ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΠΑΡΑΚΟΛΟΥΘΗΣΗ ΚΙΝΗΣΗΣ ΤΟΥ ΙΔΙΟΥ ΑΝΤΙΚΕΙΜΕΝΟΥ ΣΕ ΠΟΛΛΑΠΛΕΣ ΚΑΜΕΡΕΣ ΑΣΦΑΛΕΙΑΣ

Εισηγητής : Μαλάμος Αθανάσιος
Σπουδαστής : Σταθόπουλος Κωνσταντίνος του Παναγιώτη
Α. Μ. : 1051

Ηράκλειο 2007

Ευρετήριο

Πρόλογος

Ο άνθρωπος εδώ και πολλά χρόνια κατέχεται από έναν φόβο σχετικά με την ασφάλεια στο σπίτι του, στο μαγαζί που ίσως να έχει, στην τράπεζα που διευθύνει και σε πολλά άλλα μέρη που θα δελέαζαν διάφορους επιτήδειους να προβούν σε απόπειρα κάποιας ληστείας.

Για αυτό το λόγο μέσα στα χρόνια αναπτύχθηκαν διάφορα συστήματα ασφάλειας και ελέγχου για την προστασία του πολίτη. Το ποιο διαδεδομένο όμως σύστημα ασφαλείας είναι με κάμερες οι οποίες έχουν σταθερή τοποθέτηση και παρακολουθούν η κάθε μια ένα κομμάτι. Αυτό όμως καθιστά απαραίτητη την ύπαρξη ενός χειριστή ο οποίος θα πρέπει να παρακολουθεί, είτε σε πραγματικό χρόνο όλες τις οθόνες που προβάλουν την λαμβανόμενη από τις κάμερες εικόνα με σκοπό να εντοπίσει εγκαίρως κάποια ύποπτη κίνηση, είτε να παρακολουθήσει κάποια άλλη στιγμή τα καταγεγραμμένα video από τις κάμερες. Αυτό όμως καθίσταται απίστευτα ασύμφορο, χρονοβόρο καθώς και όχι 100% αξιόπιστο. Ασύμφορο διότι όταν το κύκλωμα παρακολούθησης είναι αρκετά μεγάλο, δηλαδή να διαθέτει πάνε από 5 κάμερες για ένα κτίριο οι οποίες να είναι σε λειτουργία είκοσι τέσσερις ώρες το είκοσι τετράωρο, με αποτέλεσμα να χρειάζεται το σύστημα μεγάλους αποθηκευτικούς χορούς (σκληρούς δίσκους πολλών gigabytes). Χρονοβόρο διότι ο χειριστής θα πρέπει επί ώρες να παρακολουθεί τα καταγεγραμμένα video, παρακολουθώντας πολλές φορές άχρηστες πληροφορίες. Και όχι 100% αξιόπιστο διότι για να προλαβαίνει ο χειριστής να δει όλα τα video και να μην μαζεύεται μεγάλο φόρτο εργασίας καθότι θα συνεχίζεται η συσσώρευση εισερχομένων video από τις κάμερες που είναι σε λειτουργία θα αναγκάζεται, είτε να κοιτάζει σε fast forward τα video, είτε να τα κοιτάζει κομματιαστά. Αυτό έχει σαν αποτέλεσμα κάποια στιγμή ίσως να μην παρατήρηση κάτι σημαντικό σε κάποιο video.

Τι θα γινόταν όμως αν αυτό το σύστημα υποστηριζόταν από ένα πρόγραμμα που να λειτουργούσε σε πραγματικό χρόνο, να ανέλυε το κάθε video από κάθε κάμερα ξεχωριστά, με απώτερο σκοπό την αναγνώριση αντικειμένων που αποτελούν στόχο (π.χ. έναν άνθρωπο που κινείται), και τότε μόνο να γινόταν καταγραφή video ή να σήμανε κάποιος συναγερμός ή να ειδοποιούταν μέσω κάποιου είδους alert ο χειριστής; Σίγουρα ένα τέτοιο σύστημα θα ήταν πολύ ποιο αποδοτικό από αυτό που περιγράφτηκε προηγουμένως. Αλλά όλα αυτά στην σφαίρα της φαντασίας.

Πίσω τώρα στην πραγματικότητα, το αντικείμενο της παρούσας πτυχιακής εργασίας είναι η σχεδίαση και η υλοποίηση ενός συστήματος το οποίο θα αποτελέσει την βάση για την πραγματοποίηση ενός έξυπνου συστήματος παρακολούθησης και θα βγάλει από την σφαίρα της φαντασίας το προαναφερθέν σύστημα και θα το φέρει στην σφαίρα της πραγματικότητας.

Με άλλα λόγια στα πλαίσια αυτής της πτυχιακής εργασίας πραγματοποιήθηκε έρευνα καθώς και υλοποίηση ενός συστήματος το οποίο κάνει χρήση μιας κάμερας, κάνει λήψη του εισερχόμενου από αυτήν video το αναλύει σε πραγματικό χρόνο και με την χρήση πολλών και διάφορων αλγόριθμων ανιχνεύει την ύπαρξη αντικειμένων – στόχος που δομούνται με βάση το επιλεγμένο ή τα επιλεγμένα την δεδομένη στιγμή χρώματα και αν ανιχνευθεί κάποιο αντικείμενο αποθηκεύει για αυτό ένα video με όνομα το χρώμα του και την ημερομηνία και ώρα του συστήματος.

Κεφάλαιο 1. Θεωρητικό μέρος

1.1 Εισαγωγή

Σε αυτό το κεφάλαιο γίνεται μια θεωρητική παρουσίαση τεχνολογιών που χρησιμοποιήθηκαν για την δημιουργία της εφαρμογής όπως της γλώσσας προγραμματισμού Java, της δομής των εικόνων και των video, του αλγόριθμου που αναγνωρίζει και ξεχωρίζει τις χρωματικές περιοχές και του αλγόριθμου Convex Hull και της υλοποίησής του που χρησιμοποιήθηκε στην παρούσα εφαρμογή καθώς.

1.2 Εισαγωγή στην Java

1.2.1 Εισαγωγή

Η Java είναι δύο πράγματα μαζί, γλώσσα προγραμματισμού και πλατφόρμα. Η Java είναι υψηλού επιπέδου γλώσσα με τα ακόλουθα χαρακτηριστικά:

- **Αντικειμενοστραφής (Object oriented)**
- **Ασφαλής (secure):** Απαγορεύει στους εισβολείς να γράφουν προγράμματα που μπορούν να χαλάσουν τα συστήματα των χρηστών.
- **Μεταφερτή (portable):** Μπορεί να εκτελείται σε διαφορετικές πλατφόρμες χωρίς τροποποίηση όπως σε Windows και σε Macintosh.
- **Κατανεμημένη (distributed):** Δηλαδή διαφορετικά κομμάτια ενός προγράμματος μπορούν να προέλθουν από διαφορετικές πηγές.
- **Πολυνηματική (multithreaded):** Δηλαδή ένα απλό πρόγραμμα σε Java μπορεί να κάνει πολλά προγράμματα ανεξάρτητα και αλληλεπιδρώντα.

1.2.2 Η Java δεν εξαρτάται από πλατφόρμα

Ανεξαρτησία από πλατφόρμα είναι η δυνατότητα το ίδιο πρόγραμμα να εκτελείται σε διαφορετικές πλατφόρμες και λειτουργικά συστήματα. Τα

προγράμματα Java επιτυγχάνουν αυτή την ανεξαρτησία μέσω της χρήσης μιας εικονικής μηχανής (Virtual Machine) η οποία είναι κάτι σαν ένας υπολογιστής μέσα στον υπολογιστή. Ο πηγαίος κώδικας της Java μεταγλωττίζεται μέσω του compiler (μεταγλωττιστή). Η εικονική μηχανή παίρνει τα μεταγλωττισμένα προγράμματα Java και μετατρέπει τις εντολές τους σε εντολές που μπορεί να χειριστεί ένα λειτουργικό σύστημα. Το ίδιο μεταγλωττισμένο πρόγραμμα, που υπάρχει σε μια μορφή που καλείται bytecode, μπορεί να εκτελεσθεί σε οποιαδήποτε άλλη πλατφόρμα και λειτουργικό σύστημα που διαθέτει μια εικονική μηχανή της Java. Ο bytecode ενός προγράμματος Java μπορεί να τρέξει σε Windows NT, Solaris, Macintosh χωρίς καμιά αλλαγή.

Bytecode είναι η έκδοση κώδικα μηχανής της εικονικής μηχανής (VM) της Java, δηλαδή οι εντολές που καταλαβαίνει άμεσα. Η εικονική μηχανή (VM) είναι επίσης γνωστή σαν διερμηνευτής Java (Java interpreter).

Η Java είναι επίσης ανεξάρτητη πλατφόρμας σε επίπεδο πηγαίου κώδικα. Τα προγράμματα της Java αποθηκεύονται σαν αρχεία κειμένου πριν να μεταγλωττιστούν και τα αρχεία αυτά μπορούν να δημιουργηθούν σε οποιαδήποτε πλατφόρμα που υποστηρίζει την Java. Π.χ. μπορείτε να γράψετε ένα πρόγραμμα της Java σε ένα Macintosh και να το μεταγλωττίσετε σε bytecode σε ένα μηχάνημα με Windows 95.

Η δυνατότητα ένα αρχείο bytecode να εκτελείται σε διάφορες πλατφόρμες είναι αυτή που κάνει την Java να εργάζεται στο World Wide Web, επειδή το ίδιο το Web είναι ανεξάρτητο από πλατφόρμα. Όπως τα αρχεία HTML μπορούν να διαβαστούν σε οποιαδήποτε πλατφόρμα, οι μικρό-εφαρμογές (applets) της Java μπορούν να εκτελεστούν σε οποιαδήποτε πλατφόρμα που διαθέτει ένα browser (Internet Explorer, Mozilla, NetScape κ.α.) με δυνατότητες Java.

1.2.3 Η πλατφόρμα Java

Η πλατφόρμα Java περιλαμβάνει δύο συστατικά:

- Την εικονική μηχανή (Virtual Machine)
- Το Application Programming Interface (API).

Το Java API είναι μια μεγάλη συλλογή από έτοιμα προγράμματα της Java τα οποία παρέχουν χρήσιμες δυνατότητες σε ένα υπό κατασκευή πρόγραμμα. Το Java API είναι οργανωμένο σε βιβλιοθήκες (packages) κώδικα από συστατικά που σχετίζονται μεταξύ τους.

1.2.4 Περιβάλλον και εργαλεία της Java

Υπάρχουν πολλά πακέτα λογισμικού που μπορείτε να χρησιμοποιήσετε για να δημιουργήσετε προγράμματα σε Java. Ένα από τα πιο δημοφιλή πακέτα είναι το Java Development Kit (JDK) της Sun Microsystems. Το JDK χρησιμοποιεί την γραμμή εντολών (DOS prompt) στα συστήματα Windows 95/98/NT/2000. Αναλυτικές πληροφορίες για το JDK υπάρχουν στο site http://java.sun.com. Όπως με τις περισσότερες γλώσσες προγραμματισμού, τα πηγαία αρχεία Java αποθηκεύονται σαν αρχεία καθαρού κειμένου. Μπορείτε να τα δημιουργήσετε με οποιοδήποτε κειμενογράφο ή επεξεργαστή κειμένου που μπορεί να αποθηκεύσει καθαρό κείμενο, μια μορφοποίηση που καλείται επίσης ASCII ή DOS text.

Τα πηγαία αρχεία Java πρέπει να αποθηκευτούν με επέκταση .java. Τα πηγαία αρχεία Java μεταγλωττίζονται σε bytecode με επέκταση ονόματος αρχείου .class. Τα αρχεία αυτά ονομάζονται κλάσεις. Κατά κάποιο τρόπο, ο όρος κλάση (class) στην Java είναι συνώνυμος με τον όρο πρόγραμμα. Τα τρία πιο χρήσιμα εργαλεία της Java είναι ο μεταγλωττιστής javac, ο διερμηνευτής java και ο εξεταστής appletviewer .

Ο μεταγλωττιστής javac, μετατρέπει τον πηγαίο κώδικα Java σε ένα ή περισσότερα αρχεία κλάσεων bytecode που μπορούν να εκτελεστούν από ένα διερμηνευτής java. Για την μεταγλώττιση ενός αρχείου πηγαίου κώδικα Java σε bytecode, το εργαλείο javac εκτελείται με το όνομα του αρχείου σαν όρισμα.

Π.χ.
javac MyProgram.java

Το παραγόμενο αρχείο από την παραπάνω εντολή είναι το MyProgram.class' το οποίο είναι ένα αρχείο κλάσης και είναι σε μορφή bytecode. Ο διερμηνευτής java, χρησιμοποιείται για εκτέλεση εφαρμογών Java από την γραμμή εντολών. Δέχεται σαν όρισμα το όνομα ενός αρχείου κλάσης προς εκτέλεση.

Π.χ.
java MyProgram

Αν και τα αρχεία κλάσης της Java έχουν επέκταση .class η επέκταση αυτή δεν καθορίζεται στην εντολή εκτέλεσής τους. Το αρχείο κλάσης που φορτώνεται από τον διερμηνευτή java πρέπει να περιέχει μια μέθοδο (συνάρτηση) main(...) που έχει την παρακάτω μορφή διαφορετικά θα εμφανισθεί μια ένδειξη λάθους.

Π.χ.

```
public static void main(String[] arguments){
        // Κώδικας μεθόδου
}
```

1.3 Η δομή μιας εικόνας

Μια εικόνα αποτελεί έναν δισδιάστατο πίνακα με διαστάσεις τις διαστάσεις ανάλυσης της εικόνας. Δηλαδή μια εικόνα ανάλυσης 640 πλάτος * 480 ύψος θα έχει έναν δισδιάστατο πίνακα διαστάσεων 640 πλάτος * 480 ύψος. Το κάθε στοιχείο του πίνακα αναπαριστά το κάθε pixel της εικόνας και περιέχει πληροφορίες για αυτό όσον αφορά τις RGB τιμές του pixel αυτού, σε περίπτωση έγχρωμης εικόνας. (R = Red, G = Green, B = Blue). Η επεξεργασία, η ανάλυση, η τροποποίηση αυτού του πίνακα αντιστοιχεί στην επεξεργασία μιας εικόνας. Μας παρέχει όλες τις πληροφορίες που χρειάζονται για να πραγματοποιηθεί η οποιαδήποτε επεξεργασία μιας εικόνας, χρησιμοποιώντας ανά περίπτωση τους κατάλληλους αλγόριθμους.

1.4 Η δομή ενός video

Ένα video δεν είναι τίποτα άλλο από την συλλογή πολλών φωτογραφιών, οι οποίες έχουν συνέχεια μεταξύ τους η μια με την άλλη, και οι οποίες αν προβληθούν με μια ταχύτητα μεγαλύτερης της ταχύτητας του ματιού, δίνουν την εντύπωση της κίνησης. Το ανθρώπινο μάτι, δηλαδή, αν λάβει μια πληροφορία η οποία αποτελείται με πάνω από 15 φωτογραφίες το δευτερόλεπτο, δέχεται την πληροφορία αυτή ως συνεχή κίνηση. Επομένως βάζοντας στη σειρά, φωτογραφίες οι οποίες έχουν συνέχεια μεταξύ τους και φτιάχνοντας ένα πρόγραμμα να αλλάζει την εκάστοτε προβαλλόμενη φωτογραφία με την επόμενή της δημιουργείται μια μορφή video.

Με την πάροδο των χρόνων αυτή η ιδέα έχει εξελιχθεί σε μεγάλο βαθμό. Υπάρχουν πολλές τεχνικές και αλγόριθμοι δημιουργίας, λήψης, επεξεργασίας, κωδικοποίησης και αναπαραγωγής video. Η ιδέα όμως παραμένει ίδια. Η ταχύτητα εναλλαγής των εικόνων είναι διαφορετική, ανάλογα με την κωδικοποίηση

που χρησιμοποιείται, για παράδειγμα με PAL κωδικοποίηση είναι 25 εικόνες το δευτερόλεπτο, η αλλιώς 25 frames το δευτερόλεπτο.

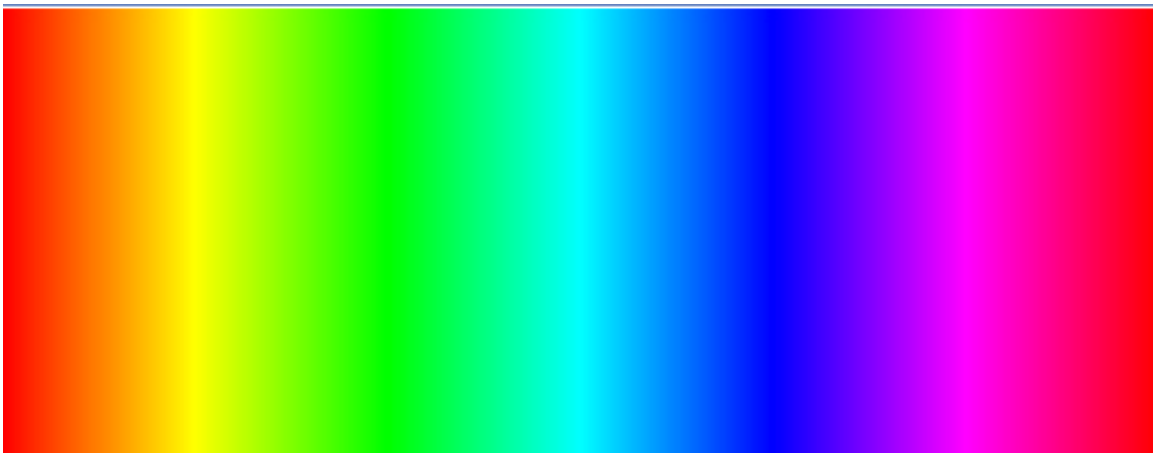
1.5 Το μοντέλο χρωμάτων H.S.I.

Η εφαρμογή βασίζεται στο μοντέλο χρωμάτων του συστήματος **H.S.I.**

Τι σημαίνει όμως **H.S.I.** και πως αναπαρίσταται; Το **H.S.I.** δίνει τη δυνατότητα να ορίζεται η χροιά (H - Hue), το επίπεδο κορεσμού (S - Saturation) και η φωτεινότητα (I – Intensity) ενός χρώματος. Η γραφική αναπαράσταση ολόκληρου του φάσματος του **H.S.I.** φαίνεται στην εικόνα 2.14, με το **Hue** από το ελάχιστο έως το μέγιστο**,** από αριστερά προς τα δεξιά της εικόνας αντίστοιχα, και τα **Saturation** και **Intensity** από το ελάχιστο έως το μέγιστο, από πάνω προς τα κάτω αυτής αντίστοιχα.

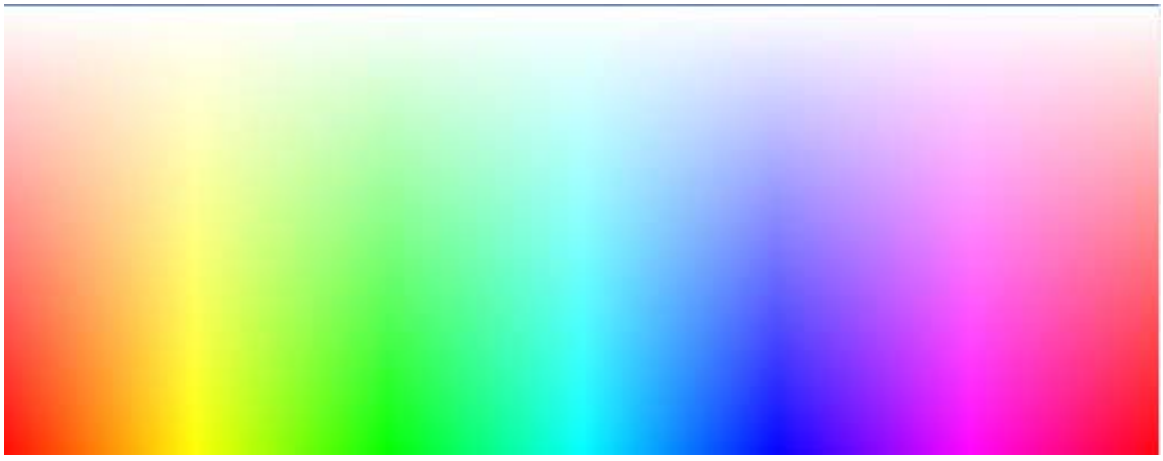
Εικόνα 2.14

Το **Hue** παίρνει τιμές από 0 έως 360 με κάθε τιμή από αυτές να αναπαριστά την τιμή που παίρνει το **Hue** στο H.S.I. μοντέλο χρωμάτων. Η γραφική αναπαράσταση ολόκληρου του φάσματος του **Hue** με **Saturation** και **Intensity** στο μέγιστο φαίνεται στην εικόνα 2.15.



Εικόνα 2.15

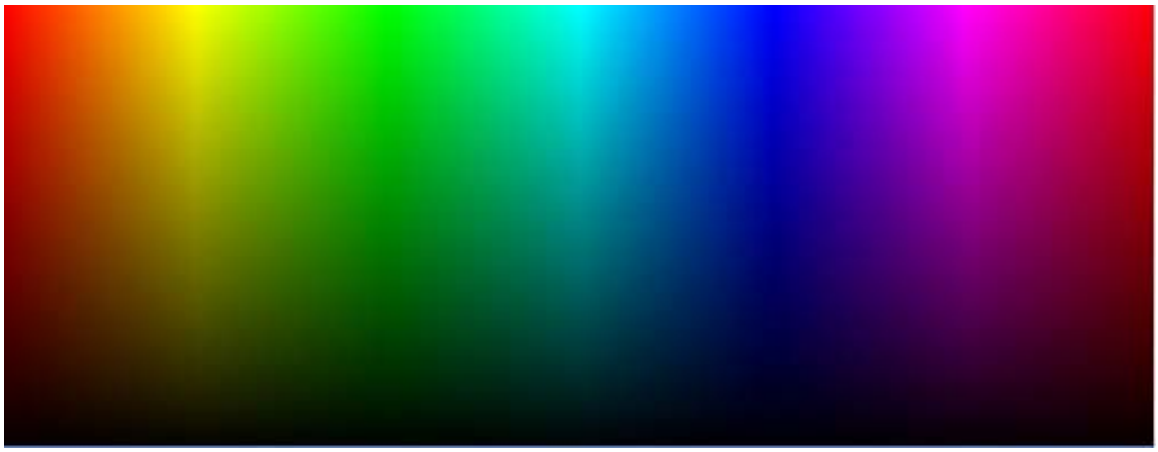
Το **Saturation** παίρνει τιμές από 0.0 έως 1.0 με κάθε τιμή από αυτές να αναπαριστά το επίπεδο κορεσμού του χρωματικού φάσματος. Η γραφική αναπαράσταση του **Saturation** σε όλο το φάσμα του **H.S.I.** φαίνεται στην εικόνα 2.16 με μέγιστο **Intensity** και το **Saturation** από ελάχιστο έως μέγιστο, από πάνω προς τα κάτω της εικόνας αντίστοιχα, και στην εικόνα 2.17 με ελάχιστο **Intensity** και **Saturation** όπως και στην εικόνα 2.16.



Εικόνα 2.16

Εικόνα 2.17

Το **Intensity** παίρνει τιμές από 0.0 έως 1.0 με κάθε τιμή από αυτές να αναπαριστά το επίπεδο φωτεινότητας του χρωματικού φάσματος. Η γραφική αναπαράσταση του **Intensity** σε όλο το φάσμα του **H.S.I.** φαίνεται στην εικόνα 2.18 με μέγιστο **Saturation** και το **Intensity** από ελάχιστο έως μέγιστο, από πάνω προς τα κάτω της εικόνας αντίστοιχα, και στην εικόνα 2.19 με ελάχιστο **Saturation** και **Intensity** όπως και στην εικόνα 2.18.



Εικόνα 2.18

Εικόνα 2.19

Για προγραμματιστικούς λόγους το **Hue** από 0 – 360 έχει μετατραπεί σε 0.0 – 1000 και τα **Saturation**, **Intensity** από 0.0 – 1.0 επίσης σε 0.0 – 1000.

