακόμα και όταν τα αντικείμενα κινούνται ή όταν το κεφάλι κινείται.

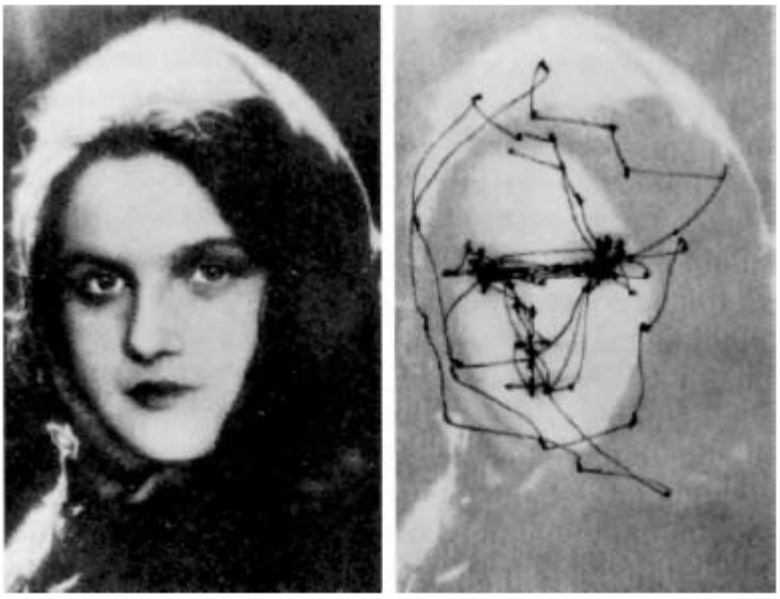
Τα τρία υποσυστήματα οφθαλμοκινητικού ελέγχου είναι το οφθαλμοκινητικό σύστημα (oculomotor system), το σύστημα σταθεροποίησης (fixation system) και το σύστημα κίνησης του κεφαλιού (headmovement system).

2.2.2 Το οφθαλμοκινητικό σύστημα

Το οφθαλμοκινητικό σύστημα υποστηρίζει την παρατήρηση ακίνητων και μη εικόνων. Είναι υπεύθυνο για την ύπαρξη τριών ειδών κινήσεων. Τις σακκαδικές, τις ομαλές κινήσεις παρακολούθησης (smooth pursuit movements) και τις κινήσεις σύγκλισης − απόκλισης (vergence movements).

2.2.3 Σακκαδικές κινήσεις (saccades)

Είναι εκούσιες κινήσεις των ματιών (πάνω από 100.000 ημερησίως) με τις οποίες «σαρώνουμε» μία εικόνα (Εικόνα). Είναι οι πιο γρήγορες κινήσεις από όλες τις κινήσεις του ανθρωπίνου σώματος (~700ο/sec) και είναι πολύ σύντομες (~ 50 msec)3. Το πλάτος και η κατεύθυνσή τους είναι εκούσια (1-20ο) ενώ η ταχύτητά τους είναι ακούσια και καθορίζεται από την εκκεντρότητα του στόχου. Επίσης η εκκεντρότητα και η προβλεψιμότητα του στόχου καθορίζει την καθυστέρησή τους. Τέλος, υφίστανται και χωρίς την απαραίτητη ύπαρξη στόχου ακόμα δηλαδή στο σκοτάδι. Οι σακκαδικές κινήσεις επηρεάζονται από τα χαρακτηριστικά του στόχου και είναι σχεδόν όμοιες στους δύο οφθαλμούς

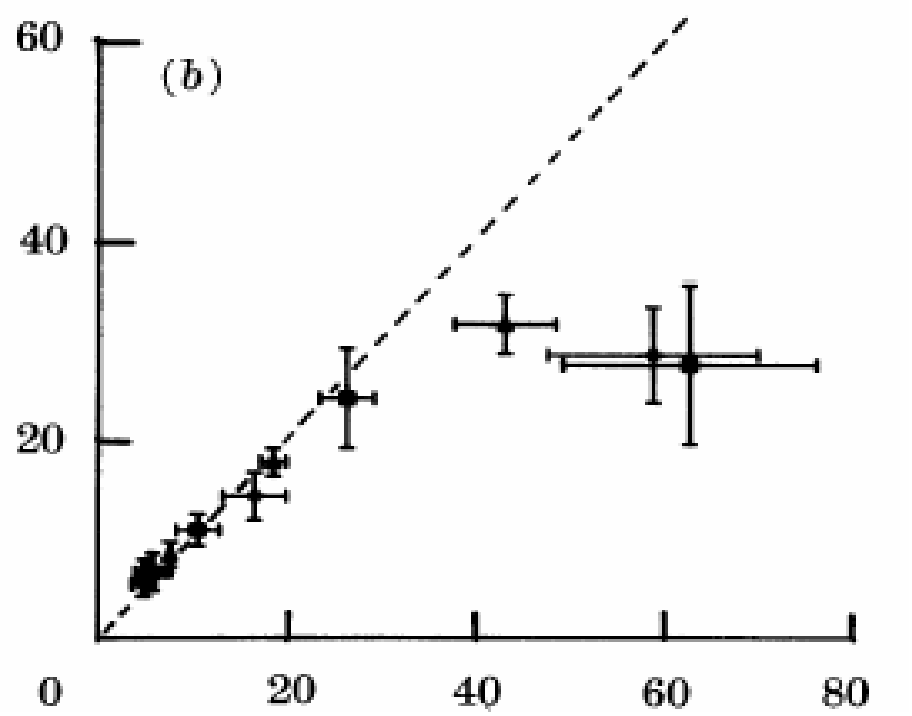


Εικόνα 9 : παράδειγμα σακκαδικών κινήσεων
Η τροχιά των σακκαδικών κινήσεων κατά την διάρκεια «σάρωσης» μιας εικόνας. Οι μαύρες κουκίδες ανάμεσα σε δυο σακκαδικές κινήσεις αντιπροσωπεύουν προσωρινά σημεία σταθεροποίησης.

2.2.4 Κινήσεις σύγκλισης _ απόκλισης (vergence movements)

Είναι «ασύζευκτες» οφθαλμικές κινήσεις κατά τις οποίες τα μάτια είτε συγκλίνουν (convergence), κατά την διάρκεια της προσαρμογής, είτε απομακρύνονται (divergence) το ένα από το άλλο. Λαμβάνουν μέρος όταν ένα αντικείμενο πλησιάζει ή απομακρύνεται αντίστοιχα. Καθοδηγούνται από τον βαθμό ανομοιότητας του αμφιβληστροειδικού ειδώλου (το είδωλο του αντικειμένου προβάλλεται σε διαφορετικές περιοχές του αμφιβληστροειδή σε κάθε μάτι (retinal disparity).

Η σχετική αυτή κίνηση των ματιών αποτελεί μια πολύπλοκη διαδικασία
η οποία απαιτεί ανώτερη επεξεργασία στον φλοιό του εγκεφάλου κάτι το οποίο συνεπάγεται καθυστέρηση στην απόκριση της κίνησης (Εικόνα). Η μέγιστη ταχύτητά τους (περίπου 100ο/sec) είναι πολύ μικρότερη από αυτή των σακκαδικών κινήσεων, ενώ η ταχύτητα απόκρισης τους έχει να κάνει με το κατά πόσο είναι προβλέψιμη ή όχι η κατεύθυνση του κινούμενου στόχου-αντικειμένου.

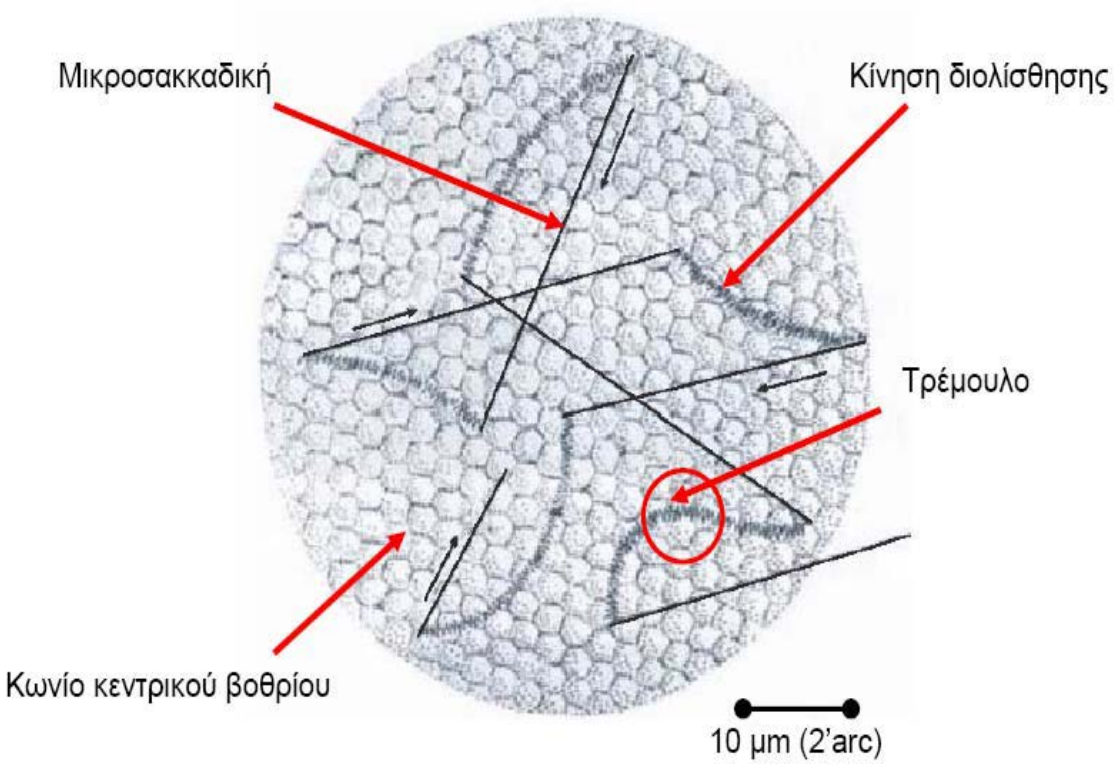


Εικόνα 10 :
Απόκριση σύγκλισης (ocular vergence) συναρτήσει ταχύτητας στόχου (target vergence).
Για μεγάλες ταχύτητες στόχου γίνεται φανερή η καθυστέρηση στην απόκριση της κίνησης
Ocular vergence (deg/sec) Target vergence (deg/sec)

2.2.5 Το σύστημα σταθεροποίησης

Το σύστημα σταθεροποίησης προσδιορίζεται από τις οφθαλμικές κινήσεις που λαμβάνουν μέρος κατά την διάρκεια που προσηλώνουμε σε ένα σταθερό σημείο. Είναι υπεύθυνο για την ύπαρξη τριών ειδών κινήσεων. Τις τρέμουλο κινήσεις (tremors) τις κινήσεις διολίσθησης (drifts) και τις μικροσακκαδικές κινήσεις (microsaccades).

- Το τρέμουλο (tremors) ή διαφορετικά φυσιολογικός νυσταγμός είναι μια σειρά ταλαντευτικών κινήσεων που κρατούν τους νευρώνες σε συνεχή λειτουργία. Πρόκειται για απεριοδικές κυματοειδείς κινήσεις με συχνότητα περίπου 90Hz. Είναι οι μικρότερες οφθαλμικές κινήσεις, με πλάτος που είναι περίπου ίσο με τη διάμετρο του κωνίου στην περιοχή του βοθρίου. Η καταγραφή του τρέμουλου είναι δύσκολη, ενώ η συγκεκριμένη κίνηση διαφέρει για τα δύο μάτια.
- Οι κινήσεις διολίσθησης (drifts) είναι καμπυλοειδείς κινήσεις που πραγματοποιούνται ταυτόχρονα με το τρέμουλο και μεταξύ των μικροσακκαδικών κινήσεων. Με την πραγματοποίηση των κινήσεων διολίσθησης, το είδωλο του αντικειμένου κινείται δια μέσου των φωτοϋποδοχέων. Οι κινήσεις διολίσθησης έχουν καταγραφεί και ως συζυγείς και ως μη-συζυγείς κινήσεις.
- Οι μικροσακκαδικές κινήσεις (microsaccades) είναι απότομες κινήσεις που εμφανίζονται κατά τη διάρκεια της προσήλωσης και μεταφέρουν το είδωλο στο βοθρίο μετά την απομάκρυνση του λόγω των κινήσεων διολίσθησης.

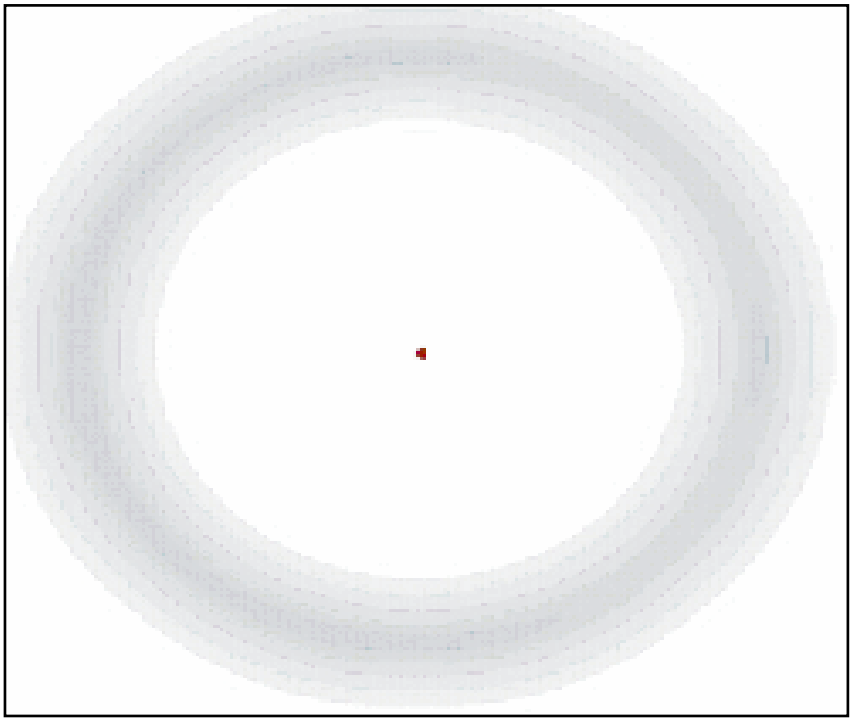


Εικόνα 11:
Οι οφθαλμικές κινήσεις προσήλωσης σε σχέση με το μέγεθος των κωνίων στον κέντρο του αμφιβληστροειδή

2.2.6 Ο ρόλος των οφθαλμικών κινήσεων προσήλωσης

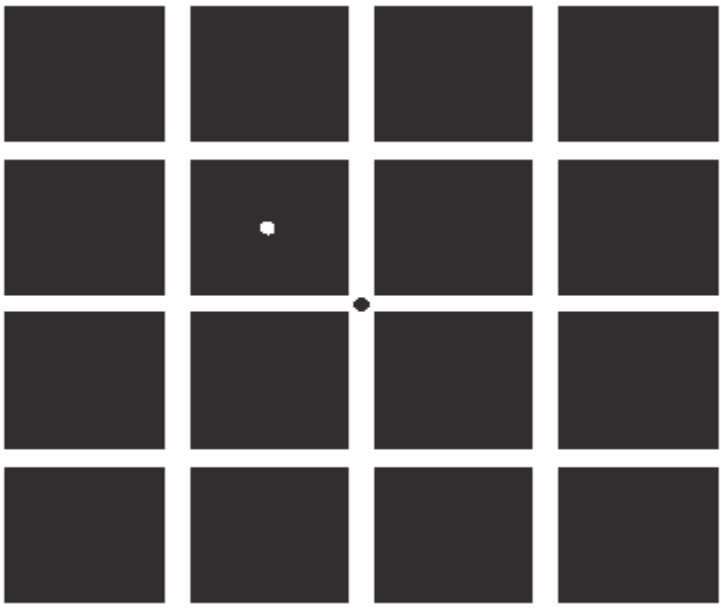
Ο ρόλος των οφθαλμικών κινήσεων κατά την διάρκεια που προσηλώνουμε σε ένα στόχο (αλλά και κατά την διάρκεια που ανιχνεύουμε το οπτικό μας περιβάλλον) είναι πολύ σημαντικός για την λειτουργικότητα της όρασης. Μέσω των οφθαλμικών κινήσεων τα είδωλα των αντικειμένων κατευθύνονται στην κεντρική περιοχή του αμφιβληστροειδή, στην οποία υπάρχει και μεγαλύτερη πυκνότητα κωνίων, από όπου ξεκινάει μία πολύπλοκη διαδικασία επεξεργασίας της πληροφορίας σε ανώτερες εγκεφαλικές οδούς.

Λόγω του ότι το περιβάλλον μας είναι συνεχώς μεταβαλλόμενο το νευρικό μας σύστημα έχει εξελιχθεί έτσι ώστε να μπορούμε να ανιχνεύουμε «ξαφνικές» αλλαγές. Στάσιμες εικόνες προκαλούν «νευρωνική προσαρμογή» (αδρανοποίηση των οπτικών νευρώνων) με αποτέλεσμα την αποτυχία της οπτικής μας αντίληψης .



Εικόνα 12 : Troxler’s effect

Προσηλώνοντας στην κόκκινη κηλίδα για λίγα δευτερόλεπτα ο κυκλικός δίσκος που την περιβάλλει εξαφανίζεται σταδιακά εξαιτίας της περιφερικής νευρωνικής προσαρμογής. Η απλούστερη εξήγηση αυτής της αποτυχίας της οπτικής μας αντίληψης είναι ότι το μέγεθος των μικροκινήσεων προσήλωσης είναι μικρότερο από το μέγεθος των υποδεκτικών πεδίων της περιφέρειας του αμφιβληστροειδή με αποτέλεσμα να μην είναι ικανές να αποτρέψουν την περιφερική οπτική εξαφάνιση ειδικά για ερέθισμα χαμηλής φωτεινής αντίθεσης. Κίνηση των ματιών κάνει άμεσα τον κυκλικό δίσκο οπτικά αντιληπτό.
Στην πραγματικότητα η νευρωνική προσαρμογή αποφεύγεται λόγω του ότι η εικόνα που προβάλλεται στον αμφιβληστροειδή δεν είναι ποτέ σταθερή εξαιτίας της συνεχούς ακούσιας και εκούσιας κίνησης των ματιών μας ακόμα και όταν προσηλώνουμε το βλέμμα μας σε ένα ακίνητο στόχο.



Εικόνα 13 : 1961 Taylor & Francis
Τα μάτια μας είναι σε συνεχή κίνηση. Εστιάζοντας στην μαύρη κηλίδα για περίπου ένα λεπτό και αμέσως μετά στην άσπρη, γίνονται αντιληπτές οι οφθαλμικές κινήσεις προσήλωσης

Κεφάλαιο 3 Συστήματα Εντοπισμού οφθαλμικών κινήσεων

3.1 Αρχές ιδανικού συστήματος εντοπισμού οφθαλμικής κίνησης

Αυτόματες καταγραφές της κατεύθυνσης του βλέμματος ξεκίνησαν ήδη από το 1936. Ο βασικός στόχος που επιτεύχθηκε με τις πρώτες τεχνικές καταγραφής ήταν απλά η συνεισφορά στην έρευνα για την συλλογή δεδομένων της συμπεριφοράς του οφθαλμοκινητικού συστήματος του ανθρώπου. Η συνεχής εξέλιξη της τεχνολογίας οδήγησε στην ανάπτυξη νέων εφαρμογών, ιδανικών για πιο ακριβή καταγραφή των οφθαλμικών κινήσεων.

Ο εντοπισμός των κινήσεων οφθαλμού (eye tracking) είναι η διαδικασία της μέτρησης και της καταγραφής είτε της κίνησης των ματιών σε σχέση με το κεφάλι ενός παρατηρητή είτε του σημείου του βλέμματος του (gaze point) πάνω σε μια οπτική σκηνή (stimulus window). Η οπτική σκηνή μπορεί να είναι η εικόνα που προβάλλεται σε μια οθόνη
υπολογιστή, σε μια τηλεόραση ή να αποτελεί το φυσικό χώρο.

Ένα σύστημα εντοπισμού οφθαλμού (eye tracker) είναι η συσκευή που
μετράει την θέση του οφθαλμού καθώς και τις οφθαλμικές κινήσεις πάνω σε μια περιοχή ενδιαφέροντος. Τα συστήματα εντοπισμού του οφθαλμού (eye tracking) και της κατεύθυνσης παρατήρησης (direction of eye-gaze) μέχρι σήμερα ποικίλουν και παρουσιάζουν σημαντικές διαφορές μεταξύ τους.

Σύμφωνα με τους Scott και Findley (1993), ένα ιδανικό σύστημα εντοπισμού και καταγραφής της οφθαλμικής κίνησης θα πρέπει να ικανοποιεί τις παρακάτω προϋποθέσεις:

- Να μην παρεμποδίζει το οπτικό πεδίο του χρήστη.
- Να μην έρχεται σε επαφή με τον χρήστη ή στην περίπτωση όπου εφαρμόζει πάνω του να μην τον παρεμποδίζει στην παρατήρηση
- Να έχει ακρίβεια τουλάχιστον 1% ή μερικών λεπτών της μοίρας,(η ακρίβεια επηρεάζεται από την επίδραση της μη γραμμικότητας , του θορύβου, της παραμόρφωσης και άλλων πηγών σφαλμάτων
- Να έχει την διακριτική ικανότητα 1 λεπτού της μοίρας, ώστε να ανιχνεύει ακόμα και την μικρότερη δυνατή κίνηση του οφθαλμού.
- Να έχει τη δυνατότητα καταγραφής κινήσεων από 1’ έως 45ο με ταχύτητες από 1 arcmin/sec μέχρι 800ο/sec
- Να διαθέτει συχνότητα δειγματοληψίας πάνω από 100Hz
- Να έχει τη δυνατότητα απόκρισης σε πραγματικό χρόνο
- Να έχει τη δυνατότητα καταγραφής οποιασδήποτε κίνησης του οφθαλμού οριζόντια, κάθετη, κυκλοστροφική.
- Να έχει τη διόφθαλμης καταγραφής
- Να μπορεί να καταγράφει κινήσεις κεφαλιού και σώματος
- Να μπορεί να χρησιμοποιηθεί με ευκολία από το ευρύτερο κοινό.

Αν και οι παραπάνω προϋποθέσεις θα παρείχαν ένα ιδανικό σύστημα καταγραφής οφθαλμικών κινήσεων το όποιο θα ήταν και το επιθυμητό δεν είναι και αναγκαίες για την αποδοχή μιας συσκευής ή διεπαφής. Άλλωστε με τα χρόνια το τι θεωρείται φυσιολογικό όσο αναφορά την χρήση τεχνολογικών συσκευών αλλάζει έτσι μελλοντικά ίσως δούμε στην καθημερινότητα την χρήση ειδικών γυαλιών προσαρμοσμένα με κάμερα είτε ακουστικών .

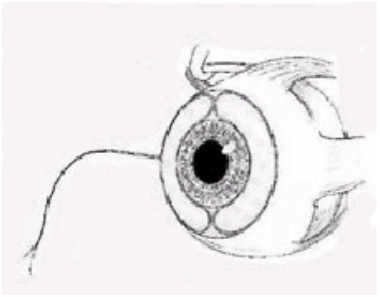
3.2 Τεχνικές εντοπισμού οφθαλμού

Οι σύγχρονες συσκευές εντοπισμού και καταγραφής της κίνησής του οφθαλμού με χρήση κριτήρια που αφορούν τον τρόπο επαφής με το χρήστη, το είδος των δεδομένων που συλλέγονται και τη μεθοδολογία ανάλυσής τους, διακρίνονται στις ακόλουθες τρεις βασικές τεχνικές:

- Τεχνικές που βασίζονται σε χρήση φακών επαφής.
- Τεχνική που βασίζεται στην καταγραφή του ηλεκτρικού δυναμικού του δέρματος / ElectroOculoGraphy.
- Τεχνικές που βασίζονται σε ανακλώμενες ακτίνες.

3.2.1 Τεχνικές που βασίζονται σε χρήση φακών επαφής

Οι οφθαλμικές κινήσεις είναι δυνατόν να καταγραφούν με ακρίβεια και με τη χρήση ειδικών φακών επαφής. Υπάρχουν δύο τεχνικές καταγραφής με χρήση φακών επαφής.
Με την πρώτη ένας ή περισσότεροι επίπεδοι καθρέφτες στηρίζονται πάνω στον φακό και οι ανακλάσεις από φωτεινές ακτίνες χρησιμοποιούνται για την καταγραφή της κίνησης των ματιών.
Με την δεύτερη τεχνική ένα μικροσκοπικό επαγωγικό πηνίο εμφυτεύεται μέσα στον φακό (scleral search coil) και η ακριβής θέση του μπορεί να καταγραφεί με την χρήση ηλεκτρομαγνητικού πεδίου μεγάλης συχνότητας το οποίο τοποθετείται γύρω από το κεφάλι του εξεταζομένου (Εικόνα .15). Αυτές οι δύο τεχνικές δεν συνίστανται για καθημερινή χρήση διότι αφενός δεν έχουν προσδιοριστεί οι επιπτώσεις στην υγεία του χρήστη που προκύπτουν από το ηλεκτρομαγνητικό πεδίο και αφετέρου είναι άβολη και κουραστική.



Εικόνα 14 : Εμφυτευμένο επαγωγικό πηνίο

3.2.2 Τεχνική που βασίζεται στην καταγραφή του ηλεκτρικού δυναμικού του δέρματος / ElectroOculoGraphy

Η τεχνική αυτή στηρίζεται στο γεγονός ότι ένα ηλεκτροστατικό πεδίο περιστρέφεται μαζί με το μάτι. Καταγράφοντας τις μικρές αλλαγές του ηλεκτρικού δυναμικού γύρω από το μάτι η θέση του μπορεί να ανιχνευθεί. Η τεχνική αυτή δεν απαιτεί καθαρό οπτικό πεδίο κάτι το οποίο συνεπάγεται καταγραφή κινήσεων σε ένα εύρος περίπου ±70ο. Το μειονέκτημα της είναι ότι είναι ενοχλητική για τον εξεταζόμενο αφού απαιτείται άμεση επαφή ηλεκτροδίων γύρω από το μάτι του (Εικόνα 14). Οι οφθαλμικές κινήσεις καταγράφονται από το βιοηλεκτρικό δυναμικό μεταξύ δύο ηλεκτροδίων που τοποθετούνται δερματικά σε κάθε πλευρά του άξονα κερατοειδή − αμφιβληστροειδή.
Περιορίζεται σε καταγραφή καθέτων και οριζοντίων κινήσεων αλλά επιτρέπει την διόφθαλμη καταγραφή.

Εικόνα 15 : ElectroOculoGraphy

3.2.3 Τεχνικές που βασίζονται σε ανακλώμενες ακτίνες

1. Καταγραφή της κίνησης του σκληροκερατοειδικού ορίου (limbus tracking)

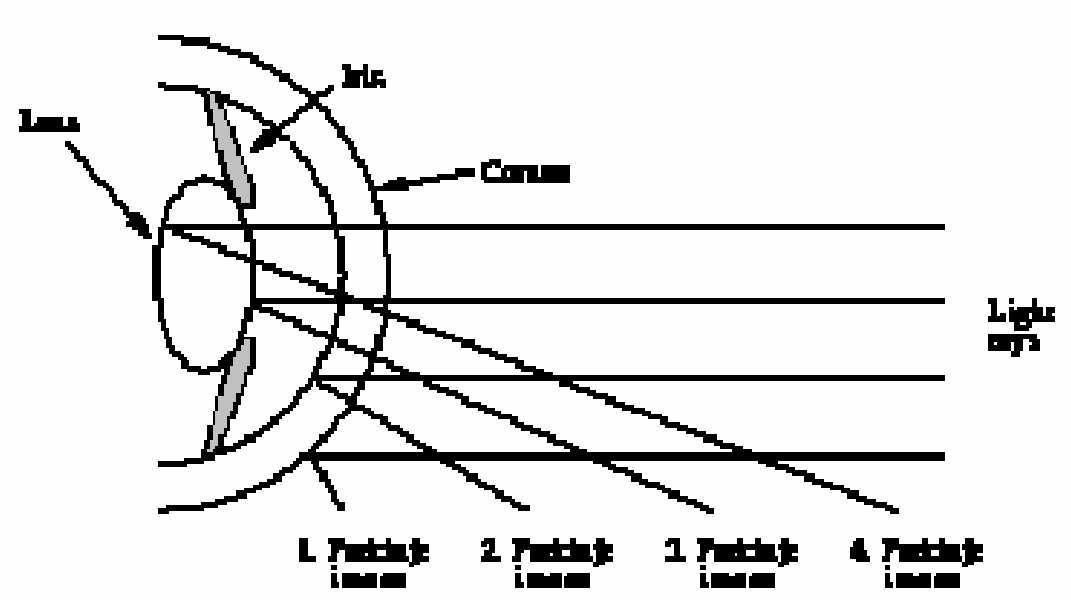
Εξαιτίας του γεγονότος ότι το σκληροκερατοειδικό όριο (limbus) είναι οπτικά ανιχνεύσιμο (ο σκληρός χιτώνας είναι άσπρος ενώ η ίριδα σκουρόχρωμη) η κίνηση του μπορεί εύκολα να καταγραφεί. Λόγω του ότι αυτή η τεχνική βασίζεται στη καταγραφή της θέσης του limbus σε σχέση με το κεφάλι πρέπει ή το κεφάλι να είναι κάπου σταθεροποιημένο ή ο μηχανισμός καταγραφής να είναι σταθεροποιημένος στο κεφάλι του εξεταζομένου. Επίσης εξαιτίας της μερικής κάλυψης του limbus από τα βλέφαρα αυτή η τεχνική είναι κατάλληλη για καταγραφή οριζοντίων κινήσεων μόνο.

2. Καταγραφή της κίνησης του ορίου ίριδας − κόρης (pupil tracking)

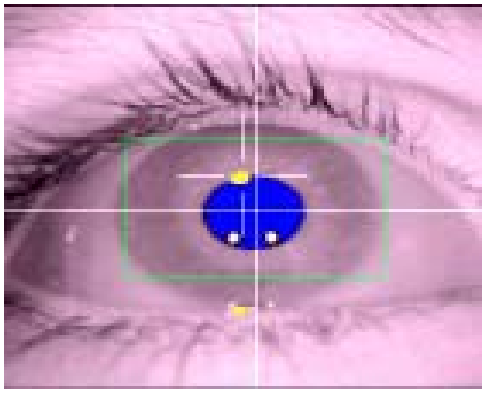
Η τεχνική αυτή είναι παρόμοια με την προηγούμενη μόνο που εδώ χρησιμοποιείται το όριο μεταξύ κόρης και ίριδας. Και σε αυτή την τεχνική ο μηχανισμός καταγραφής πρέπει να σταθεροποιείται στο κεφάλι του χρήστη. Τα πλεονεκτήματα σε σχέση με την προηγούμενη τεχνική είναι ότι υπάρχει δυνατότητα καταγραφής και των καθέτων κινήσεων (δεν υπάρχει επικάλυψη του ορίου από τα βλέφαρα) και ότι υπάρχει καλύτερη διακριτική ικανότητα λόγω του γεγονότος ότι το όριο ίριδας − κόρης είναι πιο «αιχμηρό». Το μειονέκτημα σε σχέση με την προηγούμενη τεχνική είναι η δυσκολία ανίχνευσης του ορίου λόγω του μικρότερου contrast του.

3. Καταγραφή της σχετικής θέσης κερατοειδή − κόρης / VideoOculoGraphy

Όταν υπέρυθρη ακτινοβολία προσπίπτει στο μάτι του χρήστη λαμβάνουν χώρα πολλαπλές ανακλάσεις, οι λεγόμενες εικόνες Purkinje (βλ. Εικόνα.16). Η πρώτη εικόνα Purkinje μαζί με την ανάκλαση από το σκληροκερατοειδικό όριο μπορούν να βιντεοσκοπηθούν χρησιμοποιώντας μια υπέρυθρη κάμερα ως ένα φωτεινό σημείο και ένα λιγότερο φωτεινό αντίστοιχα. Όταν το μάτι κινείται οριζόντια ή κάθετα η σχετική θέση των δύο ανακλάσεων αλλάζουν ανάλογα και η θέση του ματιού μπορεί να υπολογιστεί από αυτές τις σχετικές θέσεις. Το μειονέκτημα αυτής της μεθόδου είναι ότι πλευρικές μετατοπίσεις του κεφαλιού μπορεί να τοποθετήσουν την εικόνα του ματιού εκτός εστίας ή ακόμα και εκτός του οπτικού πεδίου της υπέρυθρης κάμερας (βλ. Εικόνα 17).



Εικόνα 16 : Purkinje



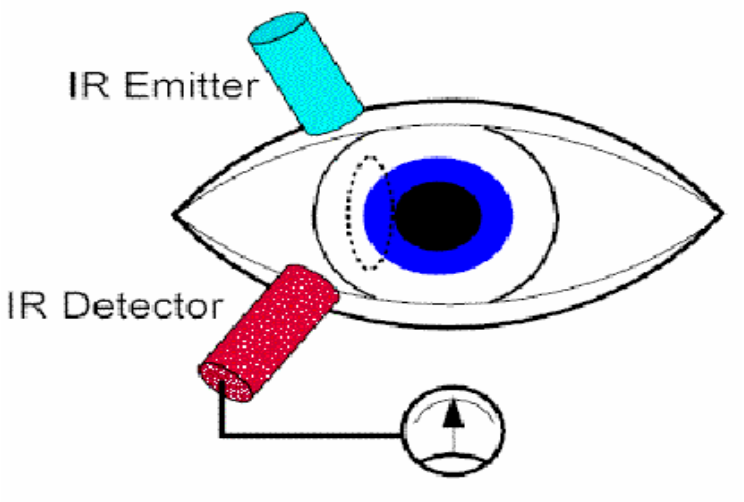
Εικόνα 17 : Απεικόνιση VOG

Μια άλλη τεχνική που χρησιμοποιεί τις εικόνες Purkinje είναι η Dual − Purkinje κατά την οποία χρησιμοποιούνται η πρώτη (ανάκλαση από την πρόσθια επιφάνεια του κερατοειδή) και η τέταρτη (ανάκλαση από την οπίσθια επιφάνεια του φακού) εικόνα. Η τεχνική αυτή είναι πιο ακριβής από τις άλλες και προσφέρει συχνότητα δειγματοληψίας μεγαλύτερη από 4000Hz. Το μειονέκτημα αυτής της τεχνικής είναι ότι η τέταρτη εικόνα Purkinje είναι σχετικά ασθενής οπότε ο περιβάλλοντας φωτισμός πρέπει να είναι ελεγχόμενος.

4. Καταγραφή με χρήση πομπού − δέκτη υπερύθρων / InfraRed OculoGraphy

Εάν μια σταθεροποιημένη πηγή φωτός κατευθύνεται προς το μάτι του εξεταζομένου το ποσό της ακτινοβολίας που ανακλάται και συλλέγεται από τον δέκτη εξαρτάται από τη θέση του ματιού (βλ. Εικονα.18). Αυτή την αρχή εκμεταλλεύονται πολλά εμπορικά διαθέσιμα συστήματα καταγραφής οφθαλμικών κινήσεων (eye trackers).

Η ακτινοβολία που χρησιμοποιείται είναι υπέρυθρη, καθώς είναι «αόρατη» για το ανθρώπινο μάτι, και έτσι έχει το πλεονέκτημα ότι δεν επηρεάζει τον εξεταζόμενο. Εφόσον οι ανιχνευτές υπερύθρων δεν επηρεάζονται από άλλες εκτεταμένες πηγές φωτός, ο περιβάλλον φωτισμός δεν επηρεάζει τις μετρήσεις. Η χωρική διακριτική ικανότητα (το μέγεθος της μικρότερης κίνησης που μπορεί να ανιχνευθεί) είναι καλή, της τάξης των 0.1ο και χρονική διακριτική ικανότητα της τάξης του 1ms μπορεί να επιτευχθεί.



Εικόνα 18 : Αρχή τεχνικής IROG

Εικόνα 19:
Σύστημα καταγραφής της οφθαλμικής κίνησης, προσαρμοσμένο σε ομοίωμα κεφαλής.

Κεφάλαιο 4. Ανάπτυξη συστημάτων εντοπισμού- παρακολούθησης οφθαλμού.

4.1 Μοντέλο διπλής κατάστασης ματιού.

Στον τομέα της αναγνώρισης νεύματος, έχουν χρησιμοποιηθεί μοντέλα που βασίζονται στην κατάσταση . Ένα νεύμα ορίζεται συχνά ως μια ακολουθία από καταστάσεις σε ένα χώρο μέτρησης ή διαμόρφωσης. Διάφορες μεταβάσεις μπορεί να προκύψουν μεταξύ των καταστάσεων αυτών. Η επαναληψιμότητα και η ποικιλομορφία των φάσεων στον χώρο καταστάσεων μπορεί να μετρηθεί με παραδείγματα εκπαίδευσης.

α) frame 1 β) frame 26

γ) frame 32 δ) frame 39

Εικόνα 20 :
Σύστημα εντοπισμού οφθαλμού χρησιμοποιώντας το χαρακτηριστικό σημείο αναγνώρισης στην κατάσταση μάτια κλειστά. Καθώς τα μάτια κλείνουν το χαρακτηριστικό σημείο αναγνώρισης εξαφανίζεται και έτσι το σύστημα αποτυγχάνει.
Αφού τα μάτια ανοίξουν η ίριδα στο αριστερό μάτι και ο κερατοειδής στο δεξί μάτι έχουν χαθεί.

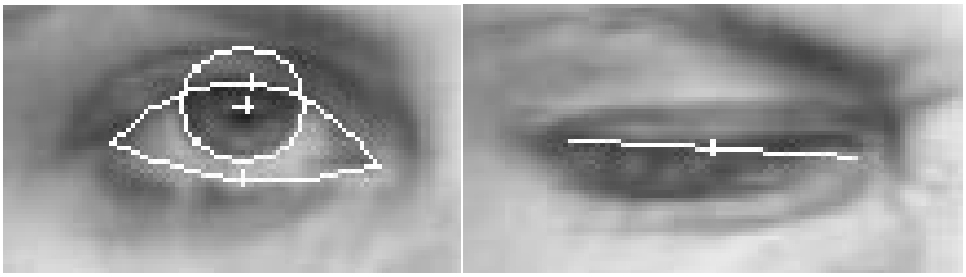
Θα δούμε μια ισχυρή και ακριβής μέθοδο παρακολούθησης των θέσεων των ματιών, εντοπισμού των καταστάσεων των ματιών, και εκτίμησης των παραμέτρων των ματιών για κάθε καρέ σε μια ακολουθία.

Σκοπός είναι να αναπτυχθεί ένα σύστημα εντοπισμού παρακολούθησης οφθαλμού που είναι ανθεκτικό σε τυχόν ανοιγόκλειμα του ματιού, το οποίο ανακτά με ακρίβεια τις παραμέτρους του ματιού.

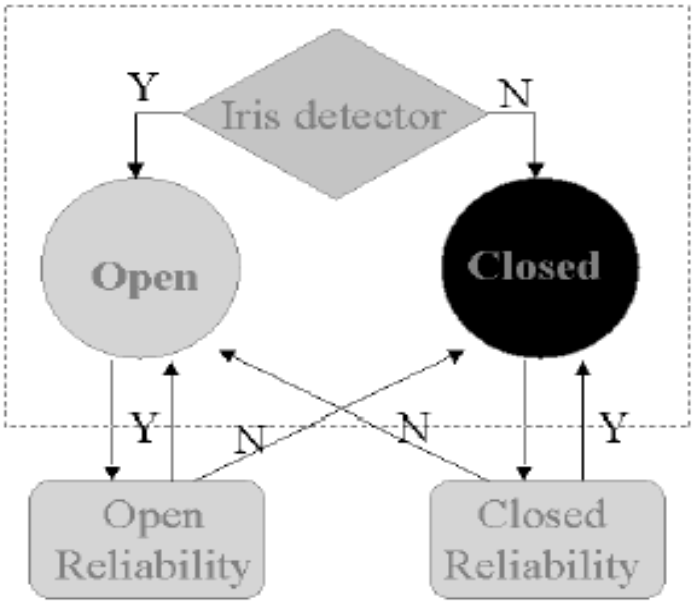
Αρκετές διαφορετικές τεχνικές παρακολούθησης χρησιμοποιούνται στο σύστημα όπως η feature point tracking, και η masked edge filtering. Ένα μοντέλο ματιού διπλής κατάστασης χρησιμοποιείται για να αναγνωρίσει την τις διαφορετικές καταστάσεις του ματιού, μάτια ανοιχτά και μάτια κλειστά. Αφού αρχικοποιηθεί το πρότυπο του ματιού στο πρώτο καρέ, η εσωτερική γωνία του ματιού μπορεί να παρακολουθείται με ακρίβεια χρησιμοποιώντας την τεχνική feature point tracking υποθέτουμε ότι οι εξωτερικές γωνίες των ματιών είναι στη γραμμή που συνδέει τις δύο εσωτερικές γωνίες των ματιών. Στη συνέχεια, οι εξωτερικές γωνίες μπορούν να ληφθούν από τις πληροφορίες σχήματος του ματιού που υπολογίζονται από το πρώτο καρέ. Η άκρη και η ένταση της ίριδας χρησιμοποιούνται για την ανίχνευση των καταστάσεων του ματιού.

