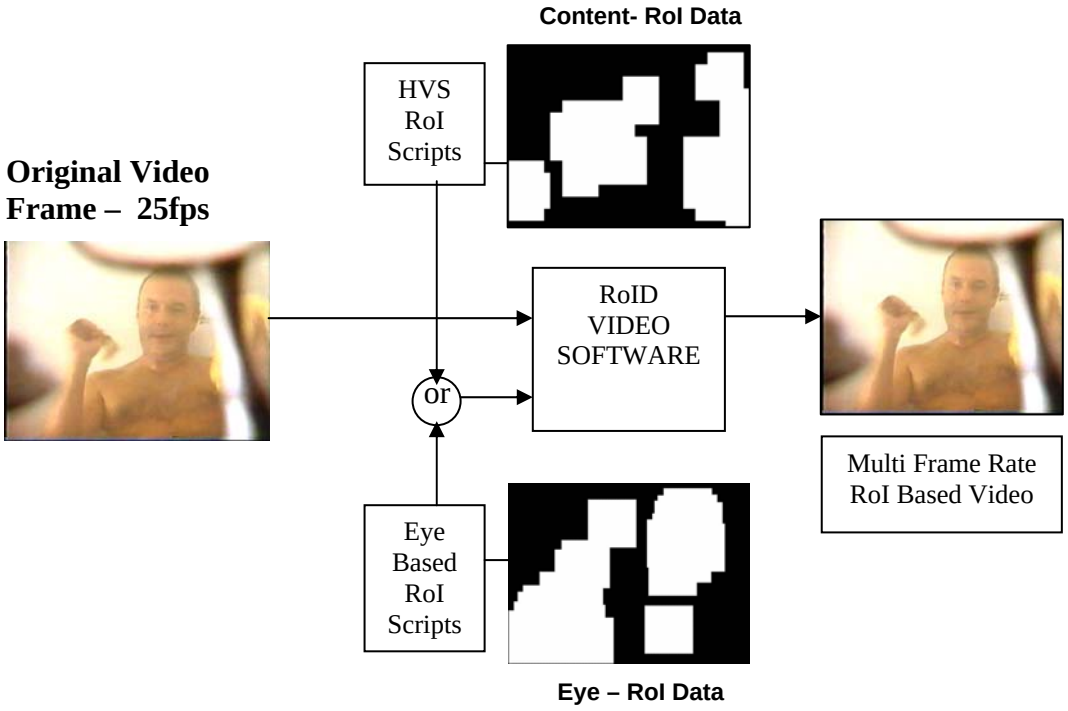
Appendix L

RoI Scripts



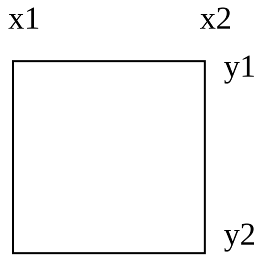
Process of adapting RoIDs from eye-based content-dependent data is dependent on RoI Data

The process of adapting RoIDs from eye-based content-dependent data is dependent on RoI Data. To ensure consistency RoI scripts were required. The following code was used to produce both content- and eye-based data RoI scripts.

RoI Script Format

A region-of-interest script starts with an (h) tag. To facilitate efficient use with a 5-frame buffer (thus enabling 5, 15 and 25fps), RoI scripts define information in 5 frame cycles. Consequently, subsequent lines define: i) the current frame number; ii) the background frame-rate and iii) the foreground frame-rate. Lines starting with a (C) tag define RoI squares extracted from participant eye-tracking data, with values (0-10000) in order

defining x1, y1, x2, y2 respectively (See Fig. 1). This file represents six RoID frames. In this example 36 RoI squares are defined, however (C) tags are not mandatory.



RoI square coordinate points.

```
h
0
5
25
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
c 0 0 0 0
f 1
c 5125 3190 5253 3318
c 5980 9922 6108 10050
c 6965 3038 7093 3166
c 7610 7265 7738 7393
c 6793 5147 6921 5275
c 4574 5566 4702 5694
c 6390 5191 6518 5319
c 7515 5471 7643 5599
c 8189 6308 8317 6436
c 5963 4119 6091 4247
c 6123 4732 6251 4860
c 6040 4683 6168 4811
c 5141 2972 5269 3100
c 6505 6981 6633 7109
c 6470 6557 6598 6685
c 7070 5828 7198 5956
c 6649 5767 6777 5895
c 6411 6019 6539 6147
c 3814 3362 3942 3490
c 5967 6391 6095 6519
c 7556 5135 7684 5263
c 7058 7133 7186 7261
c 6328 6001 6456 6129
c 6994 5745 7122 5873
c 5108 3301 5236 3429
c 6779 5940 6907 6068
c 5187 3862 5315 3990
c 2692 3770 2820 3898
c 7362 7352 7490 7480
c 4584 3838 4712 3966
c 6631 7846 6759 7974
c 4088 4485 4216 4613
c 4650 4852 4778 4980
c 5379 7369 5507 7497
c 5457 6669 5585 6797
f 2
c 5125 3190 5253 3318
c 5980 9922 6108 10050
c 6965 3038 7093 3166
c 7610 7265 7738 7393
c 6793 5147 6921 5275
c 4574 5566 4702 5694
c 6390 5191 6518 5319
c 7515 5471 7643 5599
c 8189 6308 8317 6436
c 5963 4119 6091 4247
c 6123 4732 6251 4860
c 6040 4683 6168 4811
c 5141 2972 5269 3100
c 6505 6981 6633 7109
c 6470 6557 6598 6685
c 7070 5828 7198 5956
c 6649 5767 6777 5895
c 6411 6019 6539 6147
c 3814 3362 3942 3490
c 5967 6391 6095 6519
c 7556 5135 7684 5263
c 7058 7133 7186 7261
c 6328 6001 6456 6129
c 6994 5745 7122 5873
c 5108 3301 5236 3429
c 6779 5940 6907 6068
c 5187 3862 5315 3990
c 2692 3770 2820 3898
c 7362 7352 7490 7480
c 4584 3838 4712 3966
c 6631 7846 6759 7974
```

```
c 4088 4485 4216 4613
c 4650 4852 4778 4980
c 5379 7369 5507 7497
c 5457 6669 5585 6797
f 3
c 5125 3190 5253 3318
c 5980 9922 6108 10050
c 6965 3038 7093 3166
c 7610 7265 7738 7393
c 6793 5147 6921 5275
c 4574 5566 4702 5694
c 6390 5191 6518 5319
c 7515 5471 7643 5599
c 8189 6308 8317 6436
c 5963 4119 6091 4247
c 6123 4732 6251 4860
c 6040 4683 6168 4811
c 5141 2972 5269 3100
c 6505 6981 6633 7109
c 6470 6557 6598 6685
c 7070 5828 7198 5956
c 6649 5767 6777 5895
c 6411 6019 6539 6147
c 3814 3362 3942 3490
c 5967 6391 6095 6519
c 7556 5135 7684 5263
c 7058 7133 7186 7261
c 6328 6001 6456 6129
c 6994 5745 7122 5873
c 5108 3301 5236 3429
c 6779 5940 6907 6068
c 5187 3862 5315 3990
c 2692 3770 2820 3898
c 7362 7352 7490 7480
c 4584 3838 4712 3966
c 6631 7846 6759 7974
c 4088 4485 4216 4613
c 4650 4852 4778 4980
c 5379 7369 5507 7497
c 5457 6669 5585 6797
f 4
c 5125 3190 5253 3318
c 5980 9922 6108 10050
c 6965 3038 7093 3166
c 7610 7265 7738 7393
c 6793 5147 6921 5275
c 4574 5566 4702 5694
c 6390 5191 6518 5319
c 7515 5471 7643 5599
c 8189 6308 8317 6436
c 5963 4119 6091 4247
c 6123 4732 6251 4860
c 6040 4683 6168 4811
c 5141 2972 5269 3100
c 6505 6981 6633 7109
c 6470 6557 6598 6685
c 7070 5828 7198 5956
c 6649 5767 6777 5895
c 6411 6019 6539 6147
c 3814 3362 3942 3490
c 5967 6391 6095 6519
c 7556 5135 7684 5263
c 7058 7133 7186 7261
c 6328 6001 6456 6129
c 6994 5745 7122 5873
c 5108 3301 5236 3429
c 6779 5940 6907 6068
c 5187 3862 5315 3990
c 2692 3770 2820 3898
c 7362 7352 7490 7480
c 4584 3838 4712 3966
c 6631 7846 6759 7974
c 4088 4485 4216 4613
c 4650 4852 4778 4980
c 5379 7369 5507 7497
c 5457 6669 5585 6797
5
5
25
f 0
c 5125 3190 5253 3318
c 5980 9922 6108 10050
c 6965 3038 7093 3166
c 7610 7265 7738 7393
c 6793 5147 6921 5275
c 4574 5566 4702 5694
c 6390 5191 6518 5319
c 7515 5471 7643 5599
c 8189 6308 8317 6436
c 5963 4119 6091 4247
c 6123 4732 6251 4860
c 6040 4683 6168 4811
c 5141 2972 5269 3100
c 6505 6981 6633 7109
c 6470 6557 6598 6685
c 7070 5828 7198 5956
c 6649 5767 6777 5895
c 6411 6019 6539 6147
c 3814 3362 3942 3490
c 5967 6391 6095 6519
c 7556 5135 7684 5263
c 7058 7133 7186 7261
c 6328 6001 6456 6129
c 6994 5745 7122 5873
c 5108 3301 5236 3429
c 6779 5940 6907 6068
c 5187 3862 5315 3990
c 2692 3770 2820 3898
c 7362 7352 7490 7480
c 4584 3838 4712 3966
c 6631 7846 6759 7974
c 4088 4485 4216 4613
c 4650 4852 4778 4980
c 5379 7369 5507 7497
c 5457 6669 5585 6797
```

Creating Eye-base RoI Scripts

This code is used to extract RoI scripts from eye-tracking data, which is contained in object orientated structure.

