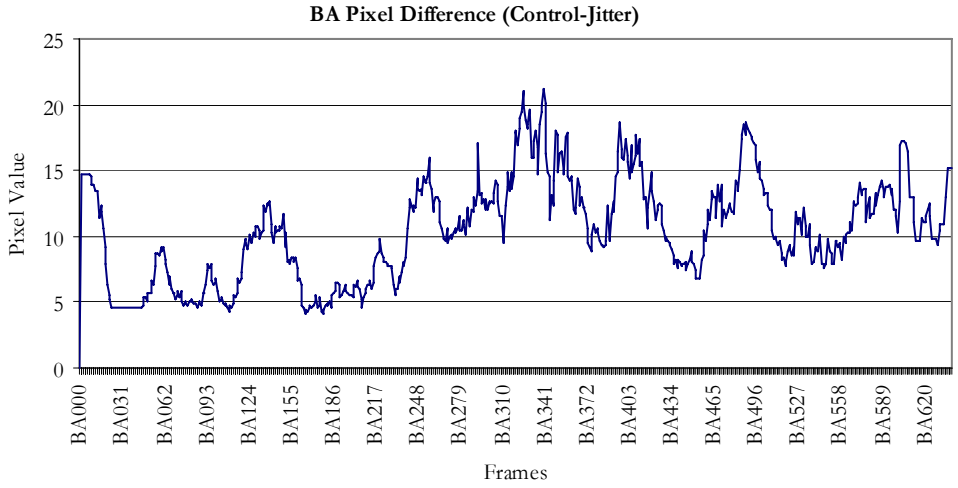
Appendix H

Image Pixel Distributions

The value of pixels was used in our work to represent information including: the distribution of participant fixations across a video frame; the existence of an edge / colour contrast in an original image frame; as well the difference between two frames, i.e. the movement. As pixels have a red, green and blue value that lies between 0 (black) and 255 (white), a change in the distribution of pixel values represents a change in the colours used in the video frame. A change over time can be observed by considering the variation in the average pixel value for each video frame. Although rudimentary, analysis of multiple factors provides a good idea of the video contents. The following code was used to calculate pixel distributions and was used in the definition of fixation maps, content-based edge, colour and movement regions of interest.



Pixel Difference between control and jitter fixation maps (BA video).

Package: definingroi

pixelData.java – object contains single pixel value

```
/* pixelData is class that facilitates the easy manipulation
of pixel information. Each pixel defines the content of a
pixelData object, therefore many hundred-thousand
objects are created for each image. */

package definingroi;
public class pixelData {
/* pixelData objects contain x, y coordinate values and
red, green and blue pixel values. */
  private int x;
  private int y;
  private int red;
  private int green;
  private int blue;

  public pixelData(int x_value, int y_value, int red_value,
int green_value, int blue_value)
//assign values from pixel at object creation
  {
   x = x_value;
   y = y_value;
   red = red_value;
   green = green_value;
   blue = blue_value;
```

```
 }

 private void setXValue(int x_value)
 {
  x = x_value;
// allows the x coordinate to be manipulated
 }

 public int getXValue()
 {
  return x;
// allows the x coordinate to be monitored

 }

 private void setYValue(int y_value)
 {
  y = y_value;
// allows the y coordinate to be manipulated

 }

 public int getYValue()
 {
  return y;
// allows the x coordinate to be monitored

 }

 private void setRedValue(int red_value)
 {
  red = red_value;
//allows the red value to be manipulated
 }

 public int getRedValue()
 {
  return red;
// allows the red value to be monitored
 }

 private void setGreenValue(int green_value)
 {
  green = green_value;
//allows the green value to be manipulated
 }

 public int getGreenValue()
 {
  return green;
// allows the green value to be monitored
 }

 private void setBlueValue(int blue_value)
  {
   blue = blue_value;
// allows the blue value to manipulated
  }

  public int getBlueValue()
  {
   return blue;
// allows the blue value to be monitored
  }

  public void description()
  {
   System.out.println("X:"+x+" Y:"+y+" red:"+red+"
green:"+ green +" blue:"+blue);
// allows the object content to be displayed
  }
}
```