Για ένα ανοικτό μάτι, τα βλέφαρα πρέπει να παρακολουθούνται από το feature point tracking. Για ένα κλειστό μάτι, τα περιγράμματα των βλέφαρων δεν χρειάζεται να παρακολουθούνται. Η συγκεκριμένη μέθοδος παρακολούθησης οφθαλμού έχει δοκιμαστεί με 500 ακολουθίες εικόνων, και λειτουργεί καλά για τις 493 ακολουθίες. Τα πειραματικά αποτελέσματα δείχνουν ότι η μέθοδος λειτουργεί καλά και για τις δυο καταστάσεις των ματιών, και οι σωστές παράμετροι των ματιών ανακτώνται, ακόμη και μετά την κατάσταση μάτια κλειστά.

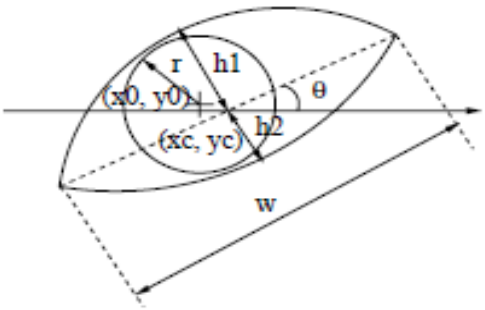
Η ίριδα μπορεί να μας παρέχει πολύτιμες πληροφορίες για την κατάσταση του ματιού διότι εάν το μάτι είναι κλειστό τότε η ίριδα δεν θα είναι ορατή εάν όμως το μάτι είναι ανοιχτό τουλάχιστον ένα μέρος της ίριδας θα είναι ορατό. Έτσι εάν η ίριδα εντοπιστεί τότε θα έχουμε την κατάσταση μάτι ανοιχτό ειδάλλως έχουμε την κατάσταση μάτι κλειστό. Για διαφορετικές καταστάσεις , θα χρησιμοποιηθούν διαφορετικά πρότυπα ματιού και συγκεκριμένοι αλγόριθμοι ώστε να παραχθούν τα χαρακτηριστικά του ματιού.



(a) (b)



(c)



(d) (e)

Εικόνα 21 : Μοντέλο ματιού διπλής κατάστασης
(a) Κατάσταση μάτι ανοιχτό , (b) Κατάσταση μάτι κλειστό , (c) Διάγραμμα μεταβατικής κατάστασης , (d) Το μοντέλο παραμέτρων σε κατάσταση ανοιχτό μάτι (e) Το μοντέλο παραμέτρων σε κατάσταση κλειστό μάτι .

Πρότυπο ματιού.

Για την κατάσταση όπου έχουμε και τα δυο μάτια ανοιχτά χρησιμοποιούμε τον ίδιο αλγόριθμο προσθέτοντας δυο σημεία τοποθετημένα στο κέντρο των λευκών περιοχών. Το πρότυπο αναπαριστάται στην Εικόνα 21 (d) , αποτελείται από ένα κύκλο με τρεις παραμέτρους (**xo,y**o,r) και δυο παραβολικά τόξα με έξι παραμέτρους (**xo,y**o,h1,h2,ω,θ).

Υποθέτουμε ότι το εξωτερικό περίγραμμα του ματιού είναι συμμετρικό σε σχέση με την κάθετη διχοτόμο προς τη γραμμή που συνδέει τις δυο γωνίες των ματιών. Για ένα κλειστό μάτι χρησιμοποιούμε μια ευθεία γραμμή για το πρότυπο. Έχει τέσσερις παραμέτρους ,δυο για κάθε τερματικά σημεία. Αυτό το πρότυπο αναπαριστάται στην Εικόνα 21 (e) , είναι επαρκής για την περιγραφή της κατάστασης μάτια κλειστά.

4.1.1 Αναγνώριση κατάστασης ματιού και παρακολούθηση ματιού.

4.1.1.1 Αρχικοποίηση θέσης ματιού

Υποθέτουμε ότι η αρχική θέση του ματιού δίνεται στο πρώτο καρέ. Ο σκοπός αυτής της δήλωσης είναι να πάρουμε την αρχική θέση του ματιού στο πρώτο καρέ από την ακολουθία των εικόνων.

4.1.1.2 Περιοχή εξομάλυνση της έντασης του ματιού

Για κάποια ακολουθία εικόνων η περιοχή του ματιού είναι ιδιαίτερα σκοτεινή λόγο ελλιπή φωτισμού ή κάποιου μακιγιάζ. Για αυτό το λόγο θα πρέπει να ισοσταθμιστεί η ένταση της ακολουθίας εικόνων. Αφού έχουν αρχικοποιηθεί οι θέσεις των ματιών , ένα σταθερού μήκους περίγραμμα μετριέται γύρω από την περιοχή των ματιών.
Οι εντάσεις στην περιοχή αυτή είναι γραμμικές εκτεινόμενες ώστε να γεμισθεί ένα φάσμα της τάξεως 0 255. Για μια έγχρωμη ακολουθία εικόνων, τα R, G, B κανάλια είναι εκτεινόμενα χωριστά.

4.1.2 Παρακολούθηση γωνίας ματιού.

4.1.2.1 Εσωτερικές γωνίες

Γνωρίζουμε ότι οι εσωτερικές γωνίες του ματιού είναι οι πιο σταθερές περιοχές στο πρόσωπο και σχετικά αδρανείς σε σχέση με τις εκφράσεις του προσώπου. Χρησιμοποιώντας ανιχνευτή γωνίας βασισμένο σε άκρα σημεία, οι εσωτερικές γωνίες μπορούν να ανιχνευθούν εύκολα. Ωστόσο, λόγω της χαμηλής αντίθεσης της έντασης στα όρια των ματιών και τις ρυτίδες γύρω από τα μάτια, μερικές ψευδείς γωνίες μπορεί να μαζί με τις πραγματικές γωνίες.

Αντί να χρησιμοποιηθεί η μέθοδος που ταιριάζει τις γωνίες, θα χρησιμοποιηθεί ένα χαρακτηριστικό σημείο μέθοδο παρακολούθησης για να παρακολουθηθούν οι εσωτερικές γωνίες των ματιών στις εναπομείναντες ακολουθίες εικόνων.

Σε αυτό το σύστημα οι εσωτερικές γωνίες των ματιών θα παρακολουθηθούν χρησιμοποιώντας μια εξελιγμένη μορφή του αλγορίθμου παρακολούθησης Lucas –Kanade. Υποθέτουμε ότι οι τιμές έντασης της κάθε περιοχής (σταθερού μήκους περίγραμμα) δεν αλλάζουν , αλλά απλώς μετατοπίζονται από την μια θέση στην άλλη.

Σκεφτείτε ένα πρότυπο χαρακτηριστικό έντασης **It(x))** πάνω από μια **xx**n περιοχή **R** στην εικόνα αναφοράς κατά το χρόνο **t**. Θέλουμε να βρούμε το **d** μετάφραση αυτής της περιοχής στο ακόλουθο καρέ **It+1(x+d)** τη χρονική στιγμή **t +1,** με την ελαχιστοποίηση μιας συνάρτησης η οποία ορίζεται ως εξής:

$$\mathrm{E} = \sum_{x \in R} [It + 1(x + d) - It(x)]^2 \qquad (1)$$

$$d_n + 1 = d_n + \left\{ \sum_{x \in R} \left(\frac{\partial I}{\partial x} \right)^T \Big| x + d_n \left[I_t(x) - I_t + 1(x) \right] \right\}$$

$$\left[\sum_{x \in R} \left(\frac{\partial I}{\partial x} \right) \left(\frac{\partial I}{\partial x} \right)^T \Big| x + d_n \right]^{-1} \qquad (2)$$

Όπου d0, είναι η αρχική εκτίμηση, μπορεί να θεωρηθεί ως μηδέν εάν μόνο πρόκειται για μικρές μετατοπίσεις.

Διαδοχικά καρέ μιας ακολουθίας εικόνων μπορεί να περιέχουν μεγάλο χαρακτηριστικό σημείο κίνησης, όπως απότομες κινήσεις της κεφαλής, γεγονός που μπορεί να προκαλέσει ελλιπή εντοπισμό ή αποτυχία εντοπισμού.

Για να παρακολουθήσει μεγάλες κινήσεις χωρίς να χάσει κάποιο sub pixel ακρίβειας, χρησιμοποιείται μια μέθοδος πυραμίδα με μειωμένη ανάλυση. Κάθε εικόνα αναλύεται σε 5 επίπεδα από το επίπεδο 0 (η αρχική καλύτερη ανάλυση εικόνας) στο επίπεδο 4 (η χονδροειδή ανάλυση εικόνας).

Στην εφαρμογή, χρησιμοποιείται ένα φίλτρο 5x5Gaussian ώστε να εξομαλύνει το θόρυβο, προκειμένου να ενισχυθεί ο υπολογισμός σύγκλισης, και μια χαρακτηριστική περιοχή 13x13 χρησιμοποιείται για όλα τα επίπεδα.

Γρήγορες και μεγάλου μετατοπίσεις μέχρι 100 pixel μπορούν να παρακολουθούνται, διατηρώντας παράλληλα την ευαισθησία των subpixel στη κίνηση του πρόσωπου

4.1.2.2 Εξωτερικές γωνίες

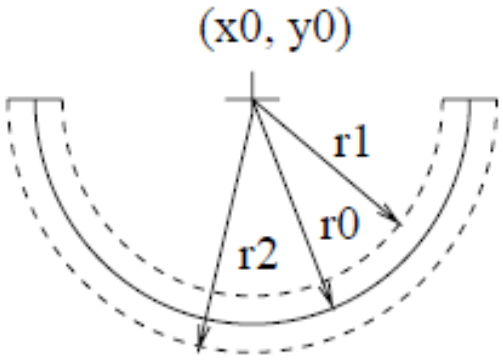
Γνωρίζουμε ότι οι εξωτερικές γωνίες του ματιού είναι δύσκολο να αναγνωριστούν και λιγότερο σταθερές από τις εσωτερικές. Υποθέτουμε ότι οι εξωτερικές γωνίες είναι συγγραμμικές με τις εσωτερικές γωνίες του ματιού. Τα χαρακτηριστικά του σχήματος του ματιού (πλάτος ματιού, απόσταση μεταξύ των εσωτερικών γωνιών) λαμβάνονται από το πρώτο καρέ. Αφού παρακολουθήσουμε τις εσωτερικές γωνίες σε κάθε καρέ , οι θέσεις των εξωτερικών γωνιών λαμβάνονται από τα χαρακτηριστικά του σχήματος του ματιού.

4.1.3 Παρακολούθηση ίριδας και αναγνώριση κατάστασης ματιού.

4.1.3.1 Ίριδα

Η ίριδα μπορεί να παρέχει σημαντικές πληροφορίες σχετικά με την κατάσταση του ματιού αφού εάν το μάτι είναι κλειστό η ίριδα δεν θα είναι ορατή ενώ αν το μάτι είναι ανοιχτό τότε η ίριδα θα είναι ορατή. Γενικά η κόρη είναι το πιο σκοτεινό σημείο του ματιού. Κάποια συστήματα εντοπισμού και παρακολούθησης οφθαλμού ,αναγνωρίζουν την κόρη ως το πιο σκοτεινό σημείο στην περιοχή του ματιού σε κάθε καρέ. Όμως κάποιες φορές αυτό το σημείο μετατοπίζεται στην άκρη του ματιού λόγο σκουρόχρωμων βλεφαρίδων ή αυξημένης σκίασης βλεφάρων. Εδώ βλέπουμε τη ένταση της ίριδας μαζί με χαρτογράφηση των άκρων ώστε να παρακολουθήσουμε το κέντρο της ίριδας.

Αναγνωρίζουμε τις άκρες μέσω της χαρτογράφησης των άκρων του ματιού. Η αναγνώριση των ακριών για διαφορετικές καταστάσεις των ματιών φαίνεται στην Εικόνα 24. Η χαρτογράφηση ακρών έχει αρκετό θόρυβο οπότε δεν την χρησιμοποιούμε κατευθείαν. Το άκρο της ίριδας είναι σχετικά καθαρό, και το ανώτερο μέρος της ίριδας είναι συχνά αποφραγμένο από το άνω βλέφαρο. Ως αποτέλεσμα, χρησιμοποιούμε μισό κύκλο για να φιλτράρουμε την ίριδα άκρη (Εικόνα 22). Η ακτίνα του πρότυπου κύκλου της ίριδας r0 μπορεί να ληφθεί από το πρώτο καρέ. Σε ακολουθίες εικόνων προσώπου, αν δεν υπάρχει μεγάλη κίνηση της κεφαλής εκτός πλάνου , τότε η ακτίνα της ίριδας δεν θα αλλάξει πολύ. Αυξάνεται και μειώνεται η ακτίνα του κύκλου (r) από r0 ώστε να δημιουργήσει το ημικύκλιο με ελάχιστη ακτίνα (r0 -δr) και μέγιστη ακτίνα (r0 + δr). Αν το πάχος της ίριδας (δr) είναι πολύ μεγάλο, πολλές άκρες που δεν αφορούν την ίριδα θα συμπεριληφθούν. Αν το (δr) είναι πολύ μικρό, η άκρη της ίριδας δεν θα συμπεριλήφθη αν υπάρχει κίνηση της κεφαλής.



Εικόνα 22 : Ημικύκλιο ίριδας, το σημείο (χ0,y0) είναι το κέντρο της ίριδας ,r0 είναι η ακτίνα της ίριδας , r1 είναι η ελάχιστη ακτίνα που χρησιμοποιείται , r2 είναι η μέγιστη ακτίνα που χρησιμοποιείται.

4.1.3.2 Παρακολούθηση ίριδας

Οι πληροφορίες έντασης και άκρων χρησιμοποιούνται για την ανίχνευση της ίριδας.
Αν η ίριδα έχει ανιχνευθεί, το μάτι είναι ανοικτό και το κέντρο της ίριδας θεωρείται το (x0,y0). Το κέντρο της ίριδας μπορεί να ληφθεί σε τέσσερα βήματα:

1. Υπολογίζεται η μέση ένταση του I0 στο κάτω μισό της ίριδας στο πρώτο καρέ.

2. Εξάγονται τα αποτελέσματα της χαρτογράφησης των ακρών στην περιοχή του ματιού και υπολογίζεται ο αριθμός των pixel ανήκουν σε άκρες E0 στην περιοχή μεταξύ r1 και r2 στο πρώτο καρέ.

3. Αναζήτηση της περιοχής του ματιού μεταξύ της εσωτερικής γωνίας και της εξωτερικής γωνίας ώστε να βρεθεί το κέντρο της ίριδας (x0,y0) με τον μεγαλύτερο αριθμό pixel ακρών E.

4. Υπολογισμός της μέσης έντασης I της ίριδας όταν είναι στην θέση με τις μεγαλύτερες άκρες. Εάν (I-Io)<T1 και E/Eo>T2 τότε η ίριδα έχει ανιχνευτεί με σημείο κέντρου (x0,y0). Όπου T1και T2 είναι τα κατώτατα όρια έντασης και ακρών αντίστοιχα.

Η κατάσταση των ματιών μπορεί να προσδιοριστεί από την εξίσωση:

$$Eyestate = \begin{cases} Open & \text{if the iris detected} \\ Closed & \text{otherwise} \end{cases}$$

(a) Original eye images. (b) Eye edge maps.

Εικόνα 23 : Χαρτογράφηση των ακρών του ματιού από την κατάσταση μάτια τελείως ανοιχτά έως το κλείσιμο. Η άκρες του κάτω μέρος της ίριδας είναι σχετικά σαφής για έναν ανοικτό μάτι.

4.1.3.3 Εντοπισμός ορίου ματιού.

Για τον εντοπισμό του ορίου ενός ανοιχτού ματιού , χρησιμοποιείται το πρότυπο ματιού ώστε να ληφθούν τα σωστά όρια αυτού. Αφού αναγνωριστεί η το πρότυπο ματιού στο πρώτο καρέ , δυο μόνο σημαντικά σημεία παρακολουθούνται στο πρότυπο ματιού (τα κεντρικά σημεία του ανώτερου μέρους και τα κατώτερα βλέφαρα) στις εναπομείναντες ακολουθίες εικόνων με μάτια ανοιχτά. Από αυτά τα δυο σημεία και από τις γωνίες των ματιών μπορούν να ληφθούν οι παράμετροι του πρότυπου ματιού. Τότε τα όρια του ματιού υπολογίζονται από τις αντίστοιχες παραμέτρους του πρότυπου ματιού.
Για ένα κλειστό μάτι χρησιμοποιείται μια απλή γραμμή από την εσωτερική στην εξωτερική γωνία ως όριο.

Εικόνα 24 : Εντοπισμός και παρακολούθηση οφθαλμού.

4.2 Σύστημα παρακολούθησης- εντοπισμού παρακολούθησης οφθαλμού με μια σταθερή κάμερα

Πολλά συστήματα eye tracking είτε απαιτούν από τον χρηστή να κρατήσει το κεφάλι του σταθερό είτε περιέχουν κάμερα ή άλλο εξοπλισμό τοποθετημένο σταθερό πάνω στο κεφάλι του χρήστη. Αν και αυτοί οι τρόποι είναι δεκτοί για εφαρμογές που αφορούν την έρευνα , οι περιορισμοί αυτοί κάνουν τα συστήματα μη ικανοποιητικά για παρατεταμένες διαδραστικές εφαρμογές. Μιας και ο σκοπός μας είναι να χρησιμοποιήσουμε μεθόδους καταγραφής οφθαλμού με προηγμένη οπτική επικοινωνία μέσο της καθοδήγησης της κόρης οφθαλμού και για λογούς Επαυξητικής και Εναλλακτικής Επικοινωνίας , μας ενδιαφέρουν λιγότερο επεμβατικές τεχνικές- μέθοδοι εντοπισμού οφθαλμού.

Τα τελευταία χρόνια έχουν παρουσιαστεί αρκετοί μέθοδοι « απομακρυσμένης» τεχνολογίας καταγραφής οφθαλμού. Αυτά τα συστήματα δεν απαιτούν κάποιο εξοπλισμό σταθεροποιημένο στον χρήστη και επιτρέπουν στον χρήστη να κινεί το κεφάλι του ελαφρά ,εντος φυσικά κάποιον ορίων.

Η πιο ακριβής μέθοδος απομακρυσμένης τεχνολογίας καταγραφής οφθαλμού που έχει περιγραφεί χρησιμοποιεί πολλαπλές κάμερες και επιτυγχάνει ακρίβεια από 0,5 έως 1,0 βαθμούς. Χρειαζόμαστε αυτήν την ακρίβεια για την εφαρμογή μας αλλά θα έμοιαζε ιδανικό να επιτευχτεί χρησιμοποιώντας μια wide-fieldof- view κάμερα για λόγους κόστους και περιπλοκότητας.

Σε αυτό το σύστημα θα χρησιμοποιηθεί μια κάμερα με απομακρυσμένη τεχνολογία καταγραφής οφθαλμού η οποία έχει σχεδιαστεί ώστε να επιτυγχάνει ακρίβεια από 0,5 έως 1,0.

Το υλικό μέρος του συστήματος καταγραφής οφθαλμού αποτελείται από τα εξής:

1. μια κάμερα υψηλής ευκρίνειας (1280x1024 pixels)
2. δύο υπέρυθρες διόδους εκπομπής φωτός (LED) σε κάθε πλευρά της κάμερας που θα φωτίζουν το πρόσωπο και θα δημιουργήσουν αντανακλαστικά του κερατοειδούς (CRS) στην επιφάνεια του κερατοειδή, και
3. μια οθόνη τοποθετημένη πάνω από τη κάμερα. Και τα LEDS (εικόνα 25)

Αυτά τα δύο στοιχεία έχουν αναπτυχθεί ανεξάρτητα και έχουν πρόσφατα ενσωματωθεί σε ένα λειτουργικό σύστημα. Μετρήσεις ακρίβειας δεν έχουν γίνει ακόμη για το ολοκληρωμένο σύστημα, αλλά κάποιες δοκιμές.

Εικόνα 25 : Τα μέρη του συστήματος, η κάμερα και τα δυο υπέρυθρα.

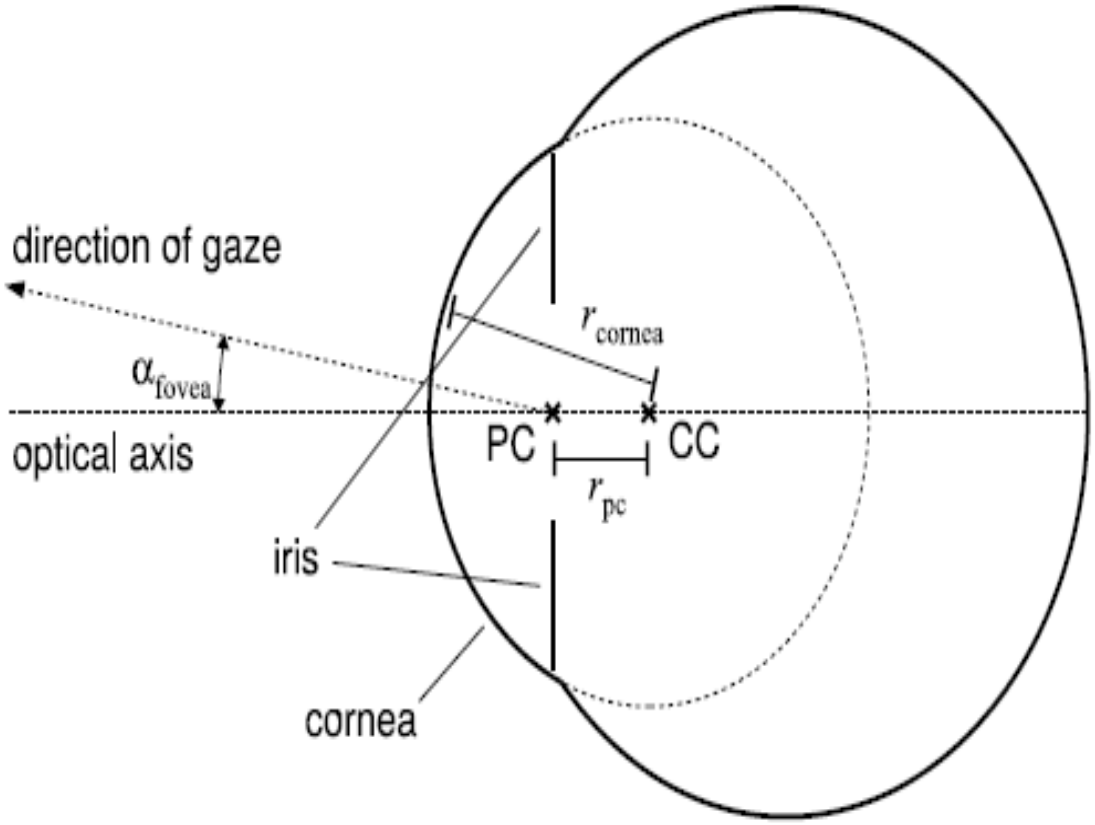
4.2.1 Αλγόριθμοι

Μπορούμε να δώσουμε μια σύντομη περιγραφή της επεξεργασίας εικόνας και των αλγόριθμων εκτίμησης της κόρης. Ο αλγόριθμος που χρησιμοποιείται για να εξαγάγει τις θέσεις της κόρης και των αντανακλαστικών του κερατοειδή από την εικόνα της κάμερας βασίζεται στον αλγόριθμο Starburst , ο οποίος τροποποιήθηκε ώστε να ταιριάξει με τις ανάγκες απομακρυσμένης τεχνολογίας καταγραφής οφθαλμού. Μια εφαρμογή ανοιχτού κώδικα αυτού του αλγορίθμου είναι διαθέσιμη υπό την επωνυμία "openEyes", αλλά εμείς επιλέξαμε να ξανά υλοποιήσουμε τον αλγόριθμο ώστε να έχουμε καλύτερη ενσωμάτωση στο πλαίσιο του υπάρχοντος computer visiom και του eye tracking framework.

Με τον αλγόριθμο επιτυγχάνεται εξαγωγή των θέσεων της κόρη και των αντανακλαστικών του κερατοειδή (CRs) ως εξής: Οι θέσεις των (CRs) εξάγονται εφαρμόζοντας διάφορα gausssian και ψάχνοντας για το μεγαλύτερο, το κατά προσέγγιση κέντρο της κόρης ορίζεται ως το πιο σκοτεινό pixel στην περιοχή των (CRs) , περιφερειακά σημεία προσδιορίζονται σε ακτίνες από το κέντρο της κόρης όπως και δευτερεύουσες ακτίνες από τα βασικά περιφερειακά σημεία.

Η εικόνα 26 δείχνει το μοντέλο του ματιού , το όποιο περιγραφεί τα ακόλουθα στοιχειά του ματιού :

Η επιφάνεια του κερατοειδή. Αυτή περιγράφεται ως μια σφαιρική επιφάνεια με ένα κέντρο καμπυλότητας CC και μια ακτίνα καμπυλότητας rcornea.. Η επιφάνεια του κερατοειδή παίζει ρολό σε δυο σημεία που είναι σχετικά με το eye tracking: Πρώτον οι αντανακλάσεις του κερατοειδή δημιουργούνται από αντανακλάσεις των υπερύθρων LED στην επιφάνεια του κερατοειδή και δεύτερον η εικόνα της κόρης όπου παρατηρούμε μέσο του κερατοειδή διαστρεβλώνεται από αντανάκλαση στην επιφάνεια του κερατοειδή.



Εικόνα 26 : Μοντέλο οφθαλμού που χρησιμοποιείται για την εκτίμηση της ίριδας.
PC Pupil Centre (κέντρο κόρης), CC Cornea Centre (κέντρο κερατοειδή).
Το μοντέλο χρησιμοποίει τρεις παραμέτρους εξαρτώμενους από τον χρήστη : **α**fovea, **r**cornea , **r**pc.

Το κέντρο της κόρης , που αναφέρεται ως PC. Αυτό είναι το σημείο που βρίσκεται στο κέντρο του δίσκου της κόρης (κάνουμε την εξιδανικευμένη υπόθεση ότι η κόρη είναι απόλυτα κυκλική). Το PC είναι μια ορισμένη απόσταση από το κέντρο του κερατοειδούς CC, και το αποκαλούμε απόσταση **r**pc

Η γωνιακή μετατόπιση μεταξύ του οπτικού άξονα του ματιού και την κατεύθυνση της ίριδας (που αναφέρονται ως **α**fovea), η οποία προκαλείται από το γεγονός ότι το βοθρίο δεν βρίσκονται στον οπτικό άξονα, αλλά αντισταθμίζεται χρονικά και ελαφρώς προς τα πάνω (έως τώρα έχουμε περιγράψει μόνο την οριζόντια συνιστώσα αυτής της μετατόπισης).

Λαμβάνοντας υπόψη τη θέση και τον προσανατολισμό του ματιού σε σχέση με το την κάμερα, το μοντέλο προβλέπει που θα πρέπει να παρατηρηθούν στη εικόνα της κάμερας η κόρη και οι αντανακλάσεις του κερατοειδή. Το αντίστροφο πρόβλημα μπορεί επίσης να λυθεί, επιτρέποντας μας να καθορίσουμε την κατεύθυνση της ίριδας από την κόρη και την θέση αντανακλαστικών του κερατοειδή. Οι τιμές του μοντέλου,

Οι παράμετροι για έναν συγκεκριμένο χρήστη καθορίζονται ζητώντας από το χρήστη να εστιάσει σε μια σειρά από σημεία βαθμονόμησης και βρίσκοντας το πιο ταιριαστό σύνολο τιμών - παραμέτρων που οδηγεί στην καλύτερη παρακολούθηση .

4.3 Ακτινοβολία

4.1 Κίνδυνος ακτινοβολίας.

Πριν εκθέσουμε το μάτι σε υπέρυθρο φως θα πρέπει να σκεφτούμε τη μέγιστη ακτινοβολία που μπορεί το μάτι να εκτεθεί. Αν υποθέσουμε ότι το σύστημα εντοπισμού και παρακολούθησης οφθαλμού θα χρησιμοποιηθεί αργότερα από τους χρήστες επί μονίμου βάσεως για περίπου μια ώρα καθημερινά ή περισσότερο, η ακτινοβόληση δεν πρέπει να υπερβαίνει τη συνιστώμενη κατ 'ανώτατο όριο. Mulvey et al. (2008) προτείνει να μην υπερβαίνει συνολικά η ακτινοβολία του κερατοειδούς από 10 mW/cm2 για μήκη κύματος των 700 - 3000 nm ανά ημέρα.

Για να υπολογίσουμε την ακτινοβολία που θα εκτεθεί το μάτι πρέπει να γνωρίζουμε πόσο από το υπέρυθρο φως των LED χτυπά τον κερατοειδή χιτώνα. Αυτό μπορεί να δοθεί από τον υπολογίζοντας την στερεάς γωνίας (στερακτίνιο ή sr).

Για τον υπολογισμό αυτό, η απόσταση μεταξύ του κερατοειδή και του υπέρυθρου LED είναι απαραίτητη, η οποία θα είναι περίπου 5 cm σε αυτήν την περίπτωση. Η απόσταση αντιστοιχεί στην ακτίνα μιας φανταστικής σφαίρας γύρω από το LED, η οποία λειτουργεί ως το κέντρο της εν λόγω σφαίρας. Για τη στερεά γωνία η περιοχή της τομής μεταξύ της σφαίρας και του κερατοειδούς εκπροσωπείται από τον υπολογισμό του ΔΑΚ. Ο ακόλουθος τύπος δίνεται από Mulvey et al. (2008). Η στερεά γωνία δίνεται από:

$$\Omega = 4\pi \frac{\Delta A_K}{A_K} \quad (1)$$

or with $A_K = 4\pi r^2$

$$\Omega = \frac{\Delta A_K}{r^2} \quad (2)$$

Για τον υπολογισμό του ΔΑΚ μπορεί να γίνει μια αλλαγή στην γωνία όπως φαίνεται στο (3). Φ είναι η γωνία μεταξύ της παρακείμενης πλευράς (που είναι η ακτίνα της σφαίρας) και της υποτείνουσας του τριγώνου. Από το μέγεθος του σφαιρικού καλύμματος που δόθηκε από Δ ΑΚ = 2 πrh έχουμε:

$$\Delta A_K = 2\pi r^2 (1 - \cos \varphi) \quad (3)$$

Αντικαθιστώντας την (2) έχουμε:

$$\Omega = 2\pi (1 - \cos \varphi) \quad (4)$$

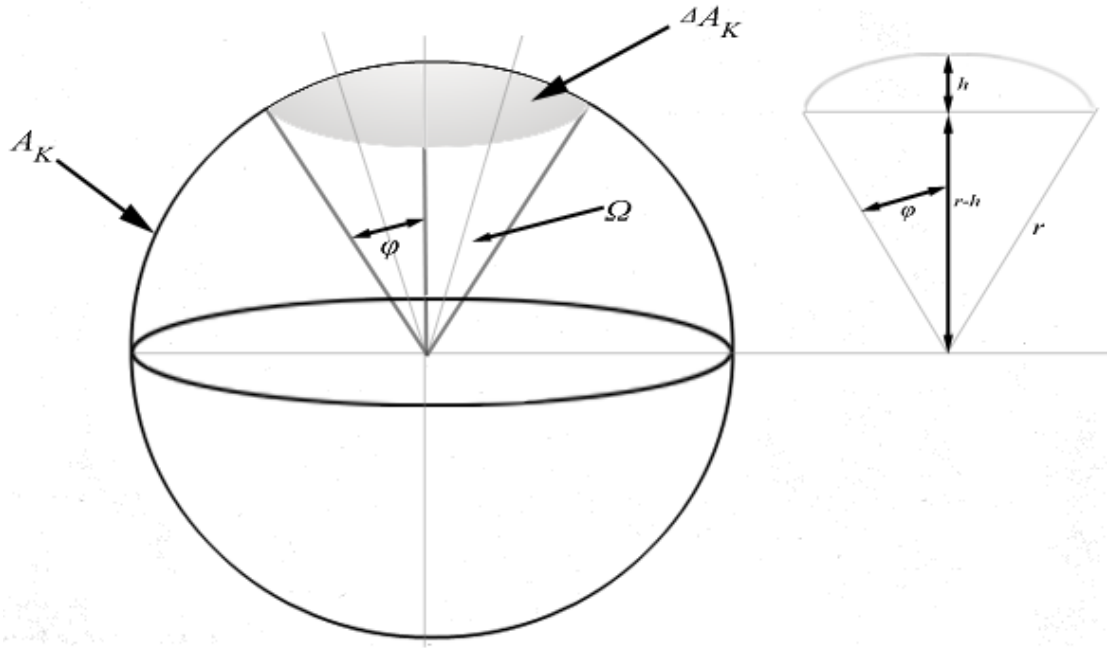
Τώρα χρειάζεται η περιοχή της τομής που δίνεται από την επιφάνεια του κερατοειδούς. Ο κερατοειδής χιτώνας είναι ένα ελλειψοειδές με μέσο μέγεθος τάξεως του 11,7 × 10,6 mm ((Duke-Elder and Wybar 1958).

Για τον επόμενο υπολογισμό χρησιμοποιείται η οριζόντια ακτίνα του κερατοειδή ώστε να δοθεί κατά προσέγγιση η επιφάνεια με π(5.85*mm*)^2 =107.5*mm*^2 .

Χρησιμοποιώντας το μεγαλύτερο κύκλο θα οδηγήσει σε αυστηρότερο ανώτατο όριο για την υπέρυθρη ακτινοβολία. Πρώτον, πρέπει να υπολογιστεί το φ. Όπως περιγράφηκε πριν το φ είναι η γωνία μεταξύ της παρακείμενης πλευράς (που είναι η ακτίνα της σφαίρας) και της υποτείνουσας του τριγώνου . Λόγω της tan (φ) = απέναντι/παρακείμενη και το γεγονός ότι η απέναντι συσχετίζεται με την ακτίνα του κερατοειδούς αυτό οδηγεί σε 5,85 mm/50 mm = 0,117. Με Φ = tan^-1 (5,85/50) αυτό μπορεί να αντικατασταθεί στην (4) και να προκύψει :

$$\Omega = 2\pi(1 - \cos(\tan^{-1}(\frac{5.85}{50}))) \approx 0.043\, sr$$

Μετά τον υπολογισμό του στερακτίνιου και γνωρίζοντας την επιφάνεια του κερατοειδούς η μέγιστη υπέρυθρη LED ακτινοβολία μπορεί να βρεθεί. Η επιφάνεια του κερατοειδούς είναι 1,075 cm^2 και η συνιστώμενη μέγιστη έκθεση είναι από
10 mW/cm^2, έτσι δεν πρέπει να υπάρχει περισσότερα από 10,75 mW ακτινοβολίας στον κερατοειδή χιτώνα. Επειδή η στερεά γωνία για ένα LED σε απόσταση 5 εκατοστών από κερατοειδή χιτώνα είναι 0,043 sr (περίπου (1/ 23) Sr), η συνολική ακτινοβολία του LED δεν πρέπει να υπερβαίνει το 23 B · 10.75mW= 247.25mW / sr.



Εικόνα 27 :
Το σφαιρικό ΔΑΚ αντιπροσωπεύει την περιοχή της τομής ανάμεσα στον φανταστικό χώρο γύρω από το LED και του κερατοειδούς χιτώνα. για τον υπολογισμό της στερεάς γωνίας (Ω) ένας μετασχηματισμός στη γωνία φ είναι απαραίτητος.
Πηγή: Mulvey et al. (2008).

4.2 Ένταση ακτινοβολίας.

Το υπέρυθρο LED έχει ένταση ακτινοβολίας 60 mW / SR, εάν τροφοδοτείται με 100 mA, η οποία είναι πολύ κάτω από το πλαίσιο 247,25 mW / sr .Η ακτινοβολία θα πρέπει να βρίσκεται στη χαμηλότερη δυνατή τιμή. Αυτό μπορεί να γίνει με την προσθήκη ενός ποτενσιόμετρου στο κύκλωμα που κάνει την τροφοδοσία των LED ρυθμιζόμενη. Η ρύθμιση αυτή γίνεται και με μια δίοδο και μια αντίσταση με 39 ohm ώστε να παραχθούν 100 mA . Εικόνα 4

Εικόνα 28 : Ένα απλό κύκλωμα LED

Κεφάλαιο 5. Διαδικασία κατασκευής συσκευής

5.1 Βήματα κατασκευής:

1.Ετοιμασια γυαλιών:
Αρχικά απομακρύνουμε τους φακούς από τον σκελετό των γυαλιών.

Εικόνα 29 : Βήμα κατασκευής 1

2.Βαση σταθεροποίησης κάμερας:
Κόβουμε ένα κομμάτι αλουμινένιου καλωδίου, για 5 cm απόσταση κάμερας από το μάτι κόβουμε περίπου 22-25 cm, και λυγίζουμε το καλώδιο κατά 90 μοίρες για να δημιουργήσουμε μια θέση για την κάμερα. Το μήκος των πλευρών της θέσης είναι περίπου 3,8 cm (το μέγεθος της κάμερας είναι 3,6 x3,6 cm). Εφαρμόζουμε το καλώδιο στον σκελετό των γυαλιών με ταινία.

Εικόνα 30 : Βήμα κατασκευής 2

3. Τοποθέτηση της κάμερας:

Περιφερειακά της κάμερας υπάρχουν τέσσερις τρύπες ,οι οποίες μα βοηθούν να σταθεροποιήσουμε την κάμερα στην υποδοχή που έχουμε φτιάξει από αλουμινένιο καλώδιο , με την βοήθεια δεματιών.

Εικόνα 31 : Βήμα κατασκευής 3

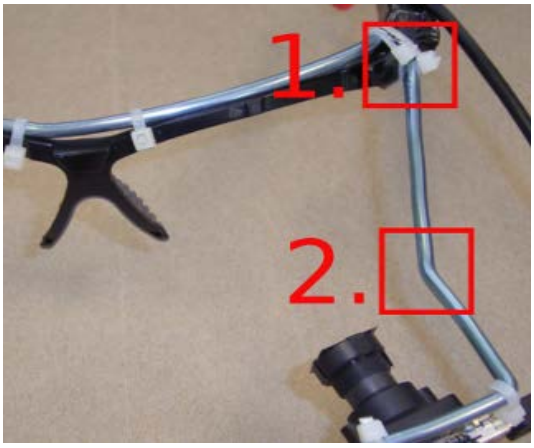
4.Σταθεροποιηση του σκελετού

Επειδή ο σκελετός των γυαλιών είναι λίγο εύκαμπτος, τον σταθεροποιούμε δένοντας κατά μήκος του καλώδιο αλουμινίου.

Εικόνα 32 : Βήμα κατασκευής 4

5. Ρύθμιση θέσης κάμερας

Μέχρι τώρα η κάμερα ήταν τοποθετημένη ακριβώς μπροστά από το μάτι και έτσι θα εμπόδιζε το οπτικό πεδίο του χρήστη. Για να λύσουμε το πρόβλημα αυτό θα πρέπει να λυγίσουμε το καλώδιο στα σημεία 1,2, όπως φαίνονται στην εικόνα διατηρώντας όμως το κέντρο της κάμερας ακριβώς απέναντι από το μάτι.



Εικόνα 33 : Βήμα κατασκευής 5

Δοκιμάζουμε την νέα θέση της κάμερας . ελέγχουμε την απόσταση μεταξύ της κάμερας και του ματιού.

Εικόνα 34 : Βήμα κατασκευής 5β

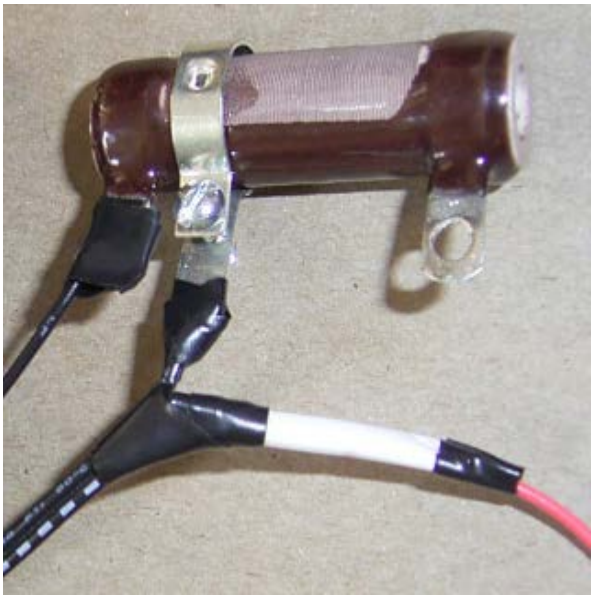
6. Προετοιμασία υπέρυθρων LED

Πριν συνδέσουμε το υπέρυθρο LED στην θέση του, χρειάζεται ένα τροφοδοτικό. Το πρώτο βήμα είναι εύκολο και περιλαμβάνει συγκόλληση της ανόδου και του καλωδίου τροφοδοσίας καθώς και της καθόδου και τοκαλώδιο γείωσης.