Package: definingroi

testScriptExtraction.java

```
package definingroi;
import java.util.*;
import java.io.*;
import java.awt.*;
```

```
class testScriptExtraction {

  public static void main(String[] args) {
// create new object structure
    topData top = new topData();
    loadData loading = new loadData();
    extractingData ext = new extractingData();
    top = loading.getTopDataList();
// load information from a data file
//Initial test varables
    String VIDEO = "BA";
//string variable - enter as filename
    int numberOfSamples = 36;
    int numberofFrames = 640;
    int internalFPS = 5;
    int externalFPS = 25;
    int frameShift = 0;

    while (VIDEO != "QUIT") {
/* loops until quit is assigned in QUIT variable */
//create dialogue box to enter video name
      Frame dialogInputFramevideo = new Frame();
      FileDialog outputFileNamevideo = new
FileDialog(dialogInputFramevideo,
          "Video Name");
      outputFileNamevideo.setVisible(true);
      do
        VIDEO = outputFileNamevideo.getFile();
      while (VIDEO == null);
      dialogInputFramevideo.dispose();
// close dialogue box
/* create dialogue box to allow the user to enter the
frame number  */
      Frame dialogInputFrameframes = new Frame();
      FileDialog outputFileNameframes = new
FileDialog(dialogInputFrameframes,
          "Number of Frames");
      outputFileNameframes.setVisible(true);
      numberofFrames =
Integer.parseInt(outputFileNameframes.getFile());
      dialogInputFrameframes.dispose();
// close dialogue box
/* create dialogue box to allow user to enter information
regarding details of internal frame rate */
      Frame dialogInputFrameinternal = new Frame();
      FileDialog outputFileNameinternal = new
FileDialog(
          dialogInputFrameinternal,
          "Internal");
      outputFileNameinternal.setVisible(true);
      internalFPS =
Integer.parseInt(outputFileNameinternal.getFile());
      dialogInputFrameinternal.dispose();
// close dialogue box
/* create dialogue box to allow user to enter information
regarding details of external frame rate */
      Frame dialogInputFrameexternal = new Frame();
      FileDialog outputFileNameexternal = new
FileDialog(
          dialogInputFrameexternal,
          "External");
      outputFileNameexternal.setVisible(true);
      externalFPS =
Integer.parseInt(outputFileNameexternal.getFile());
      dialogInputFrameexternal.dispose();
// close dialogue box
/* create dialogue box to allow user to enter information
regarding the required frame shift */
      Frame dialogInputFrameshift = new Frame();
      FileDialog outputFileNameshift = new FileDialog(
          dialogInputFrameshift,
          "FrameShift");
      outputFileNameshift.setVisible(true);
      frameShift =
Integer.parseInt(outputFileNameshift.getFile());
      dialogInputFrameshift.dispose();
// close dialogue box
      ArrayList foundVideo = new ArrayList();
      ArrayList experimentSearch = new ArrayList();
      experimentSearch = top.getExperimentList();
      Iterator experimentiterator =
experimentSearch.iterator();
// for all experiments in the object structure
      while (experimentiterator.hasNext()) {
        experiment nextexperiment = (experiment)
experimentiterator.next();
        ArrayList videoSearch = new ArrayList();
        videoSearch = nextexperiment.getVideoList();
        Iterator videoiterator = videoSearch.iterator();
        while (videoiterator.hasNext()) {
// for all videos in each experiment
          video nextVideo = (video) videoiterator.next();
          System.out.println(nextVideo.getVideoName());
          if
(nextVideo.getVideoName().startsWith(VIDEO)) {
            foundVideo.add(nextVideo);
            System.out.println("ADDED VIDEO TO
FOUND VIDEO: " +
nextVideo.getVideoDescription());
          }
        }
      }
/* create dialogue box to allow the user to define the
output of the RoI Script. */
      Frame dialogInputFrame = new Frame();
      FileDialog outputFileName = new
FileDialog(dialogInputFrame, "Defining ROI Script
File");
      outputFileName.setVisible(true);
      String selectedItem;
      do
        selectedItem = outputFileName.getDirectory() +
outputFileName.getFile();
// selecteditem defines the file name
      while (selectedItem == null);
      dialogInputFrame.dispose();
// close dialogue box
      System.out.println(selectedItem);
//create output buffer
      PrintWriter out;
      try {
        out = new PrintWriter(new
FileWriter(selectedItem));
      }
      catch (IOException ex) {
        throw new Error(ex.toString());
      }
/* something in here to extract the information from the
video */
//output started - out
      out.print("h");
      out.print("\n0");
      out.print("\n" + internalFPS);
      out.print("\n" + externalFPS);
//for frame 0 (all samples –  36)
      for (int count = 0; count < numberOfSamples;
count++) {
        out.print("\nc\t0\t0\t0\t0");
// print line to define the frame zero
      }
      for (int i = 1; i < (numberofFrames - frameShift);
i++) {// for all the frames
/* for the first of each five frames – define the frame
number and internal and external  frame number */
        if ( ( (i % 5) == 0) && (i > 0)) {
          out.print("\n" + i);
          out.print("\n" + internalFPS);
          out.print("\n" + externalFPS);
        }
/* define internal and external information in case a
change is required every five frames */
        out.print("\nf\t" + ( (i % 5)));
        for (int j = 0; j < foundVideo.size(); j++) {
```

```
// for all identified videos
        video nextVideo = ( (video) foundVideo.get(j));
        ArrayList rateSearch = new ArrayList();
        rateSearch = nextVideo.getRateList();
        Iterator rateiterator = rateSearch.iterator();
// for all identified frame rates
        while (rateiterator.hasNext()) {
          frameRate nextRate = (frameRate)
rateiterator.next();
          ArrayList frameSearch = new ArrayList();
          frameSearch = nextRate.getFrameList();
          Iterator frameiterator = frameSearch.iterator();
// for all frames in the object structure
          while (frameiterator.hasNext()) {
            videoFrame nextFrame = (videoFrame)
frameiterator.next();
            if (nextFrame.getFrameNumber() == (i +
frameShift)) {
              ArrayList coordinateSearch = new ArrayList();
              coordinateSearch =
nextFrame.getCoordinateList();
              Iterator coordinateiterator =
coordinateSearch.iterator();
// for all coordinate values in the object structure
              while (coordinateiterator.hasNext()) {
                xyCoordinates nextCoordinates =
(xyCoordinates)
                  coordinateiterator.next();
                out.print("\nc\t" +
(nextCoordinates.getXCoordinate() - 64)
                  + "\t" +
(nextCoordinates.getYCoordinate() - 64)
                  + "\t" +
(nextCoordinates.getXCoordinate() + 64)
                  + "\t" +
(nextCoordinates.getYCoordinate() + 64));
/* print the coordinate square */
              }
            }
          }
        }
      }
      out.close();
// close output file
    }
  }
}
```

Extracting RoI Primative

This code is used to determine video-content significant RoI as a result of edge, movement and colour contrast. Many sections of code, particularly in the extractData class, is duplicated for each RoI aspect (edge, movement and colour contrast).

