Adding touch screen ability to LCD using Web Cam
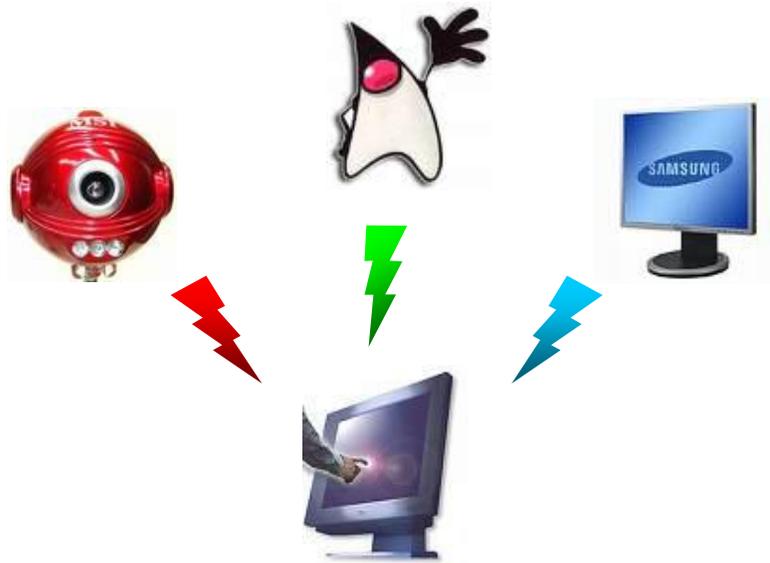
Contents

1. Introduction

Goal of this project was to transform the normal LCD into touch screen. This will be done by using single Web Cam which will be filming the LCD and detecting motion. That motion will then be transformed into adequate action like pressing left mouse button or emulating double left click.

Figures 1.1 to 1.4 show how windows control panel can be used to change screen background using your finger.
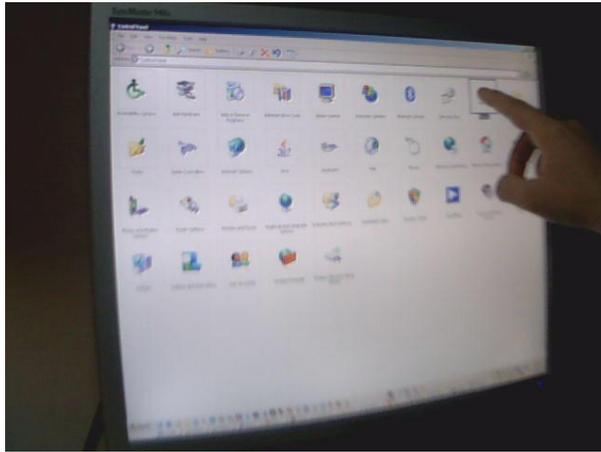

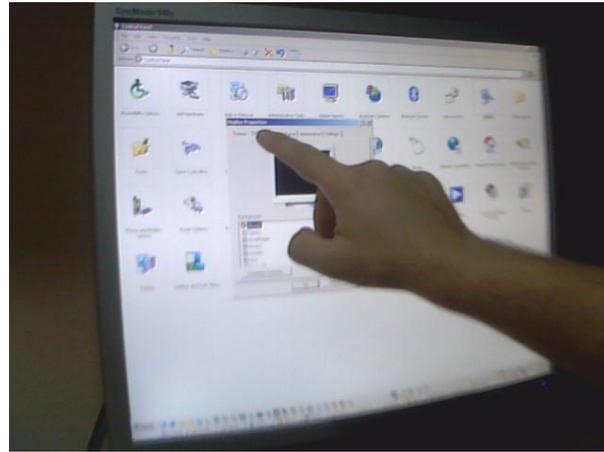Figure 1.1. Selecting Display option from Control panel.

Figure 1.2. Selecting Desktop tab from Display Properties.