Εικόνα 35 : Βήμα κατασκευής 6

7. Σύνδεση ποντεσιομτρου

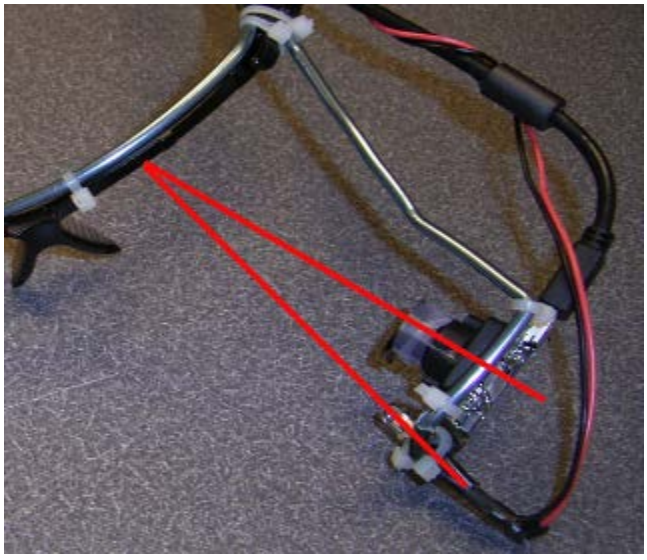
Ένα ποτενσιόμετρο επιτρέπει την ρύθμιση της ένταση της ακτινοβολίας. Για να προσθέσουμε το ποτενσιόμετρο στο κύκλωμα θα πρέπει να βρίσκεται μεταξύ καθόδου και της γείωσης ακριβώς όπως μια κανονική αντίσταση. Η ένταση της ακτινοβολίας στο μάτι πρέπει να διατηρείται κάτω από το συνιστώμενο μέγιστο.



Εικόνα 36 : Βήμα κατασκευής 7

8. Τοποθέτηση υπέρυθρων LED

Βασιζόμενοι στο τι είδους αποτέλεσμα θέλουμε (σκοτεινή ή φωτεινή κόρη) το LED θα πρέπει να τοποθετηθεί εντος ή εκτός του οπτικού άξονα της κάμερας. Οι κόκκινες γραμμές δείχνουν τους οπτικούς άξονες από την κάμερα στα αντίστοιχα σημεία των LED.



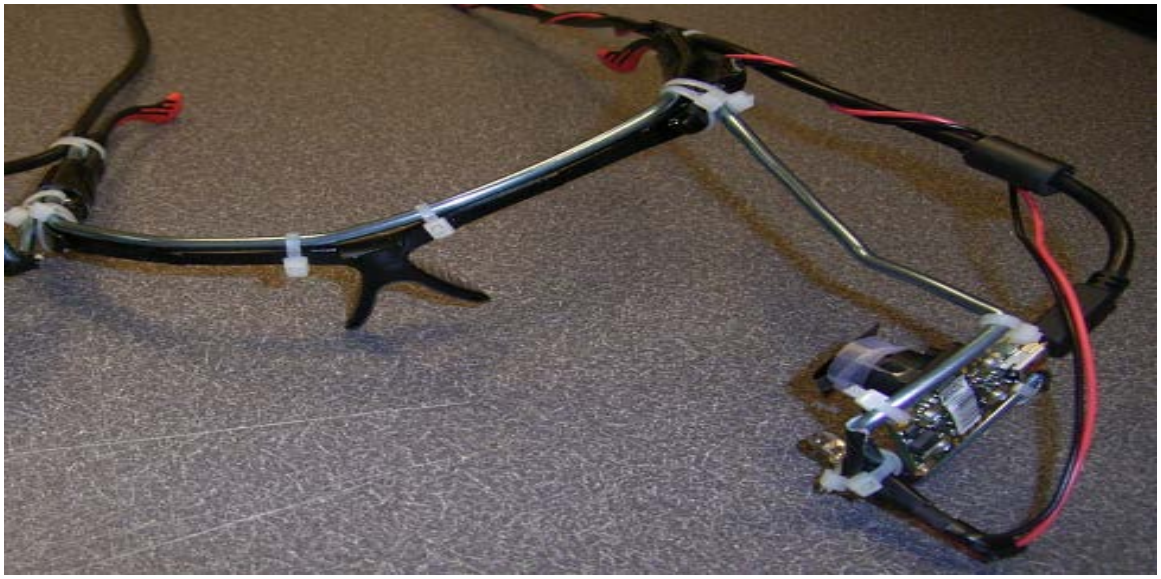
Εικόνα 37 : Βήμα κατασκευής 8

9.Τοποθετηση ενός απλού φίλτρου.

Σε περίπτωση που η κατασκευή χρησιμοποιηθεί σε εξωτερικούς χώρους η αυξημένη φωτεινότητα θα δημιουργήσει πρόβλημα καθώς το μάτι μπορεί να φωτίζεται πάρα πολύ. Για να περιορίσουμε το προβλήματα με το φως του ήλιου υπάρχει μια απλή λύση. Ένα μικρό κομμάτι αρνητικής φωτογραφίας μπορεί να τοποθετηθεί μπροστά από την κάμερα και να φιλτράρει το φως του περιβάλλοντος, ενώ το υπέρυθρο φως διαπερνά το φίλτρο. ο φωτισμός με LED υπερύθρων θα πρέπει επίσης να φιλτραριστεί, αλλά δεδομένου ότι είναι δυνατόν να αυξηθεί η ένταση της ακτινοβολίας με το ποτενσιόμετρο δεν υπάρχει κανένα πρόβλημα.

Εικόνα 38 : Βήμα κατασκευής 9

10. Ολοκληρωμένη κατασκευή.



Εικόνα 39 : Συσκευή

Κεφάλαιο 6 . Ανάλυση συστήματος που χρησιμοποιήσαμε

6.1 Εισαγωγή

Στο κεφάλαιο αυτό περιγράφεται το σύστημα εντοπισμού οφθαλμού που χρησιμοποιείται. Αρχικά, δίνεται μια αναλυτική περιγραφή των υποσυστημάτων από τα οποία αυτό μπορεί να υλοποιηθεί, καθώς και τα τεχνικά χαρακτηριστικά του επιμέρους εξοπλισμού που απαιτείται.

Το λογισμικό που αναπτύχθηκε πληροί αυστηρές προδιαγραφές για χρήση από ασθενείς με σύνδρομο ALS και άλλες παθήσεις που προϋποθέτουν περιορισμένη κινητικότητα.

Επιπλέον το λογισμικό έχει αναπτυχθεί για να είναι συμβατό και να λειτουργεί με ένα ευρύ φάσμα, και διάφορους τύπους καμερών, ούτως ώστε να μην υπάρχουν ιδιαίτεροι τεχνικοί περιορισμοί, όσο αφορά το υλικό.

Το λογισμικό Eye-tracker / Eye-mouse Lidakis Chatzaki ξεκίνησε και αναπτύχθηκε σε δύο στάδια.

Σε αρχικό στάδιο ο προγραμματισμός έγινε εξ' ολοκλήρου στην γλώσσα προγραμματισμού c++ χρησιμοποιώντας έτοιμες βιβλιοθήκες της c++, και προοριζόταν να λειτουργεί σε λειτουργικό σύστημα linux και με web-camera.

Στο δεύτερο στάδιο υλοποίησης χρησιμοποιήσαμε βιβλιοθήκες και συναρτήσεις, σε συνδυασμό με τον ITUgt αλγόριθμο στον οποίο εφαρμόσαμε φίλτρα για την βελτιστοποίηση του, και περάσαμε το πρόγραμμα σε τελείως άλλο επίπεδο, χωρίς να χρησιμοποιήσουμε σχεδόν τίποτα από τις λειτουργίες του πρώτου προγράμματος, θέλοντας να το κάνουμε περισσότερο λειτουργικό και εύκολο στην χρήση.

6.2 Μέθοδοι αναγνώρισης οφθαλμού που λήφθηκαν υπόψ'ιν για την υλοποίηση του αρχικού προγράμματος.

6.2.1Επεξεργασία εικόνων οφθαλμού – υπολογισμός παραμέτρων.

Στα πλαίσια της παρούσας εργασίας αναπτύχθηκαν και εφαρμόστηκαν δύο μέθοδοι για την αναγνώριση του οφθαλμού:

1. Υπολογισμός της κατανομής των επιπέδων του γκρι στην πάνω και κάτω περιοχή του οφθαλμού και συνδυασμός τους
2. Συνδυασμός τεχνικών επεξεργασίας εικόνων οφθαλμού για την απομόνωση της περιοχής του.

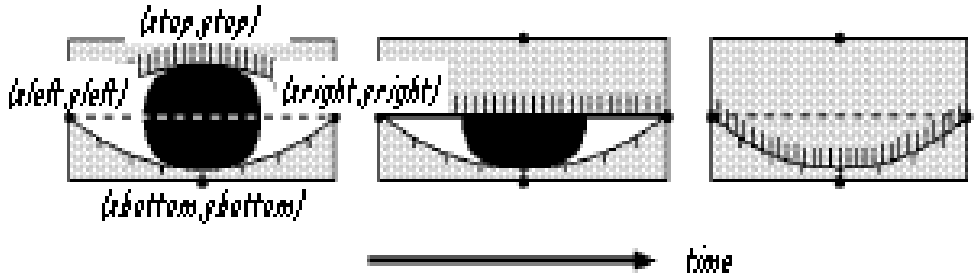
Στις επόμενες ενότητες παρουσιάζονται αναλυτικά οι δύο αυτές μέθοδοι.

Επιπλέον, και με βάση αυτές, υπολογίστηκαν οι ακόλουθοι παράμετροι που συνδέονται με την κατάσταση του οφθαλμού σε μια αλληλουχία εικόνων video:

- Αριθμός των ανοιγο-κλεισιμάτων (blinks) του οφθαλμού,
- Διάρκεια κάθε ανοιγο-κλεισίματος του οφθαλμού, και
- Διάστημα που μεσολαβεί μεταξύ διαδοχικών ανοιγο-κλεισιμάτων του οφθαλμού.

6.2.2 Πρώτη Μέθοδος Αναγνώρισης Οφθαλμού

Η πρώτη μέθοδος που χρησιμοποιήθηκε για την αναγνώριση του οφθαλμού σε μία αλληλουχία εικόνων video βασίστηκε στην εφαρμογή αλγορίθμων επεξεργασίας εικόνων, όπως προτάθηκαν από τους T. Moriyama et al. (2002). Η μέθοδος αυτή στηρίζεται στον υπολογισμό της κατανομής των επιπέδων του γκρι στην άνω και κάτω περιοχή του οφθαλμού. Συγκεκριμένα, όπως φαίνεται στο Εικόνα 40, ο οφθαλμός αποτελείται βασικά από την ίριδα, τον σκληρό χιτώνα, το άνω και κάτω βλέφαρο και τις βλεφαρίδες. Αν η περιοχή του οφθαλμού διαιρεθεί σε δύο τμήματα (άνω και κάτω περιοχές - διακεκομμένη γραμμή, Εικόνα 40), παρατηρείται μεταβολή στην κατανομή των επιπέδων του γκρι στο άνω και στο κάτω τμήμα καθώς ο οφθαλμός κλείνει και στη συνέχεια ανοίγει κατά τη διάρκεια ενός ανοιγο-κλεισίματος.



Εικόνα 40: Διδιάστατο Μοντέλο οφθαλμού με βάση τη μέθοδο των T. Moriyama et al.

Στην ακολουθία των εικόνων οφθαλμού του video που αποτελούν τα δεδομένα εισόδου στο λογισμικό που αναπτύχθηκε, ορίζεται χειροκίνητα ένα ορθογωνικό παράθυρο που περικλείει την περιοχή του οφθαλμού, για το οποίο έστω ότι ισχύει:
όπου *Xleft, Xright* οι τιμές της συντεταγμένης x που ορίζουν, αντίστοιχα, την αριστερή και τη δεξιά πλευρά της περιοχής που περικλείει τον οφθαλμό και *Ytop, Ybottom* οι τιμές της συντεταγμένης *y* που ορίζουν, αντίστοιχα, την άνω και κάτω πλευρά της περιοχής.
Υπολογίζουμε σε κάθε εικόνα οφθαλμού τη μέση φωτεινότητα για το άνω και κάτω τμήμα της περιοχής αυτής, *Iupper,mean* και *Ilower,mean*, αντίστοιχα, σύμφωνα με τις ακόλουθες εξισώσεις:
όπου *I(x,y)* η φωτεινότητα της εικόνας στο εικονοστοιχείο *(x,y), Ymiddle* η τιμή του *y* που καθορίζει τη διαχωριστική γραμμή μεταξύ των δύο περιοχών του οφθαλμού:
και *N* και *M* ο συνολικός αριθμός εικονοστοιχείων στην πάνω και κάτω περιοχή της εικόνας αντίστοιχα.

6.2.3 Δεύτερη Μέθοδος Αναγνώρισης Οφθαλμού

Η δεύτερη μέθοδος αναγνώρισης του οφθαλμού βασίζεται στην εφαρμογή αλγορίθμων επεξεργασίας εικόνων για τον αυτόματο εντοπισμό της περιοχής του οφθαλμού.
Στα βασικά βήματα της μεθόδου περιλαμβάνονται:

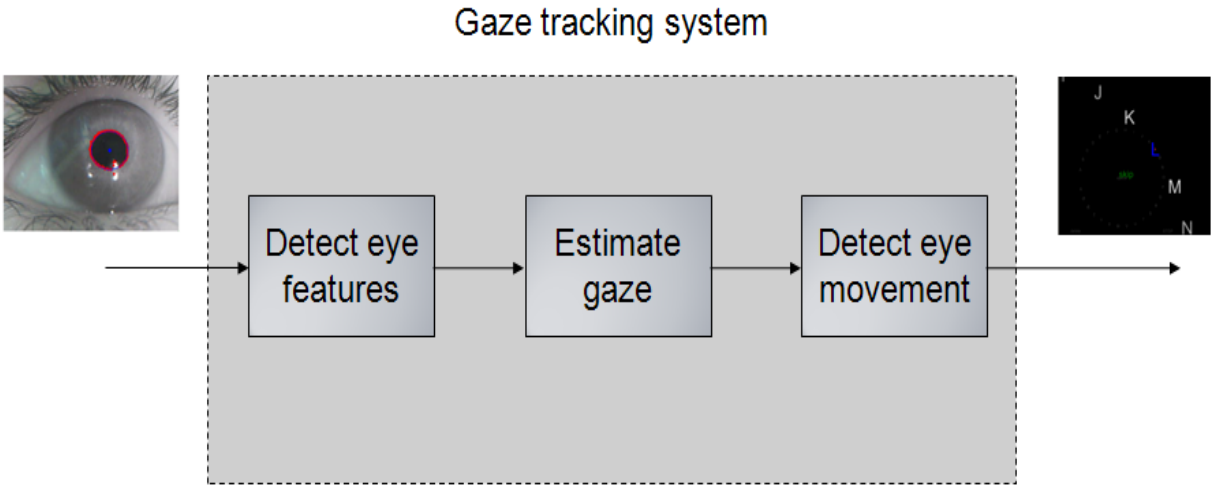
1. Η εφαρμογή αλγορίθμου ευθυγράμμισης για τη δημιουργία κοινού συστήματος συντεταγμένων όλων των εικόνων οφθαλμού που συλλέγονται από την κάμερα.
2. Η εφαρμογή δύο φίλτρων ανίχνευσης κοιλάδων και κορυφών σε κάθε εικόνα οφθαλμού.
3. Η σύντηξη των αποτελεσμάτων από την εφαρμογή των δύο φίλτρων.
4. Η μετατροπή της εικόνας σύντηξης σε δυαδική.
5. Ο υπολογισμός των παραμέτρων κατάστασης του οφθαλμού.

6.3 Τεχνολογία εντοπισμού – παρακολούθησης βλέμματος (gaze tracking)

Η καταγραφή κίνησης οφθαλμού ορίζεται ως η διαδικασία μέτρησης σημείου ενδιαφέροντος point of regard (POR).
Αυτή η διαδικασία μπορεί να χωριστεί σε δύο υποδιαδικασίες

- Eye tracking για παράδειγμα, ανίχνευση και καταγραφή του οφθαλμού και των κινήσεων του,
- Gaze Estimation υπολογισμός του βλέμματος του ματιού από τα χαρακτηριστικά του ματιού.

Ένα σύστημα υπολογισμού βλέμματος ή gaze tracker είναι ένα σύστημα που μετράει τις κινήσεις του ματιού και υπολογίζει το βλέμμα.
Η εφαρμογή μας στηρίζεται στην καταγραφή του βλέμματος του οφθαλμού σε βίντεο, που αποτελείται από μία ή περισσότερες κάμερες που καταγράφουν τα μάτια ή το μάτι του χρήστη.
Όπως στα περισσότερα συστήματα απαιτείται υπέρυθρος φωτισμός στην εικόνα για να γίνει υπολογισμός του βλέμματος / του ματιού.
΄Ένα διάγραμμα ενός γενικού συστήματος τεχνολογίας καταγραφής κίνησης βλέμματος ή οφθαλμού παρατίθεται παρακάτω.



Εικόνα 41 : Διάγραμμα συστήματος τεχνολογίας καταγραφής κίνησης βλέμματος.

Σε αρχικό στάδιο η κάμερα συλλαμβάνει μία εικόνα η οποία μεταφέρεται στον υπολογιστή. Το λογισμικό εξάγει μερικά χαρακτηριστικά του ματιού από την εικόνα όπως το κέντρο της κόρης του ματιού ή το κέντρο της ίριδας. Αυτά τα χαρακτηριστικά του ματιού χαρτογραφούνται στο σύστημα μέσα από μια διαδικασίας διαμέτρησης (calibration).
Λόγω του θορύβου, τα δεδομένα του ματιού μπορεί να είναι σπασμωδικά, και γι αυτό το λόγο μία διαδικασία διόρθωσης λαμβάνει χώρα για να εξομαλύνει τα δεδομένα ή ακόμα και την ροή των δεδομένων, όταν ανιχνευθεί κάποια ανωμαλία στις εικόνες η στις κινήσεις.
Το λογισμικό είναι σχεδιασμένο να εκτελείται στα Windows με το .net framework 3.5 sp1.
Επίσης το πρόγραμμα διαθέτει δύο λειτουργίες καταγραφής κίνησης του οφθαλμού.
Η μία είναι με συσκευή προσαρμοσμένη για να τοποθετείται στην κεφαλή του χρήστη ούτως ώστε η κάμερα να βρίσκεται κοντά στην κόρη του ματιού του χρήστη.
Η δεύτερη λειτουργία είναι, η κάμερα να είναι τοποθετημένη στο γραφείο ή στην οθόνη για μία απομακρυσμένη ρύθμιση.

6.4 Μέθοδοι καταγραφής.

Το Eye-tracker / Eye-mouse Lidakis Chatzaki υποστηρίζει δύο μεθόδους λειτουργίας, καταγραφή κόρης οφθαλμού (pupil tracking) και καταγραφή λάμψης (glint tracking). Και οι δύο μέθοδοι προϋποθέτουν καθαρή εικόνα του ματιού είτε τοποθετώντας την κάμερα κοντά στο μάτι (webcam) είτε κάνοντας εστιάζοντας (zoom) στο μάτι (video camera). Και οι δύο μέθοδοι βασίζονται στην αντανάκλαση του κερατοειδούς που δημιουργείται από το υπέρυθρο φως.

Στις μέρες μας ένας μεγάλος αριθμός προϊόντων ταιριάζει και ικανοποιεί τα κριτήρια της εφαρμογής, από χαμηλού κόστους κάμερες διαδικτύου μέχρι ακριβές βιντεοκάμερες. Τα κριτήρια αυτά αναφέρονται (στις κάμερες) ως νυχτερινή λήψη ή night-shot χρησιμοποιώντας led υπερύθρων.

Όταν χρησιμοποιείται head-mounted συσκευή η καταγραφή της κόρης γίνεται ικανοποιητικά, ωστόσο μετακινώντας την κάμερα ή το κεφάλι επηρεάζεται η καταγραφή! Για παράδειγμα αν κινεί ο χρήστης το κεφάλι του προς τα πάνω το σημείο της κόρης που καταγράφεται κινείται προς τα κάτω. Όταν τοποθετηθεί η κάμερα στο γραφείο σε απομακρυσμένη ρύθμιση η καταγραφή λάμψης πρέπει να είναι ενεργοποιημένη

6.5 Απαιτήσεις Συστήματος

Ένας τυπικός Η/Υ με επεξεργαστή δύο πυρήνων εύκολα μπορεί να επεξεργάζεται βίντεο από μία σύγχρονη web camera. Είναι σημαντικό να αναφερθεί ότι η δυνατότητα εξαρτάται από την ανάλυση του αρχικού βίντεο (raw image) και από την ταχύτητα λήψης διαδοχικών εικόνων (frame rate). Οι διαδικασίες επεξεργασίας των εικόνων εξαρτώνται από ένα αριθμό αλγορίθμων, που ο καθένας χρειάζεται κάποιο εύρος μνήμης (tempROM και RAM) κατά την εκτέλεση του, γι αυτόν τον λόγο προκύπτει και η απαίτηση αυτή. Παρόλαυτα έχει γίνει βελτιστοποίηση του κώδικα του προγράμματος ούτως ώστε κατά την εκτέλεση του να λειτουργεί με επιδόσεις και με το λιγότερο δυνατό φόρτωμα της κεντρικής μονάδας επεξεργασίας (CPU).

Πρέπει επίσης να σημειωθεί ότι σε περίπτωση που αποκολληθεί από το κύριο παράθυρο της εφαρμογής το παράθυρο όπου προβάλλεται η εικόνα της κάμερας η χρήση του επεξεργαστή αυξάνει, γι αυτόν τον λόγο προτείνεται μετά το calibration να γίνεται ελαχιστοποίηση του παραθύρου.

6.5.1 Απαιτήσεις υλικού Η/Υ

- Επεξεργαστής: INTEL ή AMD δύο ή περισσότερων πυρήνων.
- Μνήμη: 1GB RAM ή περισσότερη.

6.5.2 Απαιτήσεις Λειτουργικού συστήματος

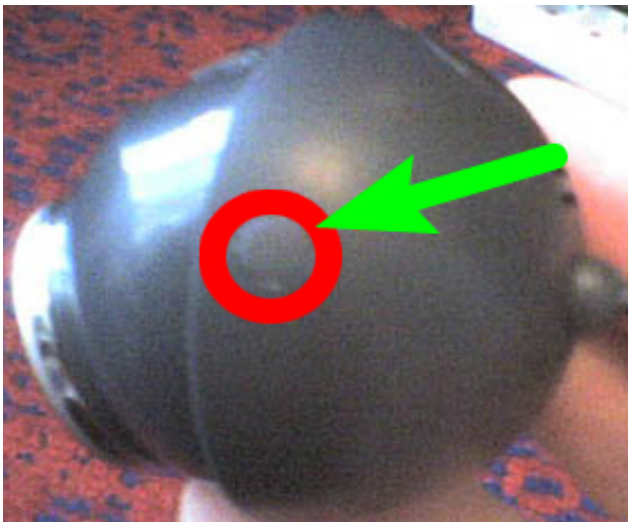
- Microsoft Windows Service Pack 3 /Microsoft Windows Vista /Microsoft Windows 7
- Microsoft .NET Framework v 3.5 SP1

6.5.3 Απαιτήσεις κάμερας

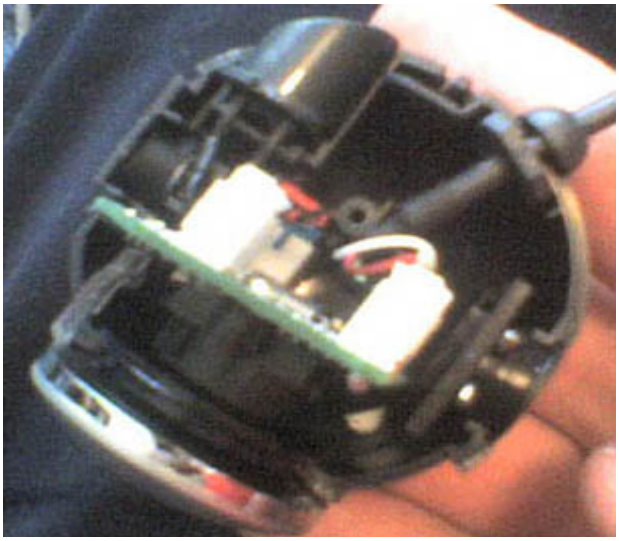
Για ένα ακριβές eye-tracking σύστημα κρίνεται απαραίτητο να μας παρέχονται υψηλής ανάλυσης εικόνες από την κάμερα. Στις περισσότερες περιπτώσεις η κόρη του ματιού πρέπει να φωτίζεται με υπέρυθρο φως και η κάμερα να μην διαθέτει φίλτρο υπερύθρων. Επίσης χρησιμοποιώντας φωτισμό του κοντινού υπερύθρου δημιουργείται και διαχωρίζεται αρκετά η κόρη του ματιού, που είναι σημαντικό για την επεξεργασία εικόνας που λαμβάνει χώρα στον Η/Υ.

6.6 Αφαιρεση φίλτρου υπερύθρων

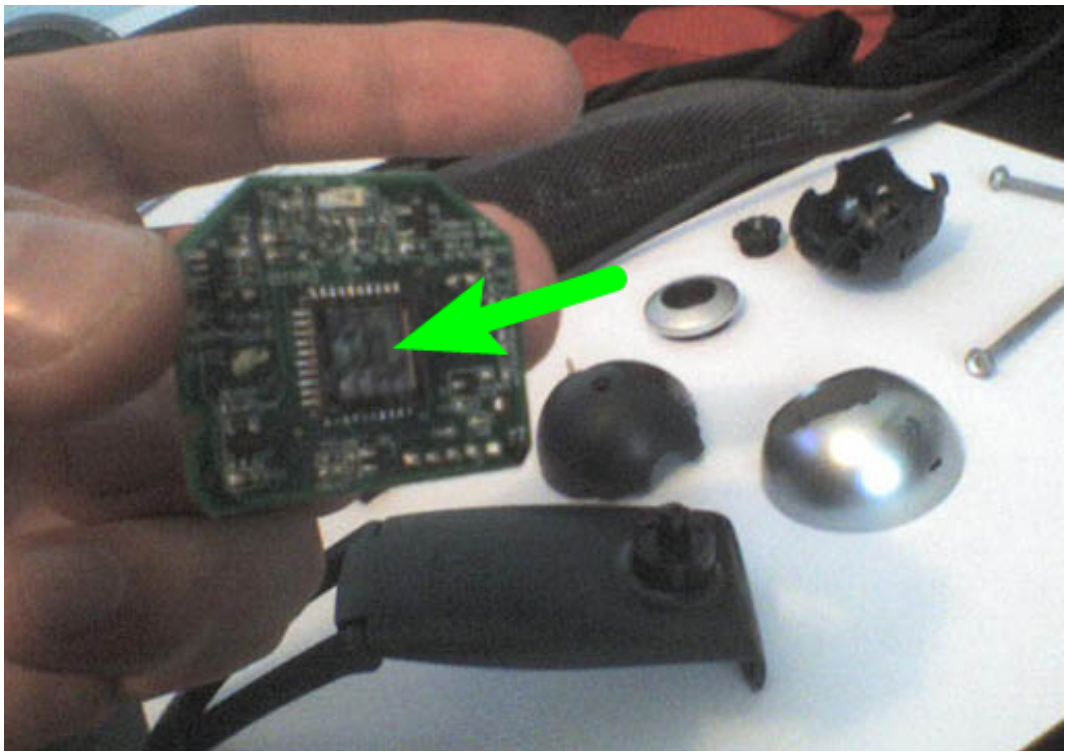
Στα πλαίσια δοκιμών της παρούσας πτυχιακής χρησιμοποιήθηκε η κάμερα Logitech Quickcam και το φίλτρο υπερύθρων αφαιρέθηκε από αυτήν. Τα στάδιατης διαδικασίας αυτής παρατίθενται παρακάτω.



Εικόνα 42 : Σημείο αποσυναρμολόγησης.

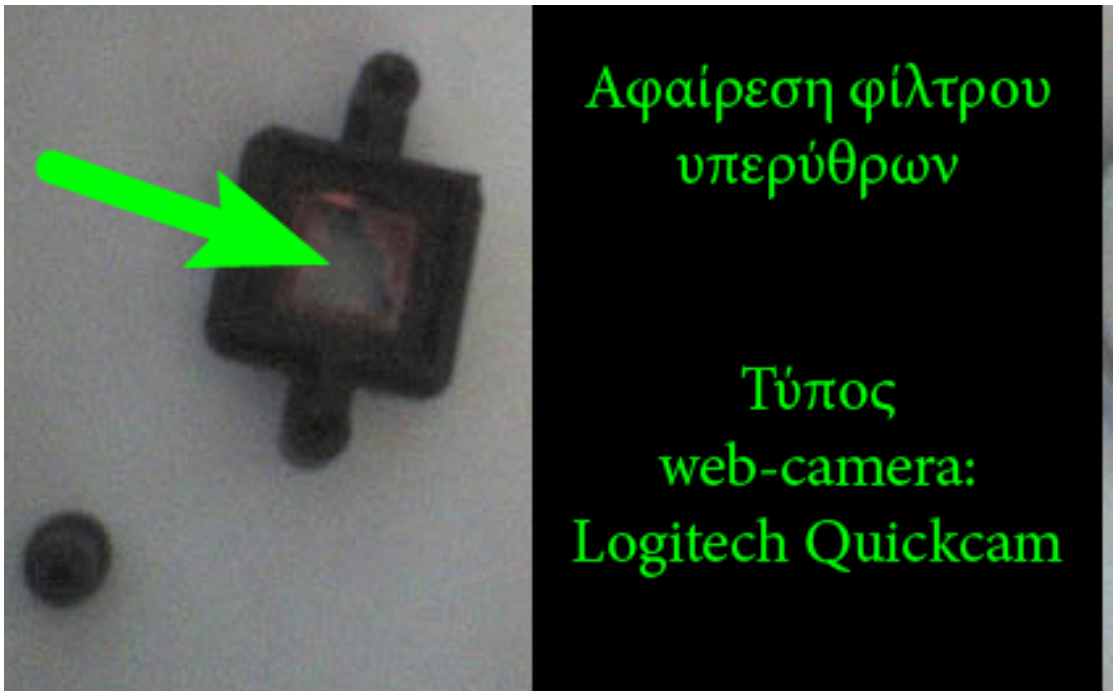


Εικόνα43 : Εσωτερικό της κάμερας.



Εικόνα 44 : Το σημείο που βρίσκεται το φίλτρο υπερύθρων

Εικόνα 45 : Επιμέρους εξαρτήματα κάμερας.



Εικόνα 46 : Η Αφαίρεση του φίλτρου υπερύθρων.

Για την χρήση κάμερας σε συσκευή που προσαρμόζεται στην κεφαλή προτείνουμε κάμερα με λειτουργία νυχτερινής λήψης, ανάλυση 640 x 480 και ρυθμό εικονοληψίας 25 frames ανά δευτερόλεπτο.

Στην συνέχεια τοποθετούμε την κάμερα κοντά στο μάτι με υποστήριγμα ένα καπέλο, σκελετό από γυαλιά ή οτιδήποτε που ταιριάζει σε αυτήν την χρήση, για να πάρουμε σωστά την εικόνα ενός ματιού.

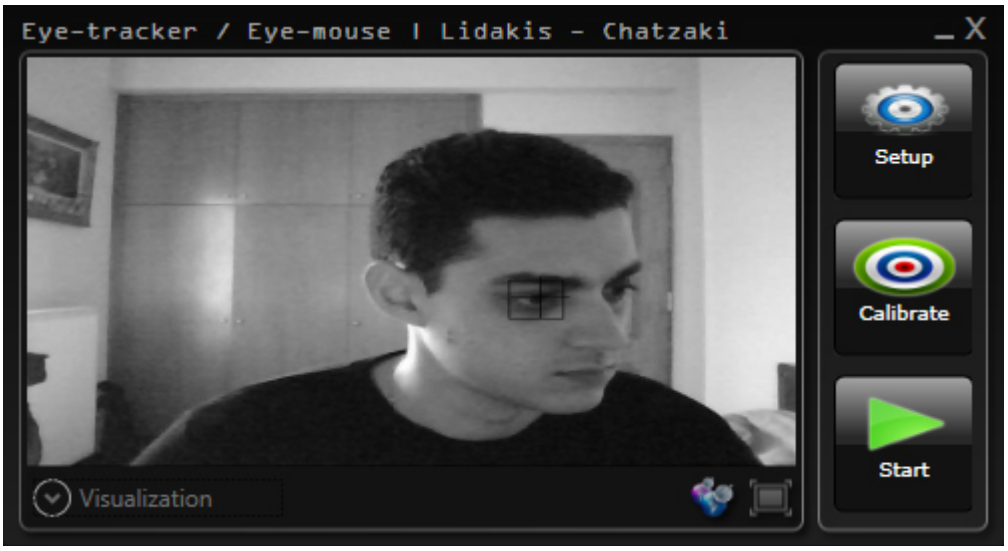
Η ιδανική απόσταση εξαρτάται από την κάμερα, αλλά κατά μέσο όρο η ιδανική απόσταση είναι από 10cm έως 15cm. Πολλές web-cameras δεν υποστηρίζουν το eye-tracking από απόσταση, διότι δεν υποστηρίζουν την λειτουργία εστίασης.

Για την χρήση του προγράμματος στην λειτουργία ‘remote eye-tracking’ δηλαδή από απόσταση καταγραφή του οφθαλμού, προϋπόθεση είναι η κάμερα να διαθέτει λειτουργία εστίασης ώστε να έχουμε μία καθαρή εικόνα του ματιού.

6.7 Περιγραφή και Αναπαράσταση Λειτουργίας της εφαρμογής σε επίπεδο χρήστη.

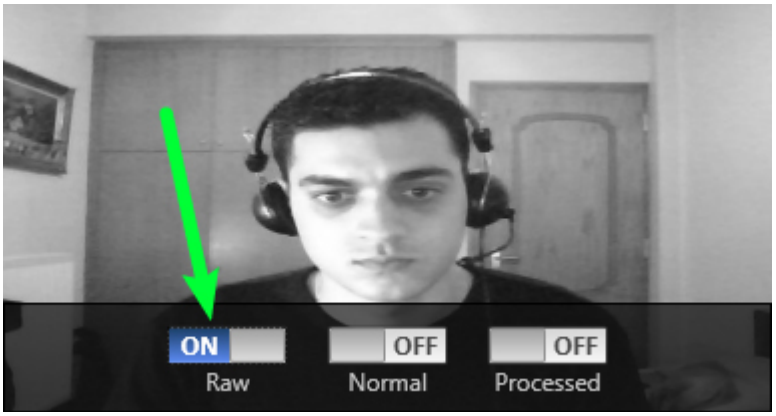
Σε αυτήν την ενότητα θα περιγράψουμε την λειτουργία της εφαρμογής σε επίπεδο χρήστη. Για να ανοίξετε την εφαρμογή κάντε διπλό κλικ στον αρχείο ‘eyetracker.exe’ που βρίσκεται στον φάκελο ‘eyetrackerLidakisChatzaki’.

Εκκινώντας λοιπόν την εφαρμογή το πρώτο παράθυρο που βλέπουμε είναι το παρακάτω.

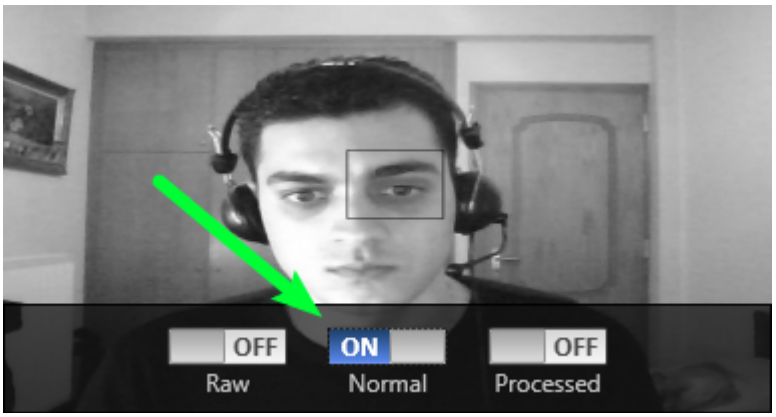


Εικόνα 47 : Αρχικό Παράθυρο

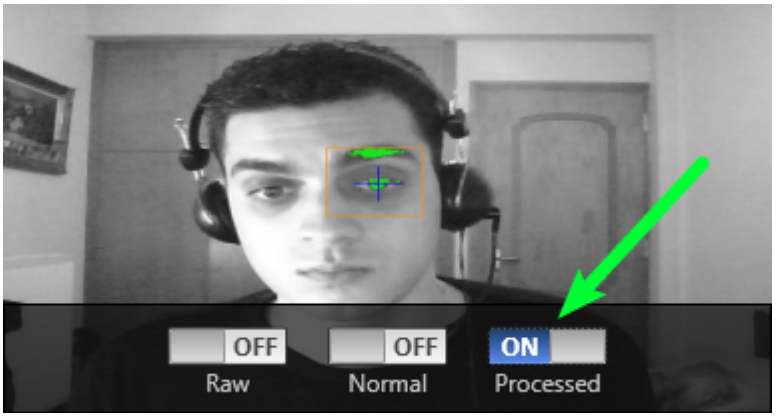
Ανοίγουμε την καρτέλα visualization και να επιλέγουμε Processed, αυτό μα δίνει την επεξεργασμένη εικόνα κατόπιν της διαδικασίας που εκτελεί το πρόγραμμα και μας βοηθάει επιλέξουμε τις σωστές ρυθμίσεις. Τρία παραδείγματα της λειτουργίας αυτής δίνονται στις παρακάτω εικόνες



Εικόνα 48 : Raw On

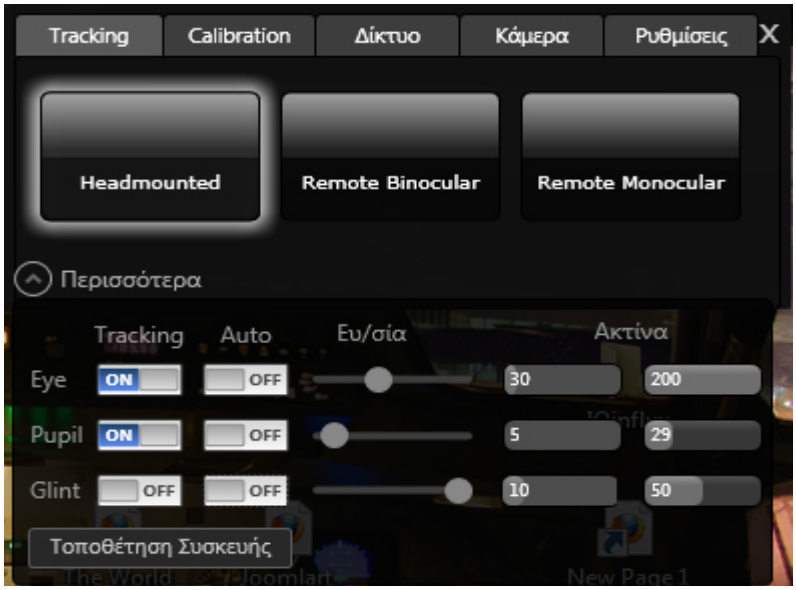


Εικόνα 49 : Normal On



Εικόνα 50 : Processed On

Επιστρέφοντας στο κεντρικό παράθυρο (Εικόνα 47) έχουμε στα δεξιά μας τρεις επιλογές, setup, calibrate και start. Στην ενότητα setup έχουμε την δυνατότητα να κάνουμε διάφορες ρυθμίσεις για τις επιλογές μας και να ορίσουμε παραμέτρους. Επιλέγοντας το κουμπί setup ανοίγει το παρακάτω παράθυρο (Εικόνα 51) στα δεξιά του κυρίου παραθύρου της εφαρμογής.