Package: videoroi

freshData.java

```
/* convert the information into consistent values and so
that the frames are printed and managed in the correct
order. */

package videoroi;
import java.util.*;
import java.io.*;
import java.awt.*;
import java.lang.Object.*;

class freshData {
  int xCoord1 = 0;
  int yCoord1 = 0;
  int framevalue = 0;

  public freshData(int currentFrame, int x1, int y1) {
    xCoord1 = x1;
    yCoord1 = y1;
     framevalue = currentFrame;
/*  constructor assigns values to xCoord, yCoord and
fame number */
   }

  public int getCurrentFrame() {
    return framevalue;
// returns int – frame number
  }

  public int getx1() {
    return xCoord1;
//return int – xCoord1
  }

  public int gety1() {
    return yCoord1;
//return int – yCoord1
  }

  public double SortKey() {
    String Tempframevalue = "";
    String TempxCoord1 = "";
    String TempyCoord1 = "";
    if (framevalue < 1000) {
      if (framevalue < 100) {
        if (framevalue < 10) {
/* frame number is converted to a consistent string
format */
          Tempframevalue = "000" + framevalue;
        } else {
          Tempframevalue = "00" + framevalue;
        } } else {
        Tempframevalue = "0" + framevalue;
      } } else {
      Tempframevalue = "" + framevalue;
    }
/* xCoord is converted to a consistent string format */
    if (xCoord1 < 1000) {
      if (xCoord1 < 100) {
        if (xCoord1 < 10) {
          TempxCoord1 = "000" + xCoord1;
        } else {
          TempxCoord1 = "00" + xCoord1;
        } } else {
        TempxCoord1 = "0" + xCoord1;
      } } else {
      TempxCoord1 = "" + xCoord1;
    }
/* yCoord is converted to a consistent string format */
    if (yCoord1 < 1000) {
      if (yCoord1 < 100) {
        if (yCoord1 < 10) {
          TempyCoord1 = "000" + yCoord1;
        } else {
```

```
        TempyCoord1 = "00" + yCoord1;
       } } else {
       TempyCoord1 = "0" + yCoord1;
      } } else {
      TempyCoord1 = "" + yCoord1;
    }
    double Tempdouble = Double.parseDouble("1" +
Tempframevalue + TempxCoord1 +
                    TempyCoord1);
/* a number is created that is consist for all -
independent of frame number */
    return Tempdouble;
  }
}
```