1.6 Ο αλγόριθμος Blob Coloring που χρησιμοποιείται για το Color Tracking

Όταν η εφαρμογή βρίσκεται υπό εκτέλεση, έχουν ενεργοποιηθεί ένα οι περισσότερα χρώματα ελέγχεται αν υπάρχει πληροφορία στο εισερχόμενο video σχετική με τα επιλεγμένα χρώματα και γίνεται ο χωρισμός των περιοχών που δομούν μια ενιαία περιοχή του κάθε χρώματος που ελέγχεται.

Όσο εισέρχεται στην εφαρμογή video από την κάμερα η εφαρμογή το αντιμετωπίζει σαν μια συνεχόμενη ροή από φωτογραφίας παίρνοντας μια προς μια τις φωτογραφίες για έλεγχο.

Αυτό σημαίνει ότι η κάθε φωτογραφία περνάει από έναν αλγόριθμο ο οποίος μέσω τον διαδικασιών που έχουν οριστεί ανιχνεύει την ύπαρξη ή όχι των επιλεγμένων χρωμάτων.

Πώς γίνεται αυτό όμως; Θεωρώντας το μπλε ως επιλεγμένο χρώμα ανίχνευσης, ο αλγόριθμος ξεκινάει, από την επάνω δεξιά γωνία της εικόνας έως την κάτω αριστερή, με οριζόντια σάρωση, παίρνοντας από κάθε pixel την τιμή R.G.B. του και αφού την μετατρέψει στην αντίστοιχη H.S.I. ελέγχει αν η τιμή αυτή

είναι μέσα στα όρια που έχουν οριστεί ότι θα κυμαίνεται το πεδίο H.S.I. για το μπλε χρώμα.

Αν το σημείο που ελέγχεται διαπιστωθεί ότι είναι σημείο ενδιαφέροντος ελέγχεται το ακριβώς αριστερό του αν είναι και αυτό σημείο ενδιαφέροντος, καθώς και ακριβώς από πάνω του. Σε περίπτωση που ούτε το αριστερό του, ούτε το επάνω του είναι σημεία ενδιαφέροντος το σημείο αυτό θεωρείται η αρχή μιας νέας περιοχής και παίρνει την τιμή 0 εάν αυτή είναι η πρώτη περιοχή που ανιχνεύθηκε ή την τιμή 1 αν είναι η δεύτερη και τα λοιπά.

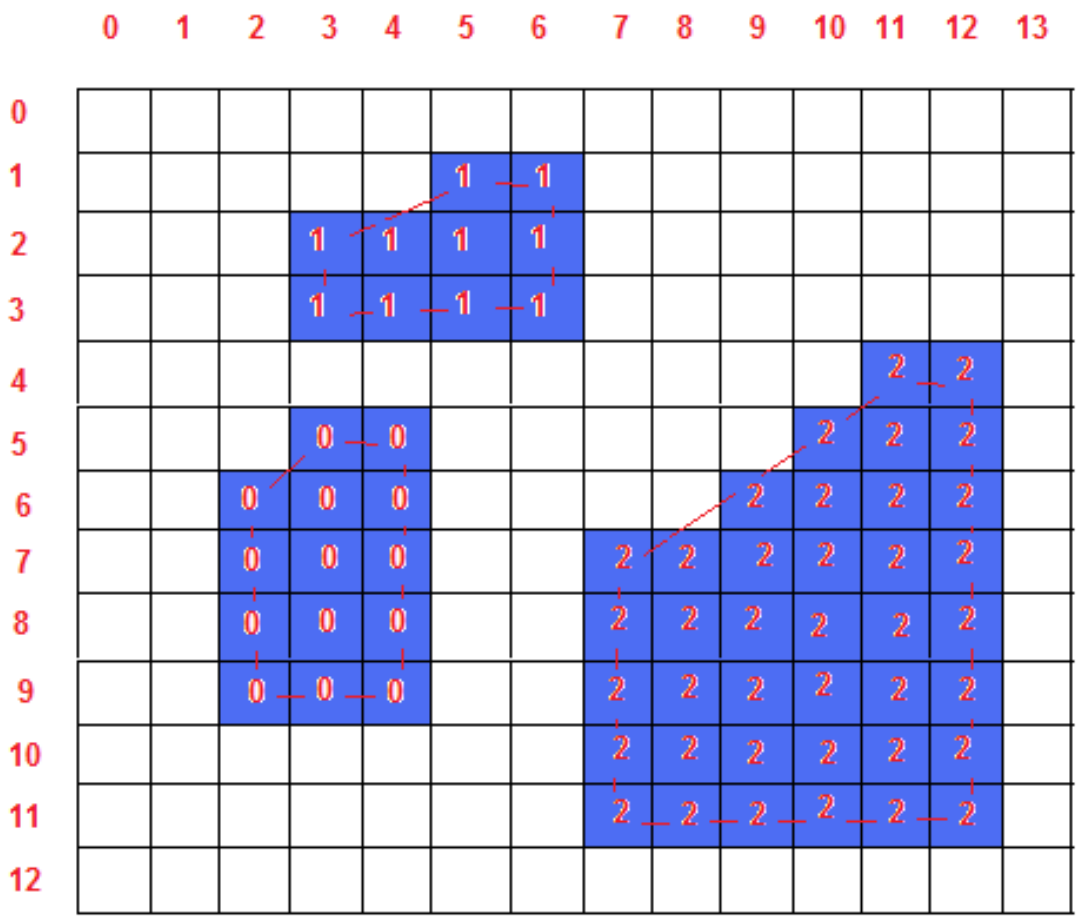
Για παράδειγμα στην εικόνα 1.1 το σημείο (6-3) είναι το πρώτο σημείο που αναγνωρίστηκε ως σημείο ενδιαφέροντος, το αμέσως αριστερό του καθώς και το ακριβώς επάνω του είναι καθαρά, άρα θεωρείται η αρχή μιας περιοχής και παίρνει την τιμή 0. Αντίστοιχα το σημείο (2-3) οριοθετεί την αρχή μιας καινούριας περιοχής. Είναι όμως η δεύτερη που αναγνωρίζεται και παίρνει σαν τιμή το 1. Εξίσου και το σημείο (7-7) που παίρνει την τιμή 2.

Στην περίπτωση που το αριστερό ή το επάνω σημείο ενός pixel είναι σημεία ενδιαφέροντος τότε το νέο αυτό pixel παίρνει την τιμή των γειτονικών του και με αυτών των τρόπο ενσωματώνεται στην ίδια περιοχή. Έτσι στην εικόνα 1.1 το σημείο (7-2) ανήκει στην περιοχή 0 και παίρνει την τιμή αυτή.

Με αυτόν τον τρόπο δημιουργούνται οι περιοχές με κάθε μια από αυτές να αποτελείται από έναν αριθμό pixel τα οποία έχουν την ίδια τιμή, ανά περιοχή.

Αυτό έχει σαν αποτέλεσμα να μπορεί η εφαρμογή να ξεχωρίσει χρωματικές περιοχές ίδιου χρώματος στην εικόνα που επεξεργάζεται και να αναγνωρίσει που αυτές αρχίζουν και που τελειώνουν.

Έτσι, μετά το τέλος αυτής της διαδικασίας θα πρέπει να αναγνωριστούν τα όρια της κάθε περιοχής από αυτές, διαδικασία η οποία πραγματοποιείται με την εφαρμογή ενός Convex Hull αλγόριθμου επάνω σε όλα τα σημεία της κάθε περιοχής ξεχωριστά.



Εικόνα 1.1

1.7 Ο αλγόριθμος Convex Hull

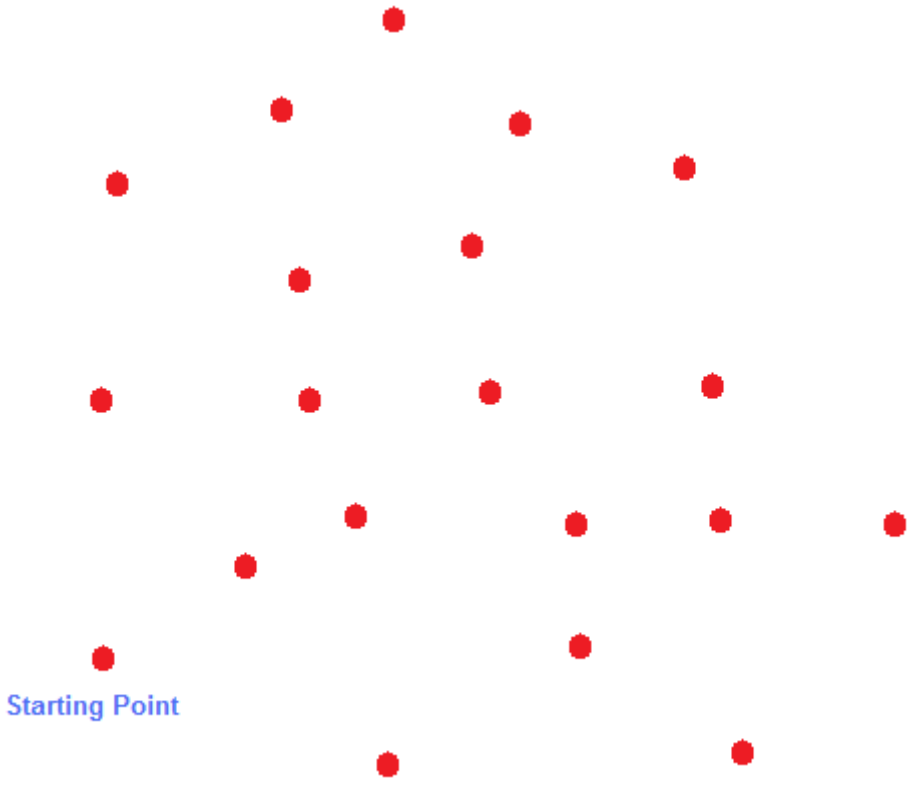
Ο αλγόριθμος Convex Hull έχει ως σκοπό να παίρνει ως είσοδό του δισδιάστατα σημεία στο χώρο (εικόνα 1.2) που εφαρμόζεται και να βρίσκει την περιβάλλουσα των σημείων αυτών φτιάχνοντας ένα πολύγονο αλλά όχι με την ακριβή έννοια του πολυγώνου. Αναλυτικότερα αν στα σημεία της εικόνας 1.2 εφαρμοστεί έναν αλγόριθμος πολυγώνου θα δοθεί το αποτέλεσμα της εικόνας 1.3. Ενώ αν εφαρμοστεί ένας αλγόριθμος Convex Hull το αποτέλεσμα θα είναι αυτό της εικόνας 1.4.

Το πολύγωνο λοιπόν που δημιουργεί ο Convex Hull παίρνει σαν σημεία του τα πλέον εξωτερικά σημεία και ποτέ δεν έχει εσοχές, ενώ όλες οι γωνίες του είναι πάντα αριστερές.

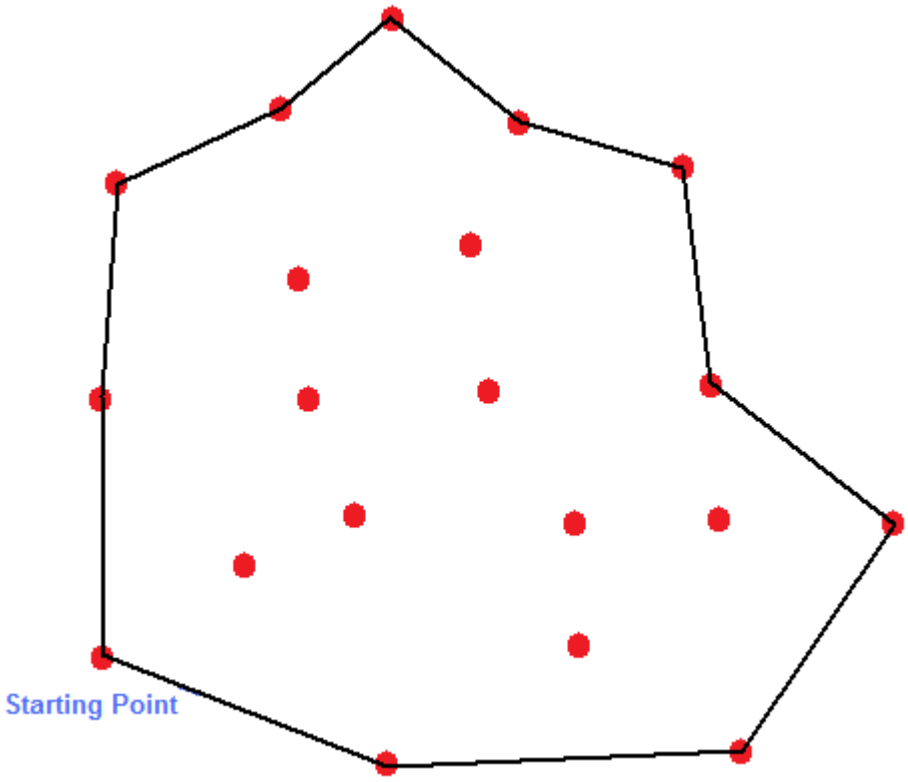
Έχουν υπάρξει διάφορες υλοποιήσεις του αλγόριθμου αυτού με τις ποιο γνωστές την Graham Scan, την Brute Force και Jarvis March όπου η τελευταία είναι και αυτή που εφαρμόστηκε στην υλοποίηση της εφαρμογής.

Η λογική της υλοποίησης του Jarvis March είναι όπως την περιτύλιξη ενός δώρου. Όπως σε ένα αντικείμενο που προορίζεται για δώρο, ο πωλητής εφαρμόζει σε αυτό το χαρτί περιτυλίγματος ξεκινώντας από μια γωνία και περνώντας από κάθε ποιο εξωτερική γωνία καταλήγοντας ξανά στην αρχική έτσι και ο Jarvis March ξεκινάει από μια γωνία και περνώντας από τις ποιο εξωτερικές γωνίες καταλήγει ξανά στην αρχική.

Αναλυτικότερα, σαν πρώτη ενέργεια βρίσκει το Νότιο - δυτικότερο σημείο από τα δοθείσα το οποίο και θα αποτελέσει το σημείο εκκίνησης Στην εικόνα 1.2 το σημείο αυτό είναι αυτό με το όνομα Starting Point.



Εικόνα 1.2

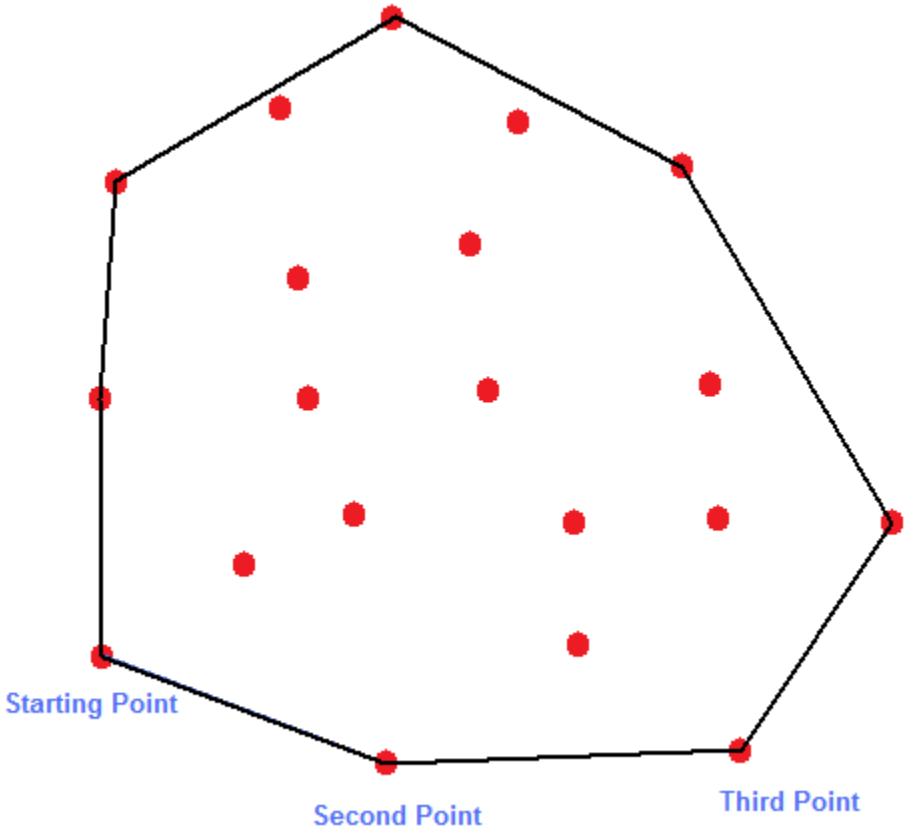


Εικόνα 1.3

Μόλις πραγματοποιηθεί αυτό μπαίνει στη δεύτερη φάση όπου πρέπει να εντοπίσει το επόμενο σημείο που θα αποτελέσει Convex Hull σημείο και μετά από το επόμενο του επόμενου έως ότου βρεθεί ο αλγόριθμος ξανά στο σημείο εκκίνησης που θα σημαίνει και το τέλος της διαδικασίας.

Πως όμως βρίσκεται ποιο σημείο θα αποτελέσει το επόμενο; Η λογική της εύρεσης του επόμενου σημείου είναι, αν γίνει νοητή προέκταση μιας ευθείας από το σημείο που βρίσκεται ο κώδικας και πραγματοποιηθεί κίνηση αυτής από την κίνηση των δεικτών του ρολογιού, το πρώτο σημείο που θα συναντήσει θα είναι και το επόμενο σημείο. Από το νέο αυτό σημείο το ίδιο ξανά για το επόμενο αυτού. Και έτσι ο αλγόριθμος φτάνει ξανά στο αρχικό και μας δίνει το «περιτύλιγμα» των σημείων που του δώσαμε.

Με άλλα λόγια το επόμενο σημείο, είναι το σημείο με την μεγαλύτερη γωνία από το σημείο που βρίσκεται εκείνη τη στιγμή ο αλγόριθμος (εικόνα 1.4).



Εικόνα 1.4

Κεφάλαιο 2. Χρησιμοποιώντας την εφαρμογή

2.1 Εισαγωγή

Στο παρόν κεφάλαιο παρουσιάζεται βήμα προς βήμα όλη η διαδικασία που είναι απαραίτητη για την εγκατάσταση των προγραμμάτων που είναι αναγκαία για την λειτουργία της συσκευής, τον τρόπο εγκατάστασης μιας κάμερας στο JMF καθώς και τον τρόπο λειτουργίας της καθεαυτό εφαρμογής.

2.2 Απαραίτητα βήματα για την εκτέλεση – τροποποίηση της εφαρμογής

2.2.1 Τι χρησιμοποιήθηκε

Η δημιουργία της εφαρμογής πραγματοποιήθηκε στο ***NetBeans 5.5***, με την χρήση του ***JDK 1.6.0*** (Java SE Development Kit 6), του ***JMF 2.1.1e windows i586*** (Java Media Framework), καθώς και με την χρήση μιας web κάμερας της ***Creative την Live! Cam Vista IM.*** Λειτουργικό σύστημα τα ***Windows Vista Ultimate 64Bit***. Καθώς και ένα Laptop της ***HP***, το ***Pavilion dv9393ea*** με ***Core 2 Duo 2GHz***, ***2GB Ram DDR2 800MHz***, και ***VGA GeForce Go 7600 512Ram DDR3***.

2.2.2 Απαραίτητο λογισμικό

Για την εκτέλεση της εφαρμογής είναι απαραίτητο να υπάρχουν εγκατεστημένα το ***JMF 2.1.1*** και το ***JRE 1.6.0*** (Java Runtime Environment) τα οποία συνοδεύονται με την εφαρμογή. Για την εκτέλεση αλλά και τροποποίηση του πηγαίου κώδικα της εφαρμογής είναι απαραίτητο να υπάρχουν εγκατεστημένα το ***JMF 2.1.1***, το ***JDK 1.6.0*** καθώς και κατά προτίμηση το ***NetBeans 5.5***.

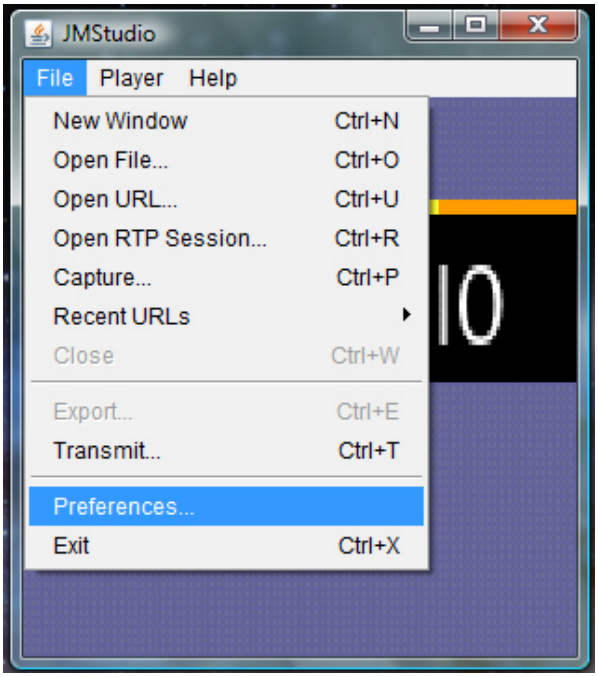
2.2.3 Απαραίτητο υλικό

Απαραίτητη και στις δυο περιπτώσεις είναι η ύπαρξη μιας κάμερας η οποία να είναι εγκατεστημένη στο λειτουργικό σύστημα του εκάστοτε ηλεκτρονικού υπολογιστή όπου θα γίνει εκτέλεση της εφαρμογής, καθώς και να είναι δηλωμένη στο ***JMF***.

2.2.4 Εγκατάσταση συσκευών εισόδου video στο JMF

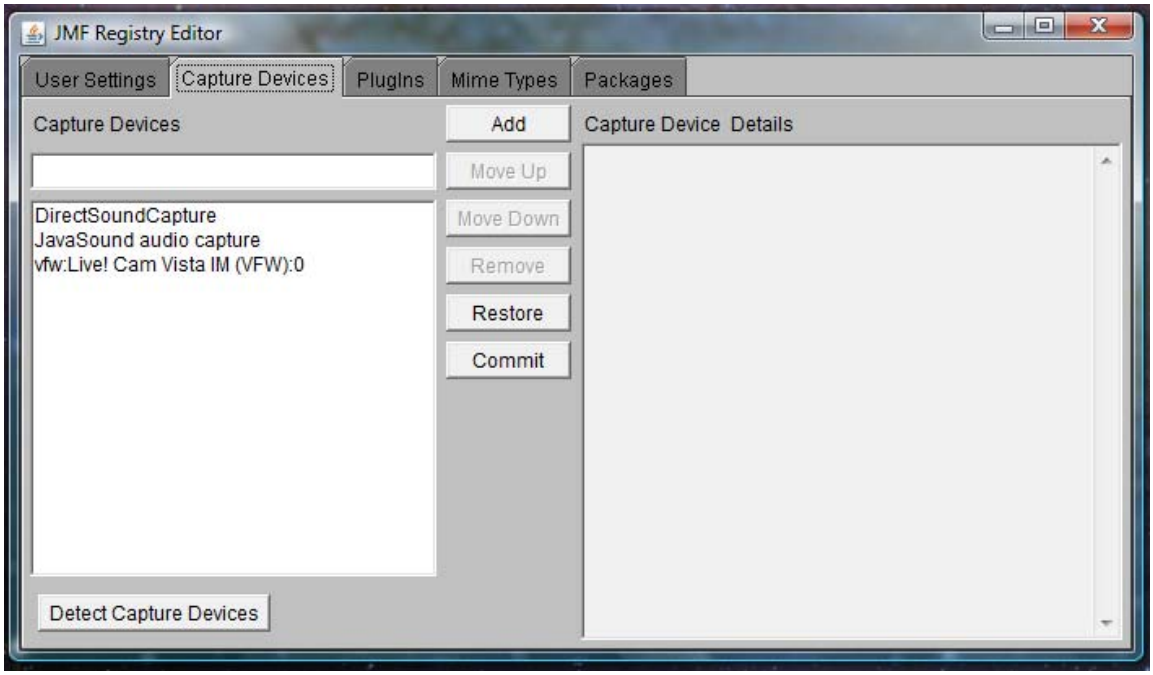
Για να δηλωθεί σωστά μια κάμερα στο ***JMF***, θα πρέπει πρώτα να εγκατασταθούν οι drivers της κάμερας και στη συνέχεια το ***JMF*** ώστε κατά την εγκατάσταση αυτού να ανιχνευθεί η κάμερα. Αν αυτό όμως δεν γίνει με αυτή τη σειρά δεν υπάρχει κανένα πρόβλημα. Υπάρχει η δυνατότητα να δηλώσουμε στο ***JMF***, που ίσως να είναι ήδη εγκατεστημένο στον ηλεκτρονικό υπολογιστή, μια κάμερα, οποιαδήποτε στιγμή.