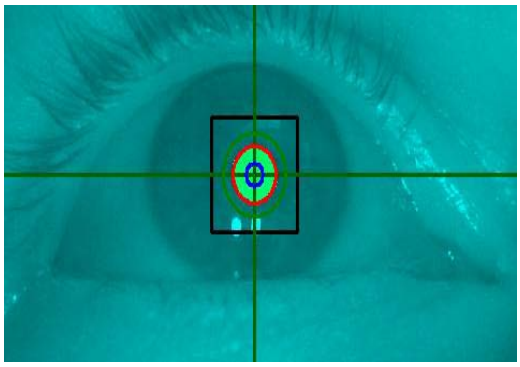


Εικόνα 51 : Παράθυρο επιλογών

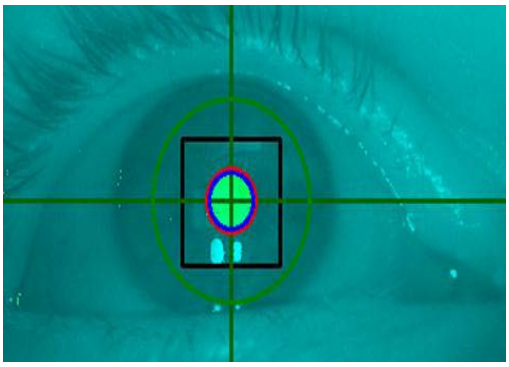
Η πρώτη καρτέλα που βλέπουμε όταν ανοίγει ο πίνακας ελέγχου της εφαρμογής είναι ή καρτέλα 'tracking'. Εδώ εμφανίζονται τρεις επιλογές για το τι συσκευή θα ρησιμοποιήσουμε, και κάποιες επιπλέον ρυθμίσεις που μπορείτε να διακρίνετε και στην εικόνα.

Για παράδειγμα μπορούμε να χρησιμοποιήσουμε συσκευή που προσαρμόζεται στο κεφάλι με μία κάμερα(head mounted), απομακρυσμένη κάμερα(Remote Monocular), ή συσκευή απομακρυσμένη που απαρτίζεται από δυο κάμερες (Remote Binocular).

Εδώ επιγραμματικά θα πρέπει να αναφέρουμε ότι σε περίπτωση που χρησιμοποιούμε head mounted συσκευή θα πρέπει το glint να είναι κλειστό. Επιπλέον είναι σημαντικό να σημειωθεί ότι κατά την διαδικασία ρύθμισης η ακτίνα της κόρης πρέπει να είναι σωστά ρυθμισμένη, (η μέγιστη και η ελάχιστη τιμή της). Στις δύο εικόνες που παρουσιάζονται στην συνέχεια (Εικόνες 52,53).
Ο μπλε κύκλος αντιπροσωπεύει την ελάχιστη τιμή και ο κόκκινος την μέγιστη, και θα πρέπει η ελάχιστη από την μέγιστη τιμή να έχουν διακριτή διαφορά (υπάρχει και η επιλογή auto για αυτόματη ρύθμιση).

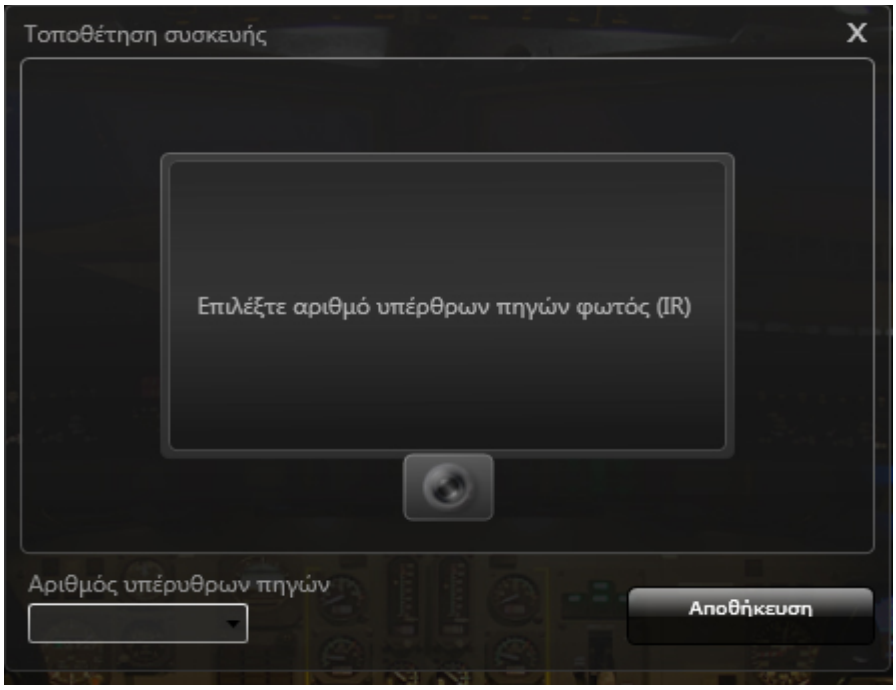


Εικόνα 52: Καλή ρύθμιση



Εικόνα 53: Όχι καλή ρύθμιση

Στην συνέχεια κάνοντας κλικ στην επιλογή για ‘remote monocular’ ή για ‘remote binocular’ εμφανίζεται ένα παράθυρο που σας ζητάει να ορίσετε πόσες πηγές υπερύθρου φωτός διαθέτετε (ένα ή δύο).



Εικόνα 54 : Επιλογή πηγών υπερύθρων

Εικόνα 55 : Ορισμός αριθμού και θέσεων πηγών υπερύθρων.

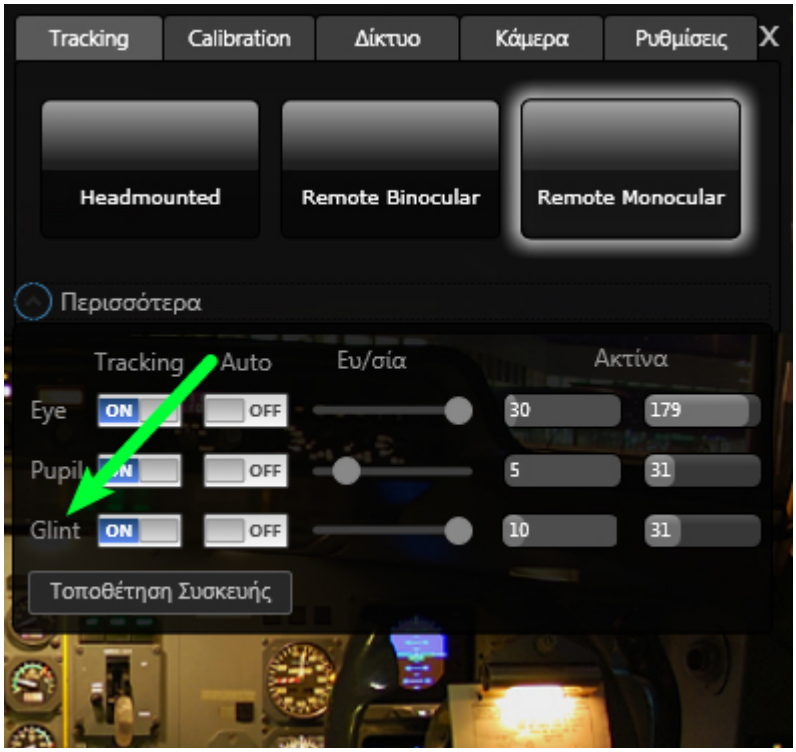
Σε αυτό το σημείο κάνοντας κλικ και κρατώντας το πατημένο, πάνω σε μία πηγή φωτός, όπως απεικονίζεται στην εικόνα 55, μπορείτε να σύρετε τα στοιχεία φωτός στην κατάλληλη θέση σε σχέση με την κάμερα αναλόγως με το πώς είναι η συσκευή της κάμερας και το που βρίσκονται οι υπέρυθρες λάμπες.

Χρησιμοποιώντας υπέρυθρες όμως δημιουργούνται αντανακλάσεις στο μάτι και μπορούν να βελτιώσουν σημαντικά το tracking. Όπως χρησιμοποιούμε σε μία απομακρυσμένη ρύθμιση όπου η κάμερα βρίσκεται μακριά από τον χρήστη.

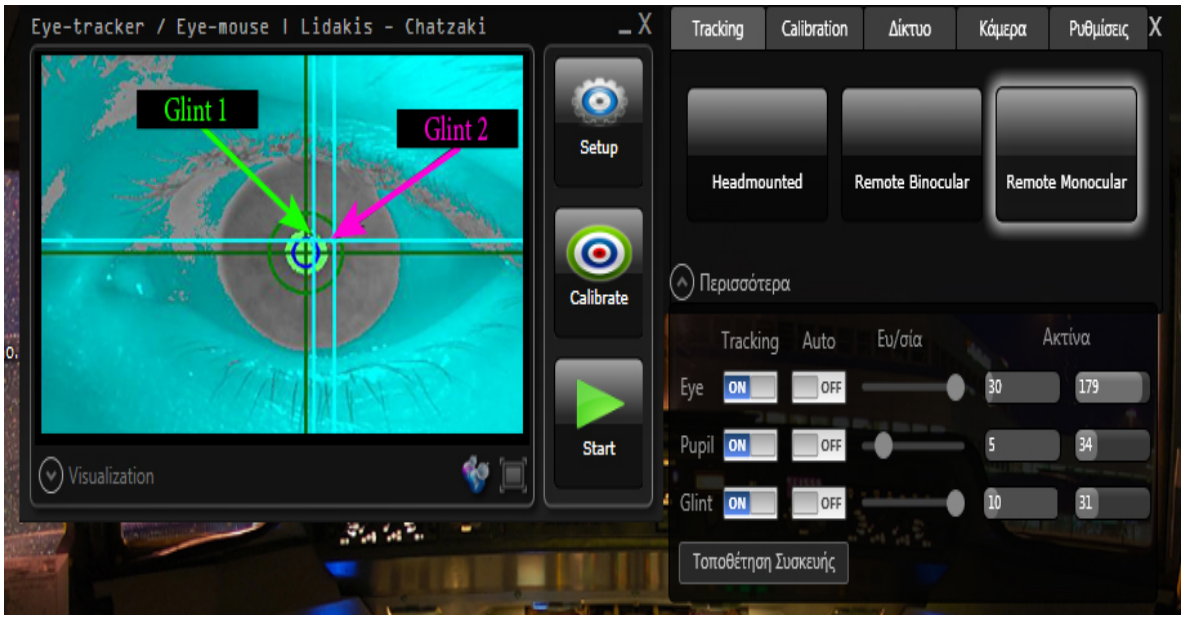
Το tracking με την μέθοδο της αντανάκλασης στην κόρη είναι πιο εξελιγμένο και το πρόγραμμα έχει σχεδιαστεί με τέτοιο τρόπο ώστε να υπάρχει ανοχή στις κινήσεις του κεφαλιού κατά την διάρκεια του tracking, αλλά ο υπέρυθρος φωτισμός πρέπει να είναι τοποθετημένος με τέτοιο τρόπο ώστε οι αντανακλάσεις να είναι ορατές και ευδιάκριτες στην κάμερα καθώς το μάτι κινείται σε όλο το εύρος της οθόνης.

Ρυθμίσεις για απομακρυσμένο tracking.

Βεβαιωθείτε ότι έχετε ενεργοποιήσει την επιλογή ‘glint’ από την ακτίνα μπορείτε να ρυθμίσετε τα glints, τα glints πρέπει να ταυτοποιηθούν από το πρόγραμμα ένα παράδειγμα δίδεται στις επόμενες δύο εικόνες (υπάρχει και η επιλογή auto για αυτόματη αναγνώριση).



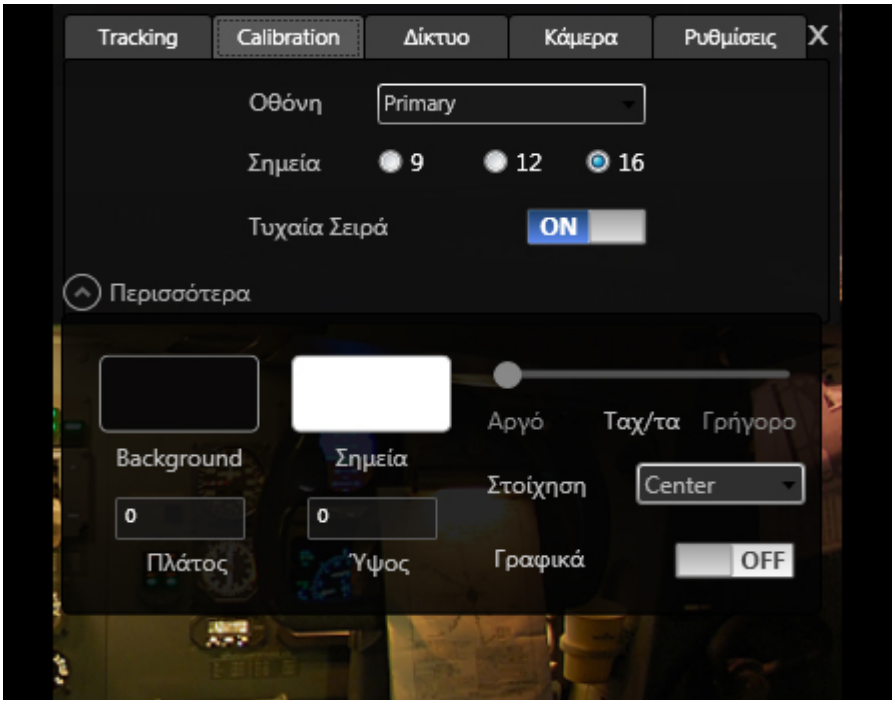
Εικόνα 56 : Ρύθμιση Glint



Εικόνα 57 : Παραμετροποιήση Glint

Η δεύτερη καρτέλα επιλογών της εφαρμογής , 'calibration' περιλαμβάνει τα στοιχεία και τις παραμέτρους που μπορούμε να εισάγουμε και είναι απαραίτητα για να εκτελεστεί το calibration σε σχέση με την οθόνη την κάμερα και το μάτι.

Εδώ μπορείτε να προσαρμόσετε την ταχύτητα αλλαγής των σημείων, πόσα σημεία να εμφανιστούν αν εμφανιστούν σε τυχαία σειρά η όχι, ακόμα και στοιχεία χρωμάτων του calibration screen. Η ποιότητα του calibration εξαρτάται από τον αριθμό εικόνων που παράγονται σε κάθε σημείο, δηλαδή από την κάμερα.



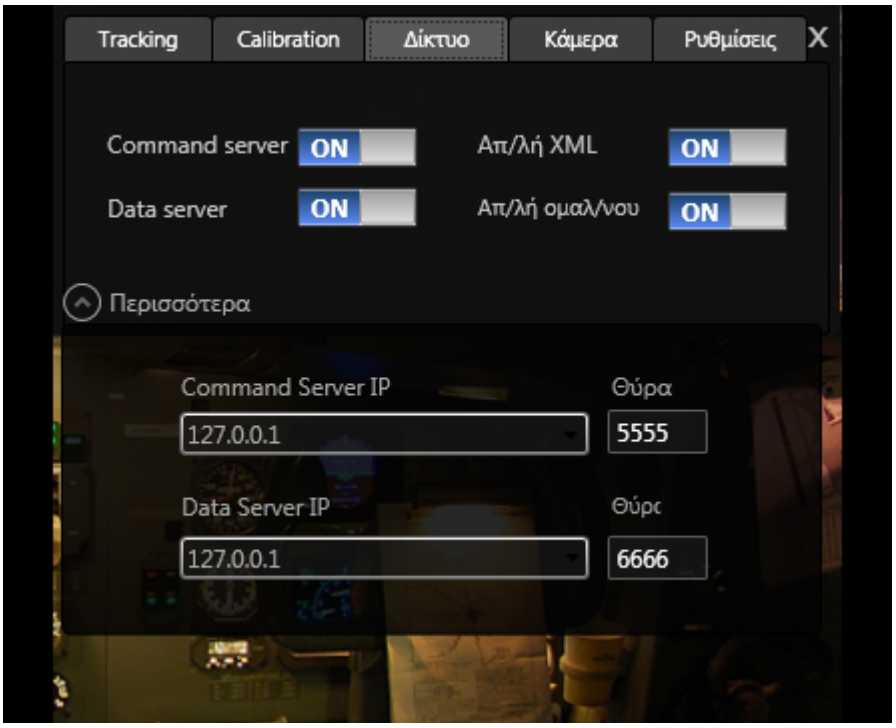
Εικόνα 58 : Καρτέλα επιλογών calibration.

Η τρίτη καρτέλα 'Δίκτυο' περιλαμβάνει ρυθμίσεις δικτύου. Για παράδειγμα η εφαρμογή μπορεί να εκτελείτε σε ένα διαφορετικό Η/Υ απ' ότι αυτόν που έχει την κάμερα και αυτή η λειτουργία αναπτύχθηκε

για να είναι η εφαρμογή ευέλικτη υπό πολλαπλές συνθήκες αφού όπως έχει προαναφερθεί ο αρχικός σκοπός της εφαρμογής είναι να χρησιμοποιηθεί σε ανθρώπους με περιορισμένη κινητικότητα.

Ο Command Server είναι ο Η/Υ όπου τρέχει η εφαρμογή και ο Data Server ο Η/Υ όπου υπάρχει η κάμερα και τα δεδομένα συλλέγονται. Επιπλέον υπάρχει η επιλογή να γίνει αποστολή των δεδομένων tracking του ματιού σε μορφή xml για άλλες χρήσεις για περεταίρω ανάλυση όπως επίσης και η αποστολή των δεδομένων αυτών κατόπιν διαδικασίας smoothing (ομαλοποίηση).

Σε περίπτωση που ο Η/Υ που τρέχει την εφαρμογή διαθέτει και την κάμερα τότε η ip που πρέπει να βάλουμε και στα δύο πεδία όπως φαίνεται στην παρακάτω εικόνα είναι η localhost δηλαδή η 127.0.0.1. Όσο αφορά τις θύρες απλά δεν πρέπει να συμπίπτουν και να μην είναι θύρες που χρησιμοποιούν άλλες εφαρμογές (π.χ 80, 25, 11536 κ.λ.π), συνιστώμενες θύρες είναι οι 5555 και 6666.

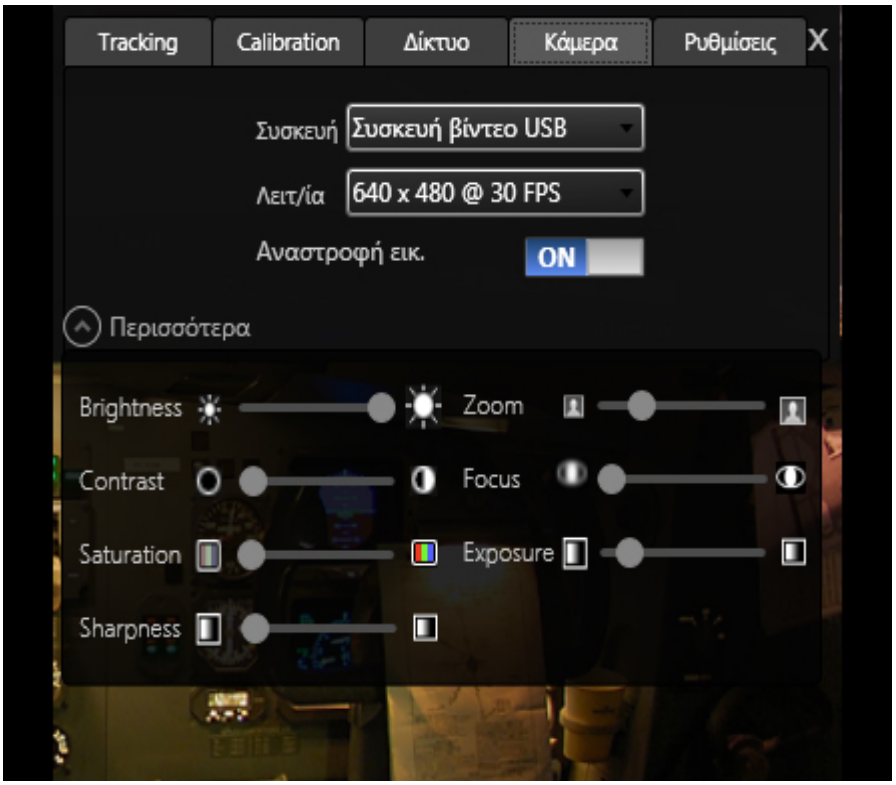


Εικονα 59 : Ρυθμίσεις Δικτύου**.**

Η τέταρτη κατά σειρά καρτέλα ‘κάμερα’ του παραθύρου επιλογών της εφαρμογής μας περιλαμβάνει τις ρυθμίσεις της κάμερας που χρησιμοποιούνται κατά την διαδικασία λήψης διαδοχικών εικόνων.

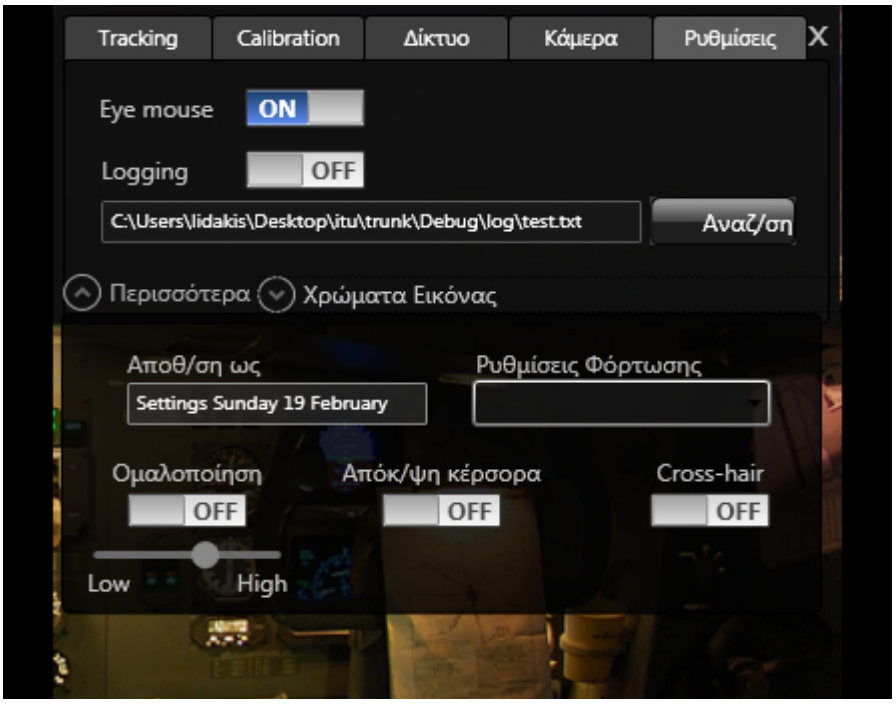
Όπως μπορείτε να δείτε μπορείτε να επιλέξετε συσκευή κάμερας σε ποια λειτουργία να είναι και άλλες διάφορες ρυθμίσεις στο κάτω μέρος του παραθύρου.

Η συνιστώμενη λειτουργία είναι η ανάλυση 640 χ 480 pixels και 30 frames ανά δευτερόλεπτο.

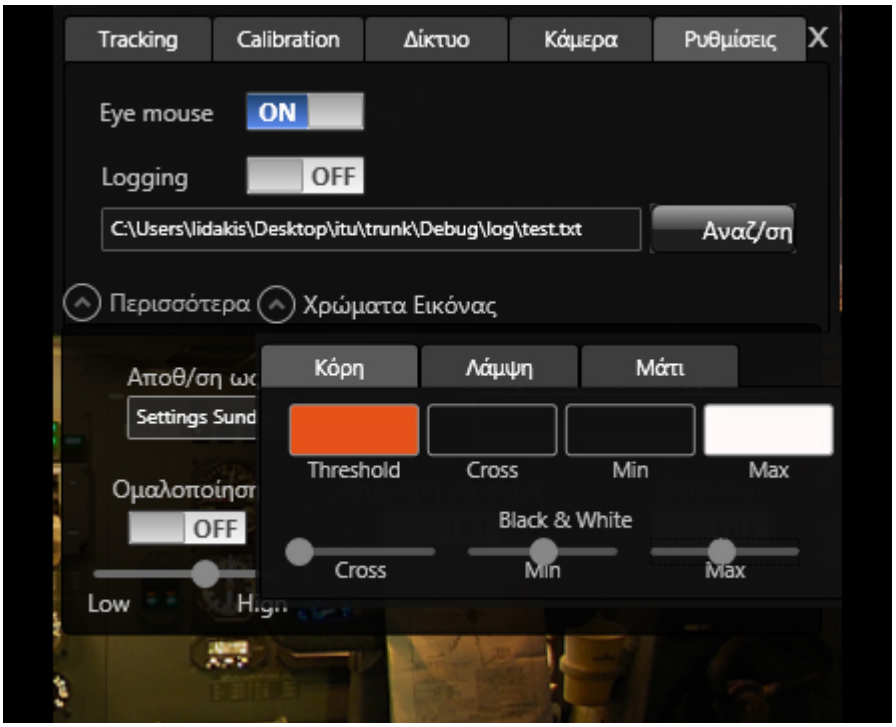


Εικόνα 60 : Ρυθμίσεις κάμερας.

Στην τελευταία καρτέλα υπάρχουν διάφορες ρυθμίσεις όπως λειτουργία η μή του eye-mouse , η εφαρμογή να κρατάει log, επιλογή για την ομαλοποίηση η μη των τιμών κίνησης κατά την διαδικασία του eye-tracking.
Επίσης υπάρχουν ρυθμίσεις για τους χρωματικούς συνδυασμούς που απεικονίζονται στην οθόνη κατά την διαδικασία αναγνώρισης και tracking του ματιού.

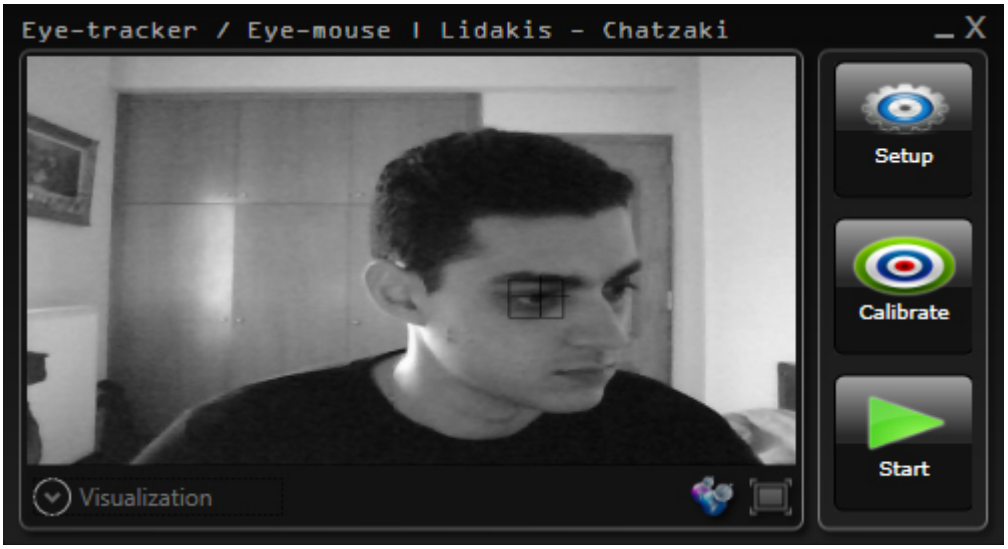


Εικόνα 61 : Ρυθμίσεις



Εικόνα 62 : Ρυθμίσεις

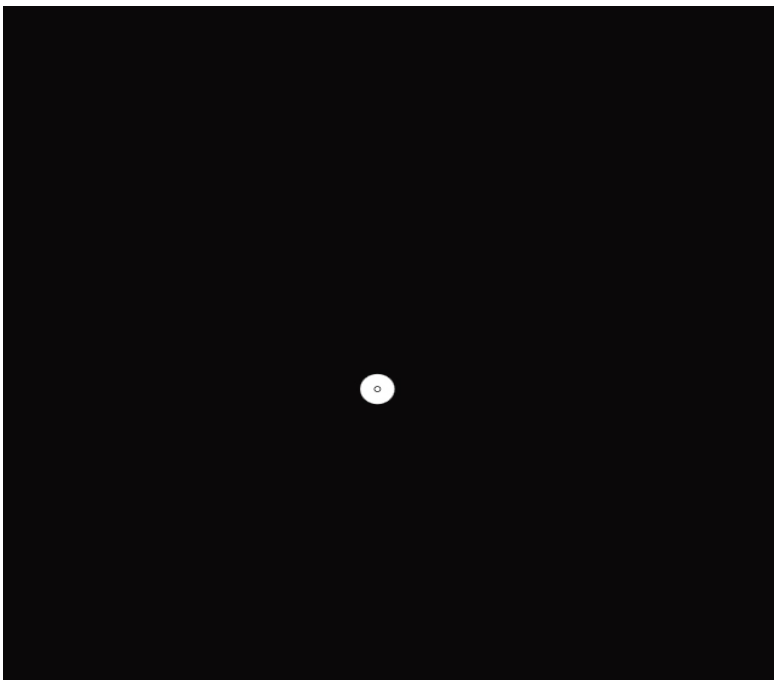
Επιστρέφοντας στο αρχικό παράθυρο αφού έχουμε εισάγει τις παραμέτρους μας.



Εικόνα 63 : Αρχικό παραθυρο

Κάνοντας κλικ στο κουμπί calibrate αυτό που θα εμφανιστεί θα είναι μία εικόνα με ένα σημείο που αλλάζει θέση, ο χρήστης πρέπει να ακολουθήσει με το βλέμμα του το σημείο έως ότου αυτό σταματήσει.

Η διαδικασία του calibration χρησιμοποιείται για να αποδοθεί το mapping μεταξύ της θέσης της κόρης του ματιού και των συντεταγμένων της οθόνης. Κατά την διαδικασία του calibration είναι σημαντικό να κοιτάμε πάνω στις κουκκίδες και να τις ακολουθούμε στην οθόνη καθ' όλη την διάρκεια που τρέχει το calibration. Πρέπει να σημειωθεί ότι οποιαδήποτε κίνηση του κεφαλιού ή της συσκευής σε σχέση με την κόρη μπορεί να μειώσει την ακρίβεια στο calibration

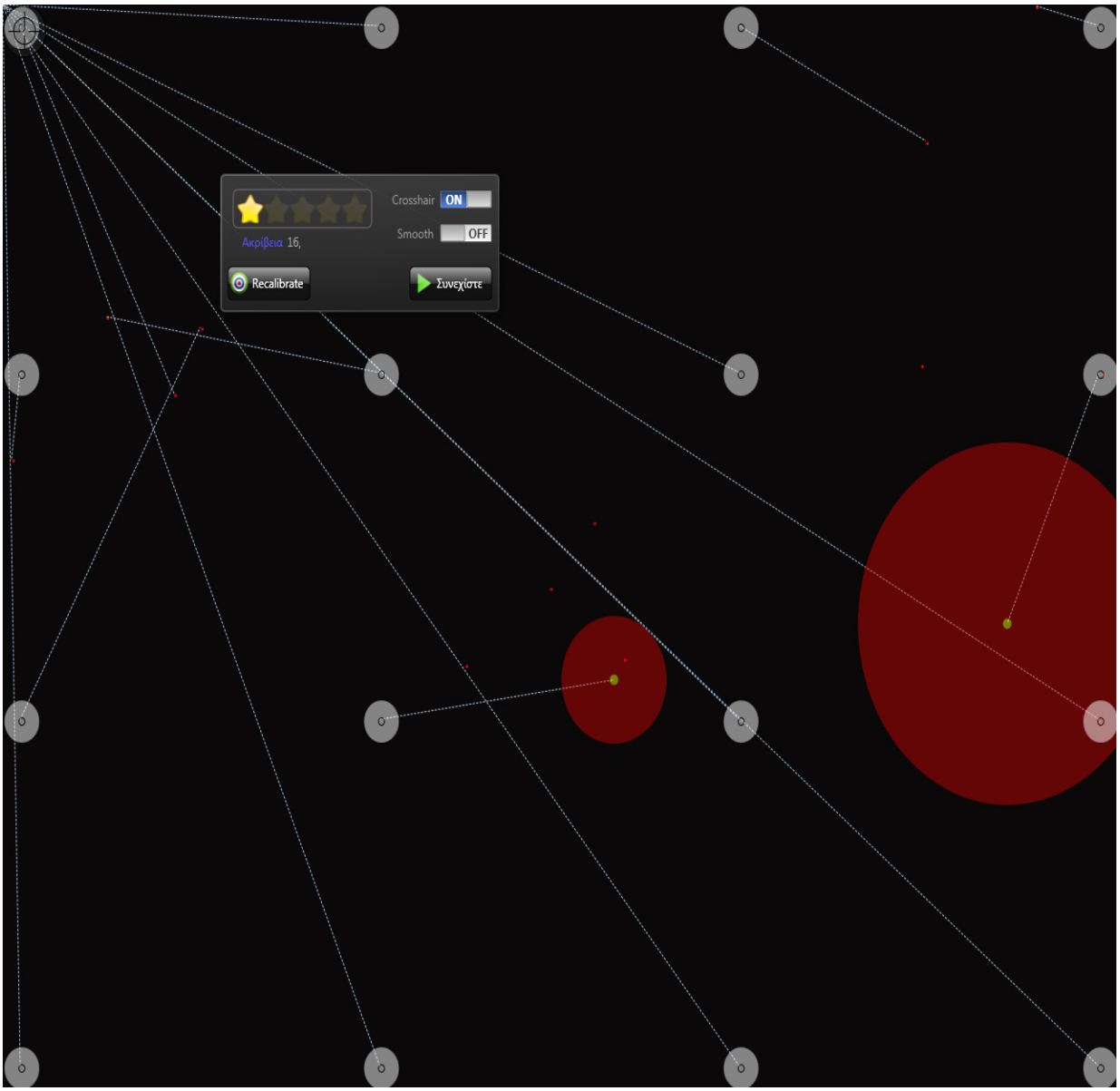


Εικόνα 64 : Κατά το calibration

Αφού περατωθεί η διαδικασία του calibration μπορείτε να δείτε την ακρίβεια. Επίσης θα δείτε ένα 'crosshair' όπως αποκαλείται ή γραμμές μεταξύ των σημείων και μπορείτε να παρατηρήσετε την ακρίβεια και την μετατόπιση.

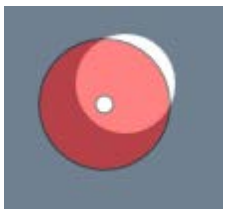
Η ποιότητα του calibration προβάλλεται με ένα σύστημα 5 αστέρων ανάλογα με το πόσο καλή είναι και με αριθμό στην κλίμακα από 0 – 100. Η ποιότητα στην οθόνη αναπαριστάνεται από το μέγεθος και το χρώμα των κύκλων όπως επίσης και την απόσταση μεταξύ των σημείων στόχευσης.

Ο μεγάλος κόκκινος κύκλος παριστάνει μεγάλη μεταβλητότητα μεταξύ των εικόνων δειγματοληψίας για αυτό το σημείο. Η διακεκομμένη γραμμή υποδεικνύει την μετατόπιση μεταξύ του σημείου στοχεύσεως και της εκτιμώμενης θέσης της ίριδας.

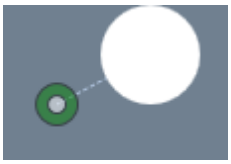


Εικόνα 65 : Κατά τον τερματισμό του calibration

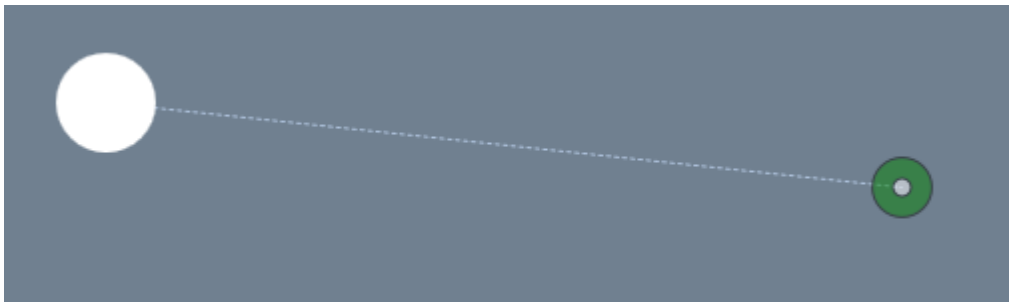
Καταστάσεις μετατόπισης μετά το calibration.



Εικόνα 66 : Μικρή μετατόπιση (ιδανικό).



Εικόνα 67: Μεσαία μετατόπιση (όχι τόσο καλό).



Εικόνα 68: Μεγάλη μετατόπιση (πιέστε recalibrate – μη αποδεκτό).

6.8 Περιγραφή εφαρμογής σε επίπεδο προγραμματισμού.

6.8.1 Εισαγωγή

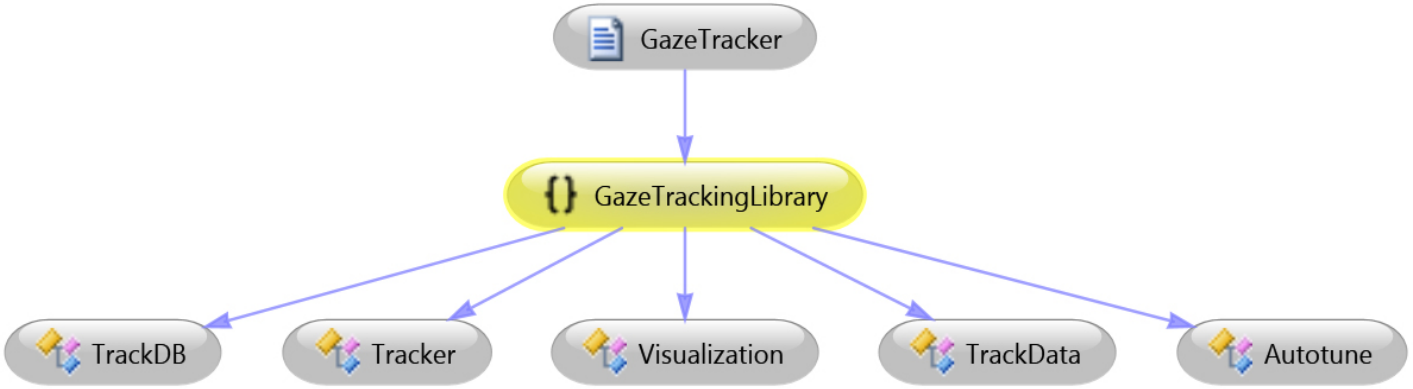
Σε αυτήν την ενότητα θα περιγράψουμε συνοπτικά την λειτουργία της εφαρμογής. Είναι σημαντικό να αναφέρουμε ξανά ότι η εφαρμογή είναι ανοικτού κώδικα, είναι βασισμένη στο ITU gaze tracker, για την πρόβλεψη των σημείων στο tracking έχουμε χρησιμοποιήσει την μέθοδο της παρεμβολής (interpolation) όπου η μέθοδος υλοποιείται στον κώδικα και ο κώδικας παραθέτεται στο παράρτημα β' με τίτλο interpolation.

Ο κώδικας που έχουμε αναπτύξει για την εν λόγω εφαρμογή, και οι αλλαγές του gaze tracking algorithm όπως και οι μέθοδοι που έχουν αναπτυχθεί για την παρούσα πτυχιακή εργασία, υπάρχουν στο online καταθετήριο κώδικα sourceforge στην διεύθυνση "*sourceforge.com/LidakisChatzakiDiplomacy*".

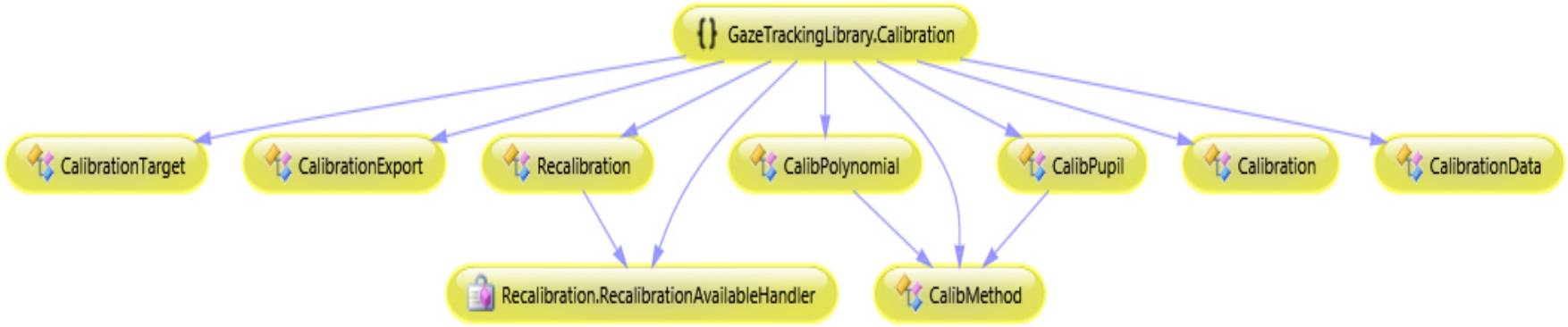
Γενικότερα οι αλληλεπιδράσεις των σημαντικότερων κλάσεων και μεθόδων της εφαρμογής περιγράφονται παρακάτω διαγραμματικά. Στο παράρτημα β' παρατίθεται ο κώδικας των σημαντικότερων σημείων που αναπτύχθηκαν και αφορούν το tracking. Ο περαιτέρω κώδικας υλοποίησης της εφαρμογής δεν περιλαμβάνεται στο παρόν βιβλίο λόγω του ότι δεν είναι στο θέμα της παρούσας πτυχιακής και λόγω του μεγάλου όγκου του.