extractData.java

```
/* extractData extracts the mean and SD for video and
frame and makes a comparison with frame squares. Once
this has been done a RoI script is created and written to
file. */

package videoroi;

import java.util.*;
import javax.swing.*;
import java.io.*;
import java.awt.*;
import java.lang.Object;
import java.awt.event.*;

public class extractData {

  public static void main(String[] args) {
    int LengthOfNumber = 3;
    int boxsize = 32;
    int frameShift = 0;
//load information from file input Vector
    String inputString = "";
/* create new dialogue box –  to allow the user to define
the location of the EDGE RoI Script */
    Frame dialogInputFrame = new Frame();
    FileDialog inputFileName = new
FileDialog(dialogInputFrame,
        "Complete EDGE Script: Mean and SD");
    inputFileName.setVisible(true);
    String selectedItem;
    do
      selectedItem = inputFileName.getDirectory() +
inputFileName.getFile();
    while (selectedItem == null);
// repeat until a item has been selected
    dialogInputFrame.dispose();
// close dialogue box
    Vector inputMeanVector = new Vector();
// create a new InputVector and input text data
    try {
      FileReader inputFile = new FileReader(selectedItem);
      BufferedReader reader = new
BufferedReader(inputFile);
      String bufferInput;
      while ( (bufferInput = reader.readLine()) != null)
        inputMeanVector.add(bufferInput);
    }
    catch (Exception e) {
      System.err.println(e.getMessage());
    }
/* The sum defines the MEAN RGB and SD RGB*/
    double SUMMeanR = 0;
    double SUMMeanG = 0;
    double SUMMeanB = 0;
    double SUMSDR = 0;
    double SUMSDG = 0;
    double SUMSDB = 0;
//values to store sun of mean and sd values
    int count = 0;
    int numOfLines = inputMeanVector.size();
    ArrayList refreshData = new ArrayList();
/* we need to identify the middle value and SD for the
entire video (condition 1) – this is required to test if the
area of a video frame is outside the mean across the
whole video. */
    for (int lineNum = 1; lineNum < (numOfLines - 3);
lineNum++) {
/* for all frame in the input file - minus three excludes
the summary at the end of each file. As always a space
before data */
      inputString = (String)
inputMeanVector.get(lineNum);
// get and print the line
      System.out.println(inputString);
      String FrameName = "";
      double MeanR = 0;
      double MeanG = 0;
      double MeanB = 0;
      double SDR = 0;
      double SDG = 0;
      double SDB = 0;
      String rubbishToken;
// to remove the unwanted tokens
      String currentLine = (String)
inputMeanVector.get(lineNum);
      StringTokenizer lineTokenizer = new
StringTokenizer(currentLine, "\t");
//Start extracting the data from the file
      if (lineTokenizer.hasMoreTokens()) {
        FrameName = lineTokenizer.nextToken();
      }
      if (lineTokenizer.hasMoreTokens()) {
        try {
          MeanR =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract mean edge red value
        }
        catch (NumberFormatException e) {
          System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
        }
      }
      if (lineTokenizer.hasMoreTokens()) {
        try {
          MeanG =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract mean green value
        }
        catch (NumberFormatException e) {
          System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
        }
      }
      if (lineTokenizer.hasMoreTokens()) {
        try {
          MeanB =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract mean blue value
        }
        catch (NumberFormatException e) {
          System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
        }
      }
```

```
    for (int i = 0; i <= 5; i++) {
     if (lineTokenizer.hasMoreTokens()) {
      rubbishToken = lineTokenizer.nextToken();
     }
    }
//ignores no relevant information
    if (lineTokenizer.hasMoreTokens()) {
     try {
      SDR =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract red standard deviation pixel value
     }
     catch (NumberFormatException e) {
      System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
     }
    }
    if (lineTokenizer.hasMoreTokens()) {
     try {
      SDG =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract green standard deviation pixel value
     }
     catch (NumberFormatException e) {
      System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
     }
    }
    if (lineTokenizer.hasMoreTokens()) {
     try {
      SDB =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract blue standard deviation pixel value
     }
     catch (NumberFormatException e) {
      System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
     }
    }
    System.out.println(MeanR + " " + MeanG + " " +
MeanB + " " + SDR + " " + SDG + " " + SDB + " ");
// print on screen
// Sum values
    SUMMeanR = SUMMeanR + MeanR;
    SUMMeanG = SUMMeanG + MeanG;
    SUMMeanB = SUMMeanB + MeanB;
    SUMSDR = SUMSDR + SDR;
    SUMSDG = SUMSDG + SDG;
    SUMSDB = SUMSDB + SDB;
    count++;
   }
   SUMMeanR = SUMMeanR / count;
   SUMMeanG = SUMMeanG / count;
   SUMMeanB = SUMMeanB / count;
   SUMSDR = SUMSDR / count;
   SUMSDG = SUMSDG / count;
   SUMSDB = SUMSDB / count;
   System.out.println("MID R: " + SUMMeanR);
   System.out.println("MID G: " + SUMMeanG);
   System.out.println("MID B: " + SUMMeanB);
   System.out.println("MID SDR: " + SUMSDR);
   System.out.println("MID SDG: " + SUMSDG);
   System.out.println("MID SDB: " + SUMSDB);

/* Provide the mean value and SD for edges in the entire
video (condition 1) – this is required to test if the area of
a video frame is outside the mean across the whole
video. We next need to find the mean value and SD for
edges in each frame (condition 2) – this is required to
test if the area of a video frame is outside the mean
across for a specific frame. */

   for (int lineNum = 1; lineNum < (numOfLines - 3);
lineNum++) {
/* for all frames in the video - minus three to exclude the
summary at the end of each file. */

/* The following extracts frame values – condition 2 */
    inputString = (String)
inputMeanVector.get(lineNum);
    System.out.println(inputString);
    String FrameName = "";
    double MeanR = 0;
    double MeanG = 0;
    double MeanB = 0;
    double SDR = 0;
    double SDG = 0;
    double SDB = 0;
    String rubbishToken;
/* rubbishTokens is used to remove the unwanted
tokens */
    String currentLine = (String)
inputMeanVector.get(lineNum);
    StringTokenizer lineTokenizer = new
StringTokenizer(currentLine, "\t");
//Start extracting the data from the file
    if (lineTokenizer.hasMoreTokens()) {
     FrameName = lineTokenizer.nextToken();
    }
    if (lineTokenizer.hasMoreTokens()) {
     try {
      MeanR =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract edge mean red value
     }
     catch (NumberFormatException e) {
      System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
     }
    }
    if (lineTokenizer.hasMoreTokens()) {
     try {
      MeanG =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract edge mean green value
     }
     catch (NumberFormatException e) {
      System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
     }
    }
    if (lineTokenizer.hasMoreTokens()) {
     try {
      MeanB =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract edge mean green value
     }
     catch (NumberFormatException e) {
      System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
     }
    }
    for (int i = 0; i <= 5; i++) {
     if (lineTokenizer.hasMoreTokens()) {
      rubbishToken = lineTokenizer.nextToken();
     }
    }
//ignores no relevant information
    if (lineTokenizer.hasMoreTokens()) {
     try {
      SDR =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract red standard deviation pixel value
     }
     catch (NumberFormatException e) {
      System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
```

```
      }
    }
    if (lineTokenizer.hasMoreTokens()) {
      try {
       SDG =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract green standard deviation pixel value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
    }
    if (lineTokenizer.hasMoreTokens()) {
      try {
       SDB =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract blue standard deviation pixel value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
    }
    System.out.println("SECOND CONDITION: " +
FrameName + " " + MeanR + " " + MeanG + " " +
MeanB + " " + SDR + " " + SDG + " " + SDB + " ");
//Passed first set of numbers
    String RootDirectory = (FrameName.substring(0,
2));
    int FrameNumber =
(Integer.parseInt(FrameName.substring(2, 5)) + 1);
/* Each frame is split into 300 squares. Therefore we
need to identify the correct text file */
    String EdgeFile = inputFileName.getDirectory() +
RootDirectory +
       "\\" + RootDirectory + "EDSQSUM" +
FrameNumber +
       ".txt";
// Outputs the EDGE and FILE NAMES
    String inputEdgeString = "";
    Vector inputEdgeVector = new Vector();
    try {
/* Transfer all information in the correct text file into
the inputVector. */
      FileReader inputFile = new FileReader(EdgeFile);
      BufferedReader reader = new
BufferedReader(inputFile);
      String bufferInput;
      while ( (bufferInput = reader.readLine()) != null)
       inputEdgeVector.add(bufferInput);
    }
    catch (Exception e) {
      System.err.println(e.getMessage());
    }
    for (int locallineNum = 1; locallineNum <
(inputEdgeVector.size());
        locallineNum++) {
/* For all lines in the input file - minus three that
excludes the summary at the end of each file (represent
information concerning the 300 distribution squares).
Extract information concerning mean and standard
deviation of distribution squares. Following define
distribution square variables. */
      inputString = (String)
inputEdgeVector.get(locallineNum);
      //System.out.println(inputString);
      String localFrameName = "";
      double localMeanR = 0;
      double localMeanG = 0;
      double localMeanB = 0;
      String localcurrentLine = (String)
inputEdgeVector.get(locallineNum);
      StringTokenizer locallineTokenizer = new
StringTokenizer(
          localcurrentLine, "\t");
//Start extracting the data from the file
      if (locallineTokenizer.hasMoreTokens()) {
       localFrameName =
locallineTokenizer.nextToken();
      }
      if (locallineTokenizer.hasMoreTokens()) {
       try {