Αυτό είναι εφικτό, αφού είναι εγκατεστημένοι οι drivers της κάμερας και το ***JMF***, ανοίγοντας το πρόγραμμα ***JMStudio*** του ***JMF*** και επιλέγοντας στο ***File*** την εντολή ***Preferences*** (εικόνα 2.1).



Εικόνα 2.1

Στο νέο παράθυρο που ανοίγει, με τίτλο «***JMF Registry Editor***», στην καρτέλα ***Capture Devices***, πρέπει να πατηθεί το κουμπί ***Detect Capture Devices*** (Εικόνα 2.2).



Εικόνα 2.2

Μετά το τέλος της ανίχνευσης συσκευών λήψης η κάμερα που είναι συνδεμένη στον ηλεκτρονικό υπολογιστή θα πρέπει να εμφανίζεται στη λίστα *Capture Devices* που φαίνεται στην εικόνα 2.2, αλλιώς ή δεν υποστηρίζεται από το ***JMF*** ή κάτι δεν έχει γίνει σωστά.

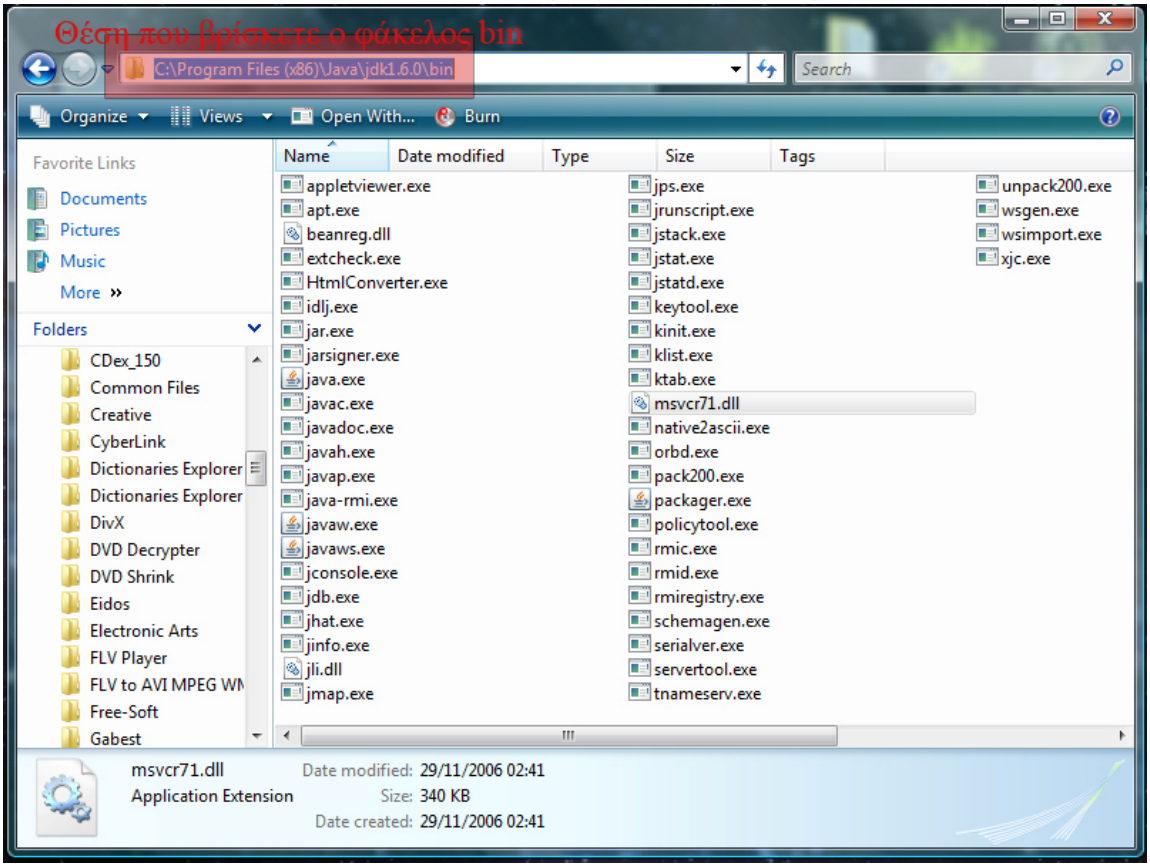
2.2.5 Δήλωση του JDK στις Environment Variables

Για να είναι εφικτό το Compile του κώδικα καθώς και η εκτέλεση της εφαρμογής, μετά τις απαραίτητες εγκαταστάσεις υλικού – λογισμικού, θα πρέπει να δηλωθεί το ***JMF*** στις ***Environment Variables***.

Για να γίνει αυτό σαν πρώτο βήμα θα πρέπει να βρεθεί ο φάκελος εγκατάστασης της java και να βρεθεί σε αυτόν ο φάκελος ***jdk*** ο οποίος περιέχει μαζί με κάποιους άλλους και έναν που λέγεται ***bin***. Για επαλήθευση ότι βρέθηκε ο

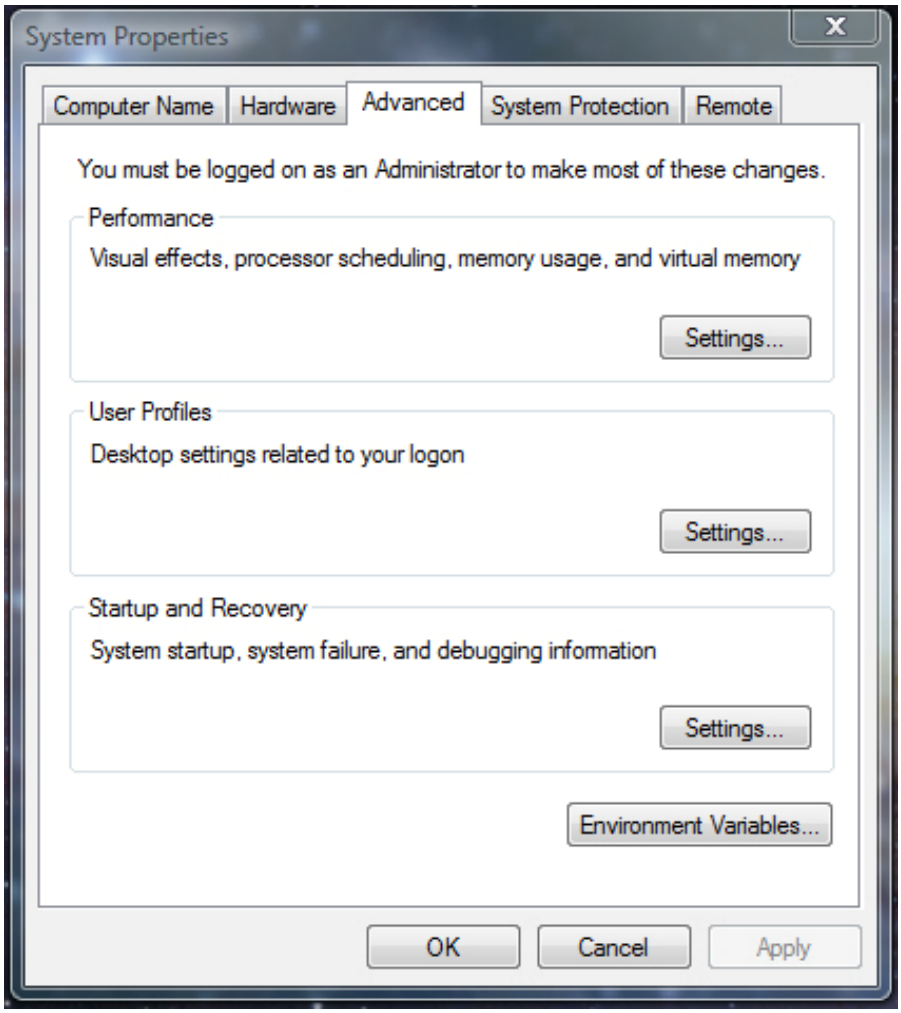
σωστός φάκελος (τον bin), θα πρέπει μέσα σε αυτόν να υπάρχουν αρχεία όπως: java.exe, javac.exe, javadoc.exe και πολλά άλλα.

Στη συνέχεια θα πρέπει να πραγματοποιηθεί αντιγραφή της θέση που βρίσκεται ο φάκελος αυτός (εικόνα 2.3).



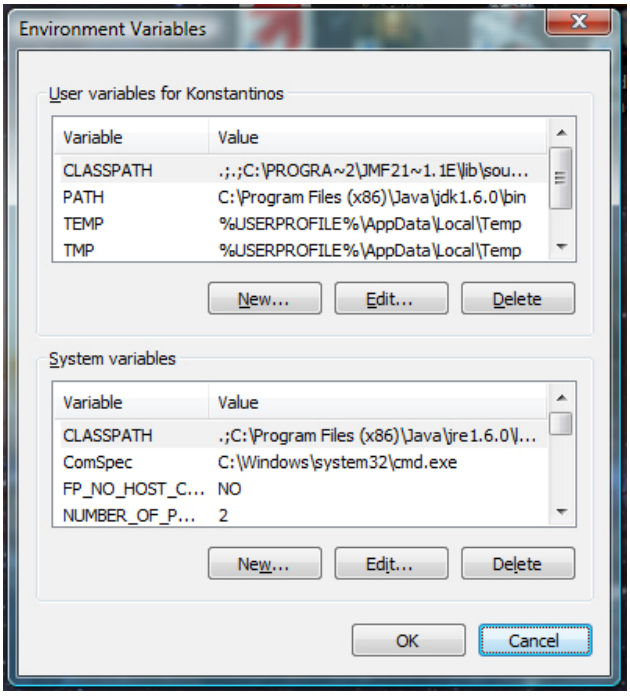
Εικόνα 2.3

Στη συνέχεια δεξί κλικ στο ***My Computer*** και επιλογή του ***properties***. Στο παράθυρο **«*System Properties*»** που ανοίγει, επιλέγεται το ***Advanced*** και εκεί κλικ στο ***Environment Variables*** (εικόνα 2.4).



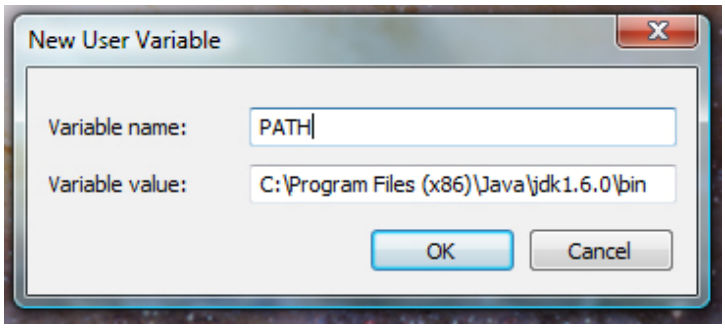
Εικόνα 2.4

Στο νέο παράθυρο που ανοίγει, με τίτλο «***Environment Variables***», κάνουμε κλικ στο κουμπί ***New…*** που αντιστοιχεί στο πεδίο ***User variables for*** **(….the user's name…)** (εικόνα 2.5).



Εικόνα 2.5

Έχουμε άλλο ένα νέο και τελευταίο παράθυρο που ανοίγει με τίτλο «***New User Variable***», στο πεδίο ***Variable name*** πρέπει να πληκτρολογηθεί ***PATH*** και στο πεδίο ***Variable value*** τοποθετείται η θέση του φακέλου ***bin***, π.χ. ***C:\Program Files (x86)\Java\jdk1.6.0\bin***. Θα πρέπει να είναι σχεδόν όπως στην εικόνα 2.6.



Εικόνα 2.6

Κλείνουμε όλα τα παράθυρα με ***OK*** και είμαστε έτοιμοι για την εκτέλεση της εφαρμογής. Όλα αυτά τα βήματα είναι απαραίτητα μια μόνο φορά, κατά την πρώτη εκτέλεση της εφαρμογής σε κάποιον ηλεκτρονικό υπολογιστή.

2.3 Compile και εκτέλεση της εφαρμογής

2.3.1 Το Compile

Η λογική του ***compile*** είναι να μετατρέψει τον υψηλού επιπέδου κώδικα, που καταλαβαίνει ο προγραμματιστής, σε χαμηλού επιπέδου κώδικα ή αλλιώς κώδικα μηχανής, που καταλαβαίνει ο ηλεκτρονικός υπολογιστής. Είναι μια διαδικασία απαραίτητη, πάντα πριν την εκτέλεση της εφαρμογής μετά από αλλαγή στον πηγαίο κώδικα. Για να πραγματοποιηθεί το compile αρκεί ένα διπλό κλικ στο εικονίδιο ***Compile.cmd*** στο φάκελο ***Runnable***. Στο command παράθυρο που παραμένει ανοιχτό παρέχονται πληροφορίες σχετικά με το compile για τυχόν λάθη στον κώδικα καθώς και το που σε αυτόν. Αν όλα εντάξει προχωρούμε στην εκτέλεση της εφαρμογής.

2.3.2 Η εκτέλεση της εφαρμογής

Η εκτέλεση της εφαρμογής γίνεται πολύ απλά με ένα διπλό κλικ στο εικονίδιο ***Run.cmd*** στο φάκελο ***Runnable***, με την προϋπόθεση ότι όλες οι ενέργειες που αναφέρθηκαν παραπάνω, για υλικό και λογισμικό, έχουν καταρχάς πραγματοποιηθεί, και κατά δεύτερον σωστά.

2.4 Το περιβάλλον της εφαρμογής

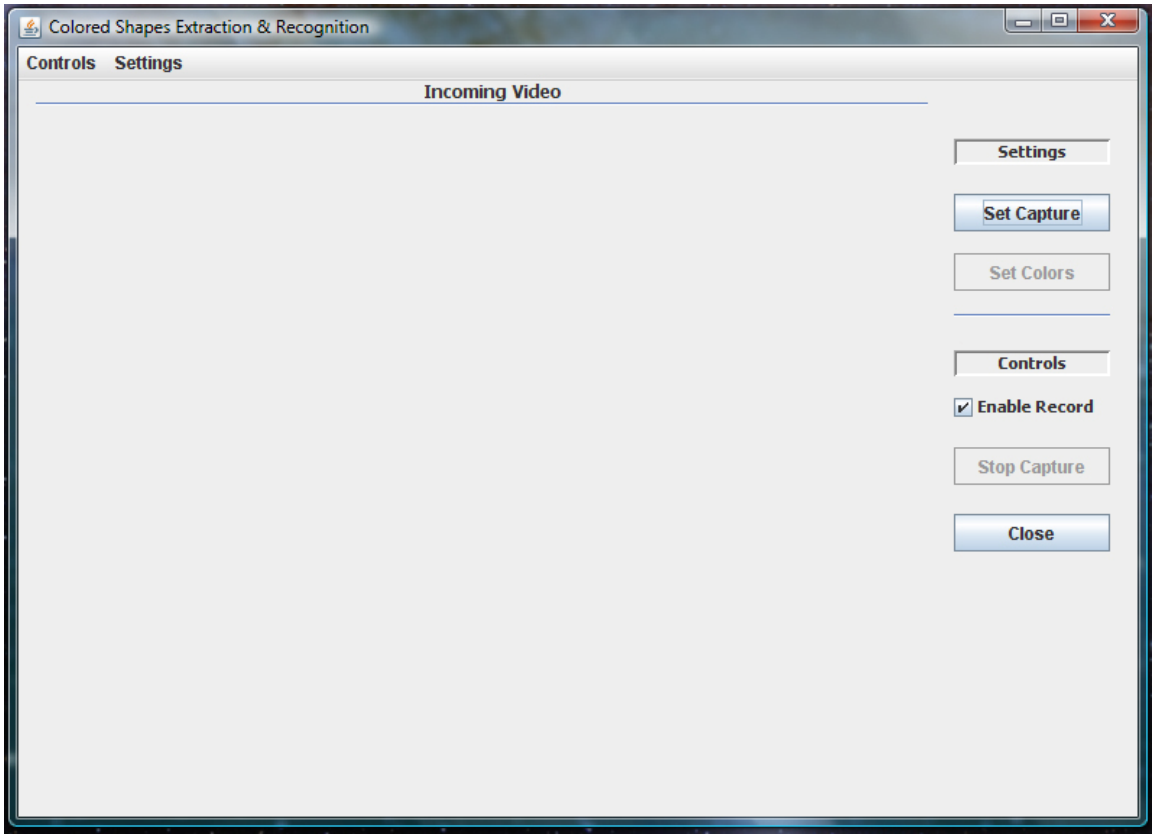
2.4.1 Μια γρήγορη ματιά

Με μια γρήγορη ματιά το περιβάλλον της εφαρμογής είναι ιδιαίτερα λιτό καθώς και απλό στη χρήση του. Αποτελείται από ένα κεντρικό παράθυρο και 2 υπό-παράθυρα του κεντρικού. Στο κεντρικό παράθυρο το οποίο λέγεται «***Colored Shapes Extraction & Recognition***», δίνετε η δυνατότητα πρόσβασης στα 2 υπό-παράθυρα «***Capture Device Selection***» και «***Color Selection & Modification***». Στο παράθυρο «***Capture Device Selection***» δίνεται η δυνατότητα να γίνει επιλογή κάμερας λήψης καθώς και του ***format*** κωδικοποίησης που θα χρησιμοποιηθεί. Στο παράθυρο «***Color Selection & Modification***» δίνεται η δυνατότητα επιλογής των χρωμάτων που θα

ανιχνεύονται καθώς και η αλλαγή του φάσματος που ορίζει το κάθε χρώμα. Στο κεντρικό παράθυρο επίσης εμφανίζεται η εικόνα από την κάμερα μετά την επεξεργασία. Όλα αυτά θα αναλυθούν με λεπτομέρεια στη συνέχεια.

2.4.2 Η χρήση της εφαρμογής

Μόλις πραγματοποιηθεί εκτέλεση της εφαρμογή το πρώτο που εμφανίζεται είναι το κεντρικό παράθυρο «***Colored Shapes Extraction & Recognition***» (εικόνα 2.7).



Εικόνα 2.7

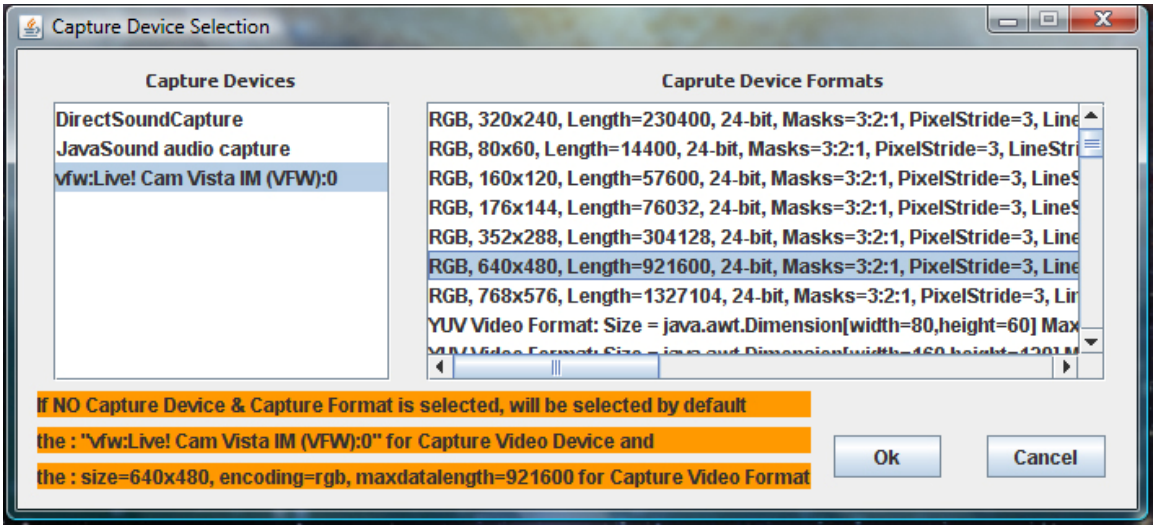
Όπως σε όλες σχεδόν τις εφαρμογές, έτσι κι εδώ, με μια πρώτη ματιά παρατηρούνται παρόμοια στοιχεία όπως κουμπιά, μενού και επιλογείς. Με λίγη παραπάνω παρατήρηση διαπιστώνεται ότι κάποια από αυτά τα στοιχεία είναι ενεργά, ενώ κάποια άλλα όχι. Αυτό σημαίνει ότι για να ενεργοποιηθεί μια εντολή θα πρέπει να ενεργοποιηθεί ή ρυθμιστεί κάποια άλλη πρώτα, όπως επίσης σε

κάποιες περιπτώσεις αφού οριστεί μια ρύθμιση, δεν είναι δυνατόν να οριστεί ξανά εάν δεν προηγηθεί μία άλλη ενέργεια πρώτα. Όλα αυτά όμως στη συνέχεια, αναλυτικότερα.

Στο κεντρικό αυτό παράθυρο λοιπόν, υπάρχουν τα κουμπιά ***Set Capture***, ***Set Colors*** στο πεδίο των ρυθμίσεων, τα κουμπιά ***Stop Capture*** και ***Close***, καθώς και ο επιλογέας ***Enable Record*** στην κατηγορία του ελέγχου, υπάρχει Μ***enu Bar*** με τις δυο κατηγορίες, των ρυθμίσεων και του ελέγχου. Τέλος υπάρχει ένα πεδίο με τίτλο ***Incoming Video*** στο οποίο προβάλεται το εισερχόμενο video.

2.4.2.1 Set Capture

Σαν πρώτη ενέργεια θα πρέπει να γίνει επιλογή κάμερας λήψης. Για να γίνει αυτό θα πρέπει να γίνει κλικ στο κουμπί ***Set Capture*** ή στο ***menu Settings*** και στη συνέχεια επιλογή του ***Set Capture***. Ένα νέο παράθυρο εμφανίζεται, με τίτλο «***Capture Device Selection***» (εικόνα 2.8).



Εικόνα 2.8

Στο παράθυρο αυτό, στο πεδίο ***Capture Devices*** εμφανίζονται όλες οι συσκευές λήψης που υπάρχουν στον ηλεκτρονικό υπολογιστή και που υποστηρίζονται από το ***JMF***. Αν μία από αυτές επιλεγεί και είναι συσκευή εισόδου video, εμφανίζονται στο πεδίο ***Capture Device Formats*** όλα ***format*** που αυτή υποστηρίζει. Αφού γίνει και επιλογή του επιθυμητού format, οι ρυθμίσεις ενεργοποιούνται μετά το πάτημα του κουμπιού ***OK***.

Σε περίπτωση που δεν οριστεί κάποια συσκευή εισόδου video ή κάποιο format η εφαρμογή θα ορίσει αυτόματα την ***Creative Live! Cam Vista IM*** ως συσκευή εισόδου video και format το ***RGB, 640x480, Length=921600, 24-bit, Mask=3:2:1, PixelStride=3, LineStride=1920, Flipped***.