Μπορείτε όμως να βρείτε ολόκληρο τον πηγαίο κώδικα του προγράμματος στον συνοδευόμενο δίσκο (cd) στον φάκελο 'source' αλλά και στην ηλεκτρονική διεύθυνση "*http://www.sourceforge.com/LidakisChatzakiDiplomacy*".

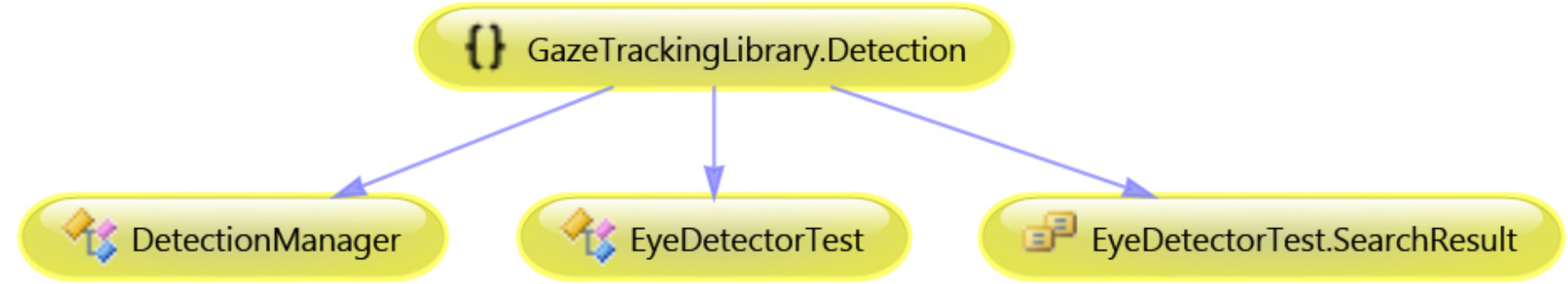
Διαγραμματική αναπαράσταση αλληλεπίδρασης των σημαντικότερων κλάσεων και μεθόδων της εφαρμογής.



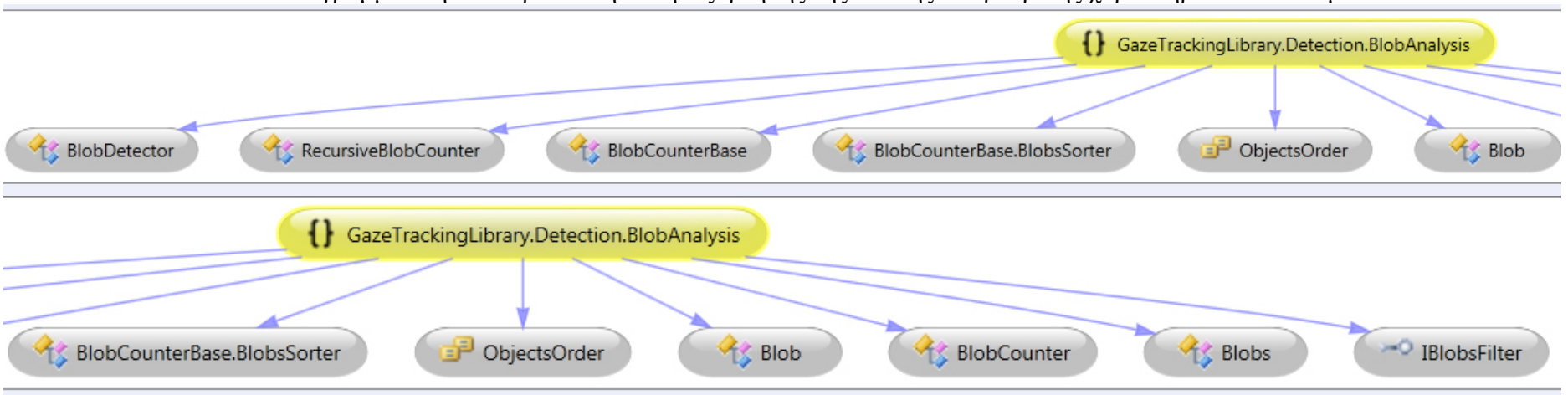
Εικόνα 69 : Διαγραμματική αναπαράσταση αλληλεξάρτησης του gaze tracking library.



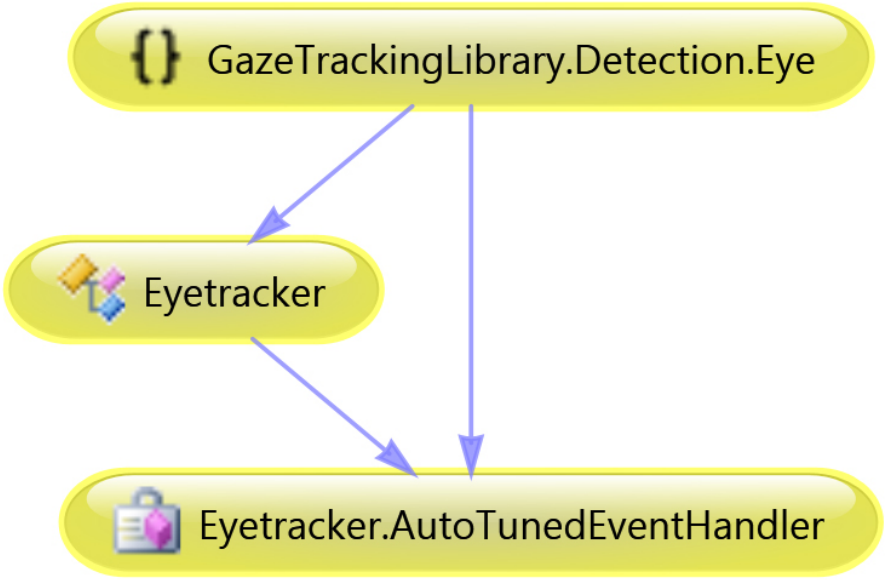
Εικόνα 70 : Διαγραμματική αναπαράσταση αλληλεξάρτησης του calibration method.



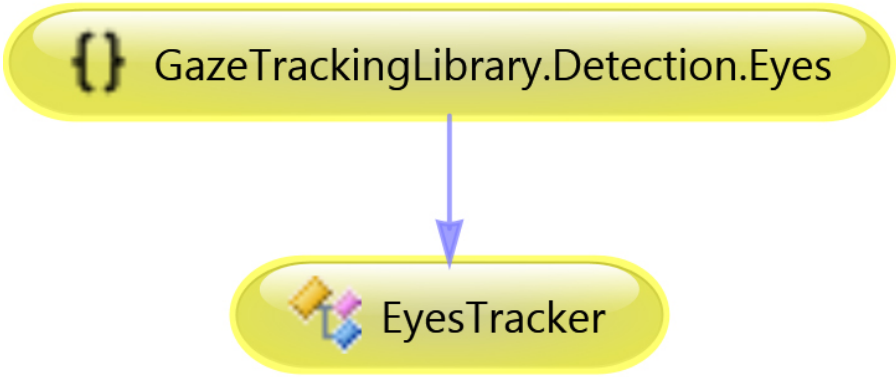
Εικόνα 71: Διαγραμματική αναπαράσταση αλληλεξάρτησης της κλάσης αναγνώρισης χαρακτηριστικών του ματιού.



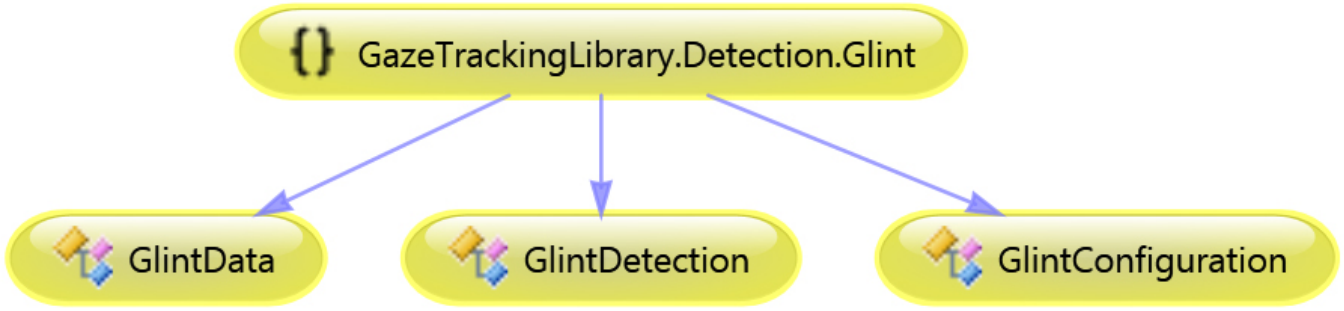
Εικόνα72: Διαγραμματική αναπαράσταση αλληλεξάρτησης της κλάσης ανάλυσης κυκλικώς χαρακτηριστικών.



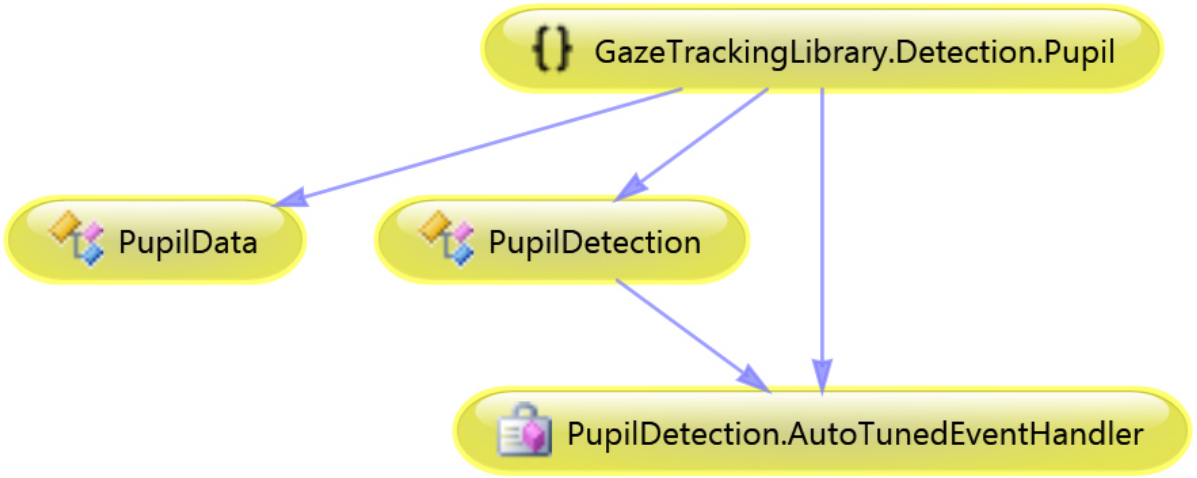
Εικόνα 73 : Διαγραμματική αναπαράσταση αλληλεξάρτησης της κλάσης αναγνώρισης του οφθαλμού



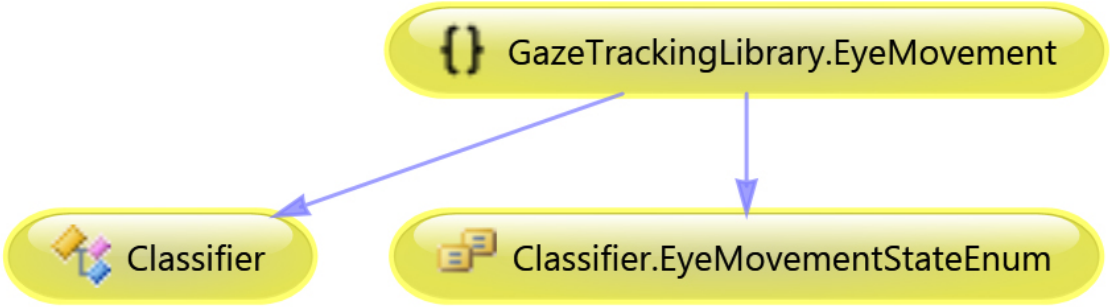
Εικόνα 74: Διαγραμματική αναπαράσταση αλληλεξάρτησης της κλάσης αναγνώρισης του οφθαλμών.



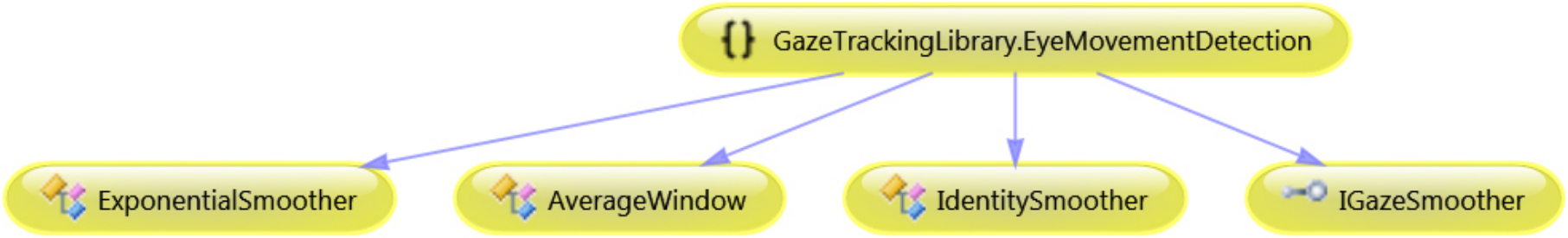
Εικόνα 75 : Διαγραμματική αναπαράσταση αλληλεξάρτησης της κλάσης αναγνώρισης αντανακλάσεων.



Εικόνα 76 : Διαγραμματική αναπαράσταση αλληλεξάρτησης της κλάσης αναγνώρισης κόρης οφθαλμού.

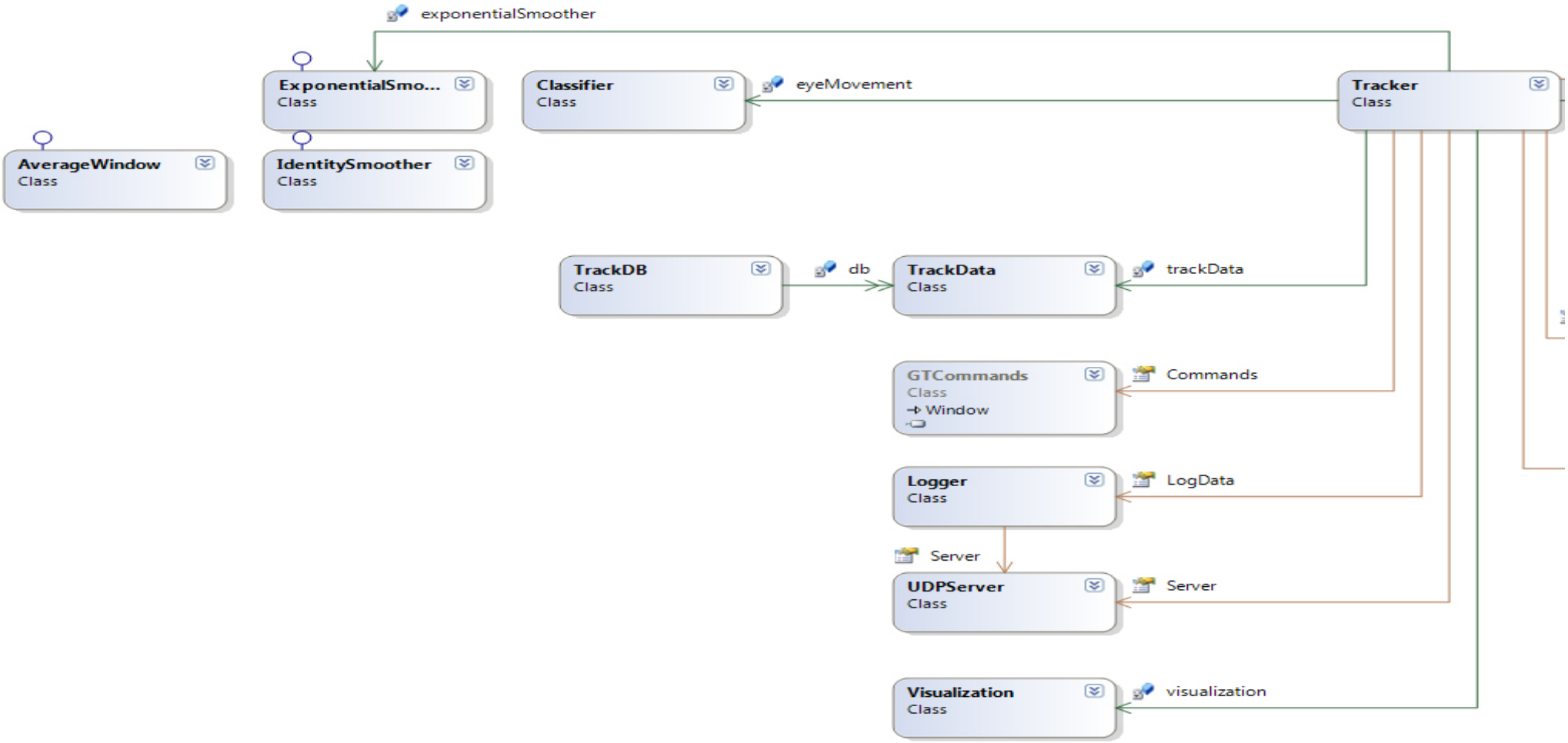


Εικόνα 77: Διαγραμματική αναπαράσταση αλληλεξάρτησης της κλάσης διαχείρισης κινήσεων του ματιού.

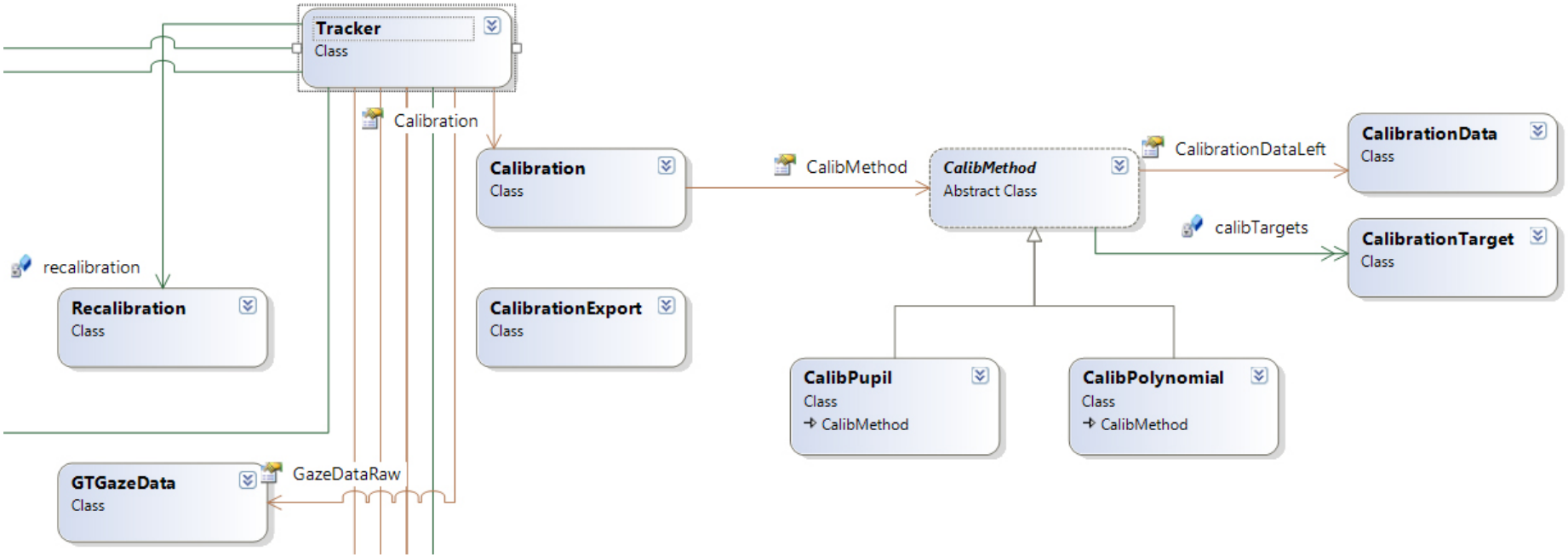


Εικόνα 78 : Διαγραμματική αναπαράσταση αλληλεξάρτησης της κλάσης αναγνώρισης κινήσεων του ματιού.

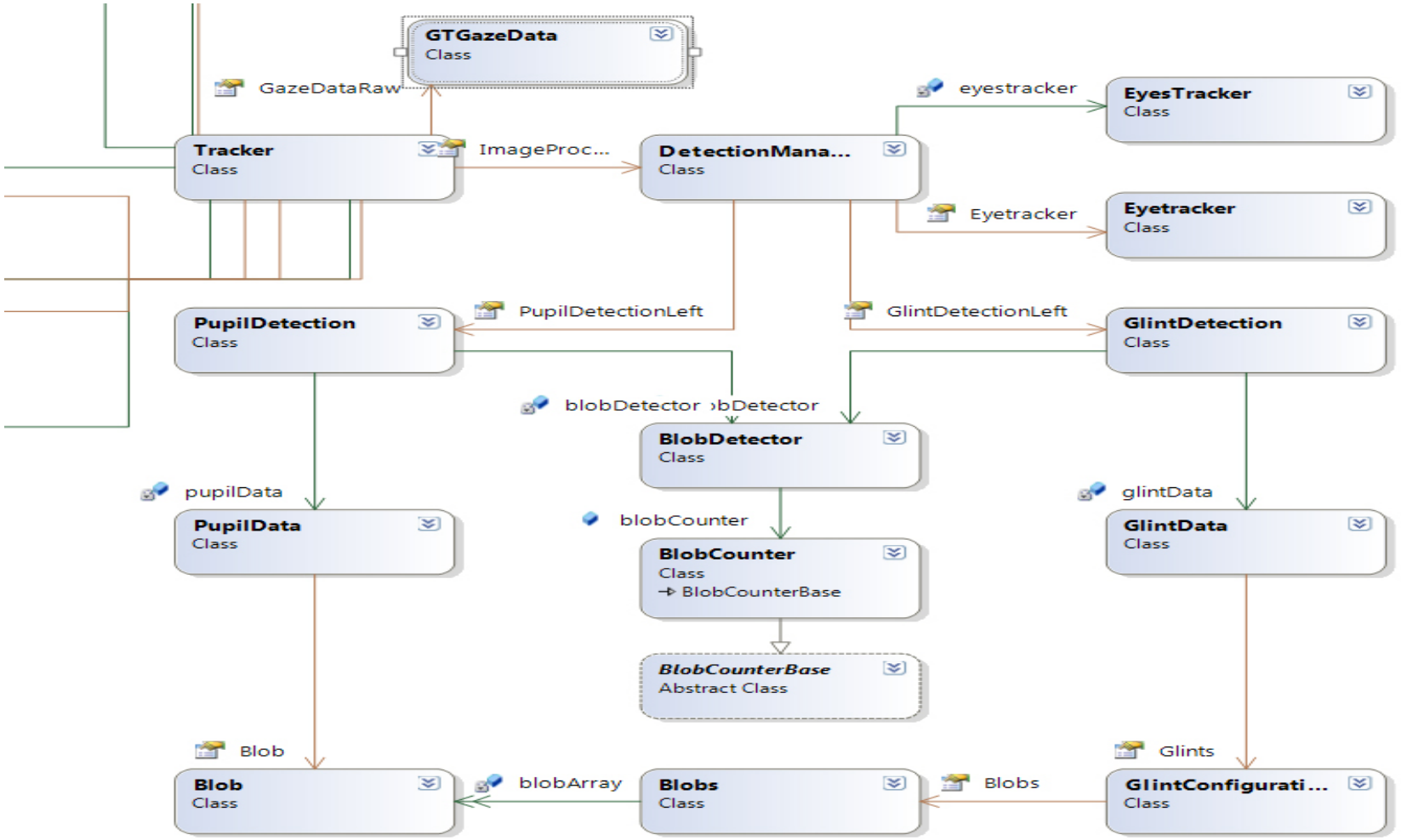
Ακολουθέι ένα εκτενέστερο διάγραμμα των σημαντικότερων κλάσεων της εφαρμογής ,παρατίθεται παρακάτω σε τρία μέρη
Κύρια κλάση, στο ακόλουθο διάγραμμα τριών μερών, αποτελεί η ‘Tracker’.



Εικόνα 79 : Διαγραμματική αναπαράσταση αλληλεξάρτησης σημαντικότερων κλάσεων μέρος 1ο.



Εικόνα 80 : Διαγραμματική αναπαράσταση αλληλεξάρτησης σημαντικότερων κλάσεων μέρος 2ο.



Εικόνα 81: Διαγραμματική αναπαράσταση αλληλεξάρτησης σημαντικότερων κλάσεων μέρος 3ο.

Συμπεράσματα

Τα συμπεράσματα που προκύπτουν από την ανάπτυξη και την ανάλυση της εφαρμογής είναι ότι ενσωματόνωντας και συνδυάζοντας μεθόδους και αλγόριθμους ήδη γνώστων τεχνικών δημιουργήθηκε μία ευέλικτη εφαρμογή με τεράστιες προοπτικές που δύναται να χρησιμοποιηθεί για διαφόρων ειδών χρήσεις από ένα μεγάλο εύρος χρηστών, και σαφώς ανταποκρίνεται απόλυτα στον τελικό σκοπό της. Η εφαρμογή έχει δυνατότητες επέκτασης από διάφορους χρήστες για διάφορους σκοπούς και δεν περιορίζει τον εκάστοτε προγραμματιστή για την ανάπτυξη του παραπέρα.Επιπλέον διανείμενετε σαν General Public Liscence Under Supervision και βρίσκεται στο repository του sourceforge.

Μελλοντική Εργασία και Επεκτάσεις

Το γεγονός ότι το σύστημα εντοπισμού και καταγραφής οφθαλμού μπορεί άνετα να αντικαταστήσει συσκευές εισόδου του υπολογιστή όπως πληκτρολόγιο ,ποντίκι αλλά και το ότι μπορεί να μάς εξάγει ως αποτέλεσμα χαρτογράφηση των σημείων όπου μετατοπίστηκε η προσοχή του χρήστη , μας δίνει την δυνατότητα πολλαπλών επεκτάσεων του συστήματος.

Αφού γίνει πρώτα εξέλιξη του συστήματος για ελαχιστοποίηση τυχόν λαθών , θα μπορούσε να συνεργαστεί – συνδεθεί με διάφορα αλλά προγράμματα έχοντας ως σκοπό την διεπαφή χρήστη υπολογιστή.

Ιδιαίτερα για τα άτομα με ειδικές ανάγκες θα μπορούσαμε να δημιουργήσουμε εφαρμογές οι οποίες θα συνεργάζονται με το σύστημα εντοπισμού και καταγραφής οφθαλμού , όπως πληκτρολόγιο οθόνης ώστε να τους παρέχεται η δυνατότητα γραφής, στην ιδία εφαρμογή μπορούμε να συνδέσουμε ακόμα ένα λογισμικό που μετατρέπει το κείμενο σε ομιλία. Αν τώρα ο χρήστης δεν είναι σε θέση να κινάει με ευκολία τα μάτια του, το πληκτρολόγιο θα μπορούσε να αντικατασταθεί από μια πολύ απλή εφαρμογή ειδοποίησης βασικών αναγκών ενός παραθύρου που θα περιέχει έτοιμα μηνύματα τα οποία θα διαβάζονται από το λογισμικό text to speech.

Μια άλλη εφαρμογή είναι αυτή του eye- painter.

Άλλη εφαρμογή ιδιαίτερα φιλόδοξη αφορά την εκπαίδευση των φοιτητών ιατρικής σε χειρουργικές επεμβάσεις . Ένας δεύτερος γιατρός ο όποιος θα έχει τοποθετημένη μια συσκευή στο κεφάλι του θα παρακολουθεί την επέμβαση από κοντά δίνοντας την δυνατότητα στους ενδιαφερομένους να παρακολουθούν την επέμβαση μέσο μιας οθόνης με κάθε δυνατή λεπτομέρεια.

Γενικά είναι ένα σύστημα το όποιο μπορεί να συμπεριληφθεί σε αρκετές εφαρμογές αφού μπορεί να αντικαταστήσει κάποια κύρια μέρη του υπολογιστή

Βιβλιογραφία

1.Διεπαφή Χρήστη υπολογιστή , Ακουμιανάκης Δημοσθένης

2..Arai M., Shimoshakai H., Watanabe H., Takei S., Recognition of Eye Direction Using Seperability Filters in Touch-type Training, Proceedings of the International Conference on Computers in Education IEEE, 2002.

3.Cleveland D., Cleveland N, Eyegaze eyetracking system. Imagina: Images Beyond Imagination. 11th Monte Carlo International Forum on New Images, Virginia USA

4.Deng J., Lai F., Region-Based Template Deformation and Masking for Eye-Feature Extraction and Description, Pattern Recognition, Vol. 30, No. 3, pp. 403-419, 1997.

5.Jacob R.J.K., Eye Tracking in Advanced Interface Design. Barfield W. & Furness T.A. (Eds.), Virtual Environments and Advanced Interface Design. Oxford University Press (1995), pp. 258-288

6.Jang J., Kim K., Lee Y., Efficient Algorithm of Eye Image Check for Robust Iris Recognition System. Lecture Notes in Computer Science 2756, (Jan 2003), pp. 301 – 308

7.J. Deng and F. Lai. Regionbased template deformation and masking for eyefeature extraction and description. *Pattern Recognition*, 30(3):403–419, 1997.

8.Miriam Spering et.al. Effects of contrast on smooth pursuit eye movements. Journal of Vision (2005),5 455-465

9.Carpenter, R. H. S. Movements of the Eyes (Pion, London,1988)

10.R.V. Abadi. Characteristics of saccadic intrusions. Vision Research 44 2675–2690 (2004)

11.Commercial uses of eyetracking By James Anthony Renshaw

12. Studying web pages using eye tracking By Kirk Ewing, Tobii Technology (2005)

Παράρτημα Ά

Ανάλυση κώδικα αρχικού προγράμματος.

Για την υλοποίηση του συστήματος καταγραφής κίνησης οφθαλμού αναπτύχθηκαν, σε γλώσσα προγραμματισμού c++ οι παρακάτω συναρτήσεις και λειτουργίες.

BlinkDetector – Calibrator – Containers – EyeExtractor – FaceDetector - FeatureDetector FMatrixAffineCompute – GaussianProcess – GazeArea – GazeTracker – GraphicalPointer – GazeTrackerGtk – GtkStore – HeadCompensation – HeadTracker – LeastSquares – MainGazeTracker – Gazer – OutputMethods – Point – PointTracker – TrackingSystem –Utils.

Παρακάτω παραθέτεται ο κώδικας του λογισμικού, με μία συνοπτική περιγραφή του κάθε αρχείου κώδικα.

BlinkDetector.cpp

Η μέθοδος BlinkDetector χρησιμοποιείται για να ανιχνέυει το ανοιγόκλειμα του ματιού, ούτως ώστε να μην χάνεται η επαφή και η λειτουργία καταγραφής κίνησης του οφθαλμού κατά την διάρκεια εκτέλεσης του προγράμματος ενώ συμβαίνει το blink του οφθαλμού.

```
1.  #include "BlinkDetector.h"
2.  #include <iostream>
3.  #include "EyeExtractor.h"
4.  #include "utils.h"
5.
6.  StateNode::StateNode(int minduration, int maxduration, double threshold):
7.      minduration(minduration), maxduration(maxduration), threshold(threshold)
8.  {
9.  }
10.
11. bool StateNode::isSatisfied(double value) const {
12.     if (threshold > 0)
13.     return value >= threshold;
14.     else
15.     return value <= -threshold;
16. }
17. LinearStateSystem::LinearStateSystem(const vector<StateNode> &states):
18.     states(states), currentState(0), duration(0)
19. {
20.     assert(!states.empty());
21. }
```

```
22.
23. void LinearStateSystem::updateState(double value) {
24.     cout << "update state: " << value << endl;
25.     duration++;
26.     if (currentState > 0 &&
27.     ((duration < states[currentState].minduration &&
28.       !states[currentState].isSatisfied(value)) ||
29.      duration > states[currentState].maxduration))
30.     setState(0);
31.
32.     if (!isFinalState() && states[currentState+1].isSatisfied(value))
33.     setState(currentState+1);
34. }
35.
36. void LinearStateSystem::setState(int state) {
37.     currentState = state;
38.     duration = 0;
39. }
40.
41. int LinearStateSystem::getState() {
42.     return currentState;
43. }
44.
45. bool LinearStateSystem::isFinalState() {
46.     return currentState == (int)states.size() - 1;
47. }
48.
49. LambdaAccumulator::LambdaAccumulator(double lambda, double initial):
50.     lambda(lambda), current(initial)
51. {
52. }
53.
54. void LambdaAccumulator::update(double value) {
55.     current = (1-lambda)*current + lambda*value;
56. }
57.
58. double LambdaAccumulator::getValue() {
59.     return current;
60. }
61. BlinkDetector::BlinkDetector():
62.     averageEye(cvCreateImage(EyeExtractor::eyesize, IPL_DEPTH_32F, 1)),
63.     acc(0.1, 1000.0),
64.     states(constructStates())
65. {
66. }
67. vector<StateNode> BlinkDetector::constructStates() {
68.     vector<StateNode> states;
69.     states.push_back(StateNode(0, 0, +0.00));
```

```
70.     states.push_back(StateNode(1, 8, +1.50));
71.     states.push_back(StateNode(1, 8, -1.20)); // blink
72.     states.push_back(StateNode(1, 8, +1.50));
73.     states.push_back(StateNode(1, 8, -1.20)); // double blink
74.     return states;
75. }
76. void BlinkDetector::update(const scoped_ptr<IplImage> &eyefloat) {
77.     double distance = cvNorm(eyefloat.get(), averageEye.get(), CV_L2);
78.     acc.update(distance);
79.     cout << "update distance" << distance << " -> " << acc.getValue() << endl;
80.     states.updateState(distance / acc.getValue());
81.     cvRunningAvg(eyefloat.get(), averageEye.get(), 0.05);
82. }
83. int BlinkDetector::getState() {
84.     return states.getState();
85. }
```

BlinkDetector.h

```
1.  #pragma once
2.  #include "utils.h"
3.
4.  using namespace std;
5.  using namespace boost;
6.
7.  struct StateNode {
8.      int minduration, maxduration;
9.      double threshold;
10.     StateNode(int minduration, int maxduration, double threshold);
11.     bool isSatisfied(double value) const;
12. };
13.
14. class LinearStateSystem {
15.     const vector<StateNode> states;
16.     int currentState, duration;
17. public:
18.     LinearStateSystem(const vector<StateNode> &states);
19.     void updateState(double value);
20.     void setState(int state);
21.     int getState();
22.     bool isFinalState();
23. };
24.
25. class LambdaAccumulator {
26.     const double lambda;
27.     double current;
28.  public:
29.     LambdaAccumulator(double lambda, double initial);
```

```
30.     void update(double value);
31.     double getValue();
32. };
33.
34. class BlinkDetector {
35.     scoped_ptr<IplImage> averageEye;
36.     LambdaAccumulator acc;
37.     LinearStateSystem states;
38.     static vector<StateNode> constructStates();
39.  public:
40.     BlinkDetector();
41.     void update(const scoped_ptr<IplImage> &image);
42.     int getState();
43. };
44. /* template <class T> */
45. /* class TwoClassDetector { */
46. /*     const T class0, class1; */
47. /*  public: */
48. /*     TwoClassDetector(const T& class0, const T& class1):  */
49. /*     class0(class0), class1(class1) {} */
50.
51. /*     double classify(const T& object) { */
52. /*  double dist0 = distance(class0, object); */
53. /*  double dist1 = distance(class0, object); */
54. /*  return dist0 / (dist0 + dist1); */
55. /*     }; */
56. /* }; */
57.
58. /* template <class T, class Func> */
59. /* class Accumulator { */
60. /*     T accumulator; */
61. /*     Func func; */
62. /*  public: */
63. /*     Accumulator(const T& initial, const Func& func):  */
64. /*     accumulator(initial), func(func) */
65. /*     {}; */
66. /*     void update(const T& object) { */
67. /*  accumulator = func(accumulator, object); */
68. /*     } */
69. /*     T get() { */
70. /*  return accumulator; */
71. /*     } */
72. /* }; */
73. /* template <T> */
74. /* class GetFurtherObject { */
75. /*     T referenceobject; */
76. /*  public: */
77. /*     T operator()(const T& one, const T& two) { */
```

```
78. /*  if (distance(referenceobject, one) > distance(referenceobject, two)) */
79. /*     return one; */
80. /*  else */
81. /*     return two; */
82. /*    } */
83. /* }; */
84. /* Accumulator<shared_ptr<IplImage> >  */
85. /* blink(currentimage, GetFurtherObject(average)); */
86.
87. /* typedef TwoClassDetector<shared_ptr<IplImage> > BlinkDetector(average, blink.get()); */
```

Calibrator.cpp

Με την παρακάτω μέθοδο υλοποιούμε το calibration, δηλαδή την θέση του οφθαλμού σε σχέση με τα σημεία που υπάρχουν στην οθόνη, ούτως ώστε όταν λαμβάνει χώρα η καταγραφή κίνησης του οφθαλμού το πεδίο περιορισμού κίνησης να συμπίπτει με τα σημεία ή την ανάλυση της οθόνης που θέλουμε να ορίσουμε. Αυτό επιτυγχάνεται με την δημιουργία σημείων που δημιουργούμε πάνω στην οθόνη ή στο πεδίο περιορισμού κάποιας εφαρμογής βασιζόμενη στην τεχνολογία καταγραφής κίνησης του οφθαλμου (π.χ eye-keyboard).

```
1.  #include "Calibrator.h"
2.  Calibrator::~Calibrator() {
3.  #ifdef DEBUG
4.      cout << "Destroying calibrator" << endl;
5.  #endif
6.  }
7.
8.  FrameFunction::~FrameFunction() {
9.  #ifdef DEBUG
10.     cout << "Destroying framefunction" << endl;
11. #endif
12. }
13.
14. MovingTarget::MovingTarget(const int &frameno,
15.             const vector<Point>& points,
16.             const shared_ptr<WindowPointer> &pointer,
17.             int dwelltime):
18.     FrameFunction(frameno),
19.     points(points), dwelltime(dwelltime), pointer(pointer)
20. {
21. };
22.
23. MovingTarget::~MovingTarget() {
24.     int id = getFrame() / dwelltime;
25. }
26.
27. void MovingTarget::process() {
```

```
28.     if (active()) {
29.     int id = getPointNo();
30.     if (getPointFrame() == 1)
31.         pointer->setPosition((int)points[id].x, (int)points[id].y);
32.     }
33.     else
34.     detach();
35. }
36.
37. bool MovingTarget::active() {
38.     return getPointNo() < (int) points.size();
39. }
40.
41. int MovingTarget::getPointNo() {
42.     return getFrame() / dwelltime;
43. }
44.
45. int MovingTarget::getPointFrame() {
46.     return getFrame() % dwelltime;
47. }
48.
49. Calibrator::Calibrator(const int &framecount,
50.                 const shared_ptr<TrackingSystem> &trackingsystem,
51.                 const vector<Point>& points,
52.                 const shared_ptr<WindowPointer> &pointer,
53.                 int dwelltime):
54.     MovingTarget(framecount, points, pointer, dwelltime),
55.     trackingsystem(trackingsystem)
56. {
57.     trackingsystem->gazetracker.clear();
58.     // todo: remove all calibration points
59. }
60.
61.
62. void Calibrator::process() {
63.     if (active()) {
64.     int id = getPointNo();
65.     int frame = getPointFrame();
66.     if (frame == 1) // start
67.         averageeye.reset(new FeatureDetector(EyeExtractor::eyesize));
68.
69.     if (frame >= dwelltime/2) // middle
70.         averageeye->addSample(trackingsystem->eyex.eyefloat.get());
71.
72.     if (frame == dwelltime-1) { // end
73.         trackingsystem->gazetracker.
74.         addExemplar(points[id], averageeye->getMean().get(),
75.                 trackingsystem->eyex.eyegrey.get());
```

```
76.     }
77.     }
78.     MovingTarget::process();
79. }
80.
81. const Point Calibrator::defaultpointarr[] = {Point(0.5, 0.5),
82.                         Point(0.1, 0.5), Point(0.9, 0.5),
83.                         Point(0.5, 0.1), Point(0.5, 0.9),
84.                         Point(0.1, 0.1), Point(0.1, 0.9),
85.                         Point(0.9, 0.9), Point(0.9, 0.1),
86.                         Point(0.3, 0.3), Point(0.3, 0.7),
87.                         Point(0.7, 0.7), Point(0.7, 0.3)};
88.
89. vector<Point>
90. Calibrator::defaultpoints(Calibrator::defaultpointarr,
91.             Calibrator::defaultpointarr+
92.             (sizeof(Calibrator::defaultpointarr) /
93.              sizeof(Calibrator::defaultpointarr[0])));
94.
95. vector<Point> Calibrator::loadpoints(istream& in) {
96.     vector<Point> result;
97.
98.     for(;;) {
99.     double x, y;
100.                         in >> x >> y;
101.                         if (in.rdstate()) break; // break if any error
102.                         result.push_back(Point(x, y));
103.                         }
104.
105.                         return result;
106.                     }
107.
108.                     vector<Point> Calibrator::scaled(const vector<Point> &points,
109.                                         double x, double y)
110.                     {
111.                     //     double dx = x > y ? (x-y)/2 : 0.0;
112.                     //     double dy = y > x ? (y-x)/2 : 0.0;
113.                     //     double scale = x > y ? y : x;
114.
115.                         vector<Point> result;
116.
117.                         xforeach(iter, points)
118.                         result.push_back(Point(iter->x * x, iter->y * y));
119.                     //  result.push_back(Point(iter->x * scale + dx, iter->y * scale + dy));
120.
121.                         return result;
122.                     }
```