        localMeanR =
Double.valueOf(locallineTokenizer.nextToken()).
          doubleValue();
// extract distribution square mean red value
       }
       catch (NumberFormatException e) {
        System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
       }
      }
      if (locallineTokenizer.hasMoreTokens()) {
       try {
        localMeanG =
Double.valueOf(locallineTokenizer.nextToken()).
          doubleValue();
// extract distribution square mean green value
       }
       catch (NumberFormatException e) {
        System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
       }
      }
      if (locallineTokenizer.hasMoreTokens()) {
       try {
        localMeanB =
Double.valueOf(locallineTokenizer.nextToken()).
          doubleValue();
       }
// extract distribution square mean blue value
       catch (NumberFormatException e) {
        System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
       }
      }
//determine x and y coordinates from local file name
/* The following must consider the fact that the frame
number can be represented as one, two, three or four
characters. */
      String XYCoordinate =
(localFrameName.substring(6, 17));
      int XCoord = 0;
      int YCoord = 0;
      if (XYCoordinate.substring(2,
3).equalsIgnoreCase("_")) {
       XCoord =
Integer.parseInt(XYCoordinate.substring(0, 2));
       if (XYCoordinate.substring(5,
6).equalsIgnoreCase("_")) {
        YCoord =
Integer.parseInt(XYCoordinate.substring(3, 5));
       }
       else {
        YCoord =
Integer.parseInt(XYCoordinate.substring(3, 6));
       }
      }
      else if (XYCoordinate.substring(3,
4).equalsIgnoreCase("_")) {
       XCoord =
Integer.parseInt(XYCoordinate.substring(0, 3));
       if (XYCoordinate.substring(6,
7).equalsIgnoreCase("_")) {
        YCoord =
Integer.parseInt(XYCoordinate.substring(4, 6));
       }
       else {
        YCoord =
Integer.parseInt(XYCoordinate.substring(4, 7));
       }
```

```
      }
      else if (XYCoordinate.substring(4,
5).equalsIgnoreCase("_")) {
        XCoord =
Integer.parseInt(XYCoordinate.substring(0, 4));
        if (XYCoordinate.substring(7,
8).equalsIgnoreCase("_")) {
          YCoord =
Integer.parseInt(XYCoordinate.substring(5, 7));
        }
        else if (XYCoordinate.substring(8,
9).equalsIgnoreCase("_")) {
          YCoord =
Integer.parseInt(XYCoordinate.substring(5, 8));
        }
        else {
          YCoord =
Integer.parseInt(XYCoordinate.substring(5, 9));
        }
      }
/*CONDITIONS TO INCLUDE OR NOT
INCLUDE – Adds a value to the Region of Interest
script is the local value is one SD greater or less than
either the frame or video SD (see chapter 5). Red, green
and blue comparisons are identical so no repeated
comments will be given. */
      boolean addtoROIScript = false;
//RED TOTAL
      if (localMeanR > (SUMMeanR + SUMSDR)) {
        addtoROIScript = true;
      }
      if (localMeanR < (SUMMeanR - SUMSDR)) {
        addtoROIScript = true;
      }
//RED LOCAL
      if (localMeanR > (MeanR + SDR)) {
        addtoROIScript = true;
      }
      if (localMeanR < (MeanR - SDR)) {
        addtoROIScript = true;
      }
//GREEN TOTAL
      if (localMeanG > (SUMMeanG + SUMSDG)) {
        addtoROIScript = true;
      }
      if (localMeanG < (SUMMeanG - SUMSDG)) {
        addtoROIScript = true;
      }
//GREEN LOCAL
      if (localMeanG > (MeanG + SDG)) {
        addtoROIScript = true;
      }
      if (localMeanG < (MeanG - SDG)) {
        addtoROIScript = true;
      }
//BLUE TOTAL
      if (localMeanB > (SUMMeanB + SUMSDB)) {
        addtoROIScript = true;
      }
      if (localMeanB < (SUMMeanB - SUMSDB)) {
        addtoROIScript = true;
      }
//BLUE LOCAL
      if (localMeanB > (MeanB + SDB)) {
        addtoROIScript = true;
      }
      if (localMeanB < (MeanB - SDB)) {
        addtoROIScript = true;
      }
// determine whether the output to be added or not
      if (addtoROIScript) {
        freshData localRefreshData = new
freshData(FrameNumber, XCoord, YCoord );
/* Create a new freshData Object and add to ArrayList -
as the information is sent to freshData the number will
also naturally order the frames. The following code adds
the new (fresh) data to the correct position in the
arraylist . */
        if (refreshData.isEmpty() == true) {
          refreshData.add(localRefreshData);
        }
        else if (refreshData.size() < 2) {
          freshData freshData1 = (freshData)
refreshData.get(0);
          if ( (localRefreshData.SortKey()) <=
(freshData1.SortKey())) {
            refreshData.add(0, localRefreshData);
          }
          else {
            refreshData.add(localRefreshData);
          }
        }
        else {
          int iterateVariable = 0;
          int nextRefreshData = iterateVariable + 1;
          int lastRefreshData = refreshData.size() - 1;
          boolean insert = false;
          while (iterateVariable < refreshData.size() &&
!insert) {
            freshData refreshData1 = (freshData)
refreshData.get(
                iterateVariable);
            freshData refreshData2 = (freshData)
refreshData.get(
                nextRefreshData);
            freshData refreshData3 = (freshData)
refreshData.get(
                lastRefreshData);
            if ( (iterateVariable == 0) &&
                (localRefreshData.SortKey() <
refreshData1.SortKey())) {
              refreshData.add(0, localRefreshData);
              insert = true;
            }
            else if (localRefreshData.SortKey() <
refreshData1.SortKey()) {
              refreshData.add(iterateVariable,
localRefreshData);
              insert = true;
            }
            else if (localRefreshData.SortKey() ==
refreshData1.SortKey()) {
              refreshData.add(iterateVariable,
localRefreshData);
              insert = true;
            }
            else if ( (localRefreshData.SortKey() >
refreshData1.SortKey()) &&
                  (iterateVariable == refreshData.size())) {
              refreshData.add(localRefreshData);
              insert = true;
            }
            else if ( (localRefreshData.SortKey() >
refreshData1.SortKey()) &&
                  (localRefreshData.SortKey() <=
refreshData2.SortKey())) {
              refreshData.add(nextRefreshData,
localRefreshData);
              insert = true;
            }
            else if (localRefreshData.SortKey() >
refreshData3.SortKey()) {
              refreshData.add(localRefreshData);
              insert = true;
            }
            else
              iterateVariable++;
          }
        }
/* New data added to the arraylist */
        System.out.println("ADDED: " +
localRefreshData.getCurrentFrame()+ "_" +
```

```
localRefreshData.getx1()+ "_" +
localRefreshData.gety1()+ "_EDGE");
      }
     }
   }
//load information from file into input Vector
   inputString = "";
/* create new dialogue box –  to allow the user to define
the location of the movement RoI Script */
   dialogInputFrame = new Frame();
   FileDialog inputFileNameMOV = new
FileDialog(dialogInputFrame,
      "Complete MOV Script: Mean and SD");
   inputFileNameMOV.setVisible(true);
   do
    selectedItem = inputFileNameMOV.getDirectory()
+ inputFileNameMOV.getFile();
   while (selectedItem == null);
// repeat until a item has been selected
   dialogInputFrame.dispose();
// close dialogue box
   inputMeanVector = new Vector();
// create a new InputVector and input text data
   try {
    FileReader inputFile = new FileReader(selectedItem);
    BufferedReader reader = new
BufferedReader(inputFile);
     String bufferInput;
     while ( (bufferInput = reader.readLine()) != null)
      inputMeanVector.add(bufferInput);
   }
   catch (Exception e) {
    System.err.println(e.getMessage());
   }
/* The sum defines the MEAN RGB and SD RGB*/
   SUMMeanR = 0;
   SUMMeanG = 0;
   SUMMeanB = 0;
   SUMSDR = 0;
   SUMSDG = 0;
   SUMSDB = 0;
//values to store sun of mean and sd values
   count = 0;
   numOfLines = inputMeanVector.size();
/* we need to identify the middle value and SD for the
entire video (condition 1) – this is required to test if the
area of a video frame is outside the mean across the
whole video. */
   for (int lineNum = 1; lineNum < (numOfLines - 3);
lineNum++) {
/* for all frame in the input file - minus three excludes
the summary at the end of each file. As always a space
before data */
     inputString = (String)
inputMeanVector.get(lineNum);
// get and print the line
     System.out.println(inputString);
     String FrameName = "";
     double MeanR = 0;
     double MeanG = 0;
     double MeanB = 0;
     double SDR = 0;
     double SDG = 0;
     double SDB = 0;
     String rubbishToken;
// to remove the unwanted tokens
     String currentLine = (String)
inputMeanVector.get(lineNum);
     StringTokenizer lineTokenizer = new
StringTokenizer(currentLine, "\t");
//Start extracting the data from the file
     if (lineTokenizer.hasMoreTokens()) {
      FrameName = lineTokenizer.nextToken();
     }
     if (lineTokenizer.hasMoreTokens()) {
      try {
       MeanR =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract mean edge red value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
     if (lineTokenizer.hasMoreTokens()) {
      try {
       MeanG =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract mean green value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
     if (lineTokenizer.hasMoreTokens()) {
      try {
       MeanB =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract mean blue value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
     for (int i = 0; i <= 5; i++) {
      if (lineTokenizer.hasMoreTokens()) {
       rubbishToken = lineTokenizer.nextToken();
      }
     }
//ignores no relevant information
     if (lineTokenizer.hasMoreTokens()) {
      try {
       SDR =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract red standard deviation pixel value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
     if (lineTokenizer.hasMoreTokens()) {
      try {
       SDG =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract green standard deviation pixel value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
     if (lineTokenizer.hasMoreTokens()) {
      try {
       SDB =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract blue standard deviation pixel value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
```

```
    System.out.println(MeanR + " " + MeanG + " " +
MeanB + " " + SDR + " " +
                SDG + " " + SDB + " ");