Εφόσον οριστούν, συσκευή εισόδου video καθώς και format κωδικοποίησης, το κουμπί ***Set Capture*** στο παράθυρο **«*Colored Shapes Extraction & Recognition*»** γίνεται ανενεργό, το κουμπί ***Set Colors*** γίνεται ενεργό ώστε να είναι δυνατό να γίνει επιλογή του ή των χρωμάτων που θα ανιχνεύει η εφαρμογή, το κουμπί ***Stop Capture*** γίνεται ενεργό, οι αντίστοιχες εντολές στο ***Menu Bar,*** των αντίστοιχων κουμπιών, ενεργοποιούνται η απενεργοποιούνται ανάλογα. Πραγματοποιείται η σύνδεση με την συσκευή εισόδου video (camera) και εμφανίζεται το εισερχόμενο video στο πεδίο ***Incoming video*** (αναλυτικότερα στο 2.4.2.5).

2.4.2.2 Set Colors

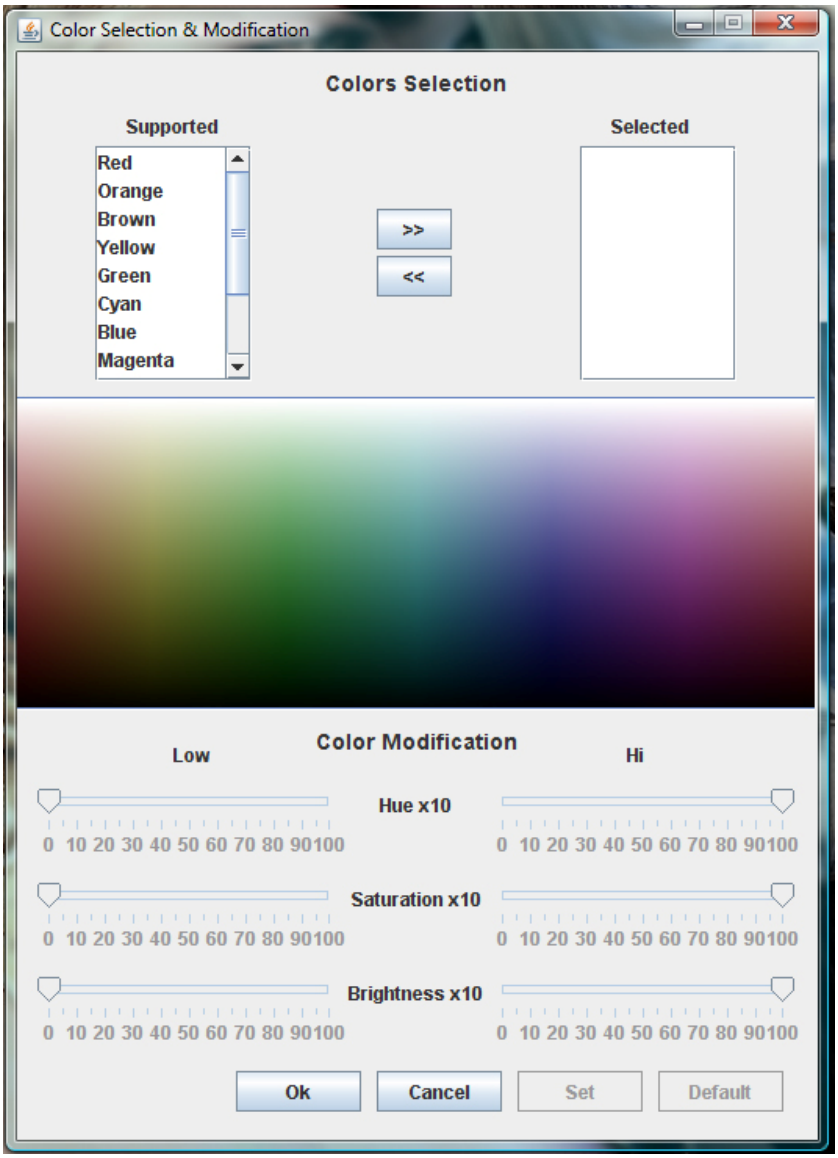
Δεύτερη βασική ενέργεια είναι η επιλογή και η πιθανή ρύθμιση του φάσματος των χρωμάτων που θα είναι ενεργά για ανίχνευση πάνω στο εισερχόμενο video.

Αυτό είναι εφικτό, αφού οριστεί συσκευή λήψης, γίνει επιτυχής σύνδεση με αυτή και ενεργοποιηθεί το κουμπί ***Set Colors***. Τότε όταν αυτό πατηθεί θα ανοίξει το παράθυρο **«*Color Selection & Modification*»**, (εικόνα 2.9) μέσω του οποίου δίνεται η δυνατότητα ρύθμισης του ποια θα είναι τα ενεργά χρώματα προς ανίχνευση, καθώς και ρύθμισης του πεδίου του φάσματος του κάθε χρώματος.

Όπως παρατηρείτε το παράθυρο αυτό, λειτουργικά, χωρίζεται σε τρία μέρη. Το πρώτο είναι το πεδίο **Colors Selection** όπου είναι εφικτή η ενεργοποίηση ή απενεργοποίηση των χρωμάτων που θα ανιχνεύονται ώστε να βρεθούν με βάση αυτά αντικείμενα στόχος για το κάθε ένα από αυτά. Στο δεύτερο πεδίο γίνεται η γραφική αναπαράσταση του **H.S.I.** και χρησιμεύει ώστε να δίνεται μια εικονική αναπαράσταση του φάσματος ενός χρώματος ώστε να βοηθάει τον χειριστή να καταλάβει που κυμαίνεται το επιλεγμένο χρώμα αλλά και να κάνει ρυθμίσεις πάνω σε αυτό. Και το τρίτο πεδίο είναι το **Color Modification** το οποίο αποτελείται από έξι μπάρες ρύθμισης που χρησιμοποιούνται για να ορίσει ο χειριστής το πεδίο τιμών του επιλεγμένου χρώματος.

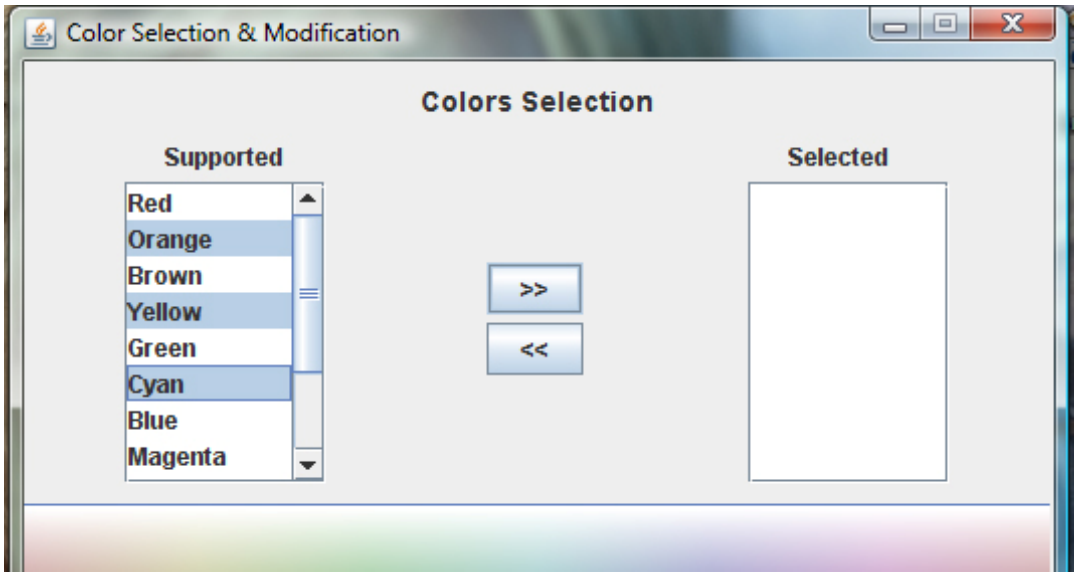
Για να ενεργοποιήσουμε ή απενεργοποιήσουμε τα χρώματα που θα ανιχνεύονται υπάρχουν στο πεδίο **Color Selection**, δύο λίστες καθώς και δύο κουμπιά (εικόνα 2.9). Η πρώτη λίστα με όνομα **Supported** περιέχει όλα τα

χρώματα που η εφαρμογή υποστηρίζει και μπορεί να πραγματοποιήσει ανίχνευση για αυτά. Η δεύτερη λίστα με όνομα **Selected** όταν ξεκινάει η εφαρμογή δεν περιέχει απολύτως τίποτα. Σε αυτήν τη λίστα μεταφέρονται – εμφανίζονται τα χρώματα που επιλέγονται να είναι ενεργά.

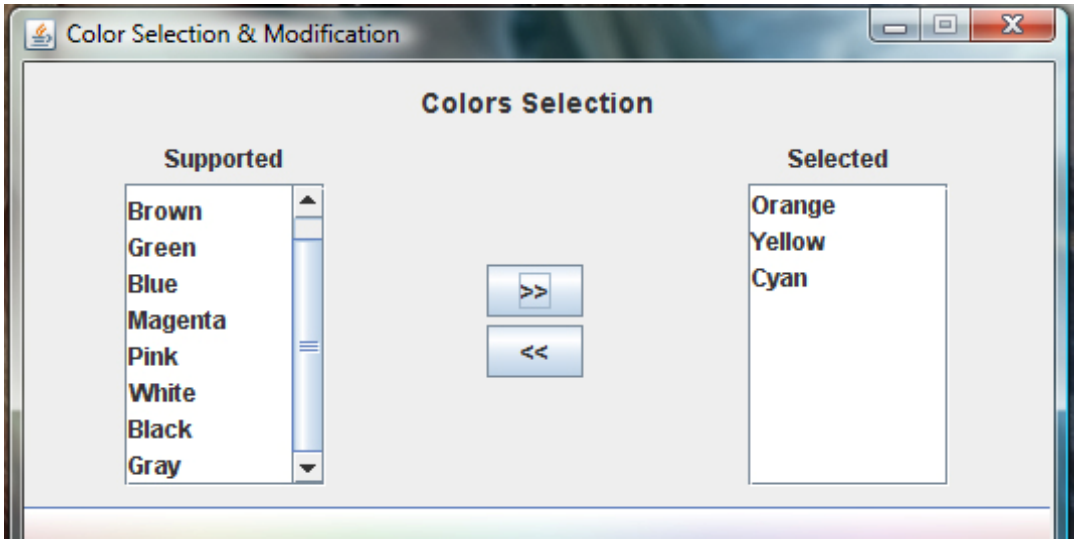


Εικόνα 2.9

Για να γίνουν ενεργά ένα ή περισσότερα χρώματα, θα πρέπει να επιλεχθούν από τη λίστα **Supported** (εικόνα 2.10) και να προστεθεί στη λίστα **Selected** πατώντας το κουμπί « **>>** » (εικόνα 2.11).

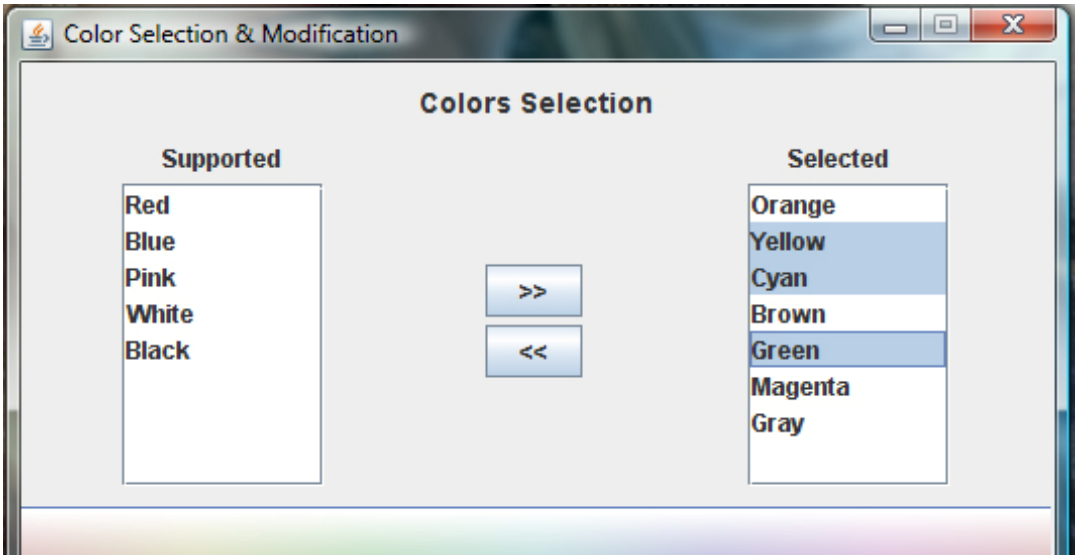


Εικόνα 2.10

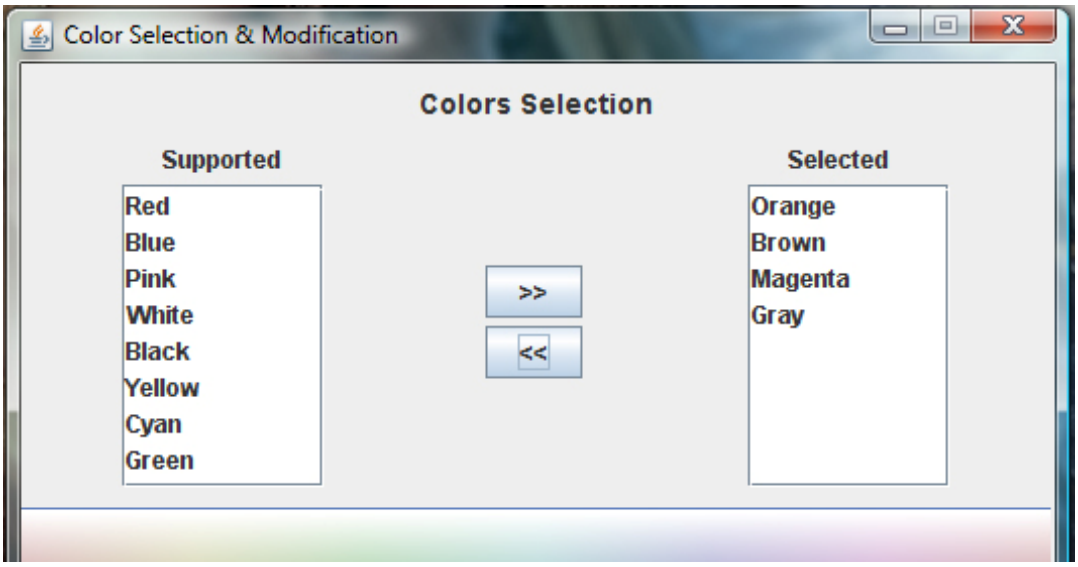


Εικόνα 2.11

Αντίστοιχα για να απενεργοποιηθούν ένα ή περισσότερα χρώματα θα πρέπει να επιλεχθούν από την λίστα **Selected** (εικόνα 2.12) και να πατηθεί το κουμπί « **<<** » (εικόνα 2.13). Τα χρώματα που απενεργοποιούνται «επιστρέφουν» στη λίστα **Supported** έτσι ώστε να είναι δυνατή μια πιθανή μελλοντική απαίτηση ενεργοποίησής τους.



Εικόνα 2.12



Εικόνα 2.13

2.4.2.3 Ρύθμιση του φάσματος ενός χρώματος

Μετά την σύντομη εξήγηση του μοντέλου χρωμάτων **H.S.I.** το επόμενο βήμα είναι η χειροκίνητη ρύθμιση του φάσματος του κάθε χρώματος.

Τι εννοούμε όμως «ρύθμιση του φάσματος»:

Όπως παρατηρείται στο παράθυρο «**Colors Selection & Modification**» (εικόνα 2.9), στην λίστα που περιέχει τα χρώματα που η εφαρμογή υποστηρίζει, παρέχονται 12 χρώματα. Τα **Red**, **Orange**, **Brown**, **Yellow**, **Green**, **Cyan**, **Blue**, **Magenta**, **Pink**, **White**, **Black** και **Gray**. Έστω ότι επιλέγεται το κόκκινο. Παρατηρώντας ένα αντικείμενο στο χώρο με βασικό χρώμα του, το κόκκινο, διαπιστώνεται ότι το χρώμα στο αντικείμενο δεν είναι παντού το ίδιο, αλλά ανάλογα με το φωτισμό, την γωνία παρατήρησης και το υλικό κατασκευής του αντικειμένου, αυτό αλλάζει σε διάφορες αποχρώσεις του κόκκινου από πολύ έντονο κόκκινο έως ίσως σε πολύ ανοιχτό κόκκινο. Αυτό είναι που αναφέρεται ως φάσμα ή αλλιώς πεδίο τιμών για κάθε χρώμα.

Σε τι όμως είναι απαραίτητο;

Αν για παράδειγμα στην εφαρμογή έχει δηλωθεί σαν κόκκινο ένας και μοναδικός τόνος χρώματος κόκκινου, η εφαρμογή θα αναγνωρίζει μόνο το σημείο πάνω στο αντικείμενο που έχει τον τόνο αυτό και όχι όλο το αντικείμενο ή ίσως και καθόλου, με αποτέλεσμα σημαντική πληροφορία να χάνεται. Για να μην συμβαίνει αυτό θα πρέπει για κάθε χρώμα που υποστηρίζει η εφαρμογή να ορίζεται ένα φάσμα. Από κατασκευής έχει οριστεί στο κάθε χρώμα το πεδίο τιμών αυτού, αλλά δίνεται η δυνατότητα στον χρήστη να αλλάξει αυτά τα πεδία τιμών οποιαδήποτε στιγμή.

Οι αλλαγές στο φάσμα ενός χρώματος γίνονται από το πεδίο **Color Modification** του παράθυρου «**Colors Selection & Modification**» αφού έχει προστεθεί στην λίστα **Selected** το χρώμα αυτό. Στη συνέχεια αφού επιλεγεί το επιθυμητό προς ρύθμιση, χρώμα, εμφανίζεται στο πεδίο αναπαράστασης φάσματος, το φάσμα του χρώματος αυτού και οι έξι μπάρες ρύθμισης παίρνουν την τιμή που αντιστοιχεί στην κάθε μια από αυτές (εικόνα 2.20).

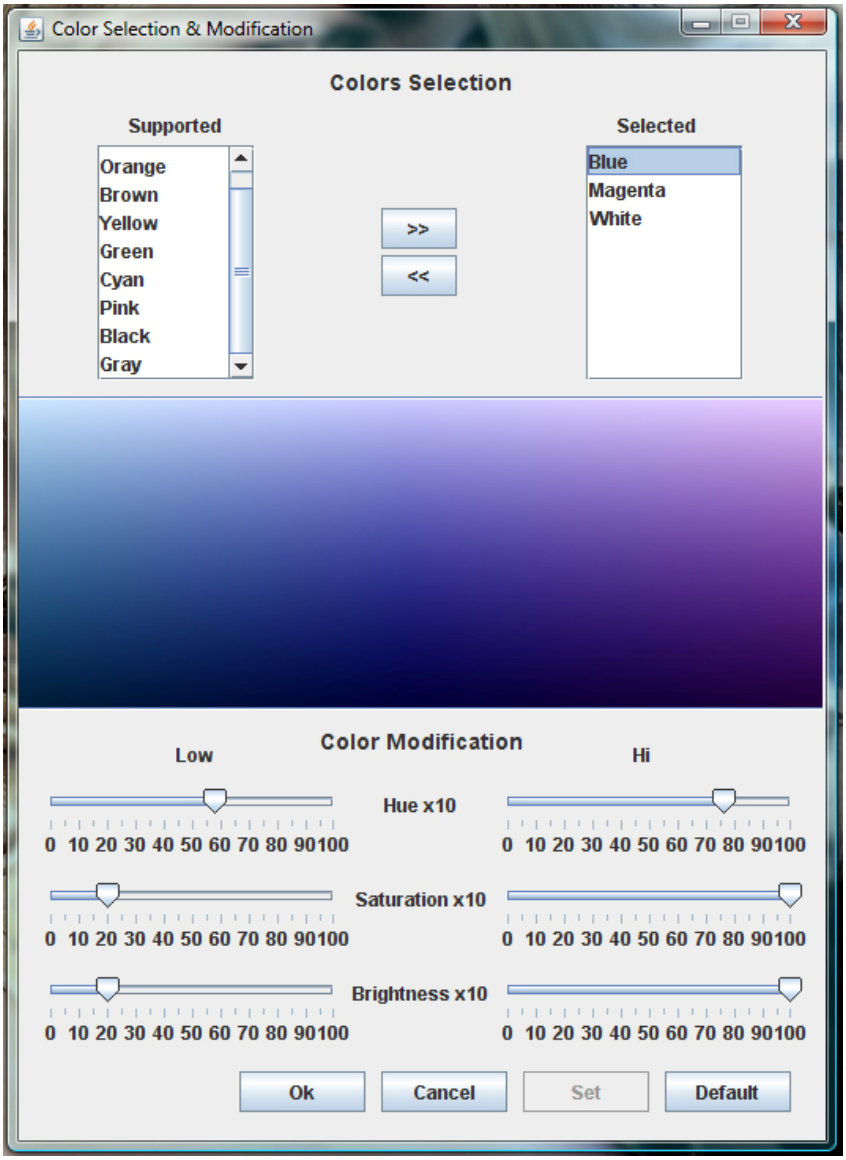
Οι δύο πρώτες μπάρες αφορούν το **Hue**, οι δύο δεύτερες το **Saturation** και οι δυο τελευταίες το **Intensity**. Και στις τρεις περιπτώσεις οι αριστερές αναπαριστούν το «Από» και οι τρεις δεξιές το «Έως».

Αναλυτικότερα, αν για παράδειγμα είναι επιθυμητό να γίνει αλλαγή στο φάσμα του χρώματος μπλε (εικόνα 2.20) του οποίου το φάσμα τιμών έχει οριστεί από κατασκευής για **Hue** 580>Hue<760 για **Saturation** 200>Saturation<1000 και για **Intensity** 200>intensity<1000 επιλέγοντάς το παρατηρείται ότι οι μπάρες αυτόματα τοποθετούνται στα νούμερα αυτά, διαιρεμένα δια δέκα. Δηλαδή η αριστερή μπάρα του Hue θα πάει στην τιμή 58 και η δεξιά στην τιμή 76. Αντίστοιχα και για τις υπόλοιπες. Αν τώρα γίνει οποιαδήποτε αλλαγή θέσης σε αυτές ταυτόχρονα γίνεται αναπαράσταση στο πεδίο φάσματος ώστε να

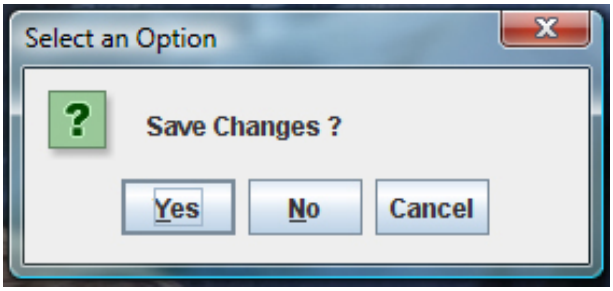
πληροφορείται παραστατικά ο χειριστής και για να τον βοηθάει στην σωστή και ποιο εύκολη αλλαγή των ρυθμίσεων.

Σε περίπτωση που γίνουν αλλαγές αποθηκεύονται με το πάτημα στο κουμπί **Set**. Σε περίπτωση που είναι επιθυμητή η επαναφορά των αρχικών ρυθμίσεων γίνεται με το πάτημα στο κουμπί «**Default**». Αν καμία αλλαγή δεν έχει γίνει το κουμπί **Set** είναι ανενεργό και ενεργοποιείται με την πρώτη αλλαγή. Απενεργοποιείται αν πατηθεί ή αν πατηθεί το «**Default**».

Σε περίπτωση αλλαγών και μετάβασης σε άλλη ενέργεια όπως επιλογή άλλου χρώματος, χωρίς να έχει γίνει **Set** ή **Default**, πριν αυτή πραγματοποιηθεί παρουσιάζεται στο χειριστή ένα μήνυμα για αποθήκευση των αλλαγών (εικόνα 2.21) με **Yes** να αποσκοπεί στην αποθήκευση των αλλαγών και πραγματοποίηση της επόμενης ενέργειας, με **No** την απόρριψη των αλλαγών και πραγματοποίηση της επόμενης ενέργειας και με **Cancel** ακύρωση της επόμενης ενέργειας και επιστροφή στην κατάσταση πριν την επιλογή μετάβασης σε άλλη ενέργεια.