distribution.java – extracts distribution for a whole frame *(later adapted for 40x40 pixel squares)*

```
package definingroi;
/* the following code takes a 640x480 jpeg image and
determines the distribution of pixels for red, green and
blue pixels */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.util.*;
import java.net.URL;
import java.io.*;
import com.sun.image.codec.jpeg.*;

public class distribution
   extends JFrame {
 Image distributionImage;
 String SelectedItem = "";
 ArrayList inputImage = new ArrayList();
//input image used to store pixelData objects

 class ImagePanel
    extends JPanel {

  public ImagePanel() {
  }

  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.drawImage(distributionImage, 10, 10, this);
    //}
  }
 }

 public void handlesinglepixel(int x, int y, int pixel, int
image) {
  int alpha = (pixel >> 24) & 0xff;
  int red = (pixel >> 16) & 0xff;
  int green = (pixel >> 8) & 0xff;
  int blue = (pixel) & 0xff;
  pixelData pg = new pixelData(x, y, red, green, blue);
  inputImage.add(pg);
/* extracts a specific pixel value into red, green and blue
values, which it assigns to a pixelData object. This is then
added to the arrayList*/
 }

 public void handlepixels(Image img, int x, int y, int w,
int h, int image) {
  int[] pixels = new int[w * h];
//creates a temporary array for pixel values
  PixelGrabber pg = new PixelGrabber(img, x, y, w, h,
pixels, 0, w);
//defines image to be used and grab area
  try {
   pg.grabPixels();
//grabs pixel single value and places values in array
  }
  catch (InterruptedException e) {
   System.err.println("interrupted waiting for pixels!");
   return;
  }
  if ( (pg.getStatus() & ImageObserver.ABORT) != 0) {
   System.err.println("image fetch aborted or errored");
   return;
  }
//repeatedly calls handlesinglepixel for each pixel value
  for (int j = 0; j < h; j++) {
   for (int i = 0; i < w; i++) {
    handlesinglepixel(x + i, y + j, pixels[j * w + i],
image);
```

```
    }
   }
  }

  public distribution(FileDialog inputFileName,
FileDialog outputFileName,
                String prefix, int length) {
    setTitle("Input Image");
    setSize(660, 500);
    Container contentPane = getContentPane();
//creates image plane, used to manipulate image
    SelectedItem = "" + inputFileName.getDirectory() +
prefix;
//define file name
    System.out.println("Selected File: " + SelectedItem);
    try { // incase of error in file name
      File imageFile = new File(SelectedItem);
      distributionImage =
getToolkit().getImage(imageFile.toURL());
//copy image to be grabbed
      repaint();
      handlepixels(distributionImage, 0, 0, 640, 480, 1);
// extract pixel values from image
      System.out.println("No of Pixels " +
inputImage.size());
      int[][] distributionIndex = new int[3][260];
      System.out.println("Array Cleaned");
// create array to hold red, green and blue distribution
      String outputDirectory;
      do
        outputDirectory = outputFileName.getDirectory();
      while (outputDirectory == null);
// get distribution output file defined
      PrintWriter out;
      String tempOut = outputDirectory +
         (prefix.substring(0, (prefix.length()) - 4)) +
         "_dis.txt";
/* define output file name, which can be changed
depending on the images being analysed */
      System.out.println("outputFile: " + tempOut);
      try {
        out = new PrintWriter(new FileWriter(tempOut));
      }
      catch (IOException ex) {
        throw new Error(ex.toString());
      }
/* for all pixels increment in the array the number of t
the red (column 0), green (column 1)and blue (column 2)
pixel values. For example if r40g120b200, then increment
[0,40] [1,120] and [2, 200]. This provides an image colour
distribution. */
      for (int i = 0; i < inputImage.size(); i++) {
        distributionIndex[0][ ( (pixelData)
inputImage.get(i)).getRedValue()]++;
        distributionIndex[1][ ( (pixelData)
inputImage.get(i)).getGreenValue()]++;
        distributionIndex[2][ ( (pixelData)
inputImage.get(i)).getBlueValue()]++;
      }
      System.out.println("Created Distribution within an
array");
      for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 260; j++) {
          out.print("\n" + i + "\t" + j + "\t" +
distributionIndex[i][j]);
//send completed information to the output file
        }
      }
      out.close();

    }
    catch (Exception e) {
      System.err.println(e.getMessage());
    }

    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });
  }
}
```