// print on screen
// Sum values
    SUMMeanR = SUMMeanR + MeanR;
    SUMMeanG = SUMMeanG + MeanG;
    SUMMeanB = SUMMeanB + MeanB;
    SUMSDR = SUMSDR + SDR;
    SUMSDG = SUMSDG + SDG;
    SUMSDB = SUMSDB + SDB;
    count++;
   }
   SUMMeanR = SUMMeanR / count;
   SUMMeanG = SUMMeanG / count;
   SUMMeanB = SUMMeanB / count;
   SUMSDR = SUMSDR / count;
   SUMSDG = SUMSDG / count;
   SUMSDB = SUMSDB / count;
   System.out.println("MID R: " + SUMMeanR);
   System.out.println("MID G: " + SUMMeanG);
   System.out.println("MID B: " + SUMMeanB);
   System.out.println("MID SDR: " + SUMSDR);
   System.out.println("MID SDG: " + SUMSDG);
   System.out.println("MID SDB: " + SUMSDB);
/* Provide the mean value and SD for movement in the
entire video (condition 1) – this is required to test if the
area of a video frame is outside the mean across the
whole video. We next need to find the mean value and
SD for movement in each frame (condition 2) – this is
required to test if the area of a video frame is outside the
mean across for a specific frame. */
   for (int lineNum = 1; lineNum < (numOfLines - 3);
lineNum++) {
/* for all frames in the video - minus three to exclude the
summary at the end of each file. */

/* The following extracts frame values – condition 2 */
    inputString = (String)
inputMeanVector.get(lineNum);
    System.out.println(inputString);
    String FrameName = "";
    double MeanR = 0;
    double MeanG = 0;
    double MeanB = 0;
    double SDR = 0;
    double SDG = 0;
    double SDB = 0;
    String rubbishToken;
/* rubbishTokens is used to remove the unwanted
tokens */
    String currentLine = (String)
inputMeanVector.get(lineNum);
    StringTokenizer lineTokenizer = new
StringTokenizer(currentLine, "\t");
//Start extracting the data from the file
    if (lineTokenizer.hasMoreTokens()) {
     FrameName = lineTokenizer.nextToken();
    }
    if (lineTokenizer.hasMoreTokens()) {
     try {
      MeanR =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
(); // extract edge mean red value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
    }
    if (lineTokenizer.hasMoreTokens()) {
     try {
      MeanG =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract edge mean green value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
    }
    if (lineTokenizer.hasMoreTokens()) {
     try {
      MeanB =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract edge mean green value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
    }
    for (int i = 0; i <= 5; i++) {
     if (lineTokenizer.hasMoreTokens()) {
      rubbishToken = lineTokenizer.nextToken();
     }
    }
//ignores no relevant information
    if (lineTokenizer.hasMoreTokens()) {
     try {
      SDR =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract red standard deviation pixel value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
    }
    if (lineTokenizer.hasMoreTokens()) {
     try {
      SDG =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract green standard deviation pixel value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
    }
    if (lineTokenizer.hasMoreTokens()) {
     try {
      SDB =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract blue standard deviation pixel value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); / in case of error
      }
    }
    System.out.println("SECOND CONDITION: " +
FrameName + " " + MeanR + " " + MeanG + " " +
MeanB + " " + SDR + " " + SDG + " " + SDB + " ");
//Passed first set of numbers
    String RootDirectory = (FrameName.substring(0,
2));
/*IDENTIFYING THE MOV FRAME NUMBER IS
NOT SOO EASY ! */
    String TempFrameNumberString =
(FrameName.substring(7, 11));
    int TempFrameNumber = 0;
    if (TempFrameNumberString.substring(1,
2).equalsIgnoreCase("_")) {
     TempFrameNumber =
Integer.parseInt(TempFrameNumberString.substring(0,
1));
    }
```

```
    else if (TempFrameNumberString.substring(2,
3).equalsIgnoreCase("_")) {
     TempFrameNumber =
Integer.parseInt(TempFrameNumberString.substring(0,
2));
    }
    else if (TempFrameNumberString.substring(3,
4).equalsIgnoreCase("_")) {
     TempFrameNumber =
Integer.parseInt(TempFrameNumberString.substring(0,
3));
    }
    else if (TempFrameNumberString.substring(4,
5).equalsIgnoreCase("_")) {
     TempFrameNumber =
Integer.parseInt(TempFrameNumberString.substring(0,
4));
    }
    int FrameNumber = TempFrameNumber + 1;
/* Each frame is split into 300 squares. Therefore we
need to identify the correct text file */
    String MovFile =
inputFileNameMOV.getDirectory() + RootDirectory +
"\\" + RootDirectory + "MVSQSUM" +
FrameNumber + ".txt";
// outputs the MOV and FILE NAMES
    String inputMovString = "";
    Vector inputMovVector = new Vector();
    try {
/* Transfer all information in the correct text file into
the inputVector. */
     FileReader inputFile = new FileReader(MovFile);
     BufferedReader reader = new
BufferedReader(inputFile);
     String bufferInput;
     while ( (bufferInput = reader.readLine()) != null)
      inputMovVector.add(bufferInput);
    }
    catch (Exception e) {
     System.err.println(e.getMessage());
    }
    for (int locallineNum = 1; locallineNum <
(inputMovVector.size());
        locallineNum++) {
/* For all lines in the input file - minus three that
excludes the summary at the end of each file (represent
information concerning the 300 distribution squares).
Extract information concerning mean and standard
deviation of distribution squares. Following define
distribution square variables. */
     inputMovString = (String)
inputMovVector.get(locallineNum);
     //System.out.println(inputMovString);
     String localFrameName = "";
     double localMeanR = 0;
     double localMeanG = 0;
     double localMeanB = 0;
     String localcurrentLine = (String)
inputMovVector.get(locallineNum);
     StringTokenizer locallineTokenizer = new
StringTokenizer(
        localcurrentLine, "\t");
//Start extracting the data from the file
     if (locallineTokenizer.hasMoreTokens()) {
      localFrameName =
locallineTokenizer.nextToken();
     }
     if (locallineTokenizer.hasMoreTokens()) {
      try {
       localMeanR =
Double.valueOf(locallineTokenizer.nextToken()).
          doubleValue();
// extract distribution square mean red value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
     if (locallineTokenizer.hasMoreTokens()) {
      try {
       localMeanG =
Double.valueOf(locallineTokenizer.nextToken()).
          doubleValue();
// extract distribution square mean green value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
     if (locallineTokenizer.hasMoreTokens()) {
      try {
       localMeanB =
Double.valueOf(locallineTokenizer.nextToken()).
          doubleValue();
// extract distribution square mean blue value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
//determine x and y coordinates from local file name
/* The following must consider the fact that the frame
number can be represented as one, two, three or four
characters. */
     String TempShift = (FrameNumber - 1) + "_" +
FrameNumber + "_";
     int ShiftNumber = TempShift.length();
//determine x and y coordinates from local file name
     String XYCoordinate =
(localFrameName.substring( (7 + ShiftNumber),
        (18 + ShiftNumber)));
     int XCoord = 0;
     int YCoord = 0;
     if (XYCoordinate.substring(2,
3).equalsIgnoreCase("_")) {
      XCoord =
Integer.parseInt(XYCoordinate.substring(0, 2));
      if (XYCoordinate.substring(5,
6).equalsIgnoreCase("_")) {
       YCoord =
Integer.parseInt(XYCoordinate.substring(3, 5));
      }
      else {
       YCoord =
Integer.parseInt(XYCoordinate.substring(3, 6));
      }
     }
     else if (XYCoordinate.substring(3,
4).equalsIgnoreCase("_")) {
      XCoord =
Integer.parseInt(XYCoordinate.substring(0, 3));
      if (XYCoordinate.substring(6,
7).equalsIgnoreCase("_")) {
       YCoord =
Integer.parseInt(XYCoordinate.substring(4, 6));
      }
      else {
       YCoord =
Integer.parseInt(XYCoordinate.substring(4, 7));
      }
     }
     else if (XYCoordinate.substring(4,
5).equalsIgnoreCase("_")) {
      XCoord =
Integer.parseInt(XYCoordinate.substring(0, 4));
      if (XYCoordinate.substring(7,
8).equalsIgnoreCase("_")) {
       YCoord =
Integer.parseInt(XYCoordinate.substring(5, 7));
      }
```

```
        else if (XYCoordinate.substring(8,
9).equalsIgnoreCase("_")) {
          YCoord =
Integer.parseInt(XYCoordinate.substring(5, 8));
        }
        else {
          YCoord =
Integer.parseInt(XYCoordinate.substring(5, 9));
        }
      }
/*CONDITIONS TO INCLUDE OR NOT INCLUDE – Adds a value to the Region of Interest script is the local value is one SD greater or less than either the frame or video SD (see chapter 5). Red, green and blue comparisons are identical so no repeated comments will be given. */
      boolean addtoROIScript = false;
//RED TOTAL
      if (localMeanR > (SUMMeanR + SUMSDR)) {
        addtoROIScript = true;
      }
      if (localMeanR < (SUMMeanR - SUMSDR)) {
        addtoROIScript = true;
      }
//RED LOCAL
      if (localMeanR > (MeanR + SDR)) {
        addtoROIScript = true;
      }
      if (localMeanR < (MeanR - SDR)) {
        addtoROIScript = true;
      }
//GREEN TOTAL
      if (localMeanG > (SUMMeanG + SUMSDG)) {
        addtoROIScript = true;
      }
      if (localMeanG < (SUMMeanG - SUMSDG)) {
        addtoROIScript = true;
      }
//GREEN LOCAL
      if (localMeanG > (MeanG + SDG)) {
        addtoROIScript = true;
      }
      if (localMeanG < (MeanG - SDG)) {
        addtoROIScript = true;
      }
//BLUE TOTAL
      if (localMeanB > (SUMMeanB + SUMSDB)) {
        addtoROIScript = true;
      }
      if (localMeanB < (SUMMeanB - SUMSDB)) {
        addtoROIScript = true;
      }
//BLUE LOCAL
      if (localMeanB > (MeanB + SDB)) {
        addtoROIScript = true;
      }
      if (localMeanB < (MeanB - SDB)) {
        addtoROIScript = true;
      }
// determine whether the output to be added or not
      if (addtoROIScript) {
        freshData localRefreshData = new
freshData(FrameNumber, XCoord, YCoord);
/* Create a new freshData Object and add to ArrayList - as the information is sent to freshData the number will also naturally order the frames. The following code adds the new (fresh) data to the correct position in the arraylist . */
        if (refreshData.isEmpty() == true) {
          refreshData.add(localRefreshData);
        }
        else if (refreshData.size() < 2) {
          freshData freshData1 = (freshData)