Εικόνα 2.20



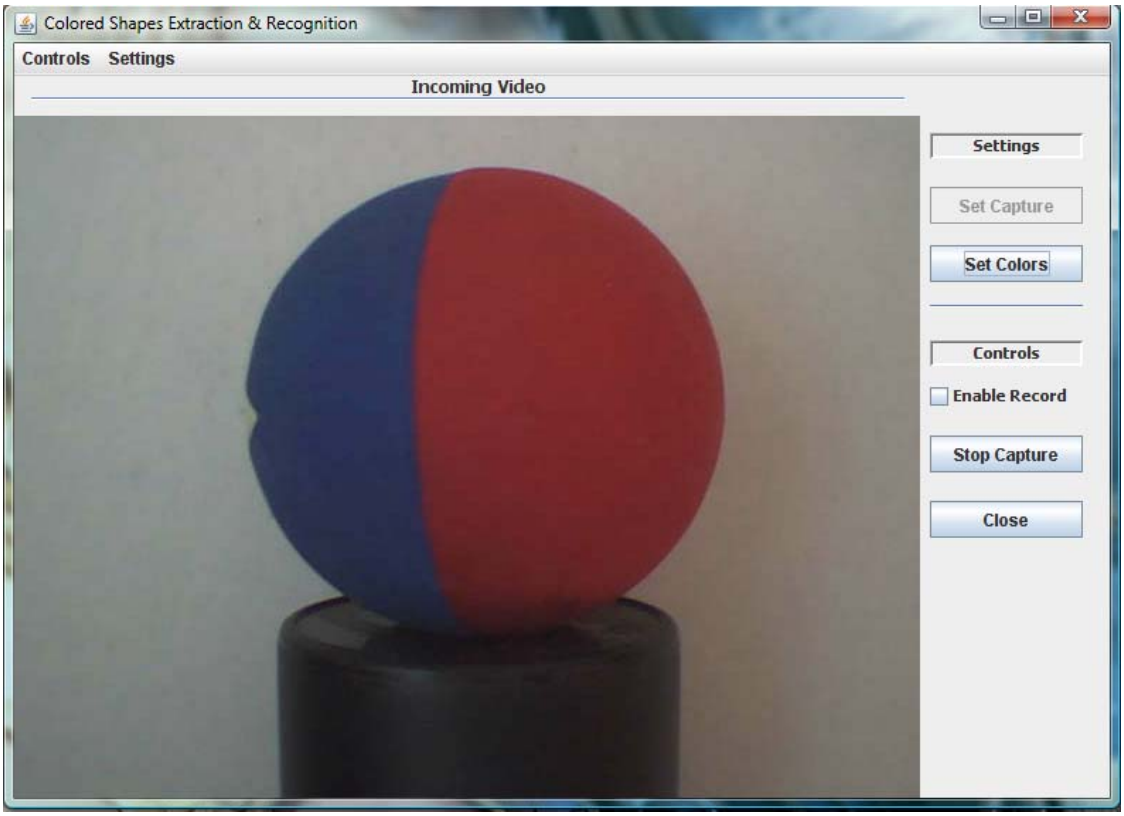
Εικόνα 2.21

Εφόσον πραγματοποιηθούν όλες οι επιθυμητές αλλαγές στο περιβάλλον «**Colors Selection & Modification**» τα επιλεγμένα προς ανίχνευση χρώματα ενεργοποιούνται με το πάτημα του κουμπιού **OK** που σημαίνει και το κλείσιμο του παράθυρου αυτού. Εάν ο χειριστής δεν επιθυμεί να ενεργοποιηθούν οι αλλαγές προσθήκης ή αφαίρεσης χρωμάτων αρκεί να πατήσει το κουμπί **Cancel.**

2.4.2.4 Incoming Video

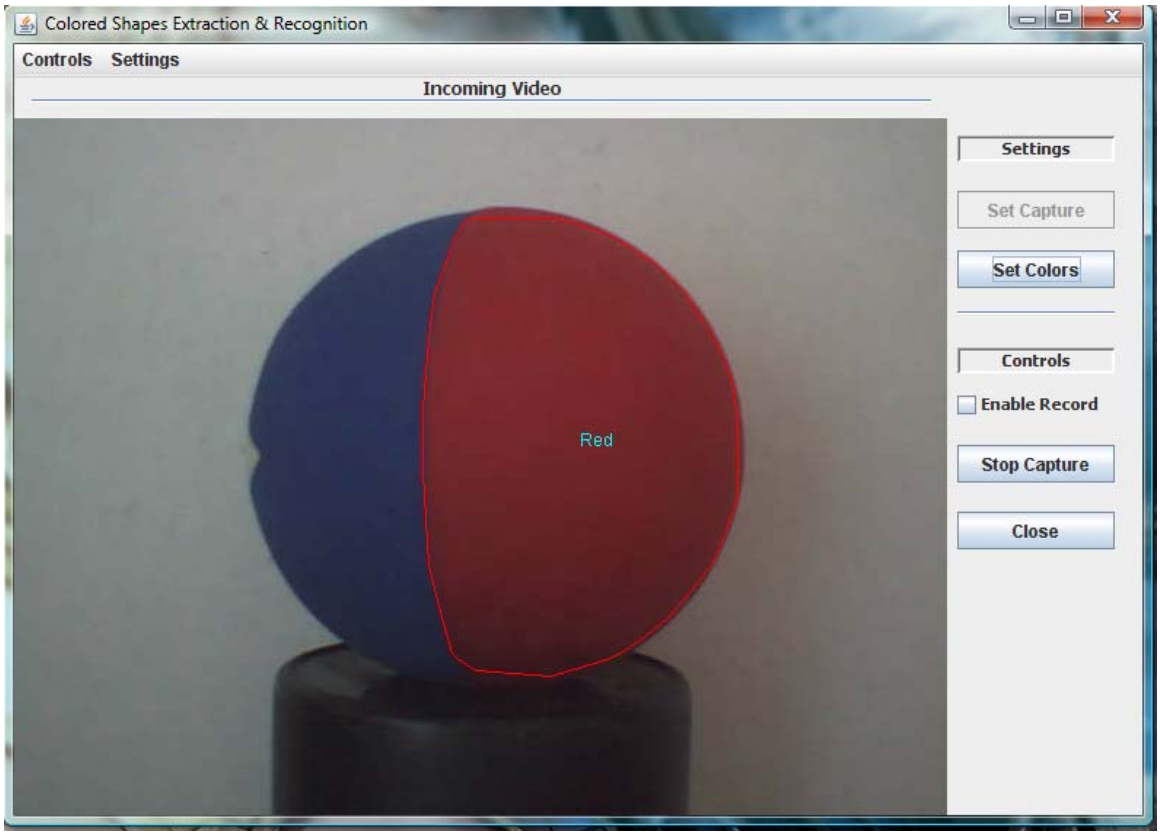
Το πεδίο με τίτλο ***Incoming Video*** προβάλλεται το εισερχόμενο video όπως έρχεται από την κάμερα εάν κανένα χρώμα δεν έχει οριστεί για ανίχνευση. Όταν οριστεί έστω ένα χρώμα και υπάρχει πληροφορία για αυτό στο πεδίο λήψης, τότε στο πεδίο ***Incoming Video*** προβάλλεται το εισερχόμενο video με μαρκαρισμένες τις περιοχές όπου το ορισμένο ή ορισμένα χρώματα ανιχνεύθηκαν.

Αναλυτικότερα, αν θεωρηθεί ότι η εφαρμογή βρίσκεται σε εκτέλεση, ότι η κάμερα είναι τοποθετημένη να παρατηρεί ένα αντικείμενο, έχει γίνει σύνδεση με αυτή, αλλά κανένα χρώμα δεν έχει οριστεί προς ανίχνευση τότε στο πλαίσιο ***Incoming Video*** θα εμφανίζεται καθαρή η εικόνα που λαμβάνει από την κάμερα (εικόνα 2.22).



Εικόνα 2.22

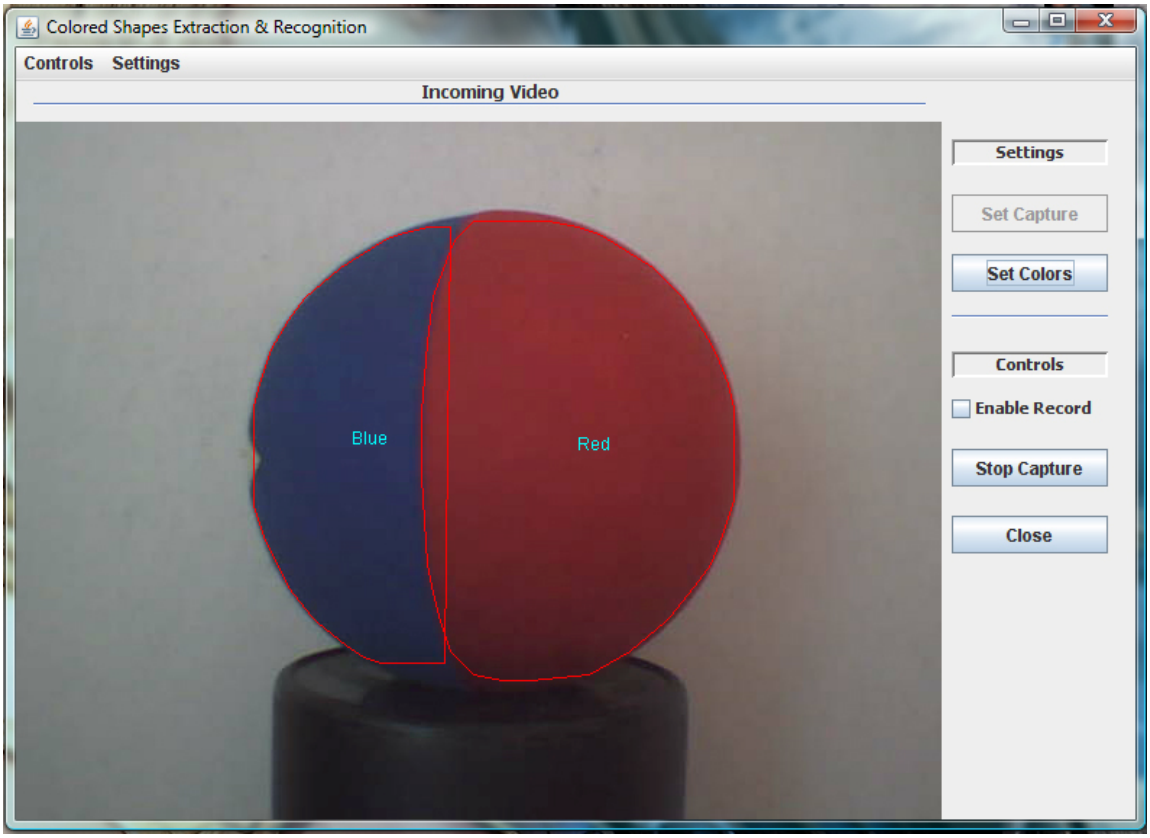
Υποθέτοντας τώρα ότι έχει ενεργοποιηθεί προς ανίχνευση το κόκκινο χρώμα. Θα παρατηρηθεί ένα πλαίσιο να κυκλώνει την περιοχή του αντικειμένου που αποτελείται από το ενεργοποιημένο χρώμα, καθώς και ένα μήνυμα, στο κέντρο του πλαισίου, με το όνομα του χρώματος αυτού για επαλήθευση (εικόνα 2.23).



Εικόνα 2.23

Αν στη συνέχεια γίνει ενεργοποίηση και του μπλε χρώματος το αποτέλεσμα θα είναι αυτό της εικόνας (2.24).

Παρατηρείται ότι ενώ είναι ένα αντικείμενο η εφαρμογή το αναγνωρίζει σαν δύο διαφορετικά. Αυτό γίνεται διότι ο αλγόριθμος που έχει εφαρμοστεί αναγνωρίζει σαν αντικείμενο – στόχος ενιαίες χρωματικές περιοχές του χρώματος προς ανίχνευση. Για παράδειγμα ένα κόκκινο αντικείμενο είναι ένα αντικείμενο – στόχος. Δύο κόκκινα αντικείμενα είναι δύο αντικείμενα – στόχος. Ένα αντικείμενο με τη μια περιοχή ενιαίο μπλε και την άλλη ενιαίο κόκκινο είναι δύο αντικείμενα στόχος.

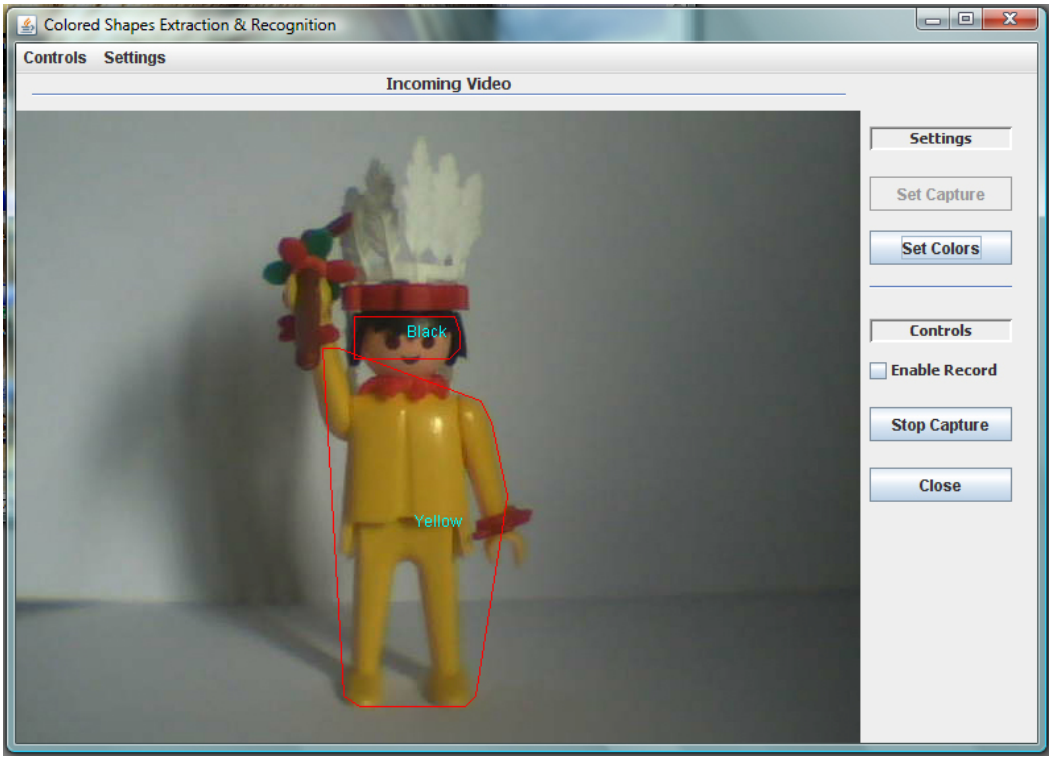


Εικόνα 2.24

Για ακόμα καλύτερη κατανόηση παρουσιάζεται ένα ακόμα παράδειγμα. Στην εικόνα 2.25 λοιπόν η κάμερα έχει ρυθμιστεί να παρατηρεί αυτό το αντικείμενο με βασικό χρώμα το κίτρινο αλλά κανένα χρώμα δεν έχει οριστεί προς ανίχνευση. Επομένως στο πεδίο ***Incoming Video*** παρουσιάζεται το video όπως έρχεται από την κάμερα.

Όταν όμως ενεργοποιηθούν το κίτρινο και το μαύρο χρώμα θα δοθεί από την εφαρμογή το αποτέλεσμα της εικόνας 2.26 όπου στο αντικείμενο η περιοχή που αποτελείται από κίτρινο χρώμα είναι κυκλωμένη με ένα πλαίσιο που το μέσον του γράφει Yellow ενώ η περιοχή που αποτελείται από μαύρο χρώμα είναι κυκλωμένη από ένα άλλο πλαίσιο με το όνομα Black.

Εικόνα 2.25



Εικόνα 2.26

2.4.2.5 Enable Record

Όταν για κάποιο ενεργοποιημένο χρώμα ανιχνευθεί πληροφορία στο εισερχόμενο video υπάρχει δυνατότητα αποθήκευσης του εισερχόμενου εκείνη τη στιγμή video. Αυτό η δυνατότητα μπαίνει σε λειτουργία εάν το κουτί επιλογής **Enable Record** είναι επιλεγμένο και απενεργοποιείται εάν αυτό από – επιλεγεί.

Τα αποθηκευμένα video τοποθετούνται στο φάκελο **Recorded** και σε αυτόν στο φάκελο που αντιστοιχεί στο όνομα του κάθε χρώματος με όνομα για κάθε νέο αρχείο video που δημιουργείται την ημερομηνία και ώρα του υπολογιστή καθώς και το όνομα του χρώματος για το οποίο γίνεται η εγγραφή.

2.4.2.6 Stop Capture

Το κουμπί που δίνει την δυνατότητα πρόσβασης στο παράθυρο **«*Capture Device Selection*»** απενεργοποιείται αφού οριστεί συσκευή εισόδου video διότι εφόσον αυτή έχει οριστεί και έχει πραγματοποιηθεί σύνδεση της εφαρμογής με αυτήν, για να οριστεί ως συσκευή εισόδου video μια άλλη θα πρέπει πρώτα να κλίσει η ήδη υπάρχουσα σύνδεση. Αυτό είναι εφικτό εάν πατηθεί το κουμπί ***Stop Capture***, ή από το ***menu*** →***Control***→***Stop Capture,*** το οποίο δίνει την εντολή στην εφαρμογή να κλίσει την σύνδεση αυτής με την συσκευή εισόδου video, με την προϋπόθεση πάντα ότι ήδη έχει γίνει μια σύνδεση της εφαρμογής με μια συσκευή εισόδου video. Αφού αυτό πραγματοποιηθεί με επιτυχία το κουμπί ***Stop Capture*** απενεργοποιείται ενώ το κουμπί ***Set Capture*** το αντίθετο δίνοντας την δυνατότητα να οριστεί από την αρχή συσκευή εισόδου video.

2.4.2.7 Close

Το **κουμπί Close ή το menu → Controls→Close** έχουν σαν αποτέλεσμα τον ασφαλή τερματισμό της εφαρμογής.

Κεφάλαιο 3. Ο κώδικας της εφαρμογής

3.1 Εισαγωγή

Μέσο αυτού το κεφαλαίου γίνεται η προσπάθεια επεξήγησης του πηγαίου κώδικα της εφαρμογής για την κατανόηση του τρόπου λειτουργίας αυτού.

3.2 Τα αρχεία κώδικα της εφαρμογής

Οι εφαρμογή αποτελείται από 12 αρχεία java τα :

- ColoredShapesTracker.java + ColoredShapesTracker.form. Είναι ένα JFrame και αποτελεί το κεντρικό αρχείο της εφαρμογής. Αυτό δίνει την εκκίνηση της λειτουργίας του κώδικα και περιλαμβάνει την φόρμα του κεντρικού παράθυρου, τα κουμπιά ρυθμίσεων και ελέγχου και ενσωματώνει το JPanel VideoStreamController που προβάλει το εισερχόμενο video.
- VideoStreamController.java + VideoStreamController.form. Είναι υπεύθυνο στο να πραγματοποιήσει την σύνδεση με την κάμερα λήψης και να κάνει εξαγωγή από το εισερχόμενο video την εικόνα που θα αποσταλεί στην υπεύθυνη για αυτή τη δουλειά κλάση, κάθε φόρα που αυτό ζητιέται. Το αρχείο αυτό είναι ένα JPanel στο οποίο προβάλλεται το εισερχόμενο video.
- CaptureDevFrame.java + CaptureDevFrame.form. Είναι ένα JFrame και αντιστοιχεί στο παράθυρου που παρουσιάζει τις συσκευές λήψης που παρέχει ο ηλεκτρονικός υπολογιστής που εκτελείται η εφαρμογή δίνοντας την δυνατότητα στον χειριστή να επιλέξει μια από αυτές για να αποτελέσει την συσκευή εισόδου video της εφαρμογής
- DeviceFormat.java. Αναλαμβάνει να ελέγχει αν υποστηρίζεται η επιλεγμένη συσκευή εισόδου καθώς και το επιλεγμένο format και

κάνει τις κατάλληλες ενέργειες ώστε αυτά που επιλέχθηκαν να ενεργοποιηθούν και να αρχίσει η λήψη.

- ColorSettings.java + ColorSettings.form. Είναι αρχείο JFrame το οποίο αφορά το παράθυρο που δίνει την δυνατότητα στο χειριστή να επιλέγει ποια θα είναι τα ενεργά προς ανίχνευση χρώματα καθώς και να ρυθμίζει το φασματικό πεδίο του κάθε χρώματος.

- Color_Chooser_Panel.java + Color_Chooser_Panel.form. Είναι αρχείο JPanel και είναι κομμάτι του αρχείου ColorSettings.java το οποίο τυπώνει στο panel του φάσμα H.S.I. του κάθε χρώματος.

- ColorValues.java + ColorValues.form. Σε αυτό το αρχείο κρατιόνται οι τιμές Hue Low – Hi, Saturaration Low – Hi, Intensity Li – Hi για το κάθε χρώμα.

- ImageProcessor.java. Η πλέον κεντρική και απαραίτητη κλάση της εφαρμογής. Σε αυτή την κλάση αποστέλλεται κάθε φορά η εικόνα προς επεξεργασία για να γίνει η ανίχνευση εάν υπάρχει αντικείμενο – στόχος. Εφαρμόζεται ο αλγόριθμος Color Tracking και εφόσον βρεθούν περιοχές ενδιαφέροντος τα σημεία που αποτελούν την κάθε περιοχή αποστέλλονται στην κλάση JarvisMarch.java.

- JarvisMarch.java. Σε αυτήν την κλάση εκτελείται ο κώδικας που υλοποιεί τον Convex Hull αλγόριθμο και δίνει την περιβάλλουσα της περιοχής που επεξεργάζεται.

- SinkData.java. Η κλάση αυτή αναλαμβάνει την εκκίνηση ή τον τερματισμό της διαδικασία εγγραφής video για το εκάστοτε χρώμα που είναι ενεργό και διαπιστώνεται ότι υπάρχει στο πεδίο λήψης

- MyDataSinkListener.java. Listener της κλάσης SinkData.java για την dataDink διαδικασία.

- SinkColorValues.java. Αυτή η κλάση λειτουργεί σαν panel με διακόπτες ON – OFF. Διαθέτει Boolean μεταβλητές για το κάθε χρώμα. Μια Boolean μεταβλητή γίνεται true όταν έχει ανιχνευθεί το αντίστοιχο χρώμα και έχει ξεκινήσει για αυτό η καταγραφή video ενώ false όταν η διαδικασία καταγραφής έχει σταματήσει ή δεν έχει ξεκινήσει καν. Αυτό είναι απαραίτητο για την εφαρμογή ώστε να μπορεί αυτή να γνωρίζει για ποια χρώματα γίνεται καταγραφή. Αυτό είναι απαραίτητο για πάρα πολλούς λόγους. Γιατί για παράδειγμά αν γίνει τερματισμός της εφαρμογής ενώ γίνεται καταγραφή, θα πρέπει

όλες οι διαδικασίες καταγραφής να τερματίσουν πριν την εφαρμογή ώστε να αποθηκευτούν τα video με ασφάλεια.