testDistribution.java – facilitate use of distribution.java

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

class testdistribution {

public static void main(String[] args) {
  Frame dialogInputFrame = new Frame();
//creates a new frame
  FileDialog inputFileName = new
FileDialog(dialogInputFrame, "TEST DIRECTORY");
  inputFileName.setVisible(true);
  String selectedInputDirectory;
//selected input dialogue box
/* Allows the user ro select the input directory that
contains the processed images */
  do {
    selectedInputDirectory =
inputFileName.getDirectory();
    System.out.println("Selected Input Directory: " +
selectedInputDirectory);
  }
  while (selectedInputDirectory == null);
  dialogInputFrame.dispose();
// remove dialogue box
  Frame dialogInputFrame2 = new Frame();
//creates a new dialogue box
  FileDialog inputFileName2 = new
FileDialog(dialogInputFrame2, "TEST OUTPUT
DIRECTORY");
  inputFileName2.setVisible(true);
  String selectedInputDirectory2;
//selected output file
/* Allows the user to select the location where
distribution text files are to be placed */
  do {
    selectedInputDirectory2 =
inputFileName2.getDirectory();
    System.out.println("Selected Output Directory: " +
selectedInputDirectory2);
  }
  while (selectedInputDirectory2 == null);
  dialogInputFrame2.dispose();
// remove dialogue box
  File temporyDirectory = new
File(selectedInputDirectory);
  String fileList[] = temporyDirectory.list();
  for (int currentFile = 0; currentFile < fileList.length;
currentFile++) {
    try {
// loads data from file into inputVector
      System.out.println("current
file:"+fileList[currentFile]);

System.out.println("prefix:"+(fileList[currentFile]).substri
ng(0,(fileList[currentFile].length()-4)));
      distribution test = new distribution(inputFileName,
inputFileName2,
        fileList[currentFile], fileList[currentFile].length());
//defines distribution settings and runs test.
      test.dispose(); // shuts frame box
```

```
   }
   catch (Exception e) {
    System.err.println(e.getMessage());
   }
```

```
  }
  }
}
```

frameDistribution.java – to identify the median and average pixel values from the created distribution files *(assuming naming format XX0000.jpg, e.g. BA0001.jpg)*

```
/* This software is used to find median and average pixel
values from the distribution
package definingroi;

import java.util.*;
import java.io.*;
import java.awt.*;

public class frameDistribution {

  public frameDistribution(FileDialog inputFileName) {

      PrintWriter out;
      String tempOut = inputFileName.getDirectory() +
         inputFileName.getFile().
         substring(0, 2) + "_AVERAGE.txt";
 /* Allows the average, median and standard deviation
for all frames to be saved in a single file. This allows the
information to be cut and pasted into data manipulation
programs – SPSS, Excel */
      System.out.println("outputFile: " + tempOut);
      try {
        out = new PrintWriter(new FileWriter(tempOut));
      }
      catch (IOException ex) {
        throw new Error(ex.toString());
      }
/* A directory called DIS should be used to contain the
distribution text files.*/
      String selectedItem = inputFileName.getDirectory()
+ "\\DIS\\";
      File temporyDirectory = new
File(inputFileName.getDirectory() + "\\DIS\\");
      String fileList[] = temporyDirectory.list();
      double averred = 0;
      double avergreen = 0;
      double averblue = 0;
      int framecount = 0;
// assign distribution variables (red, green and blue)

      for (int currentFile = 0; currentFile < fileList.length;
currentFile++) {
// for all images in the images file
// clean out variables
        double red = 0;
        double green = 0;
        double blue = 0;
        double sdred = 0;
        double sdgreen = 0;
        double sdblue = 0;
        int maxred = 0;
        int maxgreen = 0;
        int maxblue = 0;
        String inputString;
        int medianred = 0;
        boolean redflag = false;
        int mediangreen = 0;
        boolean greenflag = false;
        int medianblue = 0;
        boolean blueflag = false;

/* Adds the appropriate number of 0 to extract the
correct image file name - assuming naming format
XX0000.jpg, e.g. BA0001.jpg */
        String currentNumber = "";
        if (currentFile < 10) {currentNumber =
"000"+currentFile;}
        else if (currentFile < 100) {currentNumber =
"00"+currentFile;}
        else if (currentFile < 1000) {currentNumber =
"0"+currentFile;}
        else {currentNumber = ""+currentFile;}
        String prefix = (fileList[currentFile]).substring(0,
2)+currentNumber+"_dis.txt";
//define prefix from image file name
        System.out.println("current file:" + prefix);
        Vector inputVector = new Vector();
        try {
/* read file and input information from the text file into
a inputVector. */
         FileReader inputFile = new
FileReader(selectedItem + prefix);
         BufferedReader reader = new
BufferedReader(inputFile);
         String bufferInput;
         while ( (bufferInput = reader.readLine()) != null)
          inputVector.add(bufferInput);
        }
        catch (Exception e) {
         System.err.println(e.getMessage());
        }
        int numOfLines = inputVector.size();
// define the number of lines in the inputVector