Calibrator.h

```
1.  #pragma once
2.  #include <glibmm.h>
3.  #include <gdkmm.h>
4.  #include "utils.h"
5.  #include "TrackingSystem.h"
6.  #include "Containers.h"
7.  #include "GraphicalPointer.h"
8.  #include "FeatureDetector.h"
9.
10. class FrameProcessing;
11.
12. class FrameFunction:
13. public Containee<FrameProcessing, FrameFunction>
14. {
15.     const int &frameno;
16.     int startframe;
17.  protected:
18.     int getFrame() { return frameno - startframe; }
19. public:
20.     FrameFunction(const int &frameno): frameno(frameno), startframe(frameno) {}
21.     virtual void process()=0;
22.     virtual ~FrameFunction();
23. };
24.
25. class FrameProcessing:
26. public ProcessContainer<FrameProcessing,FrameFunction> {};
27.
28. class MovingTarget: public FrameFunction {
29.     shared_ptr<WindowPointer> pointer;
30.  public:
31.     MovingTarget(const int &frameno,
32.         const vector<Point>& points,
33.         const shared_ptr<WindowPointer> &pointer,
34.         int dwelltime=20);
35.     virtual ~MovingTarget();
36.     virtual void process();
37.  protected:
38.     vector<Point> points;
39.     const int dwelltime;
40.     int getPointNo();
41.     int getPointFrame();
42.     bool active();
43. };
```

```
44.
45. class Calibrator: public MovingTarget {
46.     static const Point defaultpointarr[];
47.     shared_ptr<TrackingSystem> trackingsystem;
48.     scoped_ptr<FeatureDetector> averageeye;
49. public:
50.     static vector<Point> defaultpoints;
51.     static vector<Point> loadpoints(istream& in);
52.     Calibrator(const int &frameno,
53.             const shared_ptr<TrackingSystem> &trackingsystem,
54.             const vector<Point>& points,
55.             const shared_ptr<WindowPointer> &pointer,
56.             int dwelltime=20);
57.     virtual ~Calibrator();
58.     virtual void process();
59.     static vector<Point> scaled(const vector<Point>& points, double x, double y);
60. };
```

Containers.cpp

Το container της εφαρμογής.

```
1.  #include "Containers.h"
2.  #include <iostream>
```

Containers.h

```
1.  #pragma once
2.  #include "utils.h"
3.  #include <glibmm.h>
4.  #include <vector>
5.  #define xforeachactive(iter,container) \
6.      for(typeof(container.begin()) iter = container.begin(); \
7.      iter != container.end(); iter++)            \
8.      if ((*iter)->parent == this)
9.  template <class ParentType, class ChildType> class Container;
10.
11. template <class ParentType, class ChildType>
12. class  Containee {
13.  protected:
14.     void detach() { parent = 0; }
15.  public:
16.     ParentType *parent;     /* set to null to request removal */
17.     Containee(): parent(0) {}
18.     virtual ~Containee() {}
19. };
```

```
20. template <class ParentType, class ChildType>
21. class Container {
22.     typedef shared_ptr<ChildType> ChildPtr;
23.     static bool isFinished(const ChildPtr &object) {
24.     return !(object && object->parent);
25.     }
26.  protected:
27.     std::vector<ChildPtr> objects;
28.     void removeFinished() {
29.     objects.erase(remove_if(objects.begin(), objects.end(), isFinished),
30.             objects.end());
31.     }
32.  public:
33.     void clear() {
34.     xforeachactive(iter, objects)
35.         (*iter)->parent = 0;
36.     removeFinished();
37.     }
38.
39.     static void addchild(ParentType *parent, const ChildPtr &child) {
40.     parent->objects.push_back(child);
41.     child->parent = parent;
42.     parent->removeFinished();
43.     }
44.
45.     virtual ~Container() {
46.     clear();
47.     }
48. };
49.
50. template <class ParentPtr, class ChildPtr>
51. class ProcessContainer: public Container<ParentPtr, ChildPtr> {
52.  public:
53.     virtual void process() {
54.     xforeachactive(iter, this->objects)
55.         (*iter)->process();
56.     this->removeFinished();
57.     }
58.     virtual ~ProcessContainer() {};
59. };
```

EyeExtraxtor.cpp

Σε αυτό το σημείο ανιχνεύουμε το εσωτερικού του ματιού χρησιμοποιώντας την μέθοδο color extraction, δηλαδή την χρωματική αλλαγή,στην κλίμακα του γκρι που υπάρχει από την κόρη του ματιού με το υπόλοιπο εσωτερικού του ματιου, και εφαρμόζοντας ένα Gaussian φίλτρο για την ομαλή μετάβαση ανίχνευσης στην περιοχές αλλαγής χρώματος – φωτεινότητας.

```
1.  #include "utils.h"
2.  #include "EyeExtractor.h"
3.  const int EyeExtractor::eyedx = 32;
4.  const int EyeExtractor::eyedy = 16;
5.  const CvSize EyeExtractor::eyesize = cvSize(eyedx*2, eyedy*2);
6.
7.  void EyeExtractor::processEye(void) {
8.      cvConvertScale(eyegrey.get(), eyefloat2.get());
9.      // todo: equalize it somehow first!
10.     cvSmooth(eyefloat2.get(), eyefloat.get(), CV_GAUSSIAN, 3);
11.     cvEqualizeHist(eyegrey.get(), eyegrey.get());
12. }
13. EyeExtractor::EyeExtractor(const PointTracker &tracker):
14.     tracker(tracker),
15.     eyefloat2(cvCreateImage( eyesize, IPL_DEPTH_32F, 1 )),
16.     eyegrey(cvCreateImage( eyesize, 8, 1 )),
17.     eyefloat(cvCreateImage( eyesize, IPL_DEPTH_32F, 1 )),
18.     eyeimage(cvCreateImage( eyesize, 8, 3 ))
19. {
20. }
21. void EyeExtractor::extractEye(const IplImage *origimage)
22.     throw (TrackingException)
23. {
24.     if (!tracker.status[tracker.eyepoint1])
25.     throw TrackingException();
26.     double x0 = tracker.currentpoints[tracker.eyepoint1].x;
27.     double y0 = tracker.currentpoints[tracker.eyepoint1].y;
28.     double x1 = tracker.currentpoints[tracker.eyepoint2].x;
29.     double y1 = tracker.currentpoints[tracker.eyepoint2].y;
30.     double factor = 0.17;
31.     double xfactor = 0.05;
32.     double yfactor = 0.20 * (x0 < x1 ? -1 : 1);
33.     double L = factor / eyedx;
34.     double LL = x0 < x1? L : -L;
35.     float matrix[6] =
36.     {LL*(x1-x0), LL*(y0-y1),
37.      x0 + factor * ((1-xfactor)*(x1-x0) + yfactor * (y0-y1)),
38.      LL*(y1-y0), LL*(x1-x0),
39.      y0 + factor * ((1-xfactor)*(y1-y0) + yfactor * (x1-x0))};
40.     CvMat M = cvMat( 2, 3, CV_32F, matrix );
41.     cvGetQuadrangleSubPix( origimage, eyeimage.get(), &M);
42.     cvCvtColor(eyeimage.get(), eyegrey.get(), CV_RGB2GRAY);
43.     processEye();
44. }
45. EyeExtractor::~EyeExtractor(void) {
46. }
```

EyeExtractor.h

```
1.  #pragma once
2.  #include "utils.h"
3.  #include "PointTracker.h"
4.
5.  class EyeExtractor {
6.      const PointTracker &tracker; /* dangerous */
7.      scoped_ptr<IplImage> eyefloat2;
8.
9.      void processEye(void);
10.
11. public:
12.     static const int eyedx;
13.     static const int eyedy;
14.     static const CvSize eyesize;
15.
16.     scoped_ptr<IplImage> eyegrey, eyefloat, eyeimage;
17.
18.     EyeExtractor(const PointTracker &tracker);
19.     void extractEye(const IplImage *origimage) throw (TrackingException);
20.     ~EyeExtractor(void);
21. };
```

FaceDetector.cpp

Ανίχνευση προσώπου (αφού η κάμερα είναι σε απόσταση από το πρόσωπό μας αρχικά πρέπει να ανιχνεύσουμε το προσώπο).

```
1.  #include "FaceDetector.h"
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.  #include <string.h>
5.  #include <assert.h>
6.  #include <math.h>
7.  #include <float.h>
8.  #include <limits.h>
9.  #include <time.h>
10. #include <ctype.h>
11.
12. FaceDetector FaceDetector::facedetector;
13.
14. FaceDetector::FaceDetector(char *cascadename):
15.     cascade((CvHaarClassifierCascade*)cvLoad(cascadename, 0, 0, 0)),
16.     storage(cvCreateMemStorage(0))
17. {}
18. FaceDetector::~FaceDetector() {
19.     cvReleaseMemStorage(&storage);
```

```
20.     // fixme: release the cascade somehow
21. }
22.
23. vector<CvRect> FaceDetector::detect(const IplImage *img) {
24.     double scale = 1.3;
25.     IplImage* gray = cvCreateImage( cvSize(img->width,img->height), 8, 1 );
26.     IplImage* small_img =
27.     cvCreateImage( cvSize( cvRound (img->width/scale),
28.                 cvRound (img->height/scale)), 8, 1 );
29.     int i;
30.
31.     cvCvtColor( img, gray, CV_BGR2GRAY );
32.     cvResize( gray, small_img, CV_INTER_LINEAR );
33.     cvEqualizeHist( small_img, small_img );
34.     cvClearMemStorage( storage );
35.
36.     CvSeq* faces =
37.     cvHaarDetectObjects( small_img, cascade, storage,
38.                 1.1, 2, 0/*CV_HAAR_DO_CANNY_PRUNING*/,
39.                 cvSize(30, 30) );
40.
41.     vector<CvRect> result;
42.     for( i = 0; i < (faces ? faces->total : 0); i++ ) {
43.     CvRect *rect = (CvRect*)cvGetSeqElem( faces, i );
44.     result.push_back(cvRect((int)(rect->x * scale),
45.                 (int)(rect->y * scale),
46.                 (int)(rect->width * scale),
47.                 (int)(rect->height * scale)));
48.     }
49.
50.     cvReleaseImage(&gray);
51.     cvReleaseImage(&small_img);
52. //     cvReleaseSeq(&faces);
53.     return result;
54. }
```

FaceDetector.h

```
1.  #include "utils.h"
2.  #include <opencv/cv.h>
3.
4.  class FaceDetector {
5.      CvMemStorage* storage;
6.      CvHaarClassifierCascade* cascade ;
7.
8.  public:
9.      static FaceDetector facedetector;
10.     FaceDetector(char *cascadename="haarcascade_frontalface_alt.xml");
```

```
11.     ~FaceDetector();
12.     vector<CvRect> detect(const IplImage *img);
13. };
```

FeatureDetector.cpp
Ανίχνευση βάθους και απόστασης του ματιου από την κάμερα.

```
1.  #include "FeatureDetector.h"
2.
3.  FeatureDetector::FeatureDetector(CvSize eyesize):
4.      eyesize(eyesize),
5.      sumimage(cvCreateImage(eyesize, IPL_DEPTH_32F, 1)),
6.      sum2image(cvCreateImage(eyesize, IPL_DEPTH_32F, 1)),
7.      temp(cvCreateImage(eyesize, IPL_DEPTH_32F, 1)),
8.      samples(0)
9.  {
10.     cvZero(sum2image.get());
11.     cvZero(sumimage.get());
12. }
13. void FeatureDetector::addSample(const IplImage *source) {
14.     cvConvertScale(source, temp.get());
15.     cvAcc(temp.get(), sumimage.get());
16.     cvSquareAcc(temp.get(), sum2image.get());
17.     samples++;
18. }
19.
20. shared_ptr<IplImage> FeatureDetector::getMean() {
21.     shared_ptr<IplImage> mean(createImage(eyesize, IPL_DEPTH_32F, 1));
22.     cvConvertScale(sumimage.get(), mean.get(), 1.0 / samples);
23.     return mean;
24. }
25.
26. shared_ptr<IplImage> FeatureDetector::getVariance() {
27.     shared_ptr<IplImage> variance(createImage(eyesize, IPL_DEPTH_32F, 1));
28.     cvMul(sumimage.get(), sumimage.get(), temp.get(), -1.0/samples);
29.     cvAdd(temp.get(), sum2image.get(), temp.get());
30.     cvScale(temp.get(), variance.get(), 1.0/samples);
31.     return variance;
32. }
```

FeatureDetector.h

```
1.  #pragma once
2.  #include "utils.h"
3.
4.  class FeatureDetector {
5.      CvSize eyesize;
```

```
6.      scoped_ptr<IplImage> sumimage, sum2image, temp;
7.      int samples;
8.  public:
9.      FeatureDetector(CvSize eyesize);
10.     void addSample(const IplImage *source);
11.     shared_ptr<IplImage> getMean();
12.     shared_ptr<IplImage> getVariance();
13. };
```

FMatrixAffineCompute.cpp

Υπολογισμός τιμών κίνησης μεταξύ σημείων διανύσματικά.

```
1.  #include <vnl/algo/vnl_svd.h>
2.
3.  template <class T>
4.
5.  T sum(vector<T> const& vector) {
6.      T sum = vector[0];
7.
8.      for(int i=1; i<vector.size(); i++)
9.      sum += vector[i];
10.
11.     return sum;
12. }
13.
14. Vector computeAffineFMatrix(vector<HomPoint> const& points1,
15.                 vector<HomPoint> const& points2)
16. {
17.     assert(points1.size() == points2.size());
18.
19.     int n = points1.size();
20.
21.     Vector centroid(4, 0.0);
22.
23.     for(int i=0; i<n; i++) {
24.     centroid[0] += points1[i].x();
25.     centroid[1] += points1[i].y();
26.     centroid[2] += points2[i].x();
27.     centroid[3] += points2[i].y();
28.     }
29.
30.     centroid /= n;
31.
32.     Matrix matrix(n, 4);
33.
34.     for(int i=0; i<n; i++) {
35.     matrix(i, 0) = points1[i].x() - centroid[0];
```

```
36.     matrix(i, 1) = points1[i].y() - centroid[1];
37.     matrix(i, 2) = points2[i].x() - centroid[2];
38.     matrix(i, 3) = points2[i].y() - centroid[3];
39.     }
40.
41. //     cout << "fmatrixcompute: " << endl << matrix << endl << endl;
42.
43.     vnl_svd<double> svd(matrix);
44.
45. //     cout << "U = " << endl << svd.U() << endl << endl;
46. //     cout << "W = " << endl << svd.W() << endl << endl;
47. //     cout << "V = " << endl << svd.V() << endl << endl;
48.
49.     Vector v = svd.V().get_column(3);
50.
51.     Vector result(5);
52.
53.     result.update(v);       // a,b,c,d
54.     result[4] = inner_product(v, centroid);
55.
56.     if (result[4] < 0)
57.     result = -result;
58.
59.
60.
61. //     cout << "mat: " << v << endl;
62. //     cout << "test: " << matrix * v << endl;
63.
64.     return result.normalize();
65. }
```

GaussianProcess.cpp

Η Gaussian μέθοδος που χρησιμοποιείται σε διάφορα σημεία μέσα στο πρόγραμμα. Εφαρμόζεται στην κίνηση του οφθαλμού κατά την διαδικασία εκτέλεσης της εφαρμογής για την ομαλότερη εισαγωγή τιμών κίνησης στην εφαρμογή (smooth κίνηση χωρίς σπασμοδικές κινήσεις).Επίσης η ενναλαγή των τιμών διεύθυνσης των διανυσμάτων κατά την διάρκεια καταγραφής σε real time ή όχι γίνεται με ομαλότητα.

```
1.  #include <vnl/algo/vnl_cholesky.h>
2.
3.  class MultiGaussian {
4.  public:
5.      Vector mean;
6.      Matrix covariance;
7.
8.      MultiGaussian(Vector const& mean, Matrix const& covariance):
9.      mean(mean), covariance(covariance) {}
10. };
11.
```

```
12. template <class T>
13. class GaussianProcess {
14. public:
15.     typedef double (*CovarianceFunction)(const T&, const T&);
16. private:
17.     vector<T> inputs;
18.     CovarianceFunction covariancefunction;
19. private:
20.     Matrix L;
21.     Vector alpha;
22.
23.     Matrix getcovariancematrix(vector<T> const& in1,
24.                 vector<T> const& in2) const
25.     {
26.     Matrix K(in1.size(), in2.size());
27.
28.     for(int i=0; i<in1.size(); i++)
29.         for(int j=0; j<in2.size(); j++)
30.         K(i,j) = covariancefunction(in1[i], in2[j]);
31.
32.     return K;
33.     }
34.
35. public:
36.     GaussianProcess(vector<T> const& inputs,
37.             Vector const& targets,
38.             CovarianceFunction covariancefunction,
39.             double noise=0.0):
40.     inputs(inputs), covariancefunction(covariancefunction)
41.     {
42.     Matrix K = getcovariancematrix(inputs, inputs);
43.     for(int i=0; i<inputs.size(); i++)
44.         K[i][i] += noise;
45.     // todo: do we need the "verbose" mode in cholesky?
46.     vnl_cholesky chol(K);
47.     L = chol.lower_triangle();
48.     alpha = chol.solve(targets);
49.     }
50.
51.     Vector getmeans(vector<T> const &tests) const {
52.     Matrix KK = getcovariancematrix(inputs, tests);
53.     return KK.transpose() * alpha;
54.     }
55.
56.     double getmean(T const& test) const {
57.     return getmeans(vector<T>(1, test))[0];
58.     }
59. };
```

```
60.
61. template <class T>
62. class MeanAdjustedGaussianProcess {
63.     double mean;
64.     const GaussianProcess<T> gp;
65.
66. public:
67.     MeanAdjustedGaussianProcess(vector<T> const& inputs,
68.                 Vector const& targets,
69.                 typename GaussianProcess<T>::CovarianceFunction
70.                   covariancefunction,
71.                 double noise=0.0):
72.     mean(targets.mean()),
73.     gp(GaussianProcess<T>(inputs, targets - mean,
74.                   covariancefunction, noise))
75.     {}
76.
77.     double getmean(T const& test) const {
78.     return gp.getmean(test) + mean;
79.     }
80. };
```

GazeArea.cpp

Στην πραγματικότητα, σε αυτό το σημείο παίρνεται η περιοχή της κόρης του ματιού και γίνεται η σύνδεση αυτής και των τιμών που έρχονται σαν είσοδο στο πεδίο περιορισμού της κίνησης ή στο πεδίο της οθόνης αντιστοιχόντας την κόρη στα σημεία του πεδίου ή της οθόνης.

```
1.  #include "GazeArea.h"
2.  #include "OutputMethods.h"
3.  #include <opencv/cv.h>
4.
5.  GazeArea::GazeArea(int argc, char **argv,
6.           const vector<shared_ptr<AbstractStore> > &stores):
7.      lastPointId(-1), gazetracker(argc, argv, stores)
8.  {
9.      set_size_request(gazetracker.canvas->width, gazetracker.canvas->height);
10.     Glib::signal_idle().connect(sigc::mem_fun(*this, &GazeArea::on_idle));
11.     add_events(Gdk::BUTTON_PRESS_MASK);
12.     add_events(Gdk::BUTTON_RELEASE_MASK);
13. }
14.
15. GazeArea::~GazeArea(void) {}
16.
```

```
17. bool GazeArea::on_idle() {
18.     gazetracker.doprocessing();
19.     queue_draw();
20.     return true;
21. }
22.
23. bool GazeArea::on_expose_event(GdkEventExpose *event) {
24.     Glib::RefPtr<Gdk::Window> window = get_window();
25.     if (window) {
26.     Gtk::Allocation allocation = get_allocation();
27.     const int width = allocation.get_width();
28.     const int height = allocation.get_height();
29.
30.     Glib::RefPtr<Gdk::GC> gc = Gdk::GC::create(window);
31.     const IplImage *image = gazetracker.canvas.get();
32.     Glib::RefPtr<Gdk::Pixbuf> pixbuf =
33.         Gdk::Pixbuf::create_from_data((guint8*) image->imageData,
34.                     Gdk::COLORSPACE_RGB,
35.                     false,
36.                     image->depth,
37.                     image->width,
38.                     image->height,
39.                     image->widthStep);
40.     window->draw_pixbuf(gc, pixbuf, 0,0,0,0, width, height,
41.                 Gdk::RGB_DITHER_NONE, 0, 0);
42.     }
43.     return true;
44. }
45.
46. bool GazeArea::on_button_press_event(GdkEventButton *event) {
47.     if (event->button == 1) {
48.     switch(event->type) {
49.     case GDK_BUTTON_PRESS: clickCount = 1; break;
50.     case GDK_2BUTTON_PRESS: clickCount = 2; break;
51.     case GDK_3BUTTON_PRESS: clickCount = 3; break;
52.     default: break;
53.     }
54.
55.     if (event->type == GDK_BUTTON_PRESS) {
56.         Point point(event->x, event->y);
57.         PointTracker &tracker = gazetracker.tracking->tracker;
58.         int closest = tracker.getClosestTracker(point);
59.         if (closest >= 0 &&
60.         point.distance(tracker.currentpoints[closest]) <= 10)
61.         lastPointId = closest;
62.         else
63.         lastPointId = -1;
64.     }
```

```
65.     return true;
66.     }
67.     else
68.     return false;
69. }
70.
71. bool GazeArea::on_button_release_event(GdkEventButton *event) {
72.     if (event->button == 1) {
73.     PointTracker &tracker = gazetracker.tracking->tracker;
74.     Point point(event->x, event->y);
75.     if (lastPointId >= 0)
76.         switch(clickCount) {
77.         case 1: tracker.updatetracker(lastPointId, point); break;
78.         case 2: tracker.removetracker(lastPointId); break;
79.         }
80.     else
81.         tracker.addtracker(point);
82.     return true;
83.     }
84.     else
85.     return false;
86. }
```

GazeArea.h

```
1.  #pragma once
2.  #include <gtkmm/drawingarea.h>
3.  #include "MainGazeTracker.h"
4.
5.  class GazeArea: public Gtk::DrawingArea {
6.      friend class GazeTrackerGtk;
7.      int lastPointId;
8.      int clickCount;
9.   public:
10.     MainGazeTracker gazetracker;
11.
12.     GazeArea(int argc, char **argv,
13.         const vector<shared_ptr<AbstractStore> > &stores);
14.     virtual ~GazeArea();
15.
16.  protected:
17.     virtual bool on_expose_event(GdkEventExpose *event);
18.     virtual bool on_button_press_event(GdkEventButton *event);
19.     virtual bool on_button_release_event(GdkEventButton *event);
20.     bool on_idle();
21. };
```

GazeTracker.cpp

Πολύ επιγραμματικά, στο παρόν σημείο γίνεται το tracking του ματιού και περνώντας τις συγκρίσεις του tracking σε μορφή διανύσματος/ων. Επίσης για να είναι επιτυχυμένο το tracking, όπως μπορέιται να δείτε και στον κώδικα παρακάτω οι τιμές του calibration λαμβάνονται υπ ‘ όψιν. Άρα η σύγκριση εικόνων από την κάμερα σε συνδυασμό με τις τιμές, τα δεδομένα, που έχουν γίνει στο calibration και τα διανύσματα από τα σημεία αναφοράς είναι σε γενικές γραμμές τα στοιχεία που απαρτίζουν την παρακάτω λειτουργία.

```
1.  #include "GazeTracker.h"
2.
3.  int Targets::getCurrentTarget(Point point) {
4.      vector<double> distances(targets.size());
5.      //    debugtee(targets);
6.      transform(targets.begin(), targets.end(), distances.begin(),
7.              sigc::mem_fun(point, &Point::distance));
8.      //    debugtee(distances);
9.      return min_element(distances.begin(), distances.end()) - distances.begin();
10. //     for(int i=0; i<targets.size(); i++)
11. //  if (point.distance(targets[i]) < 30)
12. //      return i;
13. //     return -1;
14. }
15. CalTarget::CalTarget() {}
16.
17. CalTarget::CalTarget(Point point,
18.             const IplImage* image, const IplImage* origimage):
19.     point(point),
20.     image(cvCloneImage(image), releaseImage),
21.     origimage(cvCloneImage(origimage), releaseImage)
22. {
23. }
24.
25. void CalTarget::save(CvFileStorage* out, const char* name) {
26.     cvStartWriteStruct(out, name, CV_NODE_MAP);
27.     point.save(out, "point");
28.     cvWrite(out, "image", image.get());
29.     cvWrite(out, "origimage", origimage.get());
30.     cvEndWriteStruct(out);
31. }
32.
33. void CalTarget::load(CvFileStorage* in, CvFileNode *node) {
34.     point.load(in, cvGetFileNodeByName(in, node, "point"));
35.     image.reset((IplImage*) cvReadByName(in, node, "image"));
36.     origimage.reset((IplImage*) cvReadByName(in, node, "origimage"));
37. }
38.
39. TrackerOutput::TrackerOutput(Point gazepoint, Point target, int targetid):
40.     gazepoint(gazepoint), target(target), targetid(targetid)
41. {
42. }
```

```
43.
44. template <class T, class S>
45. vector<S> getsubvector(vector<T> const& input, S T::*ptr) {
46.     vector<S> output(input.size());
47.     for(int i=0; i<input.size(); i++)
48.     output[i] = input[i].*ptr;
49.     return output;
50. }
51.
52. double GazeTracker::imagedistance(const IplImage *im1, const IplImage *im2) {
53.     double norm = cvNorm(im1, im2, CV_L2);
54.     return norm*norm;
55. }
56.
57. double GazeTracker::covariancefunction(SharedImage const& im1,
58.                     SharedImage const& im2)
59. {
60.     const double sigma = 1.0;
61.     const double lscale = 500.0;
62.     return sigma*sigma*exp(-imagedistance(im1.get(),im2.get())/(2*lscale*lscale));
63. }
64.
65. void GazeTracker::updateGPs(void) {
66.     Vector xlabels(caltargets.size());
67.     Vector ylabels(caltargets.size());
68.
69.     for(int i=0; i<caltargets.size(); i++) {
70.     xlabels[i] = caltargets[i].point.x;
71.     ylabels[i] = caltargets[i].point.y;
72.     }
73.
74.     vector<SharedImage> images =
75.     getsubvector(caltargets, &CalTarget::image);
76.
77.     gpx.reset(new ImProcess(images, xlabels, covariancefunction, 0.01));
78.     gpy.reset(new ImProcess(images, ylabels, covariancefunction, 0.01));
79.     targets.reset(new Targets(getsubvector(caltargets, &CalTarget::point)));
80. }
81.
82. void GazeTracker::clear() {
83.     caltargets.clear();
84.     // updateGPs()
85. }
86.
87. void GazeTracker::addExemplar(Point point,
88.                 const IplImage *eyefloat,
89.                 const IplImage *eyegrey)
90. {
```

```
91.     caltargets.push_back(CalTarget(point, eyefloat, eyegrey));
92.     updateGPs();
93. }
94. // void GazeTracker::updateExemplar(int id,
95. //             const IplImage *eyefloat,
96. //             const IplImage *eyegrey)
97. // {
98. //     cvConvertScale(eyegrey, caltargets[id].origimage.get());
99. //     cvAdd(caltargets[id].image.get(), eyefloat, caltargets[id].image.get());
100.                    //     cvConvertScale(caltargets[id].image.get(), caltargets[id].image.get(), 0.5);
101.                    //     updateGPs();
102.                    // }
103.
104.                    void GazeTracker::draw(IplImage *destimage, int eyedx, int eyedy) {
105.                    //     for(int i=0; i<caltargets.size(); i++) {
106.                    //  Point p = caltargets[i].point;
107.                    //  cvSetImageROI(destimage, cvRect((int)p.x - eyedx, (int)p.y - eyedy,
108.                    //                  2*eyedx, 2*eyedy));
109.                    //  cvCvtColor(caltargets[i].origimage, destimage, CV_GRAY2RGB);
110.                    //  cvRectangle(destimage, cvPoint(0,0), cvPoint(2*eyedx-1,2*eyedy-1),
111.                    //          CV_RGB(255,0,255));
112.                    //     }
113.                    //     cvResetImageROI(destimage);
114.                    }
115.
116.                    void GazeTracker::save(void) {
117.                       CvFileStorage *out =
118.                       cvOpenFileStorage("calibration.xml", NULL, CV_STORAGE_WRITE);
119.                       save(out, "GazeTracker");
120.                       cvReleaseFileStorage(&out);
121.                    }
122.
123.                    void GazeTracker::save(CvFileStorage *out, const char *name) {
124.                       cvStartWriteStruct(out, name, CV_NODE_MAP);
125.                       savevector(out, "caltargets", caltargets);
126.                       cvEndWriteStruct(out);
127.                    }
128.                    void GazeTracker::load(void) {
129.                       CvFileStorage *in =
130.                       cvOpenFileStorage("calibration.xml", NULL, CV_STORAGE_READ);
131.                       CvFileNode *root = cvGetRootFileNode(in);
132.                       load(in, cvGetFileNodeByName(in, root, "GazeTracker"));
133.                       cvReleaseFileStorage(&in);
134.                       updateGPs();
135.                    }
136.                    void GazeTracker::load(CvFileStorage *in, CvFileNode *node) {
137.                       caltargets = loadvector<CalTarget>(in, cvGetFileNodeByName(in, node,
138.                                                   "caltargets"));
```

```
139.                }
140.                static void ignore(const IplImage *) {}
141.                void GazeTracker::update(const IplImage *image) {
142.                   if (isActive()) {
143.                   output.gazepoint = Point(gpx->getmean(SharedImage(image, &ignore)),
144.                            gpy->getmean(SharedImage(image, &ignore)));
145.                   output.targetid = getTargetId(output.gazepoint);
146.                   output.target = getTarget(output.targetid);
147.                   }
148.                }
149.                int GazeTracker::getTargetId(Point point) {
150.                   return targets->getCurrentTarget(point);
151.                }
152.                Point GazeTracker::getTarget(int id) {
153.                   return targets->targets[id];
154.                }
```

GazeTracker.h

```
1.  #pragma once
2.  #include "utils.h"
3.  #include "GaussianProcess.cpp"
4.
5.  typedef MeanAdjustedGaussianProcess<SharedImage> ImProcess;
6.
7.  struct Targets {
8.     vector<Point> targets;
9.
10.    Targets(void) {};
11.    Targets(vector<Point> const& targets): targets(targets) {}
12.    int getCurrentTarget(Point point);
13. };
14.
15. struct CalTarget {
16.    Point point;
17.    SharedImage image, origimage;
18.
19.    CalTarget();
20.    CalTarget(Point point, const IplImage* image, const IplImage* origimage);
21.
22.    void save(CvFileStorage* out, const char* name=NULL);
23.    void load(CvFileStorage* in, CvFileNode *node);
24. };
25.
26. struct TrackerOutput {
27.    Point gazepoint;
28.    Point target;
```

```
29.     int targetid;
30.
31.     TrackerOutput(Point gazepoint, Point target, int targetid);
32. };
33.
34. class GazeTracker {
35.     scoped_ptr<ImProcess> gpx, gpy;
36.     vector<CalTarget> caltargets;
37.     scoped_ptr<Targets> targets;
38.
39.     static double imagedistance(const IplImage *im1, const IplImage *im2);
40.     static double covariancefunction(const SharedImage& im1,
41.                     const SharedImage& im2);
42.
43.     void updateGPs(void);
44.
45. public:
46.     TrackerOutput output;
47.
48.     GazeTracker(): targets(new Targets),
49.     output(Point(0,0), Point(0,0), -1) {}
50.
51.     bool isActive() { return gpx.get() && gpy.get(); }
52.
53.     void clear();
54.     void addExemplar(Point point,
55.             const IplImage *eyefloat, const IplImage *eyegrey);
56.     void draw(IplImage *canvas, int eyedx, int eyedy);
57.     void save(void);
58.     void save(CvFileStorage *out, const char *name);
59.     void load(void);
60.     void load(CvFileStorage *in, CvFileNode *node);
61.     void update(const IplImage *image);
62.     int getTargetId(Point point);
63.     Point getTarget(int id);
64. };
```

GazeTrackerGtk.cpp
Λειτουργίες του user interface

```
1.  #include "GazeTrackerGtk.h"
2.  #include <gtkmm.h>
3.  #include <iostream>
4.  #include "GtkStore.h"
5.
6.  static vector<shared_ptr<AbstractStore> > getStores() {
```

```
7.      vector<shared_ptr<AbstractStore> > stores;
8.
9.      stores.push_back(shared_ptr<AbstractStore>(new SocketStore()));
10.     stores.push_back(shared_ptr<AbstractStore>(new StreamStore(cout)));
11.     stores.push_back(shared_ptr<AbstractStore>
12.       (new WindowStore(WindowPointer::PointerSpec(20, 20, 0, 0, 1),
13.               WindowPointer::PointerSpec(30, 30, 1, 0, 1))));
14.
15.     return stores;
16. }
17.
18. GazeTrackerGtk::GazeTrackerGtk(int argc, char **argv):
19.     calibratebutton("Calibrate"),
20.     loadbutton("Load points"),
21.     savebutton("Save points"),
22.     clearbutton("Clear points"),
23.     picture(argc, argv, getStores())
24. {
25.     set_title("opengazer 0.1.1");
26.
27.     add(vbox);
28.     vbox.pack_start(picture);
29.     vbox.pack_start(buttonbar);
30.
31.     buttonbar.pack_start(calibratebutton);
32.     Gtk::Button *testbutton = manage(new Gtk::Button("Test"));
33.     buttonbar.pack_start(*testbutton);
34.     buttonbar.pack_start(savebutton);
35.     buttonbar.pack_start(loadbutton);
36.     buttonbar.pack_start(clearbutton);
37.
38.     calibratebutton.signal_clicked().
39.     connect(sigc::mem_fun(&picture.gazetracker,
40.                 &MainGazeTracker::startCalibration));
41.     testbutton->signal_clicked().
42.     connect(sigc::mem_fun(&picture.gazetracker,
43.                 &MainGazeTracker::startTesting));
44.     savebutton.signal_clicked().
45.     connect(sigc::mem_fun(&picture.gazetracker,
46.                 &MainGazeTracker::savepoints));
47.     loadbutton.signal_clicked().
48.     connect(sigc::mem_fun(&picture.gazetracker,
49.                 &MainGazeTracker::loadpoints));
50.     clearbutton.signal_clicked().
51.     connect(sigc::mem_fun(&picture.gazetracker,
52.                 &MainGazeTracker::clearpoints));
53.
54.     picture.show();
```

```
55.     testbutton->show();
56.     calibratebutton.show();
57.     savebutton.show();
58.     loadbutton.show();
59.     clearbutton.show();
60.     buttonbar.show();
61.     vbox.show();
62. }
63.
64.
65. GazeTrackerGtk::~GazeTrackerGtk() {
66. }
```

GazeTrackerGtk.h

```
1.  #include <gtkmm/button.h>
2.  #include <gtkmm/window.h>
3.  #include <gtkmm/drawingarea.h>
4.  #include <gtkmm/box.h>
5.
6.  #include "GazeArea.h"
7.
8.  class GazeTrackerGtk: public Gtk::Window {
9.   protected:
10.   //Member widgets:
11.     Gtk::Button calibratebutton, loadbutton, savebutton, clearbutton;
12.     Gtk::VBox vbox;
13.     Gtk::HBox buttonbar;
14.
15.  public:
16.     GazeArea picture;
17.     GazeTrackerGtk(int argc, char **argv);
18.     virtual ~GazeTrackerGtk();
19.
20. };
```

GraphicalPointer.cpp

Γραφικά, δείκτες ή σημεία που δημιουργούνται για το calibration στο user interface.

```
1.  #include "GraphicalPointer.h"
2.  #include <cairomm/context.h>
```

```
3.  WindowPointer::PointerSpec::PointerSpec(int width, int height,
4.                      double red, double green, double blue):
5.      width(width), height(height), red(red), green(green), blue(blue)
6.  {
7.  }
8.
9.  WindowPointer::GtkPointerDrawingArea::
10. GtkPointerDrawingArea(const PointerSpec &pointerspec):
11.     spec(pointerspec)
12. {
13.     set_size_request(spec.width, spec.height);
14. }
15.
16. bool WindowPointer::GtkPointerDrawingArea::
17. on_expose_event(GdkEventExpose *event)
18. {
19.     Glib::RefPtr<Gdk::Window> window = get_window();
20.     if (window) {
21.     Cairo::RefPtr<Cairo::Context> cr(new Cairo::Context(gdk_cairo_create(window->gobj()), true));
22.     if (event) {
23.         cr->rectangle(event->area.x, event->area.y,
24.                 event->area.width, event->area.height);
25.         cr->clip();
26.     }
27.     cr->set_source_rgb(1.0, 1.0, 1.0);
28.     cr->paint();
29.     cr->set_source_rgb(spec.red, spec.green, spec.blue);
30.     cr->arc(get_width()/2, get_height()/2, get_width()/3, 0, 2*M_PI);
31.     cr->fill();
32. //  Glib::RefPtr<Gdk::GC> gc = Gdk::GC::create(window);
33. //  color.set_blue(100);
34. //  gc->set_foreground(color);
35. //  window->draw_arc(gc, true, 0, 0, get_width(), get_height(), 0, 360*64);
36.     }
37.     return false;
38. }
39.
40. WindowPointer::GtkPointerWindow::
41. GtkPointerWindow(const PointerSpec &pointerspec):
42.     area(pointerspec)
43. {
44.     add(area);
45.     area.show();
46.     set_decorated(false);
47.     set_keep_above(true);
48. }
49.
50. WindowPointer::WindowPointer(const PointerSpec &pointerspec):
```

```
51.     pointerwindow(pointerspec)
52. {
53.     pointerwindow.show();
54. }
55.
56. void WindowPointer::setPosition(int x, int y) {
57.     pointerwindow.move(x, y);
58. }
```

GraphicalPointer.h

```
1.  #pragma once
2.  #include <gtkmm.h>
3.  #include "Containers.h"
4.
5.  /* represents the pointer as a small window and moves that window */
6.  class WindowPointer {
7.   public:
8.     struct PointerSpec {
9.     int width, height;
10.    double red, green, blue;
11.    PointerSpec(int width, int height,
12.            double red, double green, double blue);
13.    };
14.  private:
15.    class GtkPointerDrawingArea: public Gtk::DrawingArea {
16.    PointerSpec spec;
17.    public:
18.       GtkPointerDrawingArea(const PointerSpec &pointerspec);
19.       virtual bool on_expose_event(GdkEventExpose *event);
20.    };
21.    class GtkPointerWindow: public Gtk::Window {
22.       GtkPointerDrawingArea area;
23.    public:
24.       GtkPointerWindow(const PointerSpec &pointerspec);
25.    };
26.    GtkPointerWindow pointerwindow;
27.  public:
28.    WindowPointer(const PointerSpec &pointerspec);
29.    void setPosition(int x, int y);
30. };
31.
32. /* class PointerMark: public Gtk::DrawingArea { */
33. /*     virtual bool on_expose_event(GdkEventExpose *event); */
34. /* }; */
35.
```

```
/* class CalibrationMark: public Gtk::DrawingArea { */
/*     virtual bool on_expose_event(GdkEventExpose *event); */
/* }; */
```

GtkStore.cpp

Αποθήκευση στην RAM των δημιουργηθέντων σημείων στο user interface σε σχέση με την κόρη του ματιού ώστε να αποφασισθεί ο στόχος ή το κέντρου του ματιού που θα λειτουργεί σαν το κεντρικό σημείο ή δείκτης για το tracking.

```
#include "GtkStore.h"

WindowStore::WindowStore(const WindowPointer::PointerSpec& pointerspec,
            const WindowPointer::PointerSpec& targetspec):
    pointer(pointerspec), target(targetspec)
{
}

void WindowStore::store(const TrackerOutput &output) {
    pointer.setPosition((int) output.gazepoint.x, (int) output.gazepoint.y);
    target.setPosition((int) output.target.x, (int) output.target.y);
}
```