Η εφαρμογή ακόμα αποτελείται από 12 αρχεία properties όπου το κάθε ένα από αυτά έχει αποθηκευμένες τις τιμές Hue Low – Hi, Saturation Low – Hi, Intensity Low – Hi που ορίζονται από τον χειριστή καθώς και τις default τιμές αυτών που ορίστηκαν εκ – κατασκευής καθώς και μια μεταβλητή Boolean που δηλώνει αν έχουν τοποθετηθεί οι default τιμές ή έχουν γίνει αλλαγές. Τα αρχεία αυτά καθώς και οι τιμές τους παρουσιάζονται στη συνέχεια :

- RedProperties.properties :
  - RedHueLo = 400
  - RedHueHi = 540
  - RedSatLo = 600
  - RedSatHi = 1000
  - RedIntLo = 200
  - RedIntHi = 1000
  - RedHueLoDefault = 400
  - RedHueHiDefault = 540
  - RedSatLoDefault = 600
  - RedSatHiDefault = 1000
  - RedIntLoDefault = 200
  - RedIntHiDefault = 1000
  - Default = false
- OrangeProperties.properties :
  - OrangeHueLo = 60
  - OrangeHueHi = 110
  - OrangeSatLo = 200
  - OrangeSatHi = 1000
  - OrangeIntLo = 500
  - OrangeIntHi = 1000
  - OrangeHueLoDefault = 60
  - OrangeHueHiDefault = 110
  - OrangeSatLoDefault = 200
  - OrangeSatHiDefault = 1000
  - OrangeIntLoDefault = 500
  - OrangeIntHiDefault = 1000
  - Default = false

- BrownProperties.properties :
  - BrownHueHi = 110
  - BrownHueLo = 60
  - BrownSatHi = 1000
  - BrownSatLo = 200
  - BrownIntLo = 200
  - BrownIntHi = 500
  - BrownHueLoDefault = 60
  - BrownHueHiDefault = 110
  - BrownSatLoDefault = 200
  - BrownSatHiDefault = 1000
  - BrownIntHiDefault = 500
  - BrownIntLoDefault = 200
  - Default = false
- YellowProperties.properties :
  - YellowHueLo = 110
  - YellowHueHi = 200
  - YellowSatLo = 200
  - YellowSatHi = 1000
  - YellowIntLo = 200
  - YellowIntHi = 1000
  - YellowHueLoDefault = 110
  - YellowHueHiDefault = 200
  - YellowSatLoDefault = 200
  - YellowSatHiDefault = 1000
  - YellowIntLoDefault = 200
  - YellowIntHiDefault = 1000
  - Default = false
- GreenProperties.properties :
  - GreenHueLo = 200
  - GreenHueHi = 450
  - GreenIntLo = 0
  - GreenIntHi = 1000
  - GreenSatLo = 200
  - GreenSatHi = 1000
  - GreenHueLoDefault = 200
  - GreenHueHiDefault = 450
  - GreenSatLoDefault = 200
  - GreenSatHiDefault = 1000
  - GreenIntLoDefault = 200
  - GreenIntHiDefault = 1000
  - Default = false

- CyanProperties.properties :
  - CyanHueLo = 450
  - CyanHueHi = 580
  - CyanSatLo = 200
  - CyanSatHi = 1000
  - CyanIntLo = 200
  - CyanIntHi = 1000
  - CyanHueLoDefault = 450
  - CyanHueHiDefault = 580
  - CyanSatLoDefault = 200
  - CyanSatHiDefault = 1000
  - CyanIntLoDefault = 200
  - CyanIntHiDefault = 1000
  - Default = false
- BlueProperties.properties :
  - BlueHueLo = 580
  - BlueHueHi = 760
  - BlueSatLo = 200
  - BlueSatHi = 1000
  - BlueIntLo = 200
  - BlueIntHi = 1000
  - BlueHueLoDefault = 580
  - BlueHueHiDefault = 760
  - BlueSatLoDefault = 200
  - BlueSatHiDefault = 1000
  - BlueIntLoDefault = 200
  - BlueIntHiDefault = 1000
  - Default = false
- MagentaProperties.properties :
  - MagentaHueLo = 760
  - MagentaHueHi = 900
  - MagentaSatLo = 200
  - MagentaSatHi = 1000
  - MagentaIntLo = 200
  - MagentaIntHi = 1000
  - MagentaHueLoDefault = 760
  - MagentaHueHiDefault = 900
  - MagentaSatLoDefault = 200
  - MagentaSatHiDefault = 1000
  - MagentaIntLoDefault = 200
  - MagentaIntHiDefault = 1000
  - Default = false

- PinkProperties.properties :
  - PinkHueLo = 400
  - PinkHueHi = 540
  - PinkIntLo = 400
  - PinkIntHi = 1000
  - PinkSatLo = 200
  - PinkSatHi = 600
  - PinkHueLoDefault = 400
  - PinkHueHiDefault = 540
  - PinkSatLoDefault = 200
  - PinkSatHiDefault = 600
  - PinkIntLoDefault = 400
  - PinkIntHiDefault = 1000
  - Default = false
- WhiteProperties.properties :
  - WhiteHueLo = 0
  - WhiteHueHi = 1000
  - WhiteSatLo = 0
  - WhiteSatHi = 200
  - WhiteIntLo = 900
  - WhiteIntHi = 1000
  - WhiteHueLoDefault = 0
  - WhiteHueHiDefault = 1000
  - WhiteSatLoDefault = 0
  - WhiteSatHiDefault = 200
  - WhiteIntLoDefault = 900
  - WhiteIntHiDefault = 1000
  - Default = false
- GrayProperties.properties
  - GrayHueLo = 0
  - GrayHueHi = 1000
  - GraySatLo = 0
  - GraySatHi = 200
  - GrayIntLo = 200
  - GrayIntHi = 900
  - GrayHueLoDefault = 0
  - GrayHueHiDefault = 1000
  - GraySatLoDefault = 0
  - GraySatHiDefault = 200
  - GrayIntLoDefault = 200
  - GrayIntHiDefault = 900
  - Default = false

- BlackProperties.properties
    - BlackHueHi = 1000
    - BlackHueLo = 0
    - BlackSatLo = 0
    - BlackSatHi = 1000
    - BlackIntLo = 0
    - BlackIntHi = 190
    - BlackHueLoDefault = 0
    - BlackHueHiDefault = 1000
    - BlackSatLoDefault = 0
    - BlackSatHiDefault = 1000
    - BlackIntLoDefault = 0
    - BlackIntHiDefault = 200
    - Default = false

3.3 Ο κώδικας βήμα – βήμα

3.3.1 Η κλάση ColoredShapesTracker

Όταν γίνει διπλό κλικ στο αρχείο Run.cmd στο φάκελο Runnable σηματοδοτεί την εκκίνηση της εφαρμογής. Αυτό έχει σαν αποτέλεσμα να κλιθεί η main της κλάσης ColoredShapesTracker η οποία πυροδοτεί μια αλυσίδα ενεργειών. Ως πρώτες ενέργειες είναι η δήλωση των απαραίτητων μεταβλητών και η δημιουργία του γραφικού περιβάλλοντος. Μόλις αυτά ολοκληρωθούν η εφαρμογή «περιμένει» εντολές από το χειριστή για να προβεί στη συνέχιση των λειτουργιών που παρέχει.

Αφού ο χρήστης δηλώσει κάμερα εισόδου η εφαρμογή αναγνωρίζει αυτήν την ενέργεια μέσω ενός timer και καλεί την συνάρτηση **start()** (εικόνα 3.1)**.**

Η συνάρτηση αυτή ως πρώτη ενέργεια καλεί τρεις άλλες συναρτήσεις. Την getCaptureDeviceName(), όπου από την οποία δέχεται ως επιστροφή το όνομα της επιλεγμένης συσκευής λήψης video, την getCaptureVideoFormat(), όπου από την οποία η επιστροφή της είναι το επιλεγμένο format και την getCaptureDeviceLocator(), όπου δέχεται ως επιστροφή τον locator την κάμερας.

Στη συνέχεια γίνεται έλεγχος αν οι μεταβλητές που επέστρεψαν έχουν περιεχόμενο και αν ναι καλείται η κλάση **VideoStreamController** στέλνοντας σε

αυτή τις μεταβλητές αυτές σαν όρισμα. Αν οι μεταβλητές δεν έχουν τιμή σημαίνει ότι δεν έγινε επιλογή κάποιας συσκευής λήψης επομένως η εφαρμογή καλέσει την κλάση **VideoStreamController** αλλά σαν όρισμα θα της περάσει την default κάμερα λήψης, την : "vfw:Live! Cam Vista IM (VFW):0", καθώς και το default format, το : "size=640x480, encoding=rgb, maxdatalength=921600".

```
public void start(){
    String capDevName = getCaptureDeviceName();
    String capVidFormat = getCaptureVideoFormat();
    String capDevLocator = getCaptureDeviceLocator();
    if(capDevName != null && capVidFormat != null && capDevLocator != null){
        videoStreamController = new VideoStreamController(capDevName, capVidFormat);
        getContentPane().add(videoStreamController, BorderLayout.SOUTH);
        setVisible(true);
    }else{
        videoStreamController = new VideoStreamController(CAPTURE_DEVICE_NAME, VIDEO_FORMAT);
        getContentPane().add(videoStreamController, BorderLayout.SOUTH);
        setVisible(true);
    }
}
```

Εικόνα 3.1

Στη συνέχεια, όταν ο χειριστής κάνει επιλογή ενός οι περισσότερων χρωμάτων η κλάση αναλαμβάνει να «διαβάσει» τις τιμές Hue Low – Hi, Saturation Low – Hi, Intensity Low – Hi για το κάθε χρώμα και να δώσει τις τιμές αυτές στην κλάση **ColorValues**.

Για παράδειγμα ο κώδικας που διαβάζει αυτές τις τιμές και τις περνάει στην κλάση **ColorValues** για το χρώμα κόκκινο φαίνεται στην εικόνα 2.2 όπου γίνεται άνοιγμα του αρχείου properties και μια – μια οι τιμές διαβάζονται και στέλνονται στις αντίστοιχες μεταβλητές της κλάσης **ColorValues**. Οι συναρτήσεις όπως αυτή έχουν και μια ακόμα ενέργεια. Υποθέτοντας ότι η εφαρμογή βρίσκεται σε λειτουργία το κόκκινο χρώμα είναι στα ενεργά προς ανίχνευση χρώματα, υπάρχει πληροφορία για αυτό στο πεδίο λήψης και η διαδικασία εγγραφής video βρίσκεται σε λειτουργία. Εάν εκείνη τη στιγμή ο χειριστής απενεργοποιήσει το κόκκινο χρώμα, τότε η διαδικασία εγγραφής θα πρέπει να σταματήσει. Αυτό το σταμάτημα γίνεται από αυτού του είδους τις συναρτήσεις. Για αυτή την περίπτωση και μόνο όμως.

```
public static void setRed(boolean aColor){
    red = aColor;
    if(red == true){
        try{
            Properties settings = new Properties();
            FileInputStream in = new FileInputStream("src/properties/RedProperties.properties");
            settings.load(in);
            colorValues.setRedHueLo(Integer.parseInt(settings.getProperty("RedHueLo")));
            colorValues.setRedHueHi(Integer.parseInt(settings.getProperty("RedHueHi")));
            colorValues.setRedSatLo(Integer.parseInt(settings.getProperty("RedSatLo")));
            colorValues.setRedSatHi(Integer.parseInt(settings.getProperty("RedSatHi")));
            colorValues.setRedIntLo(Integer.parseInt(settings.getProperty("RedIntLo")));
            colorValues.setRedIntHi(Integer.parseInt(settings.getProperty("RedIntHi")));
            in.close();
        }catch (IOException inE){
            inE.printStackTrace();
        }
    }else if(red == false && VideoStreamController.sinkColorValues.getSinkRed() == true){
        VideoStreamController.sinkColorValues.setSinkRed(false);
        VideoStreamController.redSinkData.stop("Red");
    }
}
```

Εικόνα 3.2

Επίσης αυτή η κλάση είναι υπεύθυνη για τον ασφαλή τερματισμό της εφαρμογής όπου μέσω της συνάρτησης **stopRec()** σταματάει όλες τις ενεργές διαδικασίες καταγραφής video, μέσω της **stopCap()** κλίνει την σύνδεση με την κάμερα και απελευθερώνει τις μεταβλητές που αφορούν την σύνδεση και μέσω της **closeAp()** τερματίζει την εφαρμογή.

3.3.2 Η κλάση CaptureDevFrame

Η κλάση αυτή παρουσιάζει τις υποστηριζόμενες από το σύστημα συσκευές εισόδου (εικόνα 3.3) και τις προβάλει στο χειριστή μέσω μιας jlist. Από εκεί ο χει-

```
public CaptureDevFrame() {
    initComponents();
    devices = CaptureDeviceManager.getDeviceList(null);
    if (devices == null)
        System.out.println("No Capture devices known to JMF");
    else {
        for (int i = 0; i < devices.size(); i++){
            CaptureDeviceInfo captureDeviceInfo = (CaptureDeviceInfo)devices.elementAt(i);
            devicesListModel.addElement(captureDeviceInfo.getName());
        }
        jList1.setModel(devicesListModel);
    }
}
```

Εικόνα 3.3

ριστής μπορεί να επιλέξει αυτή που θέλει να χρησιμοποιήσει και τότε σε μια άλλη jlist θα εμφανιστούν τα υποστηριζόμενα από την επιλεγμένη συσκευή format για να γίνει επίσης επιλογή κάποιου από αυτά.

3.3.3 Οι κλάσεις VideoStreamController - DeviceFormat

Η κλάση **VideoStreamController** δέχεται ως όρισμα την επιλεχθείσα κάμερα εισόδου και το format που θα κωδικοποιηθεί το video. Σε συνεργασία με την κλάση **CaptureDevFrame** γίνεται η σύνδεση με την κάμερα και η ληφθείσα εικόνα γίνεται είσοδος ενός player.

Μόλις ολοκληρωθεί η σύνδεση με την κάμερα, μπαίνει σε λειτουργία ένας timer ο οποίος σε κάθε αρχή του κύκλου επεξεργασίας καλεί την συνάρτηση **step()** (εικόνα 3.4). Η συνάρτηση αυτή έχει οριστεί να καλεί την **getFrame()** (εικόνα 3.5) η οποία κάνει εξαγωγή από το video την εικόνα που λαμβάνεται εκείνη τη στιγμή μετατρέποντάς την σε Image και να την στέλνει πίσω στην **step()**

Η **step()** με την σειρά της μετατρέπει την εικόνα αυτή σε BufferedImage και την αποστέλλει στην κλάση **ImageProcessor** για επεξεργασία, εφόσον βέβαια έστω ένα χρώμα έχει επιλεγεί. Μόλις η εικόνα επιστρέψει ξανά στην κλάση **VideoStreamController** προβάλλεται στο panel μέσω της **paint()** (εικόνα 3.6) αυτής και αυτό σηματοδοτεί το τέλος του κύκλου επεξεργασίας και την αρχή του επόμενου.

Μια ακόμα ενέργεια της **step()** είναι σε περίπτωση που ενώ η εφαρμογή βρίσκεται σε λειτουργία, χρώματα προς ανίχνευση είναι ενεργά και η διαδικασία καταγραφής video βρίσκεται σε λειτουργία και όλα τα χρώματα απενεργοποιηθούν, η κλάση **ImageProcessor** δεν καλείται και όλες οι διαδικασίες εγγραφής που είναι ενεργές τερματίζονται.

```
public void step(){
    frameImage = getFrame();
    bufferedImage = new BufferedImage(frameImage.getWidth(null),
            frameImage.getHeight(null), BufferedImage.TYPE_INT_RGB);
    bufferedImage.createGraphics().drawImage(frameImage, 0, 0, this);
    //sharpen();
    boolean process = ColoredShapesTracker.getProcess();
    if(process == true){
        ImageProcessor imageProcessor = new ImageProcessor();
        imageProcessor.ImageProcessor(bufferedImage);
        bufferedImage = imageProcessor.getLiveImage();
    }else{
        if(sinkColorValues.getSinkRed() == true){
            sinkColorValues.setSinkRed(false);
            redSinkData.stop("Red");
        }
        if(sinkColorValues.getSinkOrange() == true){
            sinkColorValues.setSinkOrange(false);
            orangeSinkData.stop("Orange");
        }
        if(sinkColorValues.getSinkBrown() == true){
            sinkColorValues.setSinkBrown(false);
            brownSinkData.stop("Brown");
        }
        if(sinkColorValues.getSinkYellow() == true){
```

Εικόνα 3.4

```
public Image getFrame(){
    // Grab a frame from the capture device
    FrameGrabbingControl frameGrabber = (FrameGrabbingControl)player.
            getControl("javax.media.control.FrameGrabbingControl");
    Buffer buf = frameGrabber.grabFrame();
    // Convert frame to an buffered image and return
    Image img = (new BufferToImage((VideoFormat)buf.getFormat()).createImage(buf));
    if (img==null){
            System.out.println("Error: Capture device doesnt appear to be initialised yet.");
            return null;            // happens if video device isn't properly initialised yet
    }
    //BufferedImage buffImg = new BufferedImage(640, 480, BufferedImage.TYPE_INT_RGB);
    return img;
}
```

Εικόνα 3.5

```
public void paint(Graphics g) {
    g.clearRect(0,0,640,480);
    g.drawImage(bufferedImage, 0, 0, this);
}
```

Εικόνα 3.6

Η κλάση **DeviceFormat** έχει ως σκοπό να ορίσει ως format εισόδου στην εφαρμογή, το format που επέλεξε ο χειριστής. Καλείται από την **VideoStreamController** η συνάρτηση **setFormat()** (εικόνα 3.7) η οποία έχει ως σκοπό να εφαρμόσει το format που επιλέχθηκε ως αυτό που θα αποτελέσει το format που θα γίνεται η λήψη. Μέσω της **formatMatches()** (εικόνα 3.8) βρίσκει το αντίστοιχο με το επιλεγμένο και το εφαρμόζει αφού πρώτα ελέγξει με την **isVideo()** (εικόνα 3.9) αν είναι video format.

```
public static boolean setFormat(DataSource dataSource, Format format){
    boolean formatApplied = false;
    FormatControl formatControls[] = null;
    formatControls = ((CaptureDevice) dataSource).getFormatControls();
    for (int x = 0; x < formatControls.length; x++){
        if (formatControls[x] == null)
            continue;

        Format supportedFormats[] = formatControls[x].getSupportedFormats();
        if (supportedFormats == null)
            continue;

        if (DeviceFormat.formatMatches(format, supportedFormats) != null){
            formatControls[x].setFormat(format);
            formatApplied = true;
        }
    }
    return formatApplied;
}
```

Εικόνα 3.7

```
public static Format formatMatches(Format format, Format supported[]){
    if (supported == null)
        return null;
    for (int i = 0;  i < supported.length;  i++)
        if (supported[i].matches(format))
            return supported[i];
    return null;
}
```

Εικόνα 3.8

```
public static boolean isVideo(Format format){
    return (format instanceof VideoFormat);
}
```

Εικόνα 3.9

Οι συναρτήσεις **formatToString()** (εικόνα 3.10) και **videoFormatToString()** (εικόνα 3.11) χρησιμοποιούνται για να μετατρέψουν το format σε μορφή κειμένου ώστε να εκτυπωθεί σε κατανοητή μορφή για τον χειριστή με σκοπό την ενημέρωση αυτού.

```
public static String formatToString(Format format){
    if (isVideo(format))
        return videoFormatToString((VideoFormat) format);
    return ("--- unknown media device format ---");
}
```

Εικόνα 3.10

```
public static String videoFormatToString(VideoFormat videoFormat){
    StringBuffer result = new StringBuffer();

    // add width x height (size)
    Dimension d = videoFormat.getSize();
    result.append("size=" + (int) d.getWidth() + "x" + (int) d.getHeight() + ", ");

    // add encoding
    result.append("encoding=" + videoFormat.getEncoding() + ", ");

    // add max data length
    result.append("maxdatalength=" + videoFormat.getMaxDataLength() + "");

    return result.toString();
}
```

Εικόνα 3.11

3.3.4 Οι κλάσεις ColorSettings - Color_Chooser_Panel

Η κλάση **ColorSettings** είναι καταρχάς ένα παράθυρο αλληλεπίδρασης με το χειριστή. Προβάλλονται μέσω αυτού σε μια λίστα τα υποστηριζόμενα από την εφαρμογή χρώματα δίνοντας στον χειριστή τη δυνατότητα να επιλέξει ποιο ή ποια από αυτά θα είναι ενεργά προς ανίχνευση. Καθώς και την δυνατότητα να ρυθμίσει το φάσμα του κάθε χρώματος.

Στην περίπτωση ρύθμισης του φάσματος η συνάρτηση **colorSetup()** σε πρώτη φάση ανοίγει το αρχείο properties που αντιστοιχεί στο επιλεγμένο χρώμα διαβάζοντας της τιμές Hue Low – Hi, Saturation Low – Hi, Intensity Low – Hi, που το αφορούν. Στη συνέχεια τοποθετεί τις μπάρες κύλισης, την κάθε μια στην ανάλογη θέση της αντίστοιχης τιμής τους. Για παράδειγμα αν το Hue Lo ενός χρώματος έχει τιμή 580 η αντίστοιχη μπάρα κύλισης θα τοποθετηθεί στη τιμή 58. Έπειτα καλεί την κλάση **Color_Chooser_Panel,** στέλνοντάς της τις τιμές Hue Low – Hi, Saturation Low – Hi, Intensity Low – Hi του επιλεγμένου χρώματος, η οποία η μόνη της ενέργεια κάθε φορά που καλείται είναι να μετατρέπει μέσω της συνάρτησης **colorPrint()** (εικόνα 3.12) τις τιμές αυτές σε εικόνα και να την τυπώνει μέσω της συνάρτησης **paint()** (εικόνα 3.13) επάνω στο παράθυρο της κλάσης **ColorSettings** ώστε να δίνει στο χειριστή μια όσο το δυνατόν καλύτερη αντίληψη του φάσματος που έχει οριστεί, καθώς και κάθε φορά που γίνεται μια αλλαγή στις τιμές του φάσματος να είναι ορατή αυτή μέσω μιας αναπαράστασης.

Η συνάρτηση **colorSetup()** χρησιμοποιείται επίσης στην περίπτωση αποθήκευσης των αλλαγών που μπορεί να έχουν γίνει στο φάσμα ενός χρώματος αλλάζοντας τις αντίστοιχες τιμές στο αντίστοιχο αρχείο properties.

Μια ακόμα δυνατότητα που παρέχει η κλάση **ColorSettings** είναι η επαναφορά των ρυθμίσεων του φάσματος κάθε χρώματος οποιαδήποτε στιγμή.

```
public void colorPrint(int satHi, int satLo, int briHi, int briLo, int hueHi, int hueLo){
    bufferedImage = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    WritableRaster raster = bufferedImage.getRaster();
    if(ColorSettings.caseRed == true){
        hueHi += 500;
        hueLo += 500;
    }
    float[] hsb = new float[3];
    hsb[0] = (float)0.001 * hueLo;
    hsb[1] = (float)0.001 * satLo;
    hsb[2] = (float)0.001 * briHi;
    int k = 0;
    float hueStep = (float)(((hueHi - hueLo) * 0.001) / width);
    float satStep = (float)(((satHi - satLo) * 0.001) / height);
    float briStep = (float)(((briHi - briLo) * 0.001) / height);
    for(int x = 0; x < width; x++){
        hsb[1] = (float)0.001 * satLo;
        hsb[2] = (float)0.001 * briHi;
        for(int y = 0; y < height; y++){
            pixelArray = new int[3];
            k = color.HSBtoRGB(hsb[0], hsb[1], hsb[2]);
            r = (k & 0x00FF0000) >> 16;
            g = (k & 0x0000FF00) >> 8;
            b = (k & 0x000000FF);
            pixelArray[0] = r;
            pixelArray[1] = g;
            pixelArray[2] = b;
            raster.setPixel(x, y, pixelArray);
            hsb[1] += satStep;
            hsb[2] -= briStep;
        }
        hsb[0] += hueStep;
    }
    repaint();
}
```

Εικόνα 3.12

```
public void paint(Graphics g) {
    g.clearRect(0,0,getWidth(),getHeight());
    g.drawImage(bufferedImage, 0, 0, this);
}
```

Εικόνα 3.13

3.3.5 Η κλάση ColorValues

Σε αυτήν τη κλάση τοποθετούνται οι τιμές Hue Low – Hi, Saturation Low – Hi, Intensity Low – Hi για όλα τα ενεργά προς ανίχνευση χρώματα σε μεταβλητές όπως redHueLo, redHueHi, redSatLo, redSatHi, redIntLo, redIntHi.