        for (int lineNum = 0; lineNum < numOfLines;
lineNum++) {
// for all lines
         inputString = (String) inputVector.get(lineNum);
         String lineToken;
         StringTokenizer lineTokenizer = new
StringTokenizer(inputString, "\t");
         if (lineTokenizer.hasMoreElements()) {
          lineToken = lineTokenizer.nextToken();
          double currentInteger = (double)
Integer.parseInt(lineToken);
/* if the current value in the line starts with 0 then we
know that the information relates to red */
          if (currentInteger == 0) {
           if (lineTokenizer.hasMoreElements()) {
            currentInteger = (double)
Integer.parseInt(lineTokenizer.
              nextToken());
            if (lineTokenizer.hasMoreElements()) {
             int tempvalue =
Integer.parseInt(lineTokenizer.nextToken());
             if (redflag == false)
              medianred = medianred + tempvalue;
             if (medianred >= 153600) {
              medianred = (int) currentInteger;
              redflag = true;
/* starts at pixel value 0. Counts up the until half the
pixels have occurred – therefore median red */
             }
             if (maxred < tempvalue)
              maxred = (int) currentInteger;
             red = red + ( (currentInteger * tempvalue) /
307200);
             averred = averred + ((currentInteger *
tempvalue) / 307200);
```

```
/* Multiples the value of the pixel by the number of
pixels and adds this to the sum. Division by the total
number of pixels provides the average pixel value. */
            }
          }
        }
/* if the current value in the line starts with 1 then we
know that the information relates to green */
        if (currentInteger == 1) {
          if (lineTokenizer.hasMoreElements()) {
            currentInteger =
Integer.parseInt(lineTokenizer.nextToken());
            if (lineTokenizer.hasMoreElements()) {
              int tempvalue =
Integer.parseInt(lineTokenizer.nextToken());
              if (greenflag == false)
                mediangreen = mediangreen + tempvalue;
              if (mediangreen >= 153600) {
                mediangreen = (int) currentInteger;
                greenflag = true;
/* starts at pixel value 0. Counts up the until half the
pixels have occurred – therefore median green */
              }
              if (maxgreen < tempvalue)
                maxgreen = (int) currentInteger;
              green = green + ( (currentInteger *
tempvalue) / 307200);
              avergreen = avergreen + ((currentInteger *
tempvalue) / 307200);
/* Multiples the value of the pixel by the number of
pixels and adds this to the sum. Division by the total
number of pixels provides the average pixel value. */
            }
          }
        }
/* if the current value in the line starts with 2 then we
know that the information relates to blue */
        if (currentInteger == 2) {
          if (lineTokenizer.hasMoreElements()) {
            currentInteger =
Integer.parseInt(lineTokenizer.nextToken());
            if (lineTokenizer.hasMoreElements()) {
              int tempvalue =
Integer.parseInt(lineTokenizer.nextToken());
              if (blueflag == false)
                medianblue = medianblue + tempvalue;
              if (medianblue >= 153600) {
                medianblue = (int) currentInteger;
                blueflag = true;
/* starts at pixel value 0. Counts up the until half the
pixels have occurred – therefore median blue */
              }
              if (maxblue < tempvalue)
                maxblue = (int) currentInteger;
              blue = blue + ( (currentInteger * tempvalue)
/ 307200);
              averblue = averblue + ((currentInteger *
tempvalue) / 307200);
/* Multiples the value of the pixel by the number of
pixels and adds this to the sum. Division by the total
number of pixels provides the average pixel value. */
            }
          }
        }
      }
    }