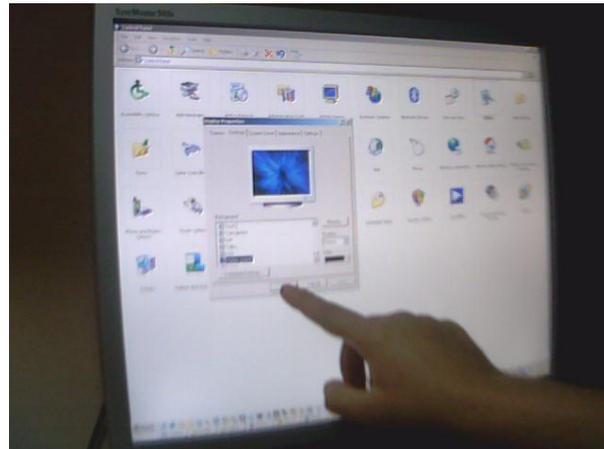Figure 1.2. Selecting background picture for desktop.

Figure 1.3. Selecting OK button.

Application works by mapping Web Cam coordinates into LCD coordinates. This means that when Web Cam detects motion it knows its coordinates inside the image it has taken. Those coordinates are then transformed into LCD coordinates by using set of precalculated coordinates of the dots which are displayed during initialization phase. Motion is detected by using skin detection on normalized RGB color space.

Application was done using JAVA programming language and Eclipse Integrated Development Environment.
Web Cam was controlled using Java MediaFramework - JMF.
Web Cam used was MSI Starcam 370i.
LCD Monitor used was SAMSUNG SyncMaster 940B.
Configuration used was AMD processor 3500+ 64 and 2GB RAM.

2.   Displaying Initialization Screens

Initialization is done by displaying three pictures on LCD.

LCD if first filled with single color as shown in figure 2.1. This will be later used to distinguish area which should be turned into touch screen from the rest of the LCD. This step presumes that there are no of such color behind LCD.

Then area is filled with lines as shown in figure 2.2. This lines will be later used to detect dots. Each line is used to define which dots belong to the same vertical coordinate.

Then area is filled with dots as shown in figure 2.3. This picture is center element in initialization process since the goal of initialization is to determine Web Cam & LCD coordinates of these dots.

After all initialization screens were displayed initialization part of application minimizes and LCD then shows whatever was on it before starting initialization. For instance it could have been control panel as shown on figure 2.4.
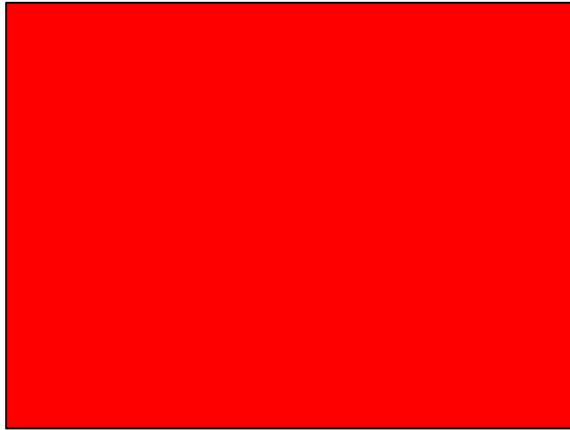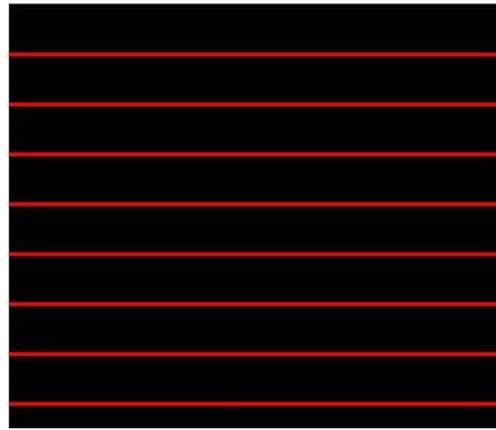

Figure 2.1. LCD is filled with single color.