Όταν ένα χρώμα ενεργοποιείται προς ανίχνευση ενημερώνεται η κλάση **ColoredShapesTracker** η οποία για κάθε χρώμα που ενεργοποιείται τοποθετεί τις τιμές που το αφορούν στις αντίστοιχες για το χρώμα τις κλάσης **ColorValues**.

Από αυτήν την κλάση αντλεί συνεχώς τις τιμές των ενεργών χρωμάτων η κλάση **ImageProcessor** σε κάθε κύκλο επεξεργασίας.

Ένα παράδειγμα κώδικα τοποθέτησης τιμής φαίνεται στην εικόνα 3.14 όπου γίνεται τοποθέτηση της τιμής Hue Low του χρώματος κόκκινου.

```
public void setRedHueLo(int redHueLo) throws IllegalArgumentException{
    /**
     * */
    if (redHueLo >= MINVALUE && redHueLo <= MAXVALUE && redHueLo <= redHueHi){
        this.redHueLo = redHueLo;
    }
    else
        throw new IllegalArgumentException();
}
```

Εικόνα 3.14

Ενώ ένα παράδειγμα κώδικα άντλησης τιμής φαίνεται στην εικόνα 3.15 όπου γίνεται άντληση της τιμής Hue Low του χρώματος κόκκινου.

```
public int getRedHueLo(){
    return redHueLo;
}
```

Εικόνα 3.15

3.3.6 Οι κλάσεις SinkData και MyDataSinkListener

Η εφαρμογή έχει την δυνατότητα να ελέγχει από ένα έως 12 προεπιλεγμένα χρώματα. Όταν η κλάση **ImageProcessor** ανιχνεύσει την ύπαρξη κάποιου ή κάποιων από τα επιλεγμένα χρώματα στο εισερχόμενο video δίνει εντολή να ενεργοποιηθεί η διαδικασία καταγραφής video, αν αυτή είναι ενεργή, για κάθε χρώμα που ανιχνεύεται και αντίστοιχα όταν διαπιστώσει ότι πλέον δεν υπάρχει το χρώμα ή τα χρώματα για τα οποία η καταγραφή βρίσκεται υπό εκτέλεση, δίνει εντολή να γίνει διακοπή της διαδικασίας αυτής για κάθε χρώμα που δεν βρίσκεται πλέον στο πεδίο λήψης.

Τις εντολές αυτές τις διαχειρίζεται η κλάση **SinkData**. Το κάθε χρώμα δημιουργεί μια «δικιά» του **SinkData** ώστε να είναι δυνατή η ταυτόχρονη καταγραφή περισσότερων του ενός χρωμάτων ταυτόχρονα, κάτι το οποίο απαιτεί την ύπαρξη τόσων διαδικασιών **SinkData** όσα και τα χρώματα.

Για την έναρξη λοιπόν της διαδικασίας καταγραφής είναι υπεύθυνη η συνάρτηση **SinkData()**. Στη συνάρτηση αυτή δημιουργείται ένας **processor** στον οποίο δίνεται σαν είσοδο η έξοδος του **DataSource** της κλάσης **VideoStreamController** (εικόνα 3.16). Αυτός χρησιμοποιείται για να δημιουργηθεί μια μεταβλητή **ProcessorModel** στην οποία δηλώνεται επίσης ο τύπος video που θα χρησιμοποιηθεί για την εγγραφή καθώς και το format κωδικοποίησης.

Από τον **ProcessorModel** δημιουργείται ένας ακόμα **processor** του οποίου η έξοδος οδηγείται σε μια μεταβλητή **DataSource** και από αυτήν τη μεταβλητή στην **DataSink** η οποία είναι υπεύθυνη για την δημιουργία του αρχείου video. Επίσης δίνεται σαν όνομα στο αρχείο που πρόκειται να δημιουργηθεί η τρέχουσα ημερομηνία και ώρα του υπολογιστή που εκτελείται η εφαρμογή (εικόνα 3.17).

Επόμενη ενέργεια είναι η έναρξη της καταγραφής που γίνεται αφού ο **processor** έρθει σε κατάσταση λειτουργίας, γίνει εκκίνηση του **DataSink** και εκκίνηση του **processor** (εικόνα 3.18).

Ο τερματισμός της διαδικασίας εγγραφής γίνεται από την συνάρτηση **stop()** της κλάσης **SinkData** (εικόνα 3.19), η οποία σταματάει, κλίνει και απελευθερώνει και τις δύο μεταβλητές **processor**, και αφού διαπιστώσει μέσω της κλάσης **MyDataSinkListener** (εικόνα 3.20) ότι το stream εισερχόμενου video άδειασε τότε κλίνει και την μεταβλητή **Datasink**.

```
        outputFormat[0] = new VideoFormat(VideoFormat.RGB);
        ds2 = ((SourceCloneable)VideoStreamController.ds1).createClone();

        try {
            sinkProcessor = Manager.createProcessor(ds2);
        }
        catch (IOException e) { System.out.println(e); }
        catch (NoProcessorException e) { System.out.println(e); }

        sinkProcessor.realize();
        while (sinkProcessor.getState( ) != Processor.Realized)
                        ;
        sinkProcessor.start();
        while (sinkProcessor.getState() != Processor.Started)
                                ;
        // create processor
        processorModel = new ProcessorModel(sinkProcessor.getDataOutput(),
                outputFormat, outputType);
        try {
            processor = Manager.createRealizedProcessor(processorModel);
        }
        catch (IOException e) { System.out.println(e); }
        catch (NoProcessorException e) { System.out.println(e); }
        catch (CannotRealizeException e) { System.out.println(e); }

        // get the output of the processor
        source = processor.getDataOutput();
```

Εικόνα 3.16

```
try {
    // create a File protocol MediaLocator with the location
    // of the file to which bits are to be written
    date = new Date();
    String fileName = (color + "_" + date.getDate() + "_" +
            date.getMonth() + "_" + (date.getYear() + 1900) +
            "_" + date.getHours() + "_" + date.getMinutes() + "_" +
            date.getSeconds());
    String destination = ("file:Recorded/" + color + "/" + fileName + ".avi");
    dest = new MediaLocator(destination);
    dataSink = Manager.createDataSink(source, dest);
    dataSinkListener = new MyDataSinkListener();
    dataSink.addDataSinkListener(dataSinkListener);
    dataSink.open();
}
```

Εικόνα 3.17

```
processor.realize();
while(processor.getState() != Processor.Realized)
    ;
processor.prefetch();
while(processor.getState() != Processor.Prefetched)
    ;

// now start the datasink and processor
try {
    dataSink.start();
}
catch (IOException e) { System.out.println(e); }

processor.start();
while(processor.getState() != Processor.Started)
    ;
System.out.println("starting capturing for color " + color + " ...");
```

Εικόνα 3.18

```
public void stop(String color){
    try{
        System.out.println("stopping processor 1...");
        processor.stop();
        System.out.println("closing processor 1...");
        processor.close();
        processor.deallocate();
        System.out.println("stopping processor 2...");
        sinkProcessor.stop();
        System.out.println("closing processor 2...");
        sinkProcessor.close();
        sinkProcessor.deallocate();
        dataSinkListener.waitEndOfStream(4000);
        System.out.println("clocing data sink ...");
        dataSink.close();
        System.out.println("data sink closed for color " + color);
    }catch(Exception e){ System.out.println(e); }
}
```

Εικόνα 3.19

```
public class MyDataSinkListener implements DataSinkListener{
    boolean endOfStream = false;
    /** The standar dataSinkUpdate function*/
    public void dataSinkUpdate(DataSinkEvent event){
        if (event instanceof javax.media.datasink.EndOfStreamEvent)
            endOfStream = true;
    }
    /** The waitEnfOfStream Function.*/
    public void waitEndOfStream(long checkTimeMs){
        while (!endOfStream){
            System.out.println("datasink: waiting for end of stream ...");
            try{
                Thread.currentThread().sleep(checkTimeMs);
            }catch(InterruptedException ie){}
        }
        System.out.println("datasink: ... end of stream reached.");
    }
}
```

Εικόνα 3.20

3.3.7 Η κλάση SinkColorValues

Για να γνωρίζει η εφαρμογή πότε μια διαδικασία καταγραφής είναι είτε ενεργή είτε όχι, υπάρχουν δώδεκα **Boolean** μεταβλητές, μια για κάθε χρώμα, από τις οποίες όταν κάποια είναι **true** σηματοδοτεί ότι η διαδικασία καταγραφής είναι εν λειτουργία ενώ το αντίθετο όταν είναι **false**.

Μια μεταβλητή αυτής τη κλάσης θα γίνει **true** μόνο από την κλάση **ImageProcessor** ενώ **false** μπορεί να γίνουν από τις κλάσης **ColoredShapesTracker**, **VideoStreamController** αλλά και από την **ImageProcessor**.

Με την αλλαγή κάποιας μεταβλητής από αυτές, από **true** σε **false** σηματοδοτείται και η λήψη της διαδικασίας εγγραφής για το χρώμα που αντιστοιχεί η μεταβλητή αυτή.

Στην εικόνα 3.21 φαίνεται ο κώδικας που δίνει τιμή στην μεταβλητή **Boolean** για το χρώμα κόκκινο, ενώ στην εικόνα 3.22 ο αντίστοιχος κώδικας που διαβάζει την τιμή της μεταβλητής αυτής.

```
public void setSinkRed(boolean sink){
    sinkRed = sink;
}
```

Εικόνα 3.21

```
public boolean getSinkRed(){
    return sinkRed;
}
```

Εικόνα 3.22

3.3.8 Η κλάση JarvisMarch

Η κλάση **JarvisMarch** είναι η υλοποίηση του αλγόριθμου **Convex Hull**. Αφού η κλάση **ImageProcessor** ανιχνεύσει μια περιοχή ενδιαφέροντος, στέλνει τις συντεταγμένες όλων των στοιχείων (σημεία) της περιοχής αυτής, μέσω δύο πινάκων **double** μεταβλητών, έναν για τα **x** και έναν για τα **y** στην κλάση **JarvisMarch** για τον υπολογισμό του **Convex Hull** πολυγώνου της περιοχής αυτής, καλώντας την συνάρτηση **forConvex()** (εικόνα 3.23). Η συνάρτηση αυτή καλεί την υποκλάση **Points** στην οποία στέλνει τα σημεία για να τα κάνει τύπου **Points** (εικόνα 3.24).

Στη συνέχεια καλεί την **JarvisMarch()** στέλνοντάς της τα σημεία τύπου **Points** (εικόνα 3.25).

```
public ArrayList forConvex(double x[], double y[]){
    startPoints = new Points(x, y);
    new JarvisMarch(startPoints);
    return arrayList;
}
```

Εικόνα 3.23

```
public static class Points{
    public double x[];
    public double y[];
    /** Creates a new instance of the <code>Points
     *  and the <code>double array y</code>, for tl
     *  */
    public Points(double[] x, double[] y){
        this.x = x;
        this.y = y;
    }
    /** The startingPoint function that founds the
     *  @see JarvisMarch#getStartingPoint
     *  @return a pointer at the possition inside
    public int startingPoint(){
        double minX = x[0];
        double minY = y[0];
        int iMin = 0;
        for(int i = 1; i < x.length; i++){
            if(x[i] < minX){
                minX = x[i];
                iMin = i;
            }else if(minX == x[i] && y[i] < minY){
                minY = y[i];
                iMin = i;
            }
        }
        return iMin;
    }
}
```

Εικόνα 3.24

Η συνάρτηση **JarvisMarch()** καλεί την συνάρτηση **calculateHull()** η οποία κάνει τον υπολογισμό του πολυγώνου και **getHullPoints()** η οποία επιστρέφει τα σημεία που αποτελούν το πολύγωνο αφού αυτό έχει υπολογιστεί.

```
public JarvisMarch(Points pts){
    arrayList = new ArrayList();
    this.pts = pts;
    int hullNumbers =  calculateHull();
    //System.out.println("Hull Numbers : " + hullNumbers);
    Points outPut = getHullPoints();
    for(int i = 0; i < outPut.x.length; i++){
        int x = (int)outPut.x[i];
        int y = (int)outPut.y[i];
        Point point = new Point(x, y);
        arrayList.add(point);
    }
}
```

Εικόνα 3.25

Η συνάρτηση **calculateHull()** πρώτα καλεί την **initializeHull()** (εικόνα 3.27) η οποία μετατρέπει τους δύο πίνακες σε ενωμένες λίστες και στη συνέχεια καλεί την **getStartingPoint()** (εικόνα 3.28) η οποία θα επιστρέψει το πρώτο σημείο από το οποίο θα ξεκινήσει ο αλγόριθμος (το πλέον Νότιο - Δυτικότερο) το οποίο βρέθηκε με την κλίση της **startingPoint()**, (εικόνα 3.24) της κλάσης **Points** η οποία κλίθηκε από την συνάρτηση **getStartingPoint()**. Στη συνέχεια καλείται η **getNextPoint()** (εικόνα 3.29) στην οποία σε συνεργασία με τις συναρτήσεις **relativeAngle()** (εικόνα 3.30), **pseudoAngle()** (εικόνα 3.31) και **quadrantOnePseudoAngle()** (εικόνα 3.32) υπολογίζεται το επόμενο σημείο που θα αποτελέσει σημείο του πολυγώνου **Convex Hull.**

```
public int calculateHull(){
    initializeHull();
    startingPoint = getStartingPoint();
    currentAngle = 0;

    addToHull(startingPoint);
    for(int p = getNextPoint(startingPoint); p != startingPoint; p = getNextPoint(p)){
        addToHull(p);
    }
    buildHullPoints();
    return hullPoints.x.length;
}
```

Εικόνα 3.26

```
private void initializeHull(){
    hx = new LinkedList<Double>();
    hy = new LinkedList<Double>();
}
```

Εικόνα 3.27

```
public int getStartingPoint(){
    return pts.startingPoint();
}
```

Εικόνα 3.28

```
private int getNextPoint(int p){
    double minAngle = MAX_ANGLE;
    int minP = startingPoint;
    for(int i = 0; i < pts.x.length; i++){
        if(i != p){
            double thisAngle = relativeAngle(i, p);
            if(thisAngle >= currentAngle && thisAngle <= minAngle){
                minP = i;
                minAngle = thisAngle;
            }
        }
    }
    currentAngle = minAngle;
    return minP;
}
```

Εικόνα 3.29

```
private double relativeAngle(int i, int p){
    return pseudoAngle(pts.x[i] - pts.x[p], pts.y[i] - pts.y[p]);
}
```

εικόνα 3.30

```
public static double pseudoAngle(double dx, double dy){
    if(dx >= 0 && dy >= 0)
        return quadrantOnePseudoAngle(dx, dy);
    if(dx >= 0 && dy < 0)
        return 1 + quadrantOnePseudoAngle(Math.abs(dy), dx);

    if(dx < 0 && dy < 0)
        return 2 + quadrantOnePseudoAngle(Math.abs(dx), Math.abs(dy));
    if(dx < 0 && dy >= 0)
        return 3 + quadrantOnePseudoAngle(dy, Math.abs(dx));
    throw new Error("Impossible");
}
```

εικόνα 3.31

```
public static double quadrantOnePseudoAngle(double dx, double dy){
    return dx / (dy + dx);
}
```

Εικόνα 3.32

Όταν βρεθεί το επόμενο σημείο καλείται η **addToHull()** (εικόνα 3.33) και προστίθεται το σημείο αυτό στη λίστα που περιέχει τα σημεία του πολυγώνου.

Μετά την ολοκλήρωση της επεξεργασίας καλείται από την **calculateHull()** η **buildHullPoints()** (εικόνα 3.34) η οποία μετατρέπει τη λίστα με τα σημεία σε πίνακες τύπου **double.**

```
private void addToHull(int p){
    hx.add(pts.x[p]);
    hy.add(pts.y[p]);
}
```

Εικόνα 3.33

```
private void buildHullPoints(){
    double[] ax = new double[hx.size()];
    double[] ay = new double[hy.size()];
    int n = 0;
    for(Iterator<Double> ix = hx.iterator(); ix.hasNext();)
        ax[n++] = ix.next();
    n = 0;
    for(Iterator<Double> iy = hy.iterator(); iy.hasNext();)
        ay[n++] = iy.next();
    hullPoints = new Points(ax, ay);
}
```

Εικόνα 3.34

Εφόσον ολοκληρωθεί η διαδικασία καλείται η **getHullPoint()** (εικόνα 3.35) από την κλάση **ImageProcessor** ώστε να πάρει τα σημεία που αποτελούν το πολύγωνο που υπολογίστηκαι και να το εμφανίσει στο πεδίο **Incoming Video**.

```
public Points getHullPoints(){
    return hullPoints;
}
```

Εικόνα 3.35

3.3.9 Η κλάση ImageProcessor

Η κλάση **ImageProcessor** είναι η πλέον σημαντική κλάση ολόκληρης της εφαρμογής. Σε αυτήν αποστέλλεται, σε κάθε κύκλο επεξεργασίας, η προς επεξεργασία εικόνα. Γίνεται η επεξεργασία της εικόνας, αναγνωρίζεται αν περιέχει αντικείμενα – στόχος για τα ενεργά χρώματα, δίνεται η εντολή υπολογισμού για αυτά, αν είναι απαραίτητο, του πολυγώνου **Convex Hull** και τυπώνεται πάνω στην εικόνα και δίνεται η εντολή εκκίνησης ή τερματισμού της διαδικασίας εγγραφής.

Αναλυτικότερα όταν γίνει εξαγωγή μιας εικόνας από την κλάση **VideoStreamController** καλείται από την συνάρτηση **step()** αυτής, η κλάση

ImageProcessor στην οποία στέλνεται ως όρισμα η προς επεξεργασία εικόνα, φορτωμένη σε μια **BufferedImage** μεταβλητή (εικόνα 3.36).

```
public void ImageProcessor(BufferedImage im){
    liveImage = im;
    processedImage = new BufferedImage(im.getWidth(),
            im.getHeight(), BufferedImage.TYPE_INT_RGB);
    processImage();
    //videoUpdateThread = new VideoUpdateThread();
    //videoUpdateThread.start();
}
```

Εικόνα 3.36

Στο σημείο αυτό γίνεται η δημιουργία μιας νέας εικόνας **BufferedImage** με διαστάσεις τις διαστάσεις αυτής που ελήφθη και **RGB** χρωματικό τύπο. Αυτή η πάνω σε αυτήν την εικόνα θα γίνει η επεξεργασία. Η **liveImage**, η οποία είναι και αυτή μεταβλητή τύπου **BufferedImage**, χρησιμοποιείται για να τυπώνονται σε αυτήν τα αποτελέσματα. Με την κλίση λοιπόν της συνάρτησης **processImage()** ξεκινάει η επεξεργασία.

Για τις ανάγκες τις επεξεργασίας δημιουργούνται 12 δισδιάστατοι πίνακες, ένας για κάθε χρώμα.

Στην αρχή λοιπόν της συνάρτησης **processImage()** (εικόνα 3.37) γίνεται αρχικοποίηση του μεγέθους των πινάκων αυτών ίσου με το μέγεθος (Ύψος - Πλάτος) της προς επεξεργασία εικόνας καθώς και ο μηδενισμός των τιμών τους σε μηδέν. Δημιουργούνται τα **WritableRaster** των **BufferedImage** καθώς και ένας πίνακας τριών θώσεων στον οποίο θα περνούν κάθε φορά που αυτό θα είναι απαραίτητο οι τιμή **RGB** κάποιου pixel.

Οι πίνακες αυτοί αποτελούν ένα είδος μάσκας για την εύρεση των περιοχών χρωματικού ενδιαφέροντος. Τι σημαίνει όμως αυτό; Υποθέτοντας ότι γίνεται ανίχνευση για το κόκκινο χρώμα, σε όποιο σημείο x – y της εικόνας θα βρίσκεται pixel με χρώμα κόκκινο, το αντίστοιχο σημείο του πίνακα που αφορά το κόκκινο χρώμα θα παίρνει την τιμή ένα. Επόμενος μετά την ολοκλήρωση της ανάλυσης της εικόνας όπου θα υπάρχει στον πίνακα η τιμή ένα θα σημαίνει ότι το αντίστοιχο σημείο της εικόνας είναι κόκκινο. Αυτό χρησιμεύει στο να κάνει των κώδικα ποιο γρήγορο και να μην χρειάζεται κάθε φορά που είναι απαραίτητη η

γνώση της τιμής ενός σημείου να γίνεται πρόσβαση στην εικόνα κάτι το οποίο μειώνει σημαντικά τους χρόνους επεξεργασίας.

Οι δηλώσεις των πινάκων έχουν γίνει στην αρχή της κλάσης και φαίνονται στην εικόνα 3.38.

```
public void processImage(){
    int height = liveImage.getHeight();
    int width = liveImage.getWidth();

    int redCount = 0, brownCount = 0, orangeCount = 0, yellowCount = 0,
        greenCount = 0, cyanCount = 0, blueCount = 0, magentaCount = 0,
        pinkCount = 0, whiteCount = 0, blackCount = 0, grayCount = 0, filter = 100;

    redMap = new int[height][width];
    orangeMap = new int[height][width];
    brownMap = new int[height][width];
    yellowMap = new int[height][width];
    greenMap = new int[height][width];
    cyanMap = new int[height][width];
    blueMap = new int[height][width];
    magentaMap = new int[height][width];
    pinkMap = new int[height][width];
    whiteMap = new int[height][width];
    blackMap = new int[height][width];
    grayMap = new int[height][width];

    processedImageRaster = processedImage.getRaster();
    liveImageRaster = liveImage.getRaster();
    hsiArray = new float[3];

    for (int y = 0; y < height; y++){
        for (int x = 0; x < width; x++){
            redMap[y][x] = 0;
            orangeMap[y][x] = 0;
            brownMap[y][x] = 0;
            yellowMap[y][x] = 0;
            greenMap[y][x] = 0;
            cyanMap[y][x] = 0;
            blueMap[y][x] = 0;
            magentaMap[y][x] = 0;
            pinkMap[y][x] = 0;
            whiteMap[y][x] = 0;
            blackMap[y][x] = 0;
            grayMap[y][x] = 0;
        }
    }
```

Εικόνα 3.37

```
private static int[][] redMap;
private static int[][] orangeMap;
private static int[][] brownMap;
private static int[][] yellowMap;
private static int[][] greenMap;
private static int[][] cyanMap;
private static int[][] blueMap;
private static int[][] magentaMap;
private static int[][] pinkMap;
private static int[][] whiteMap;
private static int[][] blackMap;
private static int[][] grayMap;
```

Εικόνα 3.38

Στη συνέχεια γίνεται η ανάλυση της εικόνας και το «γέμισμα» με τιμές των πινάκων των χρωμάτων (όσων βέβαια τα αντίστοιχα χρώματα είναι ενεργά προς επεξεργασία) (εικόνα 3.39). Η διαδικασία ανάλυσης γίνεται με οριζόντια σάρωση της εικόνας από την επάνω αριστερή γωνία της εικόνας έως την κάτω δεξιά ανά πέντε γραμμές. Ελέγχονται όλα τα χρώματα ταυτόχρονα. Διαβάζεται η τιμή του pixel στο οποίο βρίσκεται ο κώδικας και αν η τιμή του ικανοποιεί τους κανόνες ελέγχου κάποιου χρώματος η αντίστοιχη τιμή του πίνακα, με αυτή του σημείου, γίνεται ένα. Στη συνέχεια ελέγχεται το επόμενο και το επόμενο μέχρι να φτάσει ο κώδικας στο τέλος της εικόνας.

Η αρχή του κώδικα ανάλυσης γίνεται με ένα διπλό **for**. Η αρχή των κανόνων ελέγχου για κάθε χρώμα γίνεται με μια **if** του τύπου : **if(ColoredShapesTracker.red == true)** η οποία ελέγχει αν το χρώμα στο οποίο αντιστοιχεί είναι ενεργό προς ανίχνευση. Αν ναι, γίνεται έλεγχος της τιμής **H.S.I.** που διαβάστηκε από την εικόνα με το πεδίο που ορίζει το χρώμα στο οποίο βρίσκεται ο έλεγχος. Αν ικανοποιείται και αυτός ο έλεγχος η αντίστοιχη τιμή του πίνακα του χρώματος γίνεται ένα.