GtkStore.h

```
#pragma once
#include <gtkmm.h>
#include "OutputMethods.h"
#include "GraphicalPointer.h"


class WindowStore: public AbstractStore {
    WindowPointer pointer, target;
 public:
    WindowStore(const WindowPointer::PointerSpec& pointerspec,
        const WindowPointer::PointerSpec& targetspec);
    virtual void store(const TrackerOutput &output);
};
```

HeadCompensation.cpp

Η παρακάτω μέθοδος επιτυγχάνει, όταν υπάρχει περιστροφή της κεφαλής κατά την διάρκεια καταγραφής κίνησης του οφθαλμού να μην χάνεται η κόρη ή το σημείο στόχευσης που δείχνει το μάτι αφού δεν πρέπει να ξεχνάμε πως η εφαρμογή ήταν φτιαγμένη για κάμερα από απόσταση και όχι προσαρμοσμένη στο κεφάλι.

```
#include "LeastSquares.h"
class HeadCompensation {
    LeastSquares xparams, yparams;
    const HeadTracker &head;
```

```
5.      double xa0, xa1, xa2, ya0, ya1, ya2;
6.      int samples;
7.  public:
8.      HeadCompensation(HeadTracker const& head):
9.      xparams(3), yparams(3), head(head),
10.     xa0(0.0), xa1(0.0), xa2(0.0),
11.     ya0(0.0), ya1(0.0), ya2(0.0), samples(0)
12.     {}
13.
14.     void addCorrection(Point correction) {
15.     xparams.addSample(head.rotx, head.roty, 1.0, correction.x);
16.     yparams.addSample(head.rotx, head.roty, 1.0, correction.y);
17.     samples++;
18.     }
19.
20.     void updateFactors(void) {
21.     if (samples > 0) {
22.         xparams.solve(xa0, xa1, xa2);
23.         yparams.solve(ya0, ya1, ya2);
24.     }
25.     }
26.
27.     Point estimateCorrection(void) {
28.     return Point(xa0 * head.rotx + xa1 * head.roty + xa2,
29.             ya0 * head.rotx + ya1 * head.roty + ya2);
30.     }
31. };
```

HeadTracker.cpp

Όπως έχουμε προαναφέρει αρχικά το πρόγραμμα πρέπει να αναγνωρίσει την κεφαλή, στην συνέχεια να χαρακτηριστικά του προσώπου και τέλος το μάτι. Ο παρακάτω κώδικας όπως θα δείτε υλοποιεί την αναγνώριση και το tracking της κεφαλής.

```
1.  #include "HeadTracker.h"
2.  #include "FMatrixAffineCompute.cpp"
3.
4.  static double squarenorm(HomPoint point) {
5.      return square(point.x()) + square(point.y());
6.  }
7.
8.
9.  static double mean(vector<HomPoint> const& vec,
10.             double (HomPoint::*func)() const)
11. {
12.     double sum = 0.0;
13.     for(int i=0; i<vec.size(); i++)
14.     sum += (vec[i].*func)();
15.     return sum / vec.size();
```

```
16. }
17.
18.
19. static void predict(double xorig, double yorig,
20.         double a, double b, double c, double d, double depth,
21.         double &xnew, double &ynew)
22. {
23.     double A =       - a*xorig - b*yorig;
24.     double B = depth - b*xorig + a*yorig;
25.
26.     xnew = (c*A + d*B) / (d*d + c*c);
27.     ynew = (d*A - c*B) / (d*d + c*c);
28. }
29.
30.
31. static HomPoint
32. predictpoint(HomPoint p, double depth, double dmeanx, double dmeany,
33.         double rotx, double roty, double atx, double aty)
34. {
35.     HomPoint p2 (p.x() * atx - p.y() * aty,
36.         p.x() * aty + p.y() * atx);
37.
38.     return HomPoint(p2.x() + roty * depth + dmeanx,
39.           p2.y() - rotx * depth + dmeany);
40. }
41.
42. vector<bool> HeadTracker::detectinliers(vector<HomPoint> const &prev,
43.                   vector<HomPoint> const &now,
44.                   double radius)
45. {
46.     assert(prev.size() == now.size());
47.
48.     vector<HomPoint> transitions;
49.     for(int i=0; i<prev.size(); i++)
50.     transitions.push_back(now[i] - prev[i]);
51.
52. //     cout << "xtrans:";
53. //     for(int i=0; i<prev.size(); i++)
54. //  cout << " " << transitions[i].x();
55. //     cout << endl;
56.
57. //     cout << "ytrans:";
58. //     for(int i=0; i<prev.size(); i++)
59. //  cout << " " << transitions[i].y();
60. //     cout << endl;
61.
62.     vector<int> closepoints(transitions.size());
63.     for(int i=0; i<transitions.size(); i++) {
```

```
64.    closepoints[i] = 0;
65.    for(int j=0; j<transitions.size(); j++)
66.       if (squarenorm(transitions[i] - transitions[j]) <= square(radius))
67.       closepoints[i]++;
68.    }
69.
70.    int maxindex = max_element(closepoints.begin(), closepoints.end())
71.    - closepoints.begin();
72.
73.    vector<bool> inliers(transitions.size());
74.    for(int i=0; i<transitions.size(); i++)
75.    inliers[i] = (squarenorm(transitions[i] - transitions[maxindex])
76.             <= square(radius));
77.
78.    for(int i=0; i<inliers.size(); i++)
79.    if (!inliers[i]) {
80.       tracker.status[i] = false;
81.       tracker.currentpoints[i].x =
82.       0.9 * (tracker.origpoints[i].x + transitions[maxindex].x())
83.       + 0.1 * tracker.currentpoints[i].x;
84.       tracker.currentpoints[i].y =
85.       0.9 * (tracker.origpoints[i].y + transitions[maxindex].y())
86.       + 0.1 * tracker.currentpoints[i].y;
87.    }
88.
89.    return inliers;
90. }
91.
92.
93. void
94. HeadTracker::predictpoints(double xx0, double yy0, double xx1, double yy1,
95.              double rotx, double roty, double atx, double aty)
96. {
97.    double maxdiff = 0.0;
98.    int diffindex = -1;
99.
100.                        vector<HomPoint> points =
101.                        tracker.getpoints(&PointTracker::origpoints, true);
102.
103.                        for(int i=0; i<points.size(); i++) {
104.                        HomPoint p(points[i].x() - xx0, points[i].y() - yy0);
105.                        HomPoint p1 = predictpoint(p, depths[i], xx1, yy1,
106.                                 rotx, roty, atx, aty);
107.                        HomPoint p2 = predictpoint(p, -depths[i], xx1, yy1,
108.                                 rotx, roty, atx, aty);
109.
110.                        double diff1 = fabs(p1.x() - tracker.currentpoints[i].x) +
111.                           fabs(p1.y() - tracker.currentpoints[i].y);
```

```
112.
113.                 double diff2 = fabs(p2.x() - tracker.currentpoints[i].x) +
114.                     fabs(p2.y() - tracker.currentpoints[i].y);
115.
116.                 double diff = diff1 > diff2 ? diff2 : diff1;
117.
118.                 // dubious code, I'm not sure about it
119.                 if (!tracker.status[i]) {
120.                     tracker.currentpoints[i].x =
121.                     0.5 * tracker.currentpoints[i].x + 0.5 * p1.x();
122.                     tracker.currentpoints[i].y =
123.                     0.5 * tracker.currentpoints[i].y + 0.5 * p1.y();
124.                 }
125.                 }
126.             }
127.
128.             void HeadTracker::updatetracker(void) {
129.                 depths.resize(tracker.pointcount());
130.                 detectinliers(tracker.getpoints(&PointTracker::origpoints, true),
131.                     tracker.getpoints(&PointTracker::currentpoints, true));
132.
133.                 vector<HomPoint> origpoints =
134.                 tracker.getpoints(&PointTracker::origpoints, false);
135.                 vector<HomPoint> currentpoints =
136.                 tracker.getpoints(&PointTracker::currentpoints, false);
137.
138.                 double xx0 = mean(origpoints, &HomPoint::x);
139.                 double yy0 = mean(origpoints, &HomPoint::y);
140.                 double xx1 = mean(currentpoints, &HomPoint::x);
141.                 double yy1 = mean(currentpoints, &HomPoint::y);
142.
143.                 Vector fmatrix = computeAffineFMatrix(origpoints, currentpoints);
144.
145.                 double a = fmatrix[0];
146.                 double b = fmatrix[1];
147.                 double c = fmatrix[2];
148.                 double d = fmatrix[3];
149.                 double e = fmatrix[4];
150.
151.                 // compute the change
152.
153.                 vector<double> offsets(tracker.pointcount());
154.
155.                 double depthsum = 0.0001;
156.                 double offsetsum = 0.0001;
157.
158.                 for(int i=0; i<tracker.pointcount(); i++)
159.                 if (tracker.status[i]) {
```

```
160.                    double xorig = tracker.origpoints[i].x - xx0;
161.                    double yorig = tracker.origpoints[i].y - yy0;
162.                    double xnew = tracker.currentpoints[i].x - xx1;
163.                    double ynew = tracker.currentpoints[i].y - yy1;
164.                    double x0 = b*xorig - a*yorig;
165.                    double x1 = -d*xnew + c*ynew;
166.                    offsets[i] = x0 - x1;
167.                    offsetsum += offsets[i]*offsets[i];
168.                    depthsum += depths[i]*depths[i];
169.                }
170.
171.                if (tracker.areallpointsactive()) {
172.              //  cout << endl;
173.                depthsum = 1.0;
174.                }
175.
176.                double depthscale = sqrt(offsetsum / depthsum);
177.
178.                rotx = c * depthscale / hypot(a,b) / hypot(c,d);
179.                roty = d * depthscale / hypot(a,b) / hypot(c,d);
180.
181.                atx = -(a*c + b*d) / (c*c + d*d); // at = AmpliTwist
182.                aty = -(a*d - c*b) / (c*c + d*d); // at = AmpliTwist
183.
184.                // depths
185.
186.                vector<double> newdepths(tracker.pointcount());
187.
188.                for(int i=0; i<tracker.pointcount(); i++)
189.                if (tracker.status[i])
190.                   newdepths[i] = offsets[i] / depthscale;
191.
192.                if (newdepths[1] > newdepths[2]) {
193.                rotx = -rotx;
194.                roty = -roty;
195.                for(int i=0; i<tracker.pointcount(); i++)
196.                   depths[i] = -newdepths[i];
197.                }
198.                else
199.                for(int i=0; i<tracker.pointcount(); i++)
200.                   depths[i] = newdepths[i];
201.
202.
203.             //    // distance
204.             //    double distance1 = 0.0;
205.             //    double distance2 = 0.0;
206.             //    for(int i=0; i<tracker.pointcount; i++)
207.             //  if (tracker.status[i]) {
```

```
208.                //      distance1 += square(depths[i] - newdepths[i]);
209.                //      distance2 += square(depths[i] + newdepths[i]);
210.                //  }
211.
212.                //     for(int i=0; i<tracker.pointcount; i++)
213.                //  if (tracker.status[i])
214.                //      if (distance1 > distance2) {
215.                //      rotx = -rotx;
216.                //      roty = -roty;
217.                //      depths[i] = -newdepths[i];
218.                //      }
219.                //           else
220.                //      depths[i] = newdepths[i];
221.
222.                   predictpoints(xx0, yy0, xx1, yy1,
223.                        rotx, roty, atx, aty);
224.
225.                }
226.
227.                void HeadTracker::draw(IplImage *image) {
228.                //     cout << "state: "<< rotx << " " << roty << " "
229.                //   << atx << " " << aty << endl;
230.
231.                   cvLine(image,
232.                     cvPoint(320, 240),
233.                     cvPoint(320 + int(atx * 50), 240 + int(aty * 50)),
234.                     CV_RGB(255,255,255));
235.
236.                //     for(int i=0; i<tracker.pointcount; i++)
237.                //  cvLine(image,
238.                //         cvPoint(tracker.currentpoints[i].x, tracker.currentpoints[i].y),
239.                //         cvPoint(tracker.origpoints[i].x - xx0 + xx1,
240.                //            tracker.origpoints[i].y - yy0 + yy1),
241.                //         CV_RGB(255,0,0));
242.
243.                   for(int i=0; i<tracker.pointcount(); i++) {
244.                   cvLine(image,
245.                        cvPoint((int)tracker.currentpoints[i].x,
246.                           (int)tracker.currentpoints[i].y),
247.                        cvPoint((int)tracker.currentpoints[i].x,
248.                           int(tracker.currentpoints[i].y + depths[i] * 100)),
249.                        CV_RGB(0,0,255));
250.                   }
251.
252.                   double scale = 10;
253.
254.                   cvLine(image,
255.                     cvPoint(320, 240),
```

```
256.                    cvPoint(320 + int(roty * scale), 240 + int(rotx * scale)),
257.                    CV_RGB(255,0,0));
258.                }
```

HeadTracker.h

```
1.  #pragma once
2.  #include <vgl/vgl_homg_point_2d.h>
3.  #include <opencv/cv.h>
4.  #include <vector>
5.  #include "PointTracker.h"
6.
7.  using namespace std;
8.
9.  class HeadTracker {
10.     vector<double> depths;
11.
12.     vector<bool> detectinliers(vector<HomPoint> const &prev,
13.                 vector<HomPoint> const &now,
14.                 double radius = 30.0);
15.
16.     void predictpoints(double xx0, double yy0, double xx1, double yy1,
17.             double rotx, double roty, double atx, double aty);
18.
19.  public:
20.     PointTracker &tracker;
21.
22.     double rotx, roty, atx, aty;
23.
24.     void draw(IplImage *image);
25.     void updatetracker(void);
26.
27.     HeadTracker(PointTracker &tracker) : tracker(tracker) {}
28. };
```

LeastSquares.cpp

Εύρεση ελαχίστων τιμών των τετραγώνων διανυσμάτων.Χρησιμοποιείται από το headcompesation method.

```
1.  #include "LeastSquares.h"
2.  #include <vnl/algo/vnl_cholesky.h>
3.  #include <assert.h>
4.
5.  void LeastSquares::addSample(double xs[], double y) {
6.      for(int i=0; i<n; i++)
7.      for(int j=0; j<n; j++)
8.          X[i][j] += xs[i]*xs[j];
```

```
9.
10.     for(int i=0; i<n; i++)
11.     Y[i] += xs[i]*y;
12. }
13.
14. void LeastSquares::addSample(double x1, double x2, double y) {
15.     assert(n == 2);
16.     double xs[2] = {x1, x2};
17.     addSample(xs, y);
18. }
19.
20. void LeastSquares::addSample(double x1, double x2, double x3, double y) {
21.     assert(n == 3);
22.     double xs[3] = {x1, x2, x3};
23.     addSample(xs, y);
24. }
25.
26. Vector LeastSquares::solve(void) {
27.     return vnl_cholesky(X).solve(Y);
28. }
29.
30. void LeastSquares::solve(double &a0, double &a1) {
31.     assert(n == 2);
32.     Vector vec = solve();
33.     a0 = vec[0];
34.     a1 = vec[1];
35. }
36.
37. void LeastSquares::solve(double &a0, double &a1, double &a2) {
38.     assert(n == 3);
39.     Vector vec = solve();
40.     a0 = vec[0];
41.     a1 = vec[1];
42.     a2 = vec[2];
43. }
```

LeastSquares.h

```
1.  #pragma once
2.  #include <vnl/vnl_vector.h>
3.  #include <vnl/vnl_matrix.h>
4.
5.  typedef vnl_vector<double> Vector;
6.  typedef vnl_matrix<double> Matrix;
7.
8.  class LeastSquares {
9.      Matrix X;
```

```
    Vector Y;
public:
    const int n;

    LeastSquares(int n): X(n,n), Y(n), n(n) {}

    void addSample(double xs[], double y);
    void addSample(double x1, double x2, double y);
    void addSample(double x1, double x2, double x3, double y);

    Vector solve(void);
    void solve(double &a0, double &a1);
    void solve(double &a0, double &a1, double &a2);
};
```

MainGazeTracker.cpp

Στο παρακάτω αρχείο υλοποιείται το eye-tracking

```
#include "utils.h"
#include <fstream>
#include "MainGazeTracker.h"

class VideoWriter {
    CvVideoWriter *video;
public:
    VideoWriter(CvSize size) :
    video(cvCreateVideoWriter("out.avi", 0x58564944, 15.0, size))
    {}

    void write(const IplImage *image) {
    cvWriteFrame(video, image);
    }

    ~VideoWriter() {
    cvReleaseVideoWriter(&video);
    }
};



CommandLineArguments::CommandLineArguments(int argc, char** argv) {
    for(int i=1; i<argc; i++) {
    if (argv[i][0] == '-')
        options.push_back(argv[i]);
    else
        parameters.push_back(argv[i]);
    }
```

```
30. }
31.
32. bool CommandLineArguments::isoption(const char* option) {
33.     xforeach(iter, options)
34.     if (!strcmp(*iter, option))
35.         return true;
36.     return false;
37. }
38.
39. VideoInput::VideoInput():
40.     capture(cvCaptureFromCAM(0)), framecount(0),
41.     frame(cvQueryFrame(capture)), size(cvSize(frame->width, frame->height))
42. {}
43.
44. VideoInput::VideoInput(const char* filename):
45.     capture(cvCaptureFromFile(filename)), framecount(0),
46.     frame(cvQueryFrame(capture)), size(cvSize(frame->width, frame->height))
47. {}
48. void VideoInput::updateFrame() {
49.     framecount++;
50.     frame = cvQueryFrame(capture);
51. }
52.
53. VideoInput::~VideoInput() {
54.     cvReleaseCapture( &capture );
55. }
56. MainGazeTracker::MainGazeTracker(int argc, char** argv,
57.                 const vector<shared_ptr<AbstractStore> >
58.                 &stores):
59.     framestoreload(-1), stores(stores), autoreload(false)
60. //     , statemachine(shared_ptr<AlertWindow>(new AlertWindow("start")))
61. {
62.     CommandLineArguments args(argc, argv);
63.
64.     if (args.parameters.size() == 0) {
65.     videoinput.reset(new VideoInput());
66.     if (args.isoption("--record"))
67.         video.reset(new VideoWriter(videoinput->size));
68.     }
69.     else {
70.     videoinput.reset(new VideoInput(args.parameters[0]));
71. //  if (args.parameters.size() >= 2)
72. //      fileinput.reset(new FileInput(parameters[1]));
73.     }
74.
75.
76.     canvas.reset(cvCreateImage(videoinput->size, 8, 3));
77.     tracking.reset(new TrackingSystem(videoinput->size));
```

```
78.
79. }
80.
81. void MainGazeTracker::addTracker(Point point) {
82.     tracking->tracker.addtracker(point);
83. }
84.
85. void MainGazeTracker::savepoints() {
86.     try {
87.     tracking->tracker.save("tracker", "points.txt", videoinput->frame);
88.     autoreload = true;
89.     }
90.     catch (ios_base::failure &e) {
91.     cout << e.what() << endl;
92.     }
93. }
94.
95. void MainGazeTracker::loadpoints() {
96.     try {
97.     tracking->tracker.load("tracker", "points.txt", videoinput->frame);
98.     autoreload = true;
99.     }
100.                    catch (ios_base::failure &e) {
101.                    cout << e.what() << endl;
102.                    }
103.                }
104.
105.                void MainGazeTracker::clearpoints() {
106.                    tracking->tracker.cleartrackers();
107.                    autoreload = false;
108.                }
109.
110.                void MainGazeTracker::doprocessing(void) {
111.                    framecount++;
112.                    videoinput->updateFrame();
113.                    const IplImage *frame = videoinput->frame;
114.
115.                    if (video.get())
116.                    video->write(frame);
117.
118.                    canvas->origin = frame->origin;
119.                    cvCopy( frame, canvas.get(), 0 );
120.
121.                    try {
122.                    tracking->doprocessing(frame, canvas.get());
123.                    if (tracking->gazetracker.isActive()) {
124.                        xforeach(iter, stores)
125.                        (*iter)->store(tracking->gazetracker.output);
```

```
126.                //      cout << point.x << " " << point.y << " -> "
127.                //       << tracking->gazetracker.getTargetId(point) << endl;
128.                   }
129.                //  if (!tracking->tracker.areallpointsactive())
130.                //      throw TrackingException();
131.                   framestoreload = 20;
132.                   }
133.                   catch (TrackingException&) {
134.                   framestoreload--;
135.                   }
136.
137.                   framefunctions.process();
138.                //    statemachine.handleEvent(EVENT_TICK);
139.
140.                   if (autoreload && framestoreload <= 0)
141.                   loadpoints();
142.                }
143.
144.
145.                MainGazeTracker::~MainGazeTracker(void) {
146.                }
147.
148.
149.                void MainGazeTracker::addExemplar(Point exemplar) {
150.                   if (exemplar.x >= EyeExtractor::eyedx &&
151.                   exemplar.x + EyeExtractor::eyedx < videoinput->size.width &&
152.                   exemplar.y >= EyeExtractor::eyedy &&
153.                   exemplar.y + EyeExtractor::eyedy < videoinput->size.height)
154.                   tracking->gazetracker.addExemplar(exemplar,
155.                                 tracking->eyex.eyefloat.get(),
156.                                 tracking->eyex.eyegrey.get());
157.                }
158.
159.                static vector<Point> scalebyscreen(const vector<Point> &points) {
160.                   Glib::RefPtr<Gdk::Screen> screen =
161.                   Gdk::Display::get_default()->get_default_screen();
162.                   return Calibrator::scaled(points, screen->get_width(), screen->get_height());
163.                }
164.
165.                void MainGazeTracker::startCalibration() {
166.                   shared_ptr<WindowPointer>
167.                   pointer(new WindowPointer(WindowPointer::PointerSpec(30,30,1,0,0)));
168.
169.                   ifstream calfile("calpoints.txt");
170.
171.                   shared_ptr<Calibrator>
172.                   calibrator(new Calibrator(framecount, tracking,
173.                                 scalebyscreen(Calibrator::loadpoints(calfile)),
```

```
                                pointer));

                framefunctions.clear();
                framefunctions.addchild(&framefunctions, calibrator);
            }

            void MainGazeTracker::startTesting() {
                vector<Point> points;
                for(double x=0.2; x<0.9; x+=0.2)
                for(double y=0.2; y<0.9; y+=0.2)
                    points.push_back(Point(x,y));

                shared_ptr<WindowPointer>
                pointer(new WindowPointer(WindowPointer::PointerSpec(30,30,0,1,0)));

                shared_ptr<MovingTarget>
                moving(new MovingTarget(framecount, scalebyscreen(points), pointer));

                framefunctions.clear();
                framefunctions.addchild(&framefunctions, moving);
            }
```

MainGazeTracker.h

```
#pragma once
#include "utils.h"
#include "TrackingSystem.h"
//#include "Alert.h"
#include "Calibrator.h"
#include <opencv/highgui.h>
#include <gtkmm.h>

struct CommandLineArguments {
    vector<char*> parameters;
    vector<char*> options;

    CommandLineArguments(int argc, char **argv);
    bool isoption(const char* option);
};
class VideoInput {
    CvCapture* capture;

 public:
    int framecount;
    const IplImage* frame;
    const CvSize size;
    VideoInput();
```

```
24.     VideoInput(const char* avifile);
25.     ~VideoInput();
26.     void updateFrame();
27. };
28.
29. class VideoWriter;
30. /* class FileInput; */
31.
32. class MainGazeTracker {
33.     scoped_ptr<VideoWriter> video;
34.     int framestoreload;
35.     vector<shared_ptr<AbstractStore> > stores;
36.     int framecount;
37.     bool autoreload;
38. //    StateMachine<void> statemachine;
39.
40.  public:
41.     shared_ptr<TrackingSystem> tracking;
42.
43.     FrameProcessing framefunctions;
44.     scoped_ptr<IplImage> canvas;
45.     scoped_ptr<VideoInput> videoinput;
46.
47.     MainGazeTracker(int argc, char** argv,
48.             const vector<shared_ptr<AbstractStore> > &stores);
49.     void doprocessing(void);
50.     ~MainGazeTracker(void);
51.     void addTracker(Point point);
52.     void addExemplar(Point exemplar);
53.     void startCalibration();
54.     void startTesting();
55.     void savepoints();
56.     void loadpoints();
57.     void clearpoints();
58. };
```

Gazer.cpp

```
1.  #include <gtkmm.h>
2.  #include <iostream>
3.  #include "GazeTrackerGtk.h"
4.  #include "OutputMethods.h"
5.  #include "GtkStore.h"
6.
7.  int main(int argc, char **argv)
8.  {
```

```
    Gtk::Main kit(argc, argv);
    Glib::thread_init();

//    CalibrationWindow calwindow;
//    calwindow.show();

    GazeTrackerGtk helloworld(argc, argv);

    helloworld.show();

    Gtk::Main::run(helloworld);

    return 0;
}
```

OutputMethods.cpp

Υλοποίηση μεθόδων εξαγωγής δεδομένων που προκύπτουν από το tracking στις συσκευές εξόδου (π.χ οθόνη).

```
#include "OutputMethods.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

AbstractStore::~AbstractStore() {
}

MmapStore::MmapStore(const char *filename) {
    fd = open(filename, O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
    write(fd, &fd, sizeof(fd));
    write(fd, &fd, sizeof(fd));
    if (fd < 0) {perror("open");return;}
    positiontable = (int*) mmap(0, getpagesize(), PROT_READ | PROT_WRITE,
            MAP_SHARED, fd, 0);
}

void MmapStore::store(const TrackerOutput& output) {
    positiontable[0] = (int) output.gazepoint.x - 320;
    positiontable[1] = (int) output.gazepoint.y - 240;
}

MmapStore::~MmapStore() {
```

```
30.     munmap(positiontable, getpagesize());
31.     close(fd);
32. }
33.
34. StreamStore::StreamStore(ostream &stream): stream(stream) {
35. }
36.
37. StreamStore::~StreamStore() {
38. }
39.
40. void StreamStore::store(const TrackerOutput& output) {
41.     stream << (int) output.gazepoint.x << " "
42.       << (int) output.gazepoint.y << " -> "
43.       << output.targetid << endl;
44.     stream.flush();
45. }
46.
47. SocketStore::SocketStore(int port) {
48.     mysocket = socket(PF_INET, SOCK_DGRAM, 0);
49.
50.     destaddr.sin_family = AF_INET;
51.     destaddr.sin_port = htons(port);
52.     destaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
53. }
54.
55. void SocketStore::store(const TrackerOutput& output) {
56.     ostringstream stream;
57.     stream << "x " << (int) output.gazepoint.x << endl
58.       << "y " << (int) output.gazepoint.y << endl;
59.     string str = stream.str();
60.     sendto(mysocket, str.c_str(), str.size(), 0,
61.       (sockaddr*)&destaddr, sizeof(destaddr));
62. }
63.
64. SocketStore::~SocketStore(void) {
65.     close(mysocket);
66. }
```

OutputMethods.h

```
1.  #pragma once
2.  #include "utils.h"
3.  #include "GazeTracker.h"
4.  #include <netinet/in.h>
5.
6.  class AbstractStore {
7.  public:
```

```
8.      virtual void store(const TrackerOutput& output) = 0;
9.      virtual ~AbstractStore();
10. };
11.
12. class MmapStore: public AbstractStore {
13.     int fd;
14.     int *positiontable;
15. public:
16.     MmapStore(const char *filename="/tmp/gaze-mouse");
17.     virtual void store(const TrackerOutput& output);
18.     virtual ~MmapStore();
19. };
20.
21. class StreamStore: public AbstractStore {
22.     ostream &stream;
23. public:
24.     StreamStore(ostream &stream);
25.     virtual void store(const TrackerOutput& output);
26.     virtual ~StreamStore();
27. };
28.
29. class SocketStore: public AbstractStore {
30.     int mysocket;
31.     struct sockaddr_in destaddr;
32. public:
33.     SocketStore(int port=20320);
34.     virtual void store(const TrackerOutput& output);
35.     virtual ~SocketStore();
36. };
```

Point.cpp

Ορισμός συντεταγμένων των σημείων αντιστοίχισης.

```
1.  #include "utils.h"
2.
3.  void convert(const Point& point, CvPoint2D32f& p) {
4.      p.x = point.x;
5.      p.y = point.y;
6.  }
7.
8.  ostream& operator<< (ostream& out, const Point& p) {
9.      out << p.x << " " << p.y << endl;
10.     return out;
11. }
12.
```

```
13. istream& operator>> (istream& in, Point& p) {
14.     in >> p.x >> p.y;
15.     return in;
16. }
17.
18. void Point::operator=(CvPoint2D32f const& point) {
19.     x = point.x;
20.     y = point.y;
21. }
22.
23. void Point::operator=(CvPoint const& point) {
24.     x = point.x;
25.     y = point.y;
26. }
27.
28.
29. double Point::distance(Point other) const {
30.     return fabs(other.x - x) + fabs(other.y - y);
31. }
32.
33. Point Point::operator+(const Point &other) const {
34.     return Point(x + other.x, y + other.y);
35. }
36.
37. Point Point::operator-(const Point &other) const {
38.     return Point(x - other.x, y - other.y);
39. }
40.
41. void Point::save(CvFileStorage *out, const char* name) const {
42.     cvStartWriteStruct(out, name, CV_NODE_MAP);
43.     cvWriteReal(out, "x", x);
44.     cvWriteReal(out, "y", y);
45.     cvEndWriteStruct(out);
46. }
47.
48. void Point::load(CvFileStorage *in, CvFileNode *node) {
49.     x = cvReadRealByName(in, node, "x");
50.     y = cvReadRealByName(in, node, "y");
51. }
52.
53. CvPoint Point::cvpoint(void) const {
54.     return cvPoint(cvRound(x), cvRound(y));
55. }
56.
57. CvPoint2D32f Point::cvpoint32(void) const {
58.     return cvPoint2D32f(x, y);
59. }
60.
```

```
61. int Point::closestPoint(const vector<Point> &points) const {
62.     if (points.empty())
63.     return -1;
64.
65.     vector<double> distances(points.size());
66.     transform(points.begin(), points.end(), distances.begin(),
67.         sigc::mem_fun(*this, &Point::distance));
68.     return min_element(distances.begin(), distances.end()) - distances.begin();
69. }
```

Point.h

```
1.  struct Point {
2.      double x, y;
3.
4.      Point(): x(0.0), y(0.0) {}
5.      Point(double x, double y): x(x), y(y) {}
6.      Point(CvPoint2D32f const& point): x(point.x), y(point.y) {}
7.      Point(CvPoint const& point): x(point.x), y(point.y) {}
8.
9.      void operator=(CvPoint2D32f const& point);
10.     void operator=(CvPoint const& point);
11.
12.     double distance(Point other) const;
13.     Point operator+(const Point &other) const;
14.     Point operator-(const Point &other) const;
15.
16.     void save(CvFileStorage *out, const char* name) const;
17.     void load(CvFileStorage *in, CvFileNode *node);
18.
19.     CvPoint cvpoint(void) const;
20.     CvPoint2D32f cvpoint32(void) const;
21.
22.     int closestPoint(const vector<Point> &points) const;
23. };
24.
25. ostream& operator<< (ostream& out, const Point& p);
26. istream& operator>> (istream& in, Point& p);
27. void convert(const Point& point, CvPoint2D32f& p);
```

PointTracker.cpp

```
1.  #include "PointTracker.h"
2.  #include "FaceDetector.h"
3.  #include <opencv/highgui.h>
4.  #include <fstream>
5.
```

```
6.
7.  PointTracker::PointTracker(const CvSize &size):
8.      flags(CV_LKFLOW_INITIAL_GUESSES),
9.      grey(cvCreateImage(size, 8, 1)),
10.     orig_grey(cvCreateImage(size, 8, 1)),
11.     pyramid(cvCreateImage(size, 8, 1)),
12.     orig_pyramid(cvCreateImage(size, 8, 1)),
13.     last_grey(cvCreateImage(size, 8, 1)),
14.     last_pyramid(cvCreateImage(size, 8, 1))
15. //     origpoints(new CvPoint2D32f[MAX_COUNT]),
16. //     currentpoints(new CvPoint2D32f[MAX_COUNT]),
17. //     status(new char[MAX_COUNT]),
18. {}
19.
20. static Point pointbetweenrects(const Point &point, CvRect source, CvRect dest) {
21.     return Point((point.x-source.x)*(double(dest.width)/source.width)+dest.x,
22.         (point.y-source.y)*(double(dest.height)/source.height)+dest.y);
23.
24. }
25.
26. static vector<Point> pointbetweenrects(const vector<Point> &points,
27.                     CvRect source, CvRect dest)
28. {
29.     vector<Point> result;
30.     result.reserve(points.size());
31.     xforeach(iter, points)
32.     result.push_back(pointbetweenrects(*iter, source, dest));
33.     return result;
34. }
35.
36. void PointTracker::save(string filename, string newpoints,
37.             const IplImage *frame)
38. {
39.     vector<CvRect> faces = FaceDetector::facedetector.detect(frame);
40.     if (faces.size() == 1) {
41.     cvSaveImage((filename + "-orig-grey.png").c_str(), orig_grey.get());
42.     cvSaveImage((filename + "-orig-pyramid.png").c_str(), orig_pyramid.get());
43.
44.     ofstream origfile((filename + "-orig-points.txt").c_str());
45.     origfile << origpoints;
46.
47.     CvRect face = faces[0];
48.     ofstream facefile(newpoints.c_str());
49.     vector<Point> temppoints;
50.     convert(currentpoints, temppoints);
51.     facefile << pointbetweenrects(temppoints, face, cvRect(0, 0, 1, 1));
52.     }
53.     else
```

```
54.     throw ios_base::failure("No face found in the image");
55. }
56.
57. void PointTracker::load(string filename, string newpoints,
58.             const IplImage *frame)
59. {
60.     vector<CvRect> faces = FaceDetector::facedetector.detect(frame);
61.
62.     if (faces.size() == 1) {
63.     ifstream origfile((filename + "-orig-points.txt").c_str());
64.     ifstream facefile(newpoints.c_str());
65.     if (!origfile.is_open() || !facefile.is_open())
66.         throw ios_base::failure("File not found");
67.
68.     // todo: memory leak here, change to scoped_ptr!
69.     orig_grey.reset(cvLoadImage((filename + "-orig-grey.png").c_str(), 0));
70.     orig_pyramid.reset(cvLoadImage((filename + "-orig-pyramid.png").c_str(), 0));
71.
72.     vector<Point> temppoints;
73.     origfile >> temppoints;
74.     convert(temppoints, origpoints);
75.
76.     facefile >> temppoints;
77.     temppoints = pointbetweenrects(temppoints, cvRect(0,0,1,1), faces[0]);
78.     convert(temppoints, currentpoints);
79.     lastpoints = currentpoints;
80.     }
81.     else
82.     throw ios_base::failure("No face found in the image");
83. }
84.
85. int PointTracker::getClosestTracker(const Point &point) {
86.     vector<Point> points;
87.     convert(currentpoints, points);
88.     return point.closestPoint(points);
89. }
90.
91. void PointTracker::removetracker(int id) {
92.     currentpoints.erase(currentpoints.begin()+id);
93.     lastpoints.erase(lastpoints.begin()+id);
94.     origpoints.erase(origpoints.begin()+id);
95. }
96.
97. void PointTracker::synchronizepoints() {
98.     swap(orig_grey, grey);
99.     swap(orig_pyramid, pyramid);
100.                 origpoints = lastpoints = currentpoints;
101.             }
```

```
102.
103.        void PointTracker::updatetracker(int id, const Point &point) {
104.           currentpoints[id] = point.cvpoint32();
105.           synchronizepoints();
106.        }
107.
108.        void PointTracker::addtracker(const Point &point) {
109.           currentpoints.push_back(point.cvpoint32());
110.           synchronizepoints();
111.        }
112.
113.        void PointTracker::cleartrackers() {
114.           currentpoints.clear();
115.           synchronizepoints();
116.        }
117.
118.        void PointTracker::track(const IplImage *frame, int pyramiddepth)
119.        {
120.           assert(lastpoints.size() == currentpoints.size());
121.           assert(origpoints.size() == currentpoints.size());
122.           status.resize(currentpoints.size());
123.           cvCvtColor(frame, grey.get(), CV_BGR2GRAY );
124.           if (!currentpoints.empty()) {
125.           // first calculate the new position of the features based
126.           // on the (pyramidal) last frame and position estimations
127.           cvCalcOpticalFlowPyrLK(last_grey.get(), grey.get(),
128.                    last_pyramid.get(), pyramid.get(),
129.                    &lastpoints[0], &currentpoints[0], pointcount(),
130.                    cvSize(win_size,win_size), 2, &status[0], 0,
131.                    cvTermCriteria(CV_TERMCRIT_ITER|
132.                         CV_TERMCRIT_EPS,20,0.01),
133.                    flags);
134.
135.           // then calculate the position based on the original
136.           // template without any pyramids
137.           cvCalcOpticalFlowPyrLK(orig_grey.get(), grey.get(),
138.                    orig_pyramid.get(), pyramid.get(),
139.                    &origpoints[0], &currentpoints[0], pointcount(),
140.                    cvSize(win_size, win_size),
141.                    pyramiddepth, &status[0], 0,
142.                    cvTermCriteria(CV_TERMCRIT_ITER|
143.                         CV_TERMCRIT_EPS,20,0.01),
144.                    flags);
145.
146.
147.           flags |= CV_LKFLOW_PYR_A_READY;
148.           }
149.
```

```
150.                cvCopy(grey.get(), last_grey.get(), 0);
151.                cvCopy(pyramid.get(), last_pyramid.get(), 0);
152.                lastpoints = currentpoints;
153.            }
154.
155.            int PointTracker::countactivepoints(void) {
156.                return count_if(status.begin(), status.end(),
157.                        bind1st(not_equal_to<char>(), 0));
158.            }
159.
160.            bool PointTracker::areallpointsactive(void) {
161.                return count(status.begin(), status.end(), 0) == 0;
162.            }
163.
164.            void PointTracker::draw(IplImage *canvas) {
165.                for(int i=0; i< (int) currentpoints.size(); i++)
166.                cvCircle( canvas, cvPointFrom32f(currentpoints[i]), 3,
167.                    status[i]?(i == eyepoint1 || i == eyepoint2 ?
168.                        CV_RGB(255,0,0):
169.                        CV_RGB(0,255,0)):
170.                    CV_RGB(0,0,255),
171.                    -1, 8,0);
172.            }
173.
174.            int PointTracker::pointcount() {
175.                return currentpoints.size();
176.            }
177.
178.            vector<HomPoint>
179.            PointTracker::getpoints(const vector<CvPoint2D32f> PointTracker::*points,
180.                    bool allpoints)
181.            {
182.                vector<HomPoint> vec;
183.                for(int i=0; i<pointcount(); i++)
184.                if (allpoints || status[i])
185.                    vec.push_back(HomPoint((this->*points)[i].x,
186.                            (this->*points)[i].y));
187.                return vec;
188.            }
```

PointTracker.h

```
1.  #pragma once
2.  #include <opencv/cv.h>
3.  #include <vector>
4.  #include <vgl/vgl_homg_point_2d.h>
5.  #include "utils.h"
```

```
6.
7.  using namespace std;
8.  typedef vgl_vector_2d<double> HomPoint;
9.
10. class TrackingException: public exception {};
11.
12. class PointTracker {
13. public:
14.     static const int eyepoint1 = 0;
15.     static const int eyepoint2 = 1;
16.     vector<char> status;
17.     vector<CvPoint2D32f> origpoints, currentpoints, lastpoints;
18.
19.  private:
20.     static const int win_size = 11;
21.     int flags;
22.
23.     scoped_ptr<IplImage> grey, orig_grey, pyramid, orig_pyramid,
24.     last_grey, last_pyramid;
25.
26.     void synchronizepoints();
27.
28. public:
29.     PointTracker(const CvSize &size);
30.     void cleartrackers();
31.     void addtracker(const Point &point);
32.     void updatetracker(int id, const Point &point);
33.     void removetracker(int id);
34.     int getClosestTracker(const Point &point);
35.     void track(const IplImage *frame, int pyramiddepth=1);
36.     int countactivepoints(void);
37.     bool areallpointsactive(void);
38.     int pointcount();
39.     void draw(IplImage *canvas);
40.
41.     vector<HomPoint>
42.     getpoints(const vector<CvPoint2D32f> PointTracker::*points,
43.         bool allpoints=true);
44.
45.     void save(string filename, string newname, const IplImage *frame);
46.     void load(string filename, string newname, const IplImage *frame);
47. };
```