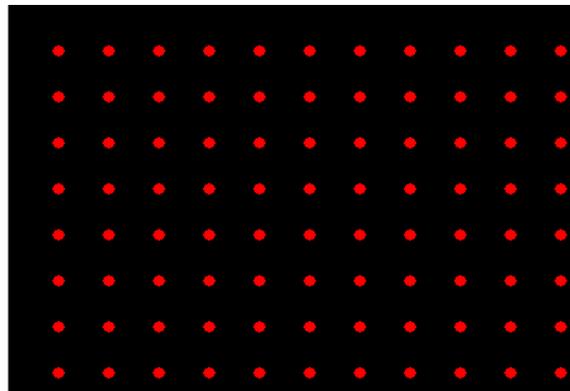Figure 2.2. LCD is filled with horizontal lines.

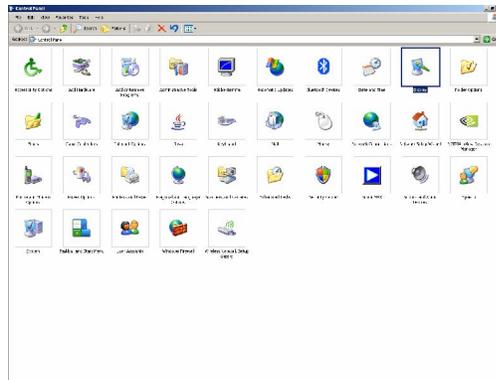Figure 2.3. Area is filled with dots.

Figure 2.4. Windows control panel.

3. Capturing Initialization Screens

Each of these pictures is captured by Web Cam. Web Cam captures not only LCD but also parts of the environment which are outside LCD. Following figures show images captured by Web Cam at the moments when each of the above initialization pictures was displayed.

After capturing initialization screens each of them is then transformed as described in following chapters. Each image goes through different steps which can all be seen in an additional frame which is displayed at the end of initialization process as collection of tabs as shown on figure 3.4.

Figure 3.1. Image taken while LCD was filled with white color. Figure 3.2. Image taken while LCD was filled with lines.
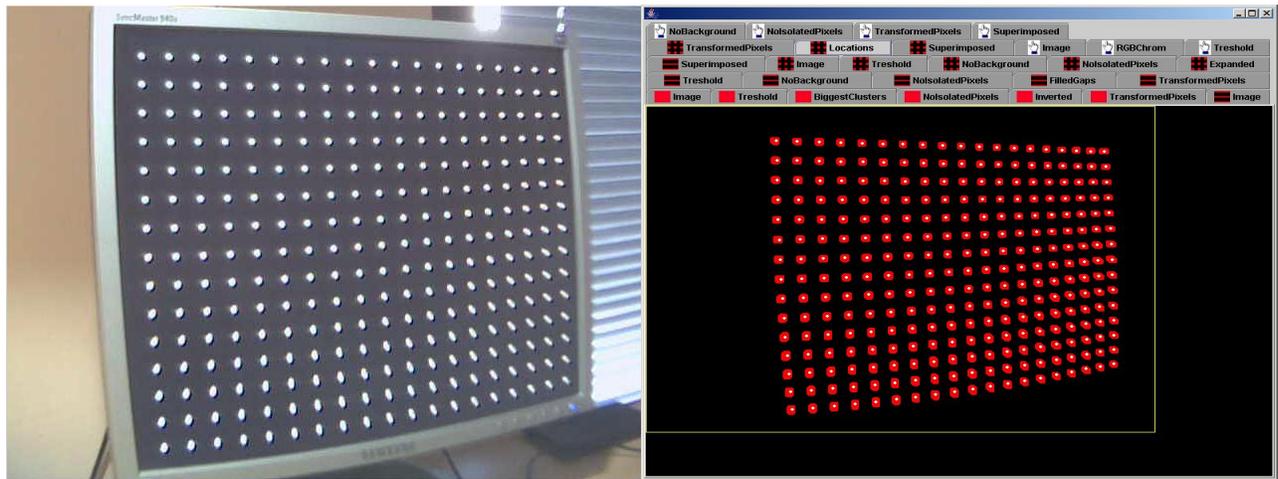

Figure 3.3. Image taken while LCD was filled with dots.   Figure 3.4. Initialization results.

4.   Analyzing Screen Containing Background

Image taken when the whole LCD was in one color is again shown in the figure 4.1

This image is then used to create new image, using 2D array, in the way that pixels in the new image are set to 1only if equivalent pixel in original image has value of all RGB components greater then some predefined value. This results in an image shown on figure 4.2.