refreshData.get(0);
          if ( (localRefreshData.SortKey()) <=
(freshData1.SortKey())) {
            refreshData.add(0, localRefreshData);
          }
          else {
            refreshData.add(localRefreshData);
          }
        }
        else {
          int iterateVariable = 0;
          int nextRefreshData = iterateVariable + 1;
          int lastRefreshData = refreshData.size() - 1;
          boolean insert = false;
          while (iterateVariable < refreshData.size() &&
!insert) {
            freshData refreshData1 = (freshData)
refreshData.get(
                iterateVariable);
            freshData refreshData2 = (freshData)
refreshData.get(
                nextRefreshData);
            freshData refreshData3 = (freshData)
refreshData.get(
                lastRefreshData);
            if ( (iterateVariable == 0) &&
                (localRefreshData.SortKey() <
refreshData1.SortKey())) {
              refreshData.add(0, localRefreshData);
              insert = true;
            }
            else if (localRefreshData.SortKey() <
refreshData1.SortKey()) {
              refreshData.add(iterateVariable,
localRefreshData);
              insert = true;
            }
            else if (localRefreshData.SortKey() ==
refreshData1.SortKey()) {
              refreshData.add(iterateVariable,
localRefreshData);
              insert = true;
            }
            else if ( (localRefreshData.SortKey() >
refreshData1.SortKey()) &&
                  (iterateVariable == refreshData.size())) {
              refreshData.add(localRefreshData);
              insert = true;
            }
            else if ( (localRefreshData.SortKey() >
refreshData1.SortKey()) &&
                  (localRefreshData.SortKey() <=
refreshData2.SortKey())) {
              refreshData.add(nextRefreshData,
localRefreshData);
              insert = true;
            }
            else if (localRefreshData.SortKey() >
refreshData3.SortKey()) {
              refreshData.add(localRefreshData);
              insert = true;
            }
            else
              iterateVariable++;
          }
        }
/* New data added to the arraylist */
        System.out.println("ADDED: " +
localRefreshData.getCurrentFrame()+ "_" +
localRefreshData.getx1()+ "_" +
localRefreshData.gety1() + "_MOV");
      }
    }
  }
//load information from file into input Vector
  inputString = "";
/* create new dialogue box – to allow the user to define the location of the movement RoI Script */
  dialogInputFrame = new Frame();
  FileDialog inputFileNameOR = new
FileDialog(dialogInputFrame,
```

```
      "Complete OR Script: Mean and SD");
   inputFileNameOR.setVisible(true);
   do
    selectedItem = inputFileNameOR.getDirectory() +
inputFileNameOR.getFile();
   while (selectedItem == null);
// repeat until a item has been selected
   dialogInputFrame.dispose();
// close dialogue box
   inputMeanVector = new Vector();
// create a new InputVector and input text data
   try {
    FileReader inputFile = new FileReader(selectedItem);
    BufferedReader reader = new
BufferedReader(inputFile);
     String bufferInput;
     while ( (bufferInput = reader.readLine()) != null)
      inputMeanVector.add(bufferInput);
   }
   catch (Exception e) {
    System.err.println(e.getMessage());
   }
/* The sum defines the MEAN RGB and SD RGB*/
   SUMMeanR = 0;
   SUMMeanG = 0;
   SUMMeanB = 0;
   SUMSDR = 0;
   SUMSDG = 0;
   SUMSDB = 0;
//values to store sun of mean and sd values
   count = 0;
   numOfLines = inputMeanVector.size();
/* we need to identify the middle value and SD for the
entire video (condition 1) – this is required to test if the
area of a video frame is outside the mean across the
whole video. */
   for (int lineNum = 1; lineNum < (numOfLines - 3);
lineNum++) {
/* for all frame in the input file - minus three excludes
the summary at the end of each file. As always a space
before data */
     inputString = (String)
inputMeanVector.get(lineNum);
// get and print the line
     System.out.println(inputString);
     String FrameName = "";
     double MeanR = 0;
     double MeanG = 0;
     double MeanB = 0;
     double SDR = 0;
     double SDG = 0;
     double SDB = 0;
     String rubbishToken;
// to remove the unwanted tokens
     String currentLine = (String)
inputMeanVector.get(lineNum);
     StringTokenizer lineTokenizer = new
StringTokenizer(currentLine, "\t");
//Start extracting the data from the file
     if (lineTokenizer.hasMoreTokens()) {
      FrameName = lineTokenizer.nextToken();
     }
     if (lineTokenizer.hasMoreTokens()) {
      try {
       MeanR =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract mean edge red value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
     if (lineTokenizer.hasMoreTokens()) {
      try {
       MeanG =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract mean green value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
     if (lineTokenizer.hasMoreTokens()) {
      try {
       MeanB =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract mean blue value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
     for (int i = 0; i <= 5; i++) {
      if (lineTokenizer.hasMoreTokens()) {
       rubbishToken = lineTokenizer.nextToken();
      }
     }
//ignores no relevant information
     if (lineTokenizer.hasMoreTokens()) {
      try {
       SDR =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract red standard deviation pixel value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
     if (lineTokenizer.hasMoreTokens()) {
      try {
       SDG =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract green standard deviation pixel value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
     if (lineTokenizer.hasMoreTokens()) {
      try {
       SDB =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract blue standard deviation pixel value
      }
      catch (NumberFormatException e) {
       System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
      }
     }
     System.out.println(MeanR + " " + MeanG + " " +
MeanB + " " + SDR + " " + SDG + " " + SDB + " ");
// print on screen
// Sum values
     SUMMeanR = SUMMeanR + MeanR;
     SUMMeanG = SUMMeanG + MeanG;
     SUMMeanB = SUMMeanB + MeanB;
     SUMSDR = SUMSDR + SDR;
     SUMSDG = SUMSDG + SDG;
     SUMSDB = SUMSDB + SDB;
     count++;
   }
   SUMMeanR = SUMMeanR / count;
```

```
    SUMMeanG = SUMMeanG / count;
    SUMMeanB = SUMMeanB / count;
    SUMSDR = SUMSDR / count;
    SUMSDG = SUMSDG / count;
    SUMSDB = SUMSDB / count;
    System.out.println("MID R: " + SUMMeanR);
    System.out.println("MID G: " + SUMMeanG);
    System.out.println("MID B: " + SUMMeanB);
    System.out.println("MID SDR: " + SUMSDR);
    System.out.println("MID SDG: " + SUMSDG);
    System.out.println("MID SDB: " + SUMSDB);
/* Provide the mean value and SD for colour contrast in
the entire video (condition 1) – this is required to test if
the area of a video frame is outside the mean across the
whole video. We next need to find the mean value and
SD for colour contrast in each frame (condition 2) – this
is required to test if the area of a video frame is outside
the mean across for a specific frame. */
    for (int lineNum = 1; lineNum < (numOfLines - 3);
lineNum++) {
/* for all frames in the video - minus three to exclude the
summary at the end of each file. */

/* The following extracts frame values – condition 2 */
      inputString = (String)
inputMeanVector.get(lineNum);
      System.out.println(inputString);
      String FrameName = "";
      double MeanR = 0;
      double MeanG = 0;
      double MeanB = 0;
      double SDR = 0;
      double SDG = 0;
      double SDB = 0;
      String rubbishToken;
/* rubbishTokens is used to remove the unwanted
tokens */
      String currentLine = (String)
inputMeanVector.get(lineNum);
      StringTokenizer lineTokenizer = new
StringTokenizer(currentLine, "\t");
//Start extracting the data from the file
      if (lineTokenizer.hasMoreTokens()) {
        FrameName = lineTokenizer.nextToken();
      }
      if (lineTokenizer.hasMoreTokens()) {
        try {
          MeanR =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract edge mean red value
        }
        catch (NumberFormatException e) {
          System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
        }
      }
      if (lineTokenizer.hasMoreTokens()) {
        try {
          MeanG =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract edge mean green value
        }
        catch (NumberFormatException e) {
          System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
        }
      }
      if (lineTokenizer.hasMoreTokens()) {
        try {
          MeanB =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract edge mean green value
        }
        catch (NumberFormatException e) {
          System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
        }
      }
      for (int i = 0; i <= 5; i++) {
        if (lineTokenizer.hasMoreTokens()) {
          rubbishToken = lineTokenizer.nextToken();
        }
      }
//ignores no relevant information
      if (lineTokenizer.hasMoreTokens()) {
        try {
          SDR =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract red standard deviation pixel value
        }
        catch (NumberFormatException e) {
          System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
        }
      }
      if (lineTokenizer.hasMoreTokens()) {
        try {
          SDG =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract green standard deviation pixel value
        }
        catch (NumberFormatException e) {
          System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
        }
      }
      if (lineTokenizer.hasMoreTokens()) {
        try {
          SDB =
Double.valueOf(lineTokenizer.nextToken()).doubleValue
();
// extract blue standard deviation pixel value
        }
        catch (NumberFormatException e) {
          System.out.print("ERROR DUE TO DOUBLE
PARSING"); / in case of error
        }
      }
      System.out.println("SECOND CONDITION: " +
FrameName + " " + MeanR + " " + MeanG + " " +
MeanB + " " + SDR + " " + SDG + " " + SDB + " ");
//Passed first set of numbers
      String RootDirectory = (FrameName.substring(0,
2));
      int FrameNumber =
(Integer.parseInt(FrameName.substring(2, 5)) + 1);

//IDENTIFYING THE MOV FRAME NUMBER
/* Each frame is split into 300 squares. Therefore we
need to identify the correct text file */