TrackingSystem.cpp
Το σύστημα που υλοποιεί το tracking.

```
1.  #include "TrackingSystem.h"
2.  #include "FeatureDetector.h"
```

```
3.  #include "BlinkDetector.h"
4.
5.  static double quadraticminimum(double xm1, double x0, double xp1) {
6.  //    cout << "values:" << xm1 << " " << x0 << " " << xp1 << endl;
7.      return (xm1 - xp1) / (2*(xp1 + xm1 - 2*x0));
8.  }
9.
10. static Point subpixelminimum(const IplImage *values) {
11.     CvPoint maxpoint;
12.     cvMinMaxLoc(values, NULL, NULL, &maxpoint);
13. //    cout << "max: " << maxpoint.x << " " << maxpoint.y << endl;
14.
15.     int x = maxpoint.x;
16.     int y = maxpoint.y;
17.
18.     Point p(x, y);
19.
20.     if (x > 0 && x < 6)
21.     p.x += quadraticminimum(cvGetReal2D(values, y, x-1),
22.                 cvGetReal2D(values, y, x+0),
23.                 cvGetReal2D(values, y, x+1));
24.
25.     if (y > 0 && y < 4)
26.     p.y += quadraticminimum(cvGetReal2D(values, y-1, x),
27.                 cvGetReal2D(values, y+0, x),
28.                 cvGetReal2D(values, y+1, x));
29.
30.     return p;
31. }
32.
33.
34. TrackingSystem::TrackingSystem(CvSize size):
35.     tracker(size), headtracker(tracker), headcomp(headtracker), eyex(tracker)
36. {}
37.
38.
39. void TrackingSystem::doprocessing(const IplImage *frame,
40.                 IplImage *image)
41. {
42.     tracker.track(frame, 2);
43.     if (tracker.countactivepoints() < 4) {
44.     tracker.draw(image);
45.     throw TrackingException();
46.     }
47.
48.     headtracker.updatetracker();
49.     eyex.extractEye(frame); // throws Tracking Exception
50.     gazetracker.update(eyex.eyefloat.get());
```

```
51.
52.     displayeye(image, 0, 0, 0, 2);
53.     tracker.draw(image);
54.     headtracker.draw(image);
55. }
56.
57. void TrackingSystem::displayeye(IplImage *image,
58.                 int basex, int basey, int stepx, int stepy)
59. {
60.     CvSize eyesize = EyeExtractor::eyesize;
61.     int eyedx = EyeExtractor::eyedx;
62.     int eyedy = EyeExtractor::eyedy;
63.
64.     static IplImage *eyegreytemp = cvCreateImage( eyesize, 8, 1 );
65.     static FeatureDetector features(EyeExtractor::eyesize);
66.
67.     features.addSample(eyex.eyegrey.get());
68.
69.     basex *= 2*eyedx; basey *= 2*eyedy;
70.     stepx *= 2*eyedx; stepy *= 2*eyedy;
71.
72.     gazetracker.draw(image, eyedx, eyedy);
73.
74.     cvSetImageROI(image, cvRect(basex, basey, eyedx*2, eyedy*2));
75.     cvCvtColor(eyex.eyegrey.get(), image, CV_GRAY2RGB);
76.
77.     cvSetImageROI(image, cvRect(basex + stepx*1, basey + stepy*1,
78.                 eyedx*2, eyedy*2));
79.     cvCvtColor(eyex.eyegrey.get(), image, CV_GRAY2RGB);
80.
81.     cvConvertScale(features.getMean().get(),  eyegreytemp);
82.     cvSetImageROI(image, cvRect(basex, basey, eyedx*2, eyedy*2));
83.     cvCvtColor(eyegreytemp, image, CV_GRAY2RGB);
84.
85. // //     features.getVariance(eyegreytemp);
86. // //     cvSetImageROI(image, cvRect(basex, basey+stepy*2, eyedx*2, eyedy*2));
87. // //     cvCvtColor(eyegreytemp, image, CV_GRAY2RGB);
88.
89. //     // compute the x-derivative
90.
91. //     static IplImage *eyegreytemp1 = cvCreateImage( eyesize, IPL_DEPTH_32F, 1 );
92. //     static scoped_ptr<IplImage>
93. //  eyegreytemp2(cvCreateImage(eyesize, IPL_DEPTH_32F, 1));
94.
95. // //     static IplImage *eyegreytemp3 = cvCreateImage( eyesize, IPL_DEPTH_32F, 1 );
96. //     static IplImage *eyegreytemp4 = cvCreateImage(cvSize(7,5),IPL_DEPTH_32F,1);
97. //     features.getMean(eyegreytemp1);
98. //     cvConvertScale(eyex.eyegrey, eyegreytemp2.get());
```

```
99. //    double distance = cvNorm(eyegreytemp1, eyegreytemp2.get(), CV_L2);
100.                    //    static BlinkDetector blinkdet;
101.                    //    blinkdet.update(eyegreytemp2);
102.                    //    cout << "distance: " << distance
103.                    //   << " blink: " << blinkdet.getState() <<endl;
104.
105.                    //    cvSetImageROI(eyegreytemp1, cvRect(2,2,eyedx*2-6,eyedy*2-4));
106.                    //    cvMatchTemplate(eyegreytemp2.get(), eyegreytemp1, eyegreytemp4,
    CV_TM_SQDIFF);
107.                    //    cvResetImageROI(eyegreytemp1);
108.
109.                    //    for(int i=0; i<5; i++) {
110.                    //  cout << endl;
111.                    //  for(int j=0; j<7; j++)
112.                    //     cout << cvGetReal2D(eyegreytemp4, i, j)/1e6 << " ";
113.                    //    }
114.                    //    cout << endl;
115.
116.                    //    CvPoint maxpoint;
117.                    //    cvMinMaxLoc(eyegreytemp4, NULL, NULL, &maxpoint);
118.                    //    cout << "max: " << maxpoint.x << " " << maxpoint.y << endl;
119.
120.                    //    cvSetImageROI(eyex.eyegrey,
121.                    //       cvRect(maxpoint.x, maxpoint.y, eyedx*2-6, eyedy*2-4));
122.                    //    cvSetImageROI(image, cvRect(basex, basey+stepy*3, eyedx*2-6, eyedy*2-4));
123.                    //    cvCvtColor(eyex.eyegrey, image, CV_GRAY2RGB);
124.                    //    cvResetImageROI(eyex.eyegrey);
125.
126.                    //    Point mxpoint = subpixelminimum(eyegreytemp4);
127.                    //    cout << "max: " << mxpoint.x << " " << mxpoint.y << endl;
128.
129.                    //    tracker.currentpoints[0].x += 0.4 * (mxpoint.x - 3.0);
130.                    //    tracker.currentpoints[0].y += 0.4 * (mxpoint.y - 2.0);
131.
132.                    //    cvSub(eyex.eyefloat, eyegreytemp1, eyegreytemp3);
133.                    //    cvSetImageROI(eyegreytemp1, cvRect(0,0,eyedx*2-1,eyedy*2));
134.                    //    cvSetImageROI(eyegreytemp2, cvRect(1,0,eyedx*2-1,eyedy*2));
135.                    //    cvCopy(eyegreytemp1, eyegreytemp2);
136.                    //    cvResetImageROI(eyegreytemp1);
137.                    //    cvResetImageROI(eyegreytemp2);
138.                    //    cvAddS(eyegreytemp1, cvScalar(128.0), eyegreytemp1);
139.                    //    cvSub(eyegreytemp1, eyegreytemp2, eyegreytemp1);
140.
141.                    //    cvSetImageROI(image, cvRect(basex, basey+stepy*2, eyedx*2, eyedy*2));
142.                    //    cvConvertScale(eyegreytemp1, eyegreytemp);
143.
144.                    // //    cvMul(eyegreytemp3, eyegreytemp1, eyegreytemp3);
145.                    // //    cout << "x movement: " << cvAvg(eyegreytemp3).val[0] << endl;
```

```
146.
147.
148.                    //    cvCvtColor(eyegreytemp, image, CV_GRAY2RGB);
149.
150.
151.
152.                       cvResetImageROI(image);
153.                    }
```

TrackingSystem.h

```
1.  #pragma once
2.  #include "utils.h"
3.  #include "PointTracker.h"
4.  #include "HeadTracker.h"
5.  #include "HeadCompensation.cpp"
6.  #include "EyeExtractor.h"
7.  #include "OutputMethods.h"
8.  #include "GazeTracker.h"
9.
10. struct TrackingSystem {
11.     PointTracker tracker;
12.     HeadTracker headtracker;
13.     HeadCompensation headcomp;
14.     EyeExtractor eyex;
15.     GazeTracker gazetracker;
16.
17.     TrackingSystem(CvSize size);
18.     void doprocessing(const IplImage *frame,
19.             IplImage *image);
20.     void displayeye(IplImage *image, int basex, int basey,
21.           int stepx, int stepy);
22.
23. };
```

Utils.cpp

```
1.  #include "utils.h"
2.
3.  void releaseImage(IplImage *image) {
4.  //     cout << "deleting shared image" << endl;
5.      cvReleaseImage(&image);
6.  }
7.
8.  shared_ptr<IplImage> createImage(const CvSize &size, int depth, int channels) {
```

```
9.      return shared_ptr<IplImage>(cvCreateImage(size, depth, channels),
10.                 releaseImage);
11. }
12.
13. namespace boost {
14.     template<>
15.     void checked_delete(IplImage *image) {
16. //  cout << "deleting scoped image" << endl;
17.     if (image)
18.         cvReleaseImage(&image);
19.     }
20. }
```

Utils.h

```
1.  #pragma once
2.
3.  #include <opencv/cv.h>
4.  #include <math.h>
5.  #include <vector>
6.  #include <iostream>
7.  #include <vnl/algo/vnl_svd.h>
8.  #include <gdkmm.h>
9.  #include <boost/shared_ptr.hpp>
10. #include <boost/scoped_ptr.hpp>
11.
12. //#define DEBUG
13.
14. using namespace std;
15. using namespace boost;
16.
17. #define xforeach(iter,container) \
18.     for(typeof(container.begin()) iter = container.begin(); \
19.     iter != container.end(); iter++)
20.
21. #define xforeachback(iter,container) \
22.     for(typeof(container.rbegin()) iter = container.rbegin();   \
23.     iter != container.rend(); iter++)
24.
25. typedef vnl_vector<double> Vector;
26. typedef vnl_matrix<double> Matrix;
27. template <class T>
28. inline T square(T a) {
29.     return a * a;
30. }
31. template <class T>
32. ostream& operator<< (ostream& out, vector<T> const& vec) {
```

```
33.     out << vec.size() << endl;
34.     xforeach(iter, vec)
35.     out << *iter << endl;
36.     return out;
37. }
38.
39. template <class T>
40. istream& operator>> (istream& in, vector<T> &vec) {
41.     int size;
42.     T element;
43.
44.     vec.clear();
45.     in >> size;
46.     for(int i=0; i<size; i++) {
47.     in >> element;
48.     vec.push_back(element);
49.     }
50.     return in;
51. }
52. template <class T>
53. T teefunction(T source, char* prefix, char *postfix="\n") {
54.     cout << prefix << source << postfix;
55.     return source;
56. }
57.
58. #define debugtee(x) teefunction(x, #x ": ")
59. template <class T>
60. void savevector(CvFileStorage *out, const char* name, vector<T>& vec) {
61.     cvStartWriteStruct(out, name, CV_NODE_SEQ);
62.     for(int i=0; i<vec.size(); i++)
63.     vec[i].save(out);
64.     cvEndWriteStruct(out);
65. }
66.
67. template <class T>
68. vector<T> loadvector(CvFileStorage *in, CvFileNode *node) {
69.     CvSeq *seq = node->data.seq;
70.     CvSeqReader reader;
71.     cvStartReadSeq(seq, &reader, 0);
72.     vector<T> result(seq->total);
73.     for(int i=0; i<seq->total; i++) {
74.     CvFileNode *item = (CvFileNode*) reader.ptr;
75.     result[i].load(in, item);
76.     CV_NEXT_SEQ_ELEM(seq->elem_size, reader);
77.     }
78.     return result;
79. }
80. #include <gtkmm.h>
```

```
81.
82. #include "Point.h"
83.
84. template <class From, class To>
85. void convert(const From& from, To& to) {
86.     to = from;
87. }
88.
89. template <class From, class To>
90. void convert(const vector<From> &from, vector<To> &to) {
91.     to.resize(from.size());
92.     for(int i=0; i< (int) from.size(); i++)
93.     convert(from[i], to[i]);
94. }
95.
96. class ConstancyDetector {
97.     int value;
98.     int counter;
99.     int maxcounter;
100.                    public:
101.                        ConstancyDetector(int maxcounter) :
102.                        value(-1), counter(0), maxcounter(maxcounter) {}
103.
104.                        bool isStable(void) {
105.                        return counter >= maxcounter;
106.                        }
107.
108.                        bool isStableExactly(void) {
109.                        return counter == maxcounter;
110.                        }
111.
112.                        bool observe(int newvalue) {
113.                        if (newvalue != value || newvalue < 0)
114.                            counter = 0;
115.                        else
116.                            counter++;
117.                        value = newvalue;
118.                        return isStable();
119.                        }
120.
121.                        void reset(void) {
122.                        counter = 0;
123.                        value = -1;
124.                        }
125.                    };
126.
127.                    // #define output(X) { cout << #X " = " << X << endl; }
128.                    template <class T>
```

```
129.        int maxabsindex(T const& vec, int size) {
130.            int maxindex = 0;
131.            for(int i=0; i<size; i++)
132.            if (fabs(vec[i]) > fabs(vec[maxindex]))
133.                maxindex = i;
134.            return maxindex;
135.        }
136.
137.        namespace boost {
138.            template <>
139.            void checked_delete(IplImage *image);
140.        }
141.
142.        void releaseImage(IplImage *image);
143.        shared_ptr<IplImage> createImage(const CvSize &size, int depth, int  channes);
144.        typedef shared_ptr<const IplImage> SharedImage;
```

Σε δέυτερη φάση θα αναλύσουμε την δεύτερη και τελική εφαρμογή που αναπτύξαμε για το Eye-tracking.
Ώπως έχουμε προαναφέρει το πρόγραμμα που αναπτύχθηκε σε δεύτερη φάση

Παράρτημα Ά

Ολόκληρο τον κώδικα της εφαρμογής μπορείτε να τον βρείτε στον δίσκο (cd) που συνοδεύει την παρούσα πτυχιακή εργασία.

Παρατίθεται ο κώδικας του Interpolation .

```
1:  using System;
2:  using System.Collections.Generic;
3:  using System.Linq;
4:  using Emgu.CV;
5:  using GazeTrackingLibrary.Detection.BlobAnalysis;
6:  using GazeTrackingLibrary.Detection.Glint;
7:  using GTCommons.Enum;
8:  using GTSettings;
9:  using GazeTrackingLibrary.Utils;
10:
11:  namespace GazeTrackingLibrary.Calibration
12:  {
13:
14:      #region Pupil-only calibration
15:
16:      /// <summary>
17:      /// Calibration of polynomial with pupil center only
18:      /// </summary>
19:      public class CalibPupil : CalibMethod
20:      {
21:          #region Constructor
22:
23:          public CalibPupil()
24:          {
25:              //Initialization
26:              IsCalibrated = false;
27:              CalibrationTargets = new List<CalibrationTarget>();
28:          }
29:
30:          #endregion
31:
32:          #region Calibrate
33:
34:          public override bool Calibrate()
35:          {
36:              if (NumImages == 0)
37:              {
```

```
  38:                 //throw new ArgumentException("numImages=0 in
Calibrate()");
  39:                 return false;
  40:             }
  41:
  42:             try
  43:             {
  44:                 CalibrationDataLeft = new CalibrationData();
  45:                 CalibrationDataRight = new CalibrationData();
  46:
  47:                 var targets = new Matrix<double>(NumImages, 3);
  48:                 var designMatrixLeft = new Matrix<double>(NumImages,
6);
  49:                 var designMatrixRight = new
Matrix<double>(NumImages, 6);
  50:
  51:                 var rowLeft = new double[6];
  52:                 var rowRight = new double[6];
  53:
  54:                 int k = 0;
  55:
  56:                 foreach (CalibrationTarget ct in CalibrationTargets)
  57:                 {
  58:                     for (int j = 0; j < ct.NumImages; j++)
  59:                     {
  60:                         targets[k, 0] = ct.targetCoordinates.X;
  61:                         targets[k, 1] = ct.targetCoordinates.Y;
  62:
  63:                         double xLeft = ct.pupilCentersLeft[j].X;
  64:                         double yLeft = ct.pupilCentersLeft[j].Y;
  65:
  66:                         rowLeft[0] = 1;
  67:                         rowLeft[1] = xLeft;
  68:                         rowLeft[2] = yLeft;
  69:                         rowLeft[3] = xLeft*yLeft;
  70:                         rowLeft[4] = xLeft*xLeft;
  71:                         rowLeft[5] = yLeft*yLeft;
  72:
  73:                         for (int r = 0; r < 6; r++)
  74:                         {
  75:                             designMatrixLeft[k, r] = rowLeft[r];
  76:                         }
  77:
  78:                         if
(Settings.Instance.Processing.TrackingMode == TrackingModeEnum.Binocular)
  79:                         {
  80:                             double xRight =
ct.pupilCentersRight[j].X;
  81:                             double yRight =
ct.pupilCentersRight[j].Y;
  82:
  83:                             rowRight[0] = 1;
  84:                             rowRight[1] = xRight;
  85:                             rowRight[2] = yRight;
  86:                             rowRight[3] = xRight*yRight;
  87:                             rowRight[4] = xRight*xRight;
  88:                             rowRight[5] = yRight*yRight;
  89:
  90:                             for (int r = 0; r < 6; r++)
```

```
 91:                                 {
 92:                                     designMatrixRight[k, r] =
rowRight[r];
 93:                                 }
 94:                             }
 95:                             k++;
 96:                         }
 97:                     }
 98:
 99:                     CalibrationDataLeft.CoeffsX = new Matrix<double>(6,
1);
100:                     CalibrationDataLeft.CoeffsY = new Matrix<double>(6,
1);
101:                     CalibrationDataLeft.CoeffsX =
Operations.SolveLeastSquares(designMatrixLeft, targets.GetCol(0));
102:                     CalibrationDataLeft.CoeffsY =
Operations.SolveLeastSquares(designMatrixLeft, targets.GetCol(1));
103:
104:                     if (Settings.Instance.Processing.TrackingMode ==
TrackingModeEnum.Binocular)
105:                     {
106:                         CalibrationDataRight.CoeffsX = new
Matrix<double>(6, 1);
107:                         CalibrationDataRight.CoeffsY = new
Matrix<double>(6, 1);
108:                         CalibrationDataRight.CoeffsX =
Operations.SolveLeastSquares(designMatrixRight, targets.GetCol(0));
109:                         CalibrationDataRight.CoeffsY =
Operations.SolveLeastSquares(designMatrixRight, targets.GetCol(1));
110:                     }
111:
112:                     // For each image we calculate the estimated gaze
coordinates
113:                     foreach (CalibrationTarget ct in CalibrationTargets)
114:                     {
115:                         // We might be recalibrating so clear
estGazeCoords first
116:                         ct.estimatedGazeCoordinatesLeft.Clear();
117:                         ct.estimatedGazeCoordinatesRight.Clear();
118:
119:                         for (int j = 0; j < ct.NumImages; j++)
120:                         {
121:                             PupilCenterLeft = ct.pupilCentersLeft[j];
122:
ct.estimatedGazeCoordinatesLeft.Add(GetGazeCoordinates(EyeEnum.Left));
123:
124:                             if
(Settings.Instance.Processing.TrackingMode == TrackingModeEnum.Binocular)
125:                             {
126:                                 PupilCenterRight =
ct.pupilCentersRight[j];
127:
ct.estimatedGazeCoordinatesRight.Add(GetGazeCoordinates(EyeEnum.Right));
128:                             }
129:                         }
130:
131:                         ct.CalculateAverageCoords();
132:                         ct.averageErrorLeft =
Operations.Distance(ct.meanGazeCoordinatesLeft, ct.targetCoordinates);
```

```
133:
134:                     if (Settings.Instance.Processing.TrackingMode ==
TrackingModeEnum.Binocular)
135:                         ct.averageErrorRight =
Operations.Distance(ct.meanGazeCoordinatesRight, ct.targetCoordinates);
136:                 }
137:
138:                 //calibrated = true;
139:                 CalibrationDataLeft.Calibrated = true;
140:                 CalculateAverageErrorLeft();
141:                 CalculateDegreesLeft();
142:
143:                 if (Settings.Instance.Processing.TrackingMode ==
TrackingModeEnum.Binocular)
144:                 {
145:                     CalibrationDataRight.Calibrated = true;
146:                     CalculateAverageErrorRight();
147:                     CalculateDegreesRight();
148:                 }
149:             }
150:             catch (Exception ex)
151:             {
152:                 //IsCalibrated = false;
153:                 return true; // what to do here
154:             }
155:
156:             IsCalibrated = true;
157:             return IsCalibrated;
158:
159:             //OnCalibrationComplete(EventArgs.Empty); // Raise event
160:         }
161:
162:         #endregion
163:
164:         #region Get Gaze Coordinates
165:
166:         public override GTPoint GetGazeCoordinates(TrackData
trackData, EyeEnum eye)
167:         {
168:             var row = new Matrix<double>(6, 1);
169:             var screenCoordinates = new Matrix<double>(2, 1);
170:
171:             var gazedPoint = new GTPoint();
172:             double x, y;
173:
174:             if (eye == EyeEnum.Left)
175:             {
176:                 x = trackData.PupilDataLeft.Center.X;
177:                 y = trackData.PupilDataLeft.Center.Y;
178:             }
179:             else
180:             {
181:                 x = trackData.PupilDataRight.Center.X;
182:                 y = trackData.PupilDataRight.Center.Y;
183:             }
184:
185:             row[0, 0] = 1;
186:             row[1, 0] = x;
187:             row[2, 0] = y;
```

```
188:                row[3, 0] = x*y;
189:                row[4, 0] = x*x;
190:                row[5, 0] = y*y;
191:
192:                if (eye == EyeEnum.Left)
193:                {
194:                    gazedPoint.X = CvInvoke.cvDotProduct(row.Ptr,
CalibrationDataLeft.CoeffsX.Ptr);
195:                    gazedPoint.Y = CvInvoke.cvDotProduct(row.Ptr,
CalibrationDataLeft.CoeffsY.Ptr);
196:                }
197:                else
198:                {
199:                    gazedPoint.X = CvInvoke.cvDotProduct(row.Ptr,
CalibrationDataRight.CoeffsX.Ptr);
200:                    gazedPoint.Y = CvInvoke.cvDotProduct(row.Ptr,
CalibrationDataRight.CoeffsY.Ptr);
201:                }
202:
203:                return gazedPoint;
204:            }
205:
206:            public GTPoint GetGazeCoordinates(EyeEnum eye)
207:            {
208:                var row = new Matrix<double>(6, 1);
209:                var screenCoordinates = new Matrix<double>(2, 1);
210:
211:                var gazedPoint = new GTPoint();
212:                double x, y;
213:
214:                if (eye == EyeEnum.Left)
215:                {
216:                    x = PupilCenterLeft.X;
217:                    y = PupilCenterLeft.Y;
218:                }
219:                else
220:                {
221:                    x = PupilCenterRight.X;
222:                    y = PupilCenterRight.Y;
223:                }
224:
225:                row[0, 0] = 1;
226:                row[1, 0] = x;
227:                row[2, 0] = y;
228:                row[3, 0] = x*y;
229:                row[4, 0] = x*x;
230:                row[5, 0] = y*y;
231:
232:                if (eye == EyeEnum.Left)
233:                {
234:                    gazedPoint.X = CvInvoke.cvDotProduct(row.Ptr,
CalibrationDataLeft.CoeffsX.Ptr);
235:                    gazedPoint.Y = CvInvoke.cvDotProduct(row.Ptr,
CalibrationDataLeft.CoeffsY.Ptr);
236:                }
237:                else
238:                {
239:                    gazedPoint.X = CvInvoke.cvDotProduct(row.Ptr,
CalibrationDataRight.CoeffsX.Ptr);
```

```
240:                     gazedPoint.Y = CvInvoke.cvDotProduct(row.Ptr,
CalibrationDataRight.CoeffsY.Ptr);
241:                 }
242:
243:                 return gazedPoint;
244:             }
245:
246:             #endregion
247:         }
248:
249:         #endregion
250:
251:         #region Pupil-glint(s) calibration
252:
253:         /// <summary>
254:         /// Calibration of a polynomial where we use the normalized
pupil center.
255:         /// We normalize using the average coordinates of the glints
stored in glintConfig.
256:         /// </summary>
257:         public class CalibPolynomial : CalibMethod
258:         {
259:             private int numOutliersRemovedLeft;
260:             private int numOutliersRemovedRight;
261:
262:             #region Constructor
263:
264:             public CalibPolynomial()
265:             {
266:                 IsCalibrated = false;
267:                 CalibrationTargets = new List<CalibrationTarget>();
268:             }
269:
270:             #endregion
271:
272:             #region Calibrate
273:
274:             public override bool Calibrate()
275:             {
276:                 if (numOutliersRemovedLeft == 0 &&
numOutliersRemovedRight == 0)
277:                     RemoveOutliers(); // Only works sometimes, tried
fixing it..
278:
279:                 if (NumImages == 0)
280:                 {
281:                     //throw new ArgumentException("numImages=0 in
Calibrate()");
282:                     IsCalibrated = false;
283:                     return false;
284:                 }
285:
286:                 #region Initialize variabels
287:
288:                 CalibrationDataLeft = new CalibrationData();
289:                 CalibrationDataRight = new CalibrationData();
290:
291:
292:                 var targets = new Matrix<double>(NumImages, 3);
```

```
293:                var designMatrixLeft = new Matrix<double>(NumImages, 6);
294:                var designMatrixRight = new Matrix<double>(NumImages,
6);
295:
296:                var rowLeft = new double[6];
297:                var rowRight = new double[6];
298:
299:                int k = 0;
300:
301:                #endregion
302:
303:                #region Build matrices
304:
305:                foreach (CalibrationTarget ct in CalibrationTargets)
306:                {
307:                    for (int j = 0; j < ct.NumImages; j++)
308:                    {
309:                        #region Left
310:
311:                        if (j < ct.pupilCentersLeft.Count && j <
ct.glintsLeft.Count)
312:                        {
313:                            GTPoint pupilCenterLeft =
ct.pupilCentersLeft.ElementAt(j);
314:                            GlintConfiguration glintsLeft =
ct.glintsLeft.ElementAt(j);
315:
316:                            if (pupilCenterLeft != null && glintsLeft !=
null && glintsLeft.Count > 0)
317:                            {
318:                                targets[k, 0] = ct.targetCoordinates.X;
319:                                targets[k, 1] = ct.targetCoordinates.Y;
320:
321:                                double xLeft = pupilCenterLeft.X -
glintsLeft.AverageCenter.X;
322:                                double yLeft = pupilCenterLeft.Y -
glintsLeft.AverageCenter.Y;
323:
324:                                rowLeft[0] = 1;
325:                                rowLeft[1] = xLeft;
326:                                rowLeft[2] = yLeft;
327:                                rowLeft[3] = xLeft*yLeft;
328:                                rowLeft[4] = xLeft*xLeft;
329:                                rowLeft[5] = yLeft*yLeft;
330:
331:                                for (int r = 0; r < 6; r++)
332:                                    designMatrixLeft[k, r] = rowLeft[r];
333:                            }
334:                        }
335:
336:                        #endregion
337:
338:                        #region Right
339:
340:                        if (Settings.Instance.Processing.TrackingMode ==
TrackingModeEnum.Binocular)
341:                        {
342:                            if (ct.pupilCentersRight.Count - 1 > j &&
ct.glintsRight.Count - 1 > j)
```

```
343:                             {
344:                                 GTPoint pupilCenterRight =
ct.pupilCentersRight.ElementAt(j);
345:                                 GlintConfiguration glintsRight =
ct.glintsRight.ElementAt(j);
346:
347:                                 if (pupilCenterRight != null &&
glintsRight != null && glintsRight.Count > 0)
348:                                 {
349:                                     double xRight = pupilCenterRight.X -
glintsRight.AverageCenter.X;
350:                                     double yRight = pupilCenterRight.Y -
glintsRight.AverageCenter.Y;
351:
352:                                     rowRight[0] = 1;
353:                                     rowRight[1] = xRight;
354:                                     rowRight[2] = yRight;
355:                                     rowRight[3] = xRight*yRight;
356:                                     rowRight[4] = xRight*xRight;
357:                                     rowRight[5] = yRight*yRight;
358:
359:                                     for (int r = 0; r < 6; r++)
360:                                     {
361:                                         designMatrixRight[k, r] =
rowRight[r];
362:                                     }
363:                                 }
364:                             }
365:                         }
366:
367:                         #endregion
368:
369:                         k++;
370:                     }
371:                 }
372:
373:                 #endregion
374:
375:                 #region SolveLeastSquares
376:
377:                 CalibrationDataLeft.CoeffsX = new Matrix<double>(6, 1);
378:                 CalibrationDataLeft.CoeffsY = new Matrix<double>(6, 1);
379:                 CalibrationDataLeft.CoeffsX =
Operations.SolveLeastSquares(designMatrixLeft, targets.GetCol(0));
380:                 CalibrationDataLeft.CoeffsY =
Operations.SolveLeastSquares(designMatrixLeft, targets.GetCol(1));
381:
382:                 if (Settings.Instance.Processing.TrackingMode ==
TrackingModeEnum.Binocular)
383:                 {
384:                     CalibrationDataRight.CoeffsX = new Matrix<double>(6,
1);
385:                     CalibrationDataRight.CoeffsY = new Matrix<double>(6,
1);
386:                     CalibrationDataRight.CoeffsX =
Operations.SolveLeastSquares(designMatrixRight, targets.GetCol(0));
387:                     CalibrationDataRight.CoeffsY =
Operations.SolveLeastSquares(designMatrixRight, targets.GetCol(1));
388:                 }
```

```
 389:
 390:                #endregion
 391:
 392:                #region Calculated est. gaze coordinates (per image)
 393:
 394:                // For each image we calculate the estimated gaze
coordinates
 395:                foreach (CalibrationTarget ct in CalibrationTargets)
 396:                {
 397:                    // We might be recalibrating so clear estGazeCoords
first
 398:                    ct.estimatedGazeCoordinatesLeft.Clear();
 399:                    ct.estimatedGazeCoordinatesRight.Clear();
 400:
 401:                    for (int j = 0; j < ct.NumImages; j++)
 402:                    {
 403:                        #region Left
 404:
 405:                        if (ct.pupilCentersLeft.Count - 1 >= j &&
ct.glintsLeft.Count - 1 >= j)
 406:                        {
 407:                            var pupilCenterLeft = new GTPoint(0, 0);
 408:                            var glintConfigLeft = new
GlintConfiguration(new Blobs());
 409:
 410:                            if (ct.pupilCentersLeft.ElementAt(j) !=
null)
 411:                                pupilCenterLeft =
ct.pupilCentersLeft[j];
 412:
 413:                            if (ct.glintsLeft.ElementAt(j) != null)
 414:                                glintConfigLeft = ct.glintsLeft[j];
 415:
 416:                            if (pupilCenterLeft.Y != 0)
 417:
ct.estimatedGazeCoordinatesLeft.Add(GetGazeCoordinates(EyeEnum.Left,
pupilCenterLeft,
 418:
glintConfigLeft));
 419:                        }
 420:
 421:                        #endregion
 422:
 423:                        #region Right
 424:
 425:                        if (Settings.Instance.Processing.TrackingMode ==
TrackingModeEnum.Binocular)
 426:                        {
 427:                            if (ct.pupilCentersRight.Count - 1 > j &&
ct.glintsRight.Count - 1 > j)
 428:                            {
 429:                                var pupilCenterRight = new GTPoint(0,
0);
 430:                                var glintConfigRight = new
GlintConfiguration(new Blobs());
 431:
 432:                                if (ct.pupilCentersRight.ElementAt(j) !=
null)
```

```
433:                                    pupilCenterRight =
ct.pupilCentersRight[j];
434:
435:                                if (ct.glintsRight.ElementAt(j) != null)
436:                                    glintConfigRight =
ct.glintsRight[j];
437:
438:                                if (pupilCenterRight.Y != 0)
439:
ct.estimatedGazeCoordinatesRight.Add(GetGazeCoordinates(EyeEnum.Right,
pupilCenterRight,
440:
glintConfigRight));
441:                            }
442:                        }
443:
444:                        #endregion
445:                    }
446:
447:                    ct.CalculateAverageCoords();
448:                    ct.averageErrorLeft =
Operations.Distance(ct.meanGazeCoordinatesLeft, ct.targetCoordinates);
449:
450:                    if (Settings.Instance.Processing.TrackingMode ==
TrackingModeEnum.Binocular)
451:                        ct.averageErrorRight =
Operations.Distance(ct.meanGazeCoordinatesRight, ct.targetCoordinates);
452:                }
453:
454:                CalibrationDataLeft.Calibrated = true;
455:                CalculateAverageErrorLeft();
456:                CalculateDegreesLeft();
457:
458:                if (Settings.Instance.Processing.TrackingMode ==
TrackingModeEnum.Binocular)
459:                {
460:                    CalibrationDataRight.Calibrated = true;
461:                    CalculateAverageErrorRight();
462:                    CalculateDegreesRight();
463:                }
464:
465:                #endregion
466:
467:                IsCalibrated = true;
468:                return IsCalibrated;
469:            }
470:
471:            private void RemoveOutliers()
472:            {
473:                var meanLeft = new GTPoint();
474:                var stddevLeft = new GTPoint();
475:                var meanRight = new GTPoint();
476:                var stddevRight = new GTPoint();
477:
478:                numOutliersRemovedLeft = 0;
479:                numOutliersRemovedRight = 0;
480:
481:                foreach (CalibrationTarget ct in CalibrationTargets)
482:                {
```

```
483:                    #region Calculate mean and std
484:
485:                    // Left
486:                    if (ct.DifferenceVectorLeft != null)
487:                    {
488:                        meanLeft =
Operations.Mean(ct.DifferenceVectorLeft);
489:                        stddevLeft =
Operations.StandardDeviation(ct.DifferenceVectorLeft);
490:                    }
491:
492:                    // Right
493:                    if (Settings.Instance.Processing.TrackingMode ==
TrackingModeEnum.Binocular)
494:                    {
495:                        if (ct.DifferenceVectorRight != null)
496:                        {
497:                            meanRight =
Operations.Mean(ct.DifferenceVectorRight);
498:                            stddevRight =
Operations.StandardDeviation(ct.DifferenceVectorRight);
499:                        }
500:                    }
501:
502:                    #endregion
503:
504:                    try
505:                    {
506:                        for (int i = 0; i < ct.NumImages - 1; i++)
507:                        {
508:                            // remove left
509:                            if (ct.DifferenceVectorLeft != null && i <=
ct.DifferenceVectorLeft.Length)
510:                                if
(Math.Abs(ct.DifferenceVectorLeft[i].X - meanLeft.X) > stddevLeft.X ||
511:
Math.Abs(ct.DifferenceVectorLeft[i].Y - meanLeft.Y) > stddevLeft.Y)
512:                                {
513:                                    if (ct.pupilCentersLeft.Count <= i)
514:                                        ct.pupilCentersLeft.RemoveAt(i -
numOutliersRemovedLeft);
515:
516:                                    if (ct.glintsLeft.Count <= i)
517:                                        ct.glintsLeft.RemoveAt(i -
numOutliersRemovedLeft);
518:
519:                                    numOutliersRemovedLeft++;
520:                                }
521:
522:                            // remove right (if binocular)
523:                            if
(Settings.Instance.Processing.TrackingMode == TrackingModeEnum.Binocular)
524:                            {
525:                                if (ct.DifferenceVectorRight != null &&
i <= ct.DifferenceVectorRight.Length)
526:                                    if
(Math.Abs(ct.DifferenceVectorRight[i].X - meanRight.X) > stddevRight.X ||
527:
Math.Abs(ct.DifferenceVectorRight[i].Y - meanRight.Y) > stddevRight.Y)
```

```
 528:                                    {
 529:                                        if (ct.pupilCentersRight.Count
<= i)
 530:
ct.pupilCentersRight.RemoveAt(i - numOutliersRemovedRight);
 531:
 532:                                        if (ct.glintsRight.Count <= i)
 533:                                            ct.glintsRight.RemoveAt(i -
numOutliersRemovedRight);
 534:
 535:                                        numOutliersRemovedRight++;
 536:                                    }
 537:                                }
 538:                                //Console.WriteLine("{0} outliers removed
out of a total of {1}, Old std: {2}, {3}, New std: {4}, {5}",
 539:                                //    numOutliersRemovedLeft, ct.NumImages,
stddevLeft.X, stddevLeft.Y,
Operations.StandardDeviation(ct.DifferenceVectorLeft).X,
 540:                                //
Operations.StandardDeviation(ct.DifferenceVectorLeft).Y);
 541:                            }
 542:                        }
 543:                        catch (Exception ex)
 544:                        {
 545:                            Console.Out.WriteLine("Calibration.cs, error
while removing outlier eye. Message: " + ex.Message);
 546:                        }
 547:                    }
 548:                }
 549:
 550:                #endregion
 551:
 552:                #region Get Gaze Coordinates
 553:
 554:                public override GTPoint GetGazeCoordinates(TrackData
trackData, EyeEnum eye)
 555:                {
 556:                    var row = new Matrix<double>(6, 1);
 557:                    var screenCoordinates = new Matrix<double>(2, 1);
 558:
 559:                    var gazedPoint = new GTPoint();
 560:
 561:                    try
 562:                    {
 563:                        double X = 0;
 564:                        double Y = 0;
 565:
 566:                        switch (eye)
 567:                        {
 568:                            case EyeEnum.Left:
 569:                                X = trackData.PupilDataLeft.Center.X -
trackData.GlintDataLeft.Glints.AverageCenter.X;
 570:                                Y = trackData.PupilDataLeft.Center.Y -
trackData.GlintDataLeft.Glints.AverageCenter.Y;
 571:                                break;
 572:                            default:
 573:                                X = trackData.PupilDataRight.Center.X -
trackData.GlintDataRight.Glints.AverageCenter.X;
```

```
574:                            Y = trackData.PupilDataRight.Center.Y -
trackData.GlintDataRight.Glints.AverageCenter.Y;
575:                            break;
576:                    }
577:
578:                    row[0, 0] = 1;
579:                    row[1, 0] = X;
580:                    row[2, 0] = Y;
581:                    row[3, 0] = X*Y;
582:                    row[4, 0] = X*X;
583:                    row[5, 0] = Y*Y;
584:
585:                    if (eye == EyeEnum.Left)
586:                    {
587:                        if (CalibrationDataLeft != null &&
CalibrationDataLeft.CoeffsX != null)
588:                        {
589:                            gazedPoint.X =
CvInvoke.cvDotProduct(row.Ptr, CalibrationDataLeft.CoeffsX.Ptr);
590:                            gazedPoint.Y =
CvInvoke.cvDotProduct(row.Ptr, CalibrationDataLeft.CoeffsY.Ptr);
591:                        }
592:                    }
593:                    else
594:                    {
595:                        if (CalibrationDataRight != null &&
CalibrationDataRight.CoeffsX != null)
596:                        {
597:                            gazedPoint.X =
CvInvoke.cvDotProduct(row.Ptr, CalibrationDataRight.CoeffsX.Ptr);
598:                            gazedPoint.Y =
CvInvoke.cvDotProduct(row.Ptr, CalibrationDataRight.CoeffsY.Ptr);
599:                        }
600:                    }
601:                }
602:                catch (Exception ex)
603:                {
604:                    Console.Out.WriteLine("Calibration.cs, exception in
GetGazeCoordinates(), message: " + ex.Message);
605:                }
606:
607:                return gazedPoint;
608:            }
609:
610:            public GTPoint GetGazeCoordinates(EyeEnum eye, GTPoint
pupilCenter, GlintConfiguration glintConfig)
611:            {
612:                var row = new Matrix<double>(6, 1);
613:                var screenCoordinates = new Matrix<double>(2, 1);
614:
615:                var gazedPoint = new GTPoint();
616:                double X, Y;
617:
618:                try
619:                {
620:                    X = pupilCenter.X - glintConfig.AverageCenter.X;
621:                    Y = pupilCenter.Y - glintConfig.AverageCenter.Y;
622:
623:                    row[0, 0] = 1;
```

```
624:                    row[1, 0] = X;
625:                    row[2, 0] = Y;
626:                    row[3, 0] = X*Y;
627:                    row[4, 0] = X*X;
628:                    row[5, 0] = Y*Y;
629:
630:                    if (eye == EyeEnum.Left)
631:                    {
632:                        gazedPoint.X = CvInvoke.cvDotProduct(row.Ptr,
CalibrationDataLeft.CoeffsX.Ptr);
633:                        gazedPoint.Y = CvInvoke.cvDotProduct(row.Ptr,
CalibrationDataLeft.CoeffsY.Ptr);
634:                    }
635:                    else
636:                    {
637:                        gazedPoint.X = CvInvoke.cvDotProduct(row.Ptr,
CalibrationDataRight.CoeffsX.Ptr);
638:                        gazedPoint.Y = CvInvoke.cvDotProduct(row.Ptr,
CalibrationDataRight.CoeffsY.Ptr);
639:                    }
640:                }
641:                catch (Exception ex)
642:                {
643:                    Console.Out.WriteLine("Calibration.cs, exception in
GetGazeCoordinates(), message: " + ex.Message);
644:                }
645:
646:                return gazedPoint;
647:            }
648:
649:            #endregion
650:        }
651:
652:        #endregion
653:    }
```