Επειδή ο έλεγχος γίνεται ανά πέντε γραμμές, για μεγαλύτερη ταχύτητα, αν το σημείο που βρίσκεται πέντε γραμμές ακριβώς πάνω από αυτό που ελέγχεται είναι ένα (στον ίδιο πάντα πίνακα) τα ενδιάμεσά τους γίνονται όλα ένα.

Επίσης υπάρχουν δώδεκα μεταβλητές τύπου **Integer** μια για κάθε χρώμα οι οποίες χρησιμεύουν σαν μετρητές για το πόσα σημεία βρέθηκαν από το κάθε χρώμα.

Τα ίδια λοιπόν ισχύουν για όλα τα χρώματα και αφού τελειώσει η διαδικασία αυτή ο κώδικας περνάει στην επόμενη φάση επεξεργασίας.

```
for (int y = 0; y < height; y+=5){
    for (int x = 0; x < width; x++){
        hsiArray = new float[3];
        liveImageRaster.getPixel(x, y, rgbArray);
        aColor.RGBtoHSB(rgbArray[0], rgbArray[1], rgbArray[2], hsiArray);
        hsiArray[0] = hsiArray[0] / DIVISOR;
        hsiArray[1] = hsiArray[1] / DIVISOR;
        hsiArray[2] = hsiArray[2] / DIVISOR;
        if(ColoredShapesTracker.red == true){
            float hue = hsiArray[0] + 500;
            if(hue > 1000)
                hue = hue - 1000;
            if (hue >= ColoredShapesTracker.colorValues.getRedHueLo() &&
                    hue <= ColoredShapesTracker.colorValues.getRedHueHi() &&
                    hsiArray[1] >= ColoredShapesTracker.colorValues.getRedSatLo() &&
                    hsiArray[1] <= ColoredShapesTracker.colorValues.getRedSatHi() &&
                    hsiArray[2] >= ColoredShapesTracker.colorValues.getRedIntLo() &&
                    hsiArray[2] <= ColoredShapesTracker.colorValues.getRedIntHi()){
                redCount++;
                processedImageRaster.setPixel(x, y, rgbArray);
                redMap[y][x] = 1;
                if((y - 5) > 0){
                    if(redMap[y - 5][x] == 1){
                       redMap[y - 1][x] = 1;
                       processedImageRaster.setPixel(x, y - 1, rgbArray);
                       redMap[y - 2][x] = 1;
                       processedImageRaster.setPixel(x, y - 2, rgbArray);
                       redMap[y - 3][x] = 1;
                       processedImageRaster.setPixel(x, y - 3, rgbArray);
                       redMap[y - 4][x] = 1;
                       processedImageRaster.setPixel(x, y - 4, rgbArray);
                       redMap[y - 5][x] = 1;
                       processedImageRaster.setPixel(x, y - 5, rgbArray);
                    }
                }
            }
        }
        if(ColoredShapesTracker.orange == true){
            if(hsiArray[0] >= ColoredShapesTracker.colorValues.getOrangeHueLo() &&
                    hsiArray[0] <= ColoredShapesTracker.colorValues.getOrangeHueHi() &&
                    hsiArray[1] >= ColoredShapesTracker.colorValues.getOrangeSatLo() &&
```

Εικόνα 3.39

Η επόμενη φάση της επεξεργασίας είναι το φιλτράρισμα (εικόνα 3.40) που καθορίζει ποιών χρωμάτων οι πίνακες θα συνεχίσουν στο υπόλοιπο της επεξεργασίας. Για να περάσει λοιπόν ένα χρώμα τον έλεγχο που ορίζει αυτό το φίλτρο θα πρέπει καταρχάς να είναι ενεργό και κατά δεύτερον να έχουν βρεθεί τουλάχιστον 100 σημεία στην εικόνα για το χρώμα που ελέγχεται. Αν λοιπόν ένα χρώμα περάσει από αυτό το σημείο καλείται η συνάρτηση **colorSpliter()** στην οποία στέλνεται ο δισδιάστατος πίνακας που αφορά το χρώμα το οποίο την κάλεσε. Αν όμως δεν περάσει από το φίλτρο αυτό θα πρέπει να γίνει έλεγχος αν υπάρχει ενεργή διαδικασία εγγραφής για το συγκεκριμένο χρώμα ώστε αυτή να τερματιστεί, διότι από την στιγμή που δεν περνάει το φίλτρο θεωρείται ότι δεν υπάρχει πληροφορία στο πεδίο λήψης σχετική με το χρώμα αυτό.

Υπάρχουν δώδεκα τέτοια φίλτρα, ένα για κάθε χρώμα.

```
if(ColoredShapesTracker.red == true && redCount > filter){
    colorSpliter(redMap, "Red");
}else if(ColoredShapesTracker.red == true && redCount < filter &&
        VideoStreamController.sinkColorValues.getSinkRed() == true){
    VideoStreamController.sinkColorValues.setSinkRed(false);
    VideoStreamController.redSinkData.stop("Red");
}else if(ColoredShapesTracker.red == false &&
        VideoStreamController.sinkColorValues.getSinkRed() == true){
    VideoStreamController.sinkColorValues.setSinkRed(false);
    VideoStreamController.redSinkData.stop("Red");
}
```

Εικόνα 3.40

Η συνάρτηση **colorSpliter()** (εικόνες 3.41.1, 3.41.2 και 341.3) καλείται από κάθε ενεργό χρώμα εφόσον αυτό εγκρίνεται από το φίλτρο (εικόνα 3.40) και παίρνει σαν όρισμα των δισδιάστατο πίνακα του χρώματος που την καλεί.

Ο πίνακας που δέχεται θα περιέχει στα στοιχεία του ανά περιοχές είτε μηδενικά, είτε άσσους.

Δουλεία λοιπόν της συνάρτησης αυτής είναι να κάνει εφαρμογή του αλγόριθμου **Blob Coloring** η ιδιότητα του οποίου είναι να ξεχωρίζει ότι μια περιοχή με άσσους οι οποίοι είναι σε «επαφή» μεταξύ τους δημιουργεί ένα

αντικείμενο – στόχος. Ο τρόπος λειτουργίας του αναφέρεται αναλυτικά στην παράγραφο έξι του κεφαλαίου ένα.

```
public void colorSpliter(int[][] colorMap, String colorCase){
    int rgb = 0, hsi = 0, h = 0, s = 0, i = 0;
    int height = liveImage.getHeight();
    int width = liveImage.getWidth();
    regionMap = new int[height][width];
    imageMap = colorMap;

    Vector regionEquivalenceTree = new Vector();
    int rgbPixelAbove = 0, rgbPixelLeft = 0, rgbPixelCurrent = 0, regionIndex = 0;
    double distLeft = 0, distAbove = 0;
    int mapFrom;
    Integer mapTo = new Integer(-1), prevMapTo = new Integer(-1), lowerRegion = new Integer(-1);

    // STEP 3: Perform blob colouring
    for (int y = 0; y < height; y++){       // scan rows
        for (int x = 0; x < width; x++){            // scan pixels in current row

            // get rgb values for neighbouring pixels
            if (x != 0)
                rgbPixelLeft = imageMap[y][x - 1];
            rgbPixelCurrent = imageMap[y][x];
            if (y != 0)
                rgbPixelAbove = imageMap[y - 1][x];

            // calculate distance
            if (x != 0){
                if ((rgbPixelLeft == EMPTY_REGION && rgbPixelCurrent != EMPTY_REGION) ||
                        (rgbPixelLeft != EMPTY_REGION && rgbPixelCurrent == EMPTY_REGION))
                    distLeft = 1;
                else
                    distLeft = 0;
            }
            if (y != 0){
                if ((rgbPixelAbove == EMPTY_REGION && rgbPixelCurrent != EMPTY_REGION) ||
                        (rgbPixelAbove != EMPTY_REGION && rgbPixelCurrent == EMPTY_REGION))
                    distAbove = 1;
                else
                    distAbove = 0;
            }
```

Εικόνα 3.41.1

```
if (x == 0 || y == 0){
    if (x != 0 && y == 0){
        if (distLeft == 1){                                                    // CASE 1
            // current pixel different to neighbour - assign new region
            regionIndex++;
            regionMap[y][x] = regionIndex;
            regionEquivalenceTree.add(new TreeSet());
        }else{                                                                 // CASE 2
            // current pixel similar to left pixel - assign to same region as left pixel
            regionMap[y][x] = regionMap[y][x - 1];
        }
    }
    else if  (x == 0 && y != 0){
        if (distAbove == 1){                                                   // CASE 1
            // current pixel different to neighbour - assign new region
            regionIndex++;
            regionMap[y][x] = regionIndex;
            regionEquivalenceTree.add(new TreeSet());
        } else{                                                                // CASE 3
            // current pixel similar to pixel above - assign to same region as pixel above
            regionMap[y][x] = regionMap[y - 1][x];
        }
    }
    else if (x == 0 && y == 0){ // First pixel in image
        // first pixel in image - assign new region number
        regionMap[y][x] = regionIndex;
        regionEquivalenceTree.add(new TreeSet());
    }
}
else{
    if ((distLeft == 1) && (distAbove == 1)){                                  // CASE 1
        // current pixel different to neighbours - assign new region
        // also add new region to equivalence map and make equivalent to itself
        regionIndex++;
        regionMap[y][x] = regionIndex;
        regionEquivalenceTree.add(new TreeSet());
    }else if ((distLeft == 0) && (distAbove == 1)){                            // CASE 2
        // current pixel similar to left pixel - assign to same region as left pixel
        regionMap[y][x] = regionMap[y][x - 1];
    }else if ((distLeft == 1) && (distAbove == 0)){                            // CASE 3
```

Εικόνα 3.41.2

```
                regionMap[y][x] = regionMap[y - 1][x];
            }else if ((distLeft == 0) && (distAbove == 0)){                                    // CASE 4
                // pixel similar to both neighbours - assign to same region as neighbours
                // if neighbours have different region numbers then add to region equivalence map
                if (regionMap[y][x - 1] != regionMap[y - 1][x]){
                    // make equivalence pointer point to lower region number
                    if (regionMap[y][x - 1] < regionMap[y - 1][x]){
                        mapFrom = regionMap[y - 1][x];
                        mapTo = new Integer(regionMap[y][x - 1]);
                    }else{
                        mapFrom = regionMap[y][x - 1];
                        mapTo = new Integer(regionMap[y - 1][x]);
                    }
                    if (mapTo.equals(prevMapTo))
                        mapTo = lowerRegion;
                    else
                        lowerRegion = new Integer(findLowestEquivalentRegion(regionEquivalenceTree,
                                mapTo.intValue()));
                    ((TreeSet)(regionEquivalenceTree.elementAt(mapFrom))).add(lowerRegion);
                    prevMapTo = mapTo;
                    regionMap[y][x] = lowerRegion.intValue();
                }
                else
                    regionMap[y][x] = regionMap[y][x - 1];
            }
        }
    }
}
```

Εικόνα 3.41.3

Στη συνέχεια αφού βρεθούν οι περιοχές που αντιστοιχούν σε αντικείμενα – στόχος, καλείται δύο φορές η συνάρτηση **flatten()** η οποία σε συνεργασία με τις συναρτήσεις **findLowestEquivalentRegion()** και **propogateLowestRegion()** κάνουν ομαλοποίηση των περιοχών που βρέθηκαν (εικόνα 3.42).

```
public static void flatten(Vector tree){
    for (int i = tree.size() - 1; i >= 0; i--){
        int lowestEquivalentRegion = findLowestEquivalentRegion(tree, i);
        propogateLowestRegion(tree, i, lowestEquivalentRegion);
    }
}
public static int findLowestEquivalentRegion(Vector tree, int region){
    TreeSet equivalentRegions = (TreeSet)(tree.elementAt(region));
    if (equivalentRegions.size() > 0){
        TreeSet lowestEquivalentRegions = new TreeSet();
        Iterator it = equivalentRegions.iterator();
        while (it.hasNext()){
            int r = ((Integer)(it.next())).intValue();
            lowestEquivalentRegions.add(new Integer(findLowestEquivalentRegion(tree, r)));
        }
        return ((Integer)(lowestEquivalentRegions.first())).intValue();
    }
    else
        return region;
}
public static void propogateLowestRegion(Vector tree, int start, int lowestEquivalentRegion){
    TreeSet regions = (TreeSet)(tree.elementAt(start));
    if (regions.size() > 0){
        Iterator it = regions.iterator();
        while (it.hasNext()){
            int r = ((Integer)(it.next())).intValue();
            propogateLowestRegion(tree, r, lowestEquivalentRegion);
        }
        regions.clear();
        regions.add(new Integer(lowestEquivalentRegion));
    }
    else
        if (start != lowestEquivalentRegion)
            regions.add(new Integer(lowestEquivalentRegion));
}
```

Εικόνα 3.42

Επόμενη ενέργεια της συνάρτησης **colorSpliter()** είναι να στείλει μια προς μια τις περιοχές που έχουν βρεθεί, αν βέβαια έχουν βρεθεί, στην κλάση **JarvisMarch** για τον υπολογισμό του **Convex Hull** πολυγώνου της κάθε περιοχής (εικόνες 3.43.1 και 3.43.2).

```
if (regionCount > 0){
    // STEP 7: Cut regions and send them at Convex Hull
    int[] regionSize = new int[regionIndex + 2];

    for (int j = 0; j < regionSize.length; j++)
        regionSize[j] = 0;

    for (int y = 0; y < height; y++)
        for (int x = 0; x < width; x++)
            if (imageMap[y][x] != EMPTY_REGION)
                regionSize[regionMap[y][x]]++;
    //System.out.println(regionSize.length);
    Polygon[] polygon = new Polygon[regionCount];
    int polygonCnt = 0;
    for (int j = 0; j < regionSize.length; j++){
    //for (int j = 45; j < 46; j++){
        if(regionSize[j] > 100){
            int count = 0;
            for (int x = 0; x < width; x += 4){
                for (int y = 0; y < height; y += 4){
                    if(regionMap[y][x] == j){
                        count++;
                    }
                }
            }
            double[] xx = new double[count];
            double[] yy = new double[count];
            count = 0;
            for (int x = 0; x < width; x += 4){
                for (int y = 0; y < height; y += 4){
                    if(regionMap[y][x] == j){
                          xx[count] = x;
                          yy[count] = y;
                          count++;
                    }
                }
```

Εικόνα 3.43.1

```
        }
        if(xx.length > 3){
            Hull2.JarvisMarch jarvis = new Hull2.JarvisMarch();
            ArrayList convexList = new ArrayList();
            convexList = jarvis.forConvex(xx, yy);
            Graphics2D g = liveImage.createGraphics();
            //Graphics2D g = processedImage.createGraphics();
            g.setColor(Color.GREEN);
            int[] xpoints = new int[convexList.size()];
            int[] ypoints = new int[convexList.size()];
            polygon[polygonCnt] = new Polygon();
            for(int m = 0; m < convexList.size(); m++){
                Point polygonPoint = (Point)convexList.get(m);
                polygon[polygonCnt].addPoint(polygonPoint.x, polygonPoint.y);
                xpoints[m] = polygonPoint.x;
                ypoints[m] = polygonPoint.y;
            }
            if(polygon[polygonCnt] != null){
                Rectangle rect = new Rectangle(polygon[polygonCnt].getBounds());
                if(rect.width > 20 && rect.width < 800 && rect.height > 20 &&
                        rect.height < 800){
                    g.draw(polygon[polygonCnt]);
                    polygonCnt++;
                }else
                    polygon[polygonCnt] = null;
            }
        }
    }
}
```

Εικόνα 3.43.2

Μετά τον υπολογισμό όλων των πολυγώνων ενός χρώματος που μπορεί να υπάρχουν στην εικόνα που δέχεται επεξεργασία υπάρχει περίπτωση πολλά πολύγωνα να περιέχονται μέσα σε μεγαλύτερα ή κάποια άλλα να μπαίνουν σε άλλα. Σε αυτή την περίπτωση θα πρέπει αυτά τα πολύγωνα να συγχωνευτούν μεταξύ τους και να υπολογιστούν για τα καινούρια εκ νέου τα πολύγωνα **Convex Hull** έως ότου να μην συμβαίνει κάτι τέτοιο. Ο κώδικας που αναλαμβάνει να φέρει εις πέρας αυτήν τη διαδικασία παρουσιάζεται στις εικόνες 3.44.1 και 3.44.2.

```
boolean intersects = true;
while(intersects == true){
    intersects = false;
    for(int l = 0; l < polygon.length - 1; l++){
        for(int k = 1; k < polygon.length; k++){
            if(polygon[l] != null && polygon[k] != null && l != k){
                int[] secondPolyXPoints = polygon[k].xpoints;
                int[] secondPolyYPoints = polygon[k].ypoints;
                for(int m = 0; m < secondPolyXPoints.length; m++)
                    if(polygon[l].contains(secondPolyXPoints[m], secondPolyYPoints[m])){
                        intersects = true;
                        double[] xPointsMerged = new double[(polygon[l].xpoints.length +
                                polygon[k].xpoints.length)];
                        double[] yPointsMerged = new double[(polygon[l].ypoints.length +
                                polygon[k].ypoints.length)];
                        int cnt = 0;
                        for(int w = 0; w < polygon[l].xpoints.length; w++){
                            if(polygon[l].xpoints[w] != 0 && polygon[l].ypoints[w] != 0){
                                xPointsMerged[cnt] = polygon[l].xpoints[w];
                                yPointsMerged[cnt++] = polygon[l].ypoints[w];
                            }
                        }

                        for(int w = 0; w < polygon[k].xpoints.length; w++){
                            if(polygon[k].xpoints[w] != 0 && polygon[k].ypoints[w] != 0){
                                xPointsMerged[cnt] = polygon[k].xpoints[w];
                                yPointsMerged[cnt++] = polygon[k].ypoints[w];
                            }
                        }

                        double[] xPointsMergedCutOfZeros = new double[cnt];
                        double[] yPointsMergedCutOfZeros = new double[cnt];
                        for(int d = 0; d < cnt; d++){
                            xPointsMergedCutOfZeros[d] = xPointsMerged[d];
                            yPointsMergedCutOfZeros[d] = yPointsMerged[d];
                        }

                        Hull2.JarvisMarch jarvis = new Hull2.JarvisMarch();
                        ArrayList convexList = new ArrayList();
                        convexList = jarvis.forConvex(xPointsMergedCutOfZeros, yPointsMergedCutOfZeros);
                        int[] xpoints = new int[convexList.size()];
```

Εικόνα 3.44.1

```
                        int[] ypoints = new int[convexList.size()];
                        polygon[l] = new Polygon();
                        for(int z = 0; z < convexList.size(); z++){
                            Point polygonPoint = (Point)convexList.get(z);
                            polygon[l].addPoint(polygonPoint.x, polygonPoint.y);
                            xpoints[z] = polygonPoint.x;
                            ypoints[z] = polygonPoint.y;
                        }
                        Graphics2D g = liveImage.createGraphics();
                        g.setColor(Color.BLUE);
                        g.drawPolygon(polygon[l]);
                        polygon[k] = null;
                        break;
                    }
                }
            }
        }
    }
```

Εικόνα 3.44.2

Προτελευταία ενέργεια της συνάρτησης είναι η εκτύπωση στην εικόνα του πολυγώνου και του ονόματος του χρώματος που αφορά το πολύγωνο.

```
for(int x = 0; x < polygon.length; x++){
    if(polygon[x] != null){
        Rectangle rectangle = polygon[x].getBounds();
        int rectangleWidth = rectangle.width;
        int rectangleHeigth = rectangle.height;
        if(rectangleWidth > 0 && rectangleWidth < 640 && rectangleHeigth > 0 && rectangleHeigth < 480){
            double rectCenterX = rectangle.getCenterX();
            double rectCenterY = rectangle.getCenterY();
            Graphics2D g = liveImage.createGraphics();
            g.drawPolygon(polygon[x]);
            g.setColor(Color.RED);
            g.setColor(Color.CYAN);
            g.drawString(colorCase, (float)rectCenterX, (float)rectCenterY);
            g.setColor(Color.ORANGE);
            //g.draw(rectangle);
        }
    }
}
Graphics2D g = liveImage.createGraphics();
g.setColor(Color.RED);
int area = 0;
for(int l = 0; l < polygon.length; l++){
    if(polygon[l] != null){
        g.draw(polygon[l]);
        area++;
    }
}
```

εικόνα 3.45

Τέλος ως τελευταία ενέργεια έχει μείνει το να ενεργοποιηθεί η διαδικασία καταγραφής video για το ανά περίπτωση χρώμα ένα υπάρχει έστω μια περιοχή ενδιαφέροντος σε αυτό που επεξεργάζεται. Αλλιώς αν δεν υπάρχει καμία περιοχή ενδιαφέροντος για το εκάστοτε χρώμα ενώ η διαδικασία εγγραφής είναι σε λειτουργία αυτή θα πρέπει να σταματήσει. Παράδειγμα κώδικα που αναλαμβάνει αυτή την ενέργεια για το κόκκινο χρώμα παρουσιάζεται στις εικόνες 3.46 και 3.47

```
if(colorCase == "Red"){
    if(area > 0 && ColoredShapesTracker.getEnableRecord() == true){
        startRedSink();
    }else{
        stopRedSink();
    }
}else if(colorCase == "Orange"){
```

Εικόνα 3.46

```
/** Starts the data sink for the Red color. */
public static void startRedSink(){
    if(VideoStreamController.sinkColorValues.getSinkRed() == false){
        VideoStreamController.sinkColorValues.setSinkRed(true);
        VideoStreamController.redSinkData = new SinkData("Red");
    }
}
/** Stops the data sink for the Red color. */
public static void stopRedSink(){
    if(VideoStreamController.sinkColorValues.getSinkRed() == true){
        VideoStreamController.sinkColorValues.setSinkRed(false);
        VideoStreamController.redSinkData.stop("Red");
    }
}
```

Εικόνα 3.47

Μετά το τέλος της επεξεργασίας η κλάση **VideoStreamController** παίρνει από την **ImageProcessor** την επεξεργασμένη εικόνα με τα τυπωμένα σε αυτήν πολύγωνα ενδιαφέροντος που βρέθηκαν, αν βρέθηκαν, καλώντας την συνάρτηση **gerProcessedImage()** (εικόνα 3.48).

```
public BufferedImage getProcessedImage(){
    synchronized (processedImage){
        return processedImage;
    }
}
```

Εικόνα 3.48

Επίλογος

Αυτό λοιπόν είναι το manual της εφαρμογής. Ελπίζεται να βρεθεί χρήσιμο, αλλά και να βοηθήσει όσο το δυνατόν περισσότερο και τον απλό χειριστή στην χρήση της εφαρμογής, αλλά και έναν έμπειρο προγραμματιστή που θα θελήσει να επέμβει στον πηγαίο κώδικα ώστε να κάνει βελτιώσεις ή να τον χρησιμοποιήσει για την δημιουργία μιας παρεμφερή εφαρμογής.

Αξίζει να σημειωθεί ότι στο πακέτο της εφαρμογής παρέχονται, στο φάκελο «Documentation» το java documentation αυτής καθώς και στο φάκελο «Needed» όλα τα απαραίτητα προγράμματα είτε για την εκτέλεση αυτής, είτε για την επέμβαση στον κώδικα.

Βιβλιογραφία

- Prentice Hall - Core Java 2 Volume I - Fundamentals (7th Edition 2004).
- Prentice Hall - Core Java 2 Volume II - Advanced Features (7th Edition 2004).
- Java 6.0 Documentation. URL: http://java.sun.com/javase/6/docs/
- JMF Documentation. URL: http://java.sun.com/products/java-media/jmf/2.1.1/documentation.html
- Convex Hull for Dynamic Data -Jorge L. Vittes -August 20, 2002 (joint work with Umut Acar, and Guy Blelloch).
- Colour Tracker – Written by David – Thursday, 03 June 2004.
- R. Gonzalez and R. Woods, Digital Image Processing, 2nd edition, Prentice Hall, 2002.