// calculates the standard deviation
    double tempdiff = 0;
    double totaldiffred = 0;
    double totaldiffgreen = 0;
    double totaldiffblue = 0;
    for (int lineNum = 0; lineNum < numOfLines;
lineNum++) {
//for all the lines in the inputVector
      inputString = (String) inputVector.get(lineNum);
      String lineToken;
      StringTokenizer lineTokenizer = new
StringTokenizer(inputString, "\t");
      if (lineTokenizer.hasMoreElements()) {
        lineToken = lineTokenizer.nextToken();
        double currentInteger = (double)
Integer.parseInt(lineToken);
/* if the current value in the line starts with 0 then we
know that the information relates to red */
        if (currentInteger == 0) {
          if (lineTokenizer.hasMoreElements()) {
            currentInteger = (double)
Integer.parseInt(lineTokenizer.
              nextToken());
            if (lineTokenizer.hasMoreElements()) {
              int tempvalue =
Integer.parseInt(lineTokenizer.nextToken());
              if (red>currentInteger){
                tempdiff = red - currentInteger;
              } else {
                tempdiff = currentInteger - red;
              }
              totaldiffred = totaldiffred +
(tempdiff*tempvalue);
// total deviation for red pixel
            }
          }
        }
/* if the current value in the line starts with 1 then we
know that the information relates to green */
        if (currentInteger == 1) {//green
          if (lineTokenizer.hasMoreElements()) {
            currentInteger =
Integer.parseInt(lineTokenizer.nextToken());
            if (lineTokenizer.hasMoreElements()) {
              int tempvalue =
Integer.parseInt(lineTokenizer.nextToken());
              if (green>currentInteger){
                tempdiff = green - currentInteger;
              } else {
                tempdiff = currentInteger - green;
              }
              totaldiffgreen = totaldiffgreen +
(tempdiff*tempvalue);
// total deviation for green pixel
            }
          }
        }
/* if the current value in the line starts with 2 then we
know that the information relates to blue */
        if (currentInteger == 2) {
          if (lineTokenizer.hasMoreElements()) {
            currentInteger =
Integer.parseInt(lineTokenizer.nextToken());
            if (lineTokenizer.hasMoreElements()) {
              int tempvalue =
Integer.parseInt(lineTokenizer.nextToken());
              if (blue>currentInteger){
                tempdiff = blue - currentInteger;
              } else {
                tempdiff = currentInteger - blue;
              }
              totaldiffblue = totaldiffblue +
(tempdiff*tempvalue);
// total deviation for blue pixel
            }
          }
        }
      }

    double SDred = totaldiffred/ 307200;
    double SDgreen = totaldiffgreen / 307200;
    double SDblue = totaldiffblue/ 307200;
/* total deviated divided by the number of pixels allows
the standard deviation for a single pixel to be identified
*/
    framecount++;
```

```
// increment frame – resets prefix
//output the frame information to the text file
      out.print("\n" + prefix + "\t" + red + "\t" + green
+ "\t" + blue + "\t" + medianred + "\t" +
mediangreen + "\t" + medianblue + "\t" + maxred +
"\t" + maxgreen + "\t" + maxblue + "\t" + SDred +
"\t" + SDgreen + "\t" + SDblue);
      System.out.println("\n" + prefix + "\t" + red +
"\t" + green + "\t" + blue);
      System.out.println("MEDIAN RED: " +
medianred + " RED FLAG: " + redflag + " MAX RED:
" + maxred);
      System.out.println("MEDIAN GREEN: " +
mediangreen + " GREEN FLAG: " + greenflag + "
MAX GREEN: " + maxgreen);
      System.out.println("MEDIAN BLUE: " +
medianblue + " BLUE FLAG: " + blueflag + " MAX
BLUE: " + maxblue);
      System.out.println("AVERAGERED "+ averred +
"AVERAGEGREEN " + avergreen +
"AVERAGEBLUE " + averblue + "FRAMECOUNT "
+ framecount);
       System.out.println("SDRED "+ SDred +
"SDGREEN " + SDgreen + "SDBLUE " + SDblue +
"FRAMECOUNT " + framecount);
    }
/* at the end print average for all pixels in all frames */
    out.println();
    out.println("Average red: " + (averred /
framecount));
    out.println("Average green: " + (avergreen /
framecount));
    out.println("Average blue: " + (averblue /
framecount));
    out.close();
  }
}
```

testframeDistribution.java – summarise frame distributions (assumes a subdirectory called DIS/ which contains the distribution *.txt documents)

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

public class testframeDistribution {
   public static void main(String[] args) {
   Frame dialogInputFrame = new Frame();
//creates a new dialog frame
   FileDialog inputFileName = new
FileDialog(dialogInputFrame,"TEST DIRECTORY");
   inputFileName.setVisible(true);
   String selectedItem;
//selected input file directory
   do {
    selectedItem = inputFileName.getDirectory();
    System.out.println("Selected File: " + selectedItem);
   }
   while (selectedItem == null);
   dialogInputFrame.dispose();
// remove dialog frame
   frameDistribution test = new
frameDistribution(inputFileName);
// run a distribution test
  }
}
```