After that clustering is used to identify cluster with most pixels. This way background noise is eliminated as shown on figure 4.3.

last step is to invert threshold image. Now all pixels that are 1 present background. This is sown in picture 4.4.
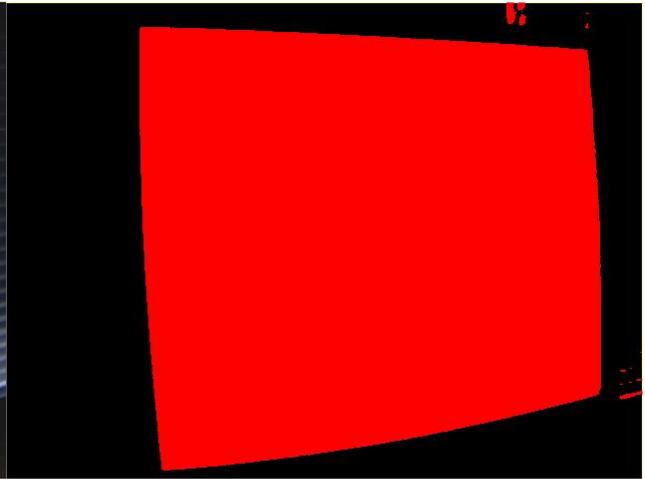
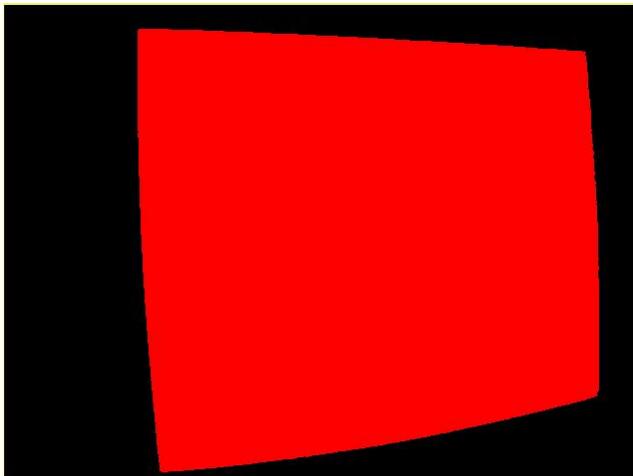Figure 4.1. Image taken while LCD was in one color   Figure 4.2   Result of thresholding.
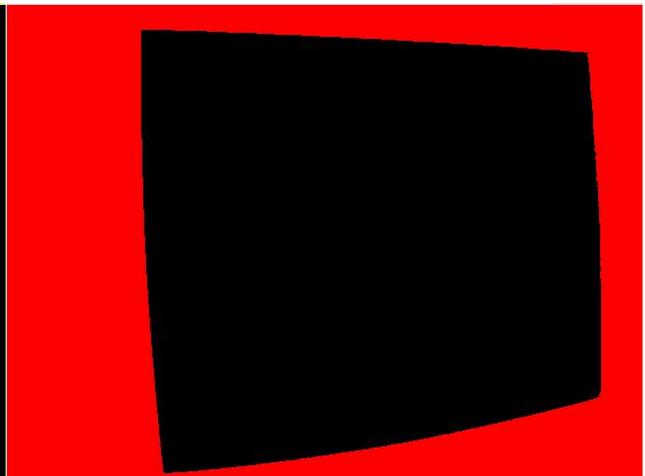
Figure 4.3.   Inverted background.   Figure 4.4.   Background without noise.

5. Analyzing Screen Containing Lines

Image taken by the Web Cam while the area was filled with lines is once again shown on the figure 5.1

Thresholding such image we get result shown in figure 5.2.

Resulting image is then transformed by removing background calculated in previous chapter. Result is shown in figure 5.3.

Then an algorithm for filling gaps is used. Result of that algorithm is shown on figure 5.4.