      String OriFile = inputFileNameOR.getDirectory() +
RootDirectory + "\\" + RootDirectory + "IMSQSUM"
+ FrameNumber + ".txt";
// outputs the ORI and FILE NAMES
      String inputORString = "";
      Vector inputORVector = new Vector();
      try {
/* Transfer all information in the correct text file into
the inputVector. */
        FileReader inputFile = new FileReader(OriFile);
        BufferedReader reader = new
BufferedReader(inputFile);
        String bufferInput;
        while ( (bufferInput = reader.readLine()) != null)
          inputORVector.add(bufferInput);
      }
      catch (Exception e) {
        System.err.println(e.getMessage());
      }
```

```
    for (int locallineNum = 1; locallineNum <
(inputORVector.size());
        locallineNum++) {
/* For all lines in the input file - minus three that
excludes the summary at the end of each file (represent
information concerning the 300 distribution squares).
Extract information concerning mean and standard
deviation of distribution squares. Following define
distribution square variables. */
      inputORString = (String)
inputORVector.get(locallineNum);
      //System.out.println(inputMovString);
      String localFrameName = "";
      double localMeanR = 0;
      double localMeanG = 0;
      double localMeanB = 0;
      String localcurrentLine = (String)
inputORVector.get(locallineNum);
      StringTokenizer locallineTokenizer = new
StringTokenizer(
          localcurrentLine, "\t");
//Start extracting the data from the file
      if (locallineTokenizer.hasMoreTokens()) {
        localFrameName =
locallineTokenizer.nextToken();
      }
      if (locallineTokenizer.hasMoreTokens()) {
        try {
          localMeanR =
Double.valueOf(locallineTokenizer.nextToken()).
              doubleValue();
// extract distribution square mean red value
        }
        catch (NumberFormatException e) {
          System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
        }
      }
      if (locallineTokenizer.hasMoreTokens()) {
        try {
          localMeanG =
Double.valueOf(locallineTokenizer.nextToken()).
              doubleValue();
// extract distribution square mean green value
        }
        catch (NumberFormatException e) {
          System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
        }
      }
      if (locallineTokenizer.hasMoreTokens()) {
        try {
          localMeanB =
Double.valueOf(locallineTokenizer.nextToken()).
              doubleValue();
// extract distribution square mean blue value
        }
        catch (NumberFormatException e) {
          System.out.print("ERROR DUE TO DOUBLE
PARSING"); // in case of error
        }
      }
//determine x and y coordinates from local file name
/* The following must consider the fact that the frame
number can be represented as one, two, three or four
characters. */
      String XYCoordinate =
(localFrameName.substring(6, 17));
      int XCoord = 0;
      int YCoord = 0;
      if (XYCoordinate.substring(2,
3).equalsIgnoreCase("_")) {
        XCoord =
Integer.parseInt(XYCoordinate.substring(0, 2));
        if (XYCoordinate.substring(5,
6).equalsIgnoreCase("_")) {
          YCoord =
Integer.parseInt(XYCoordinate.substring(3, 5));
        }
        else {
          YCoord =
Integer.parseInt(XYCoordinate.substring(3, 6));
        }
      }
      else if (XYCoordinate.substring(3,
4).equalsIgnoreCase("_")) {
        XCoord =
Integer.parseInt(XYCoordinate.substring(0, 3));
        if (XYCoordinate.substring(6,
7).equalsIgnoreCase("_")) {
          YCoord =
Integer.parseInt(XYCoordinate.substring(4, 6));
        }
        else {
          YCoord =
Integer.parseInt(XYCoordinate.substring(4, 7));
        }
      }
      else if (XYCoordinate.substring(4,
5).equalsIgnoreCase("_")) {
        XCoord =
Integer.parseInt(XYCoordinate.substring(0, 4));
        if (XYCoordinate.substring(7,
8).equalsIgnoreCase("_")) {
          YCoord =
Integer.parseInt(XYCoordinate.substring(5, 7));
        }
        else if (XYCoordinate.substring(8,
9).equalsIgnoreCase("_")) {
          YCoord =
Integer.parseInt(XYCoordinate.substring(5, 8));
        }
        else {
          YCoord =
Integer.parseInt(XYCoordinate.substring(5, 9));
        }
      }
/*CONDITIONS TO INCLUDE OR NOT INCLUDE – Adds a value to the Region of Interest script is the local value is one SD greater or less than either the frame or video SD (see chapter 5). Red, green and blue comparisons are identical so no repeated comments will be given. */
      boolean addtoROIScript = false;
//RED LOCAL
      if (localMeanR > (MeanR + SDR)) {
        addtoROIScript = true;
      }
      if (localMeanR < (MeanR - SDR)) {
        addtoROIScript = true;
      }
//GREEN LOCAL
      if (localMeanG > (MeanG + SDG)) {
        addtoROIScript = true;
      }
      if (localMeanG < (MeanG - SDG)) {
        addtoROIScript = true;
      }
//BLUE LOCAL
      if (localMeanB > (MeanB + SDB)) {
        addtoROIScript = true;
      }
      if (localMeanB < (MeanB - SDB)) {
        addtoROIScript = true;
      }
// determine whether the output to be added or not
      if (addtoROIScript) {
/* Create a new freshData Object and add to ArrayList -
as the information is sent to freshData the number will
also naturally order the frames. The following code adds
the new (fresh) data to the correct position in the
arraylist . */
```

```
        freshData localRefreshData = new
freshData(FrameNumber, XCoord, YCoord);
// Add to ArrayList - hopefully in order
        if (refreshData.isEmpty() == true) {
          refreshData.add(localRefreshData);
        }
        else if (refreshData.size() < 2) {
          freshData freshData1 = (freshData)
refreshData.get(0);
          if ( (localRefreshData.SortKey()) <=
(freshData1.SortKey())) {
            refreshData.add(0, localRefreshData);
          }
          else {
            refreshData.add(localRefreshData);
          }
        }
        else {
          int iterateVariable = 0;
          int nextRefreshData = iterateVariable + 1;
          int lastRefreshData = refreshData.size() - 1;
          boolean insert = false;
          while (iterateVariable < refreshData.size() &&
!insert) {
            freshData refreshData1 = (freshData)
refreshData.get(
              iterateVariable);
            freshData refreshData2 = (freshData)
refreshData.get(
              nextRefreshData);
            freshData refreshData3 = (freshData)
refreshData.get(
              lastRefreshData);
            if ( (iterateVariable == 0) &&
              (localRefreshData.SortKey() <
refreshData1.SortKey())) {
              refreshData.add(0, localRefreshData);
              insert = true;
            }
            else if (localRefreshData.SortKey() <
refreshData1.SortKey()) {
              refreshData.add(iterateVariable,
localRefreshData);
              insert = true;
            }
            else if (localRefreshData.SortKey() ==
refreshData1.SortKey()) {
              refreshData.add(iterateVariable,
localRefreshData);
              insert = true;
            }
            else if ( (localRefreshData.SortKey() >
refreshData1.SortKey()) &&
                  (iterateVariable == refreshData.size())) {
              refreshData.add(localRefreshData);
              insert = true;
            }
            else if ( (localRefreshData.SortKey() >
refreshData1.SortKey()) &&
                  (localRefreshData.SortKey() <=
refreshData2.SortKey())) {
              refreshData.add(nextRefreshData,
localRefreshData);
              insert = true;
            }
            else if (localRefreshData.SortKey() >
refreshData3.SortKey()) {
              refreshData.add(localRefreshData);
              insert = true;
            }
            else
              iterateVariable++;
          }
        }
/* New data added to the arraylist */
        System.out.println("ADDED: " +
localRefreshData.getCurrentFrame()
                    + "_" + localRefreshData.getx1()
                    + "_" + localRefreshData.gety1()
                    + "_ORI");
      }
    }
  }
//Select Output File for the output of System
//create a new dialogue box
  Frame dialogOutputFrame = new Frame();
  FileDialog outputFileName = new
FileDialog(dialogOutputFrame,"Defining Output File");
  outputFileName.setVisible(true);
  String selectOutput;
  do
    selectOutput = outputFileName.getDirectory() +
outputFileName.getFile();
  while (selectOutput == null);
  dialogOutputFrame.dispose();
// close the dialogue box
  PrintWriter out;
  try {
    out = new PrintWriter(new
FileWriter(selectOutput));
//create the output PrintWriter
  }
  catch (IOException ex) {
    throw new Error(ex.toString());
  }
//creating the contents of the output file
//top header
  out.print("h");
  out.print("\n0");
  out.print("\n25");
  out.print("\n5");
  out.print("\nc\t0\t0\t640\t480");

/* for all 1 –  639 (can be changed depending on the
number of frames in the video) */
  for (int i = 1; i <= 639; i++) {
    boolean noentry = true;
/* if number divided by 5 has remainer zero then print
information concerning internal and externl frame rates,
plus 'f0' header. */
    if ( (i % 5) == 0) {
      out.print("\n" + i);
      out.print("\n25");
      out.print("\n5");
      out.print("\nf\t0");
    }
/* if number divided by 5 has remainer one then add a
header 'f1' */
    if ( (i % 5) == 1) {
      out.print("\nf\t1");
    }
/* if number divided by 5 has remainer one then add a
header 'f2' */
    if ( (i % 5) == 2) {
      out.print("\nf\t2");
    }
/* if number divided by 5 has remainer one then add a
header 'f3' */
    if ( (i % 5) == 3) {
      out.print("\nf\t3");
    }
/* if number divided by 5 has remainer one then add a
header 'f4' */
    if ( (i % 5) == 4) {
      out.print("\nf\t4");
    }

    for (int j = 1; j < ( (refreshData.size() - frameShift) -
1); j++) {
// for all information in tha arraylist
      if ( ( (freshData)
(refreshData.get(j))).getCurrentFrame() == i) {
/* if the frame number matches I the add to the output
file – note coordinate values */
```

```
        out.print("\nc\t" +
                ( ( (freshData) (refreshData.get(j))).getx1() -
32)+ "\t" + ( ( (freshData) (refreshData.get(j))).gety1() -
32)+ "\t" + ( ( (freshData) (refreshData.get(j))).getx1() +
64)+ "\t" + ( ( (freshData) (refreshData.get(j))).gety1() +
64));
        noentry = false;
       }
```

```
        if (noentry) {
          out.print("\nc\t0\t0\t640\t480");
        }
       }
       out.close();
      }
     }
    }
```