Then the lines are trimmed from above and below until their height is reduced to one pixel. Algorithms used are lines_TrimLineAbove & lines_TrimLineBelow. Such image is then used to detect each line. This is done by going vertically through image from top to bottom at horizontal position in the middle of the image. Points at which lines were detected are shown as yellow dots at the middle of the lines. Now an algorithm is started which goes from that point to the left and right following line pixels until left and right line end are reached which are highlighted with yellow dots at the end of the lines. Result is shown on figure 5.5.

Figure 5.6. shows how detected lines correspond to the original lines in the image taken by Web Cam.

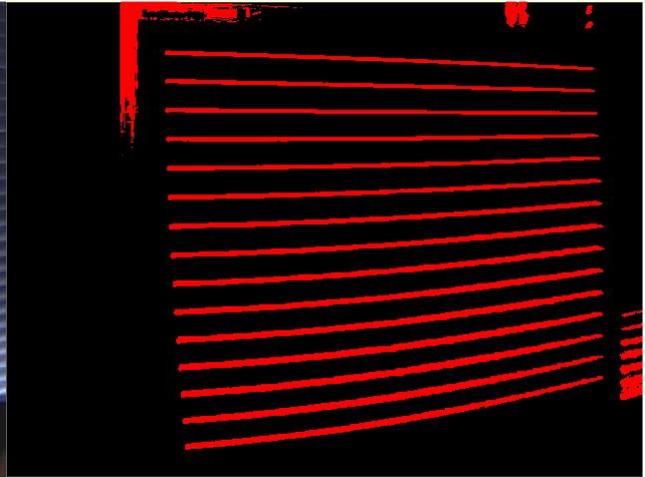Figure 5.1. Image taken while LCD was filled with lines.
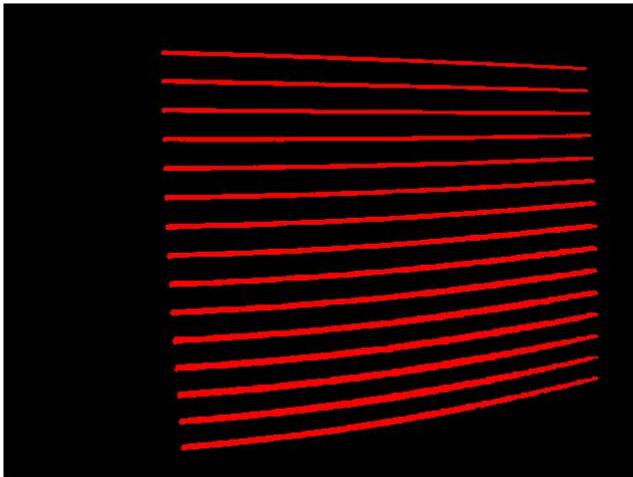

Figure 5.2. Result of thresholding.

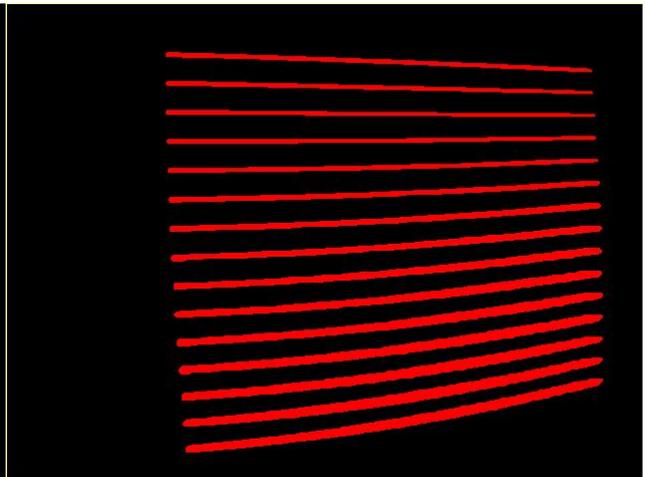Figure 5.3. Result of removing background.
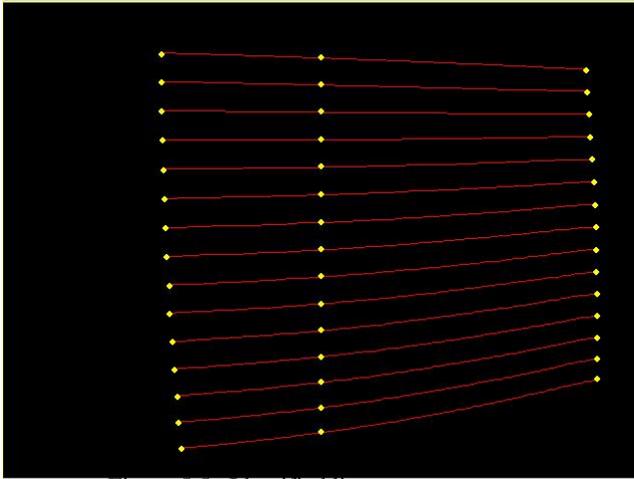

Figure 5.4. Result after filling gaps.
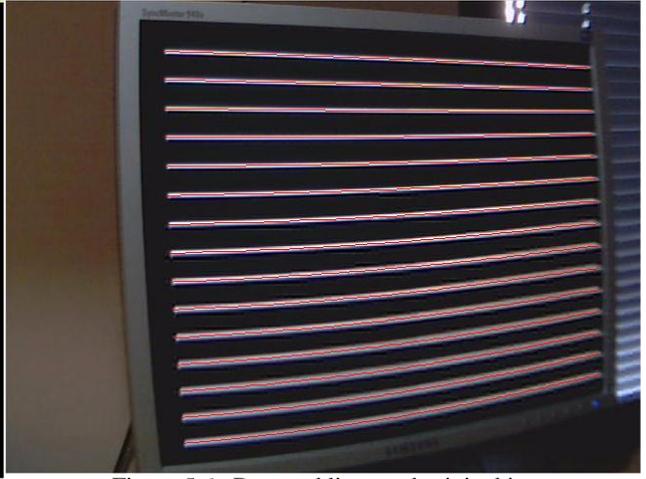
Figure 5.5. Identified lines.

Figure 5.6. Detected lines and original image.

6. Analyzing Screen Containing Dots

Image taken by the Web Cam while LCD was filled with dots is once again shown on the figure 6.1.

Thresholding such image we get result shown in figure 6.2.

Resulting image is then transformed by removing background calculated in previous chapter. Result is shown in figure 6.3.

Then an algorithm for filling gaps is used. Result of that algorithm is shown on figure 6.4.

Dots are detected by starting at the left end of each line extracted in previous chapter, following line pixels to the right and detecting when dot is reached in the image with dots. Web Cam coordinates of each dot are saved and are used later for mapping to LCD coordinates. Located dots are shown in Figure 6.4.

Figure 6.5. shows how detected dots correspond to the original dots in the image taken by Web Cam.

Figure 6.6. additionally shows effects of dialating dots to remove small gaps which might negatively influence dots detection.

Figure 6.7. demonstrates how the dots are actually being detected. Algorithm starts at left end of each lines, follows the line to the right, detects when it has entered and exited the dot and uses this two values to approximates center of the dot. This is how the white dots are calculated.
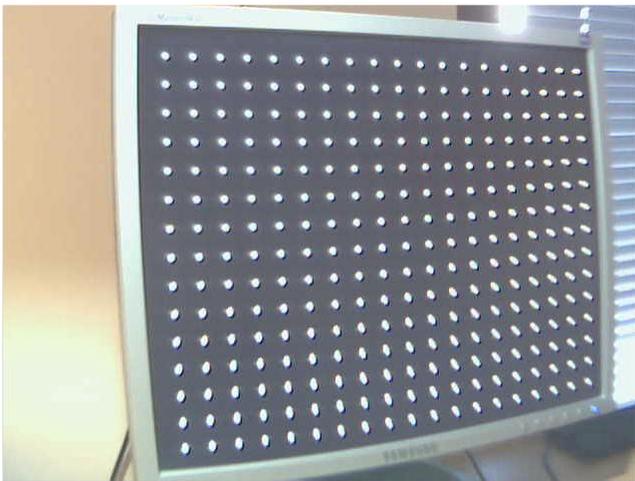

Figure 6.1. Image taken while LCD was filled with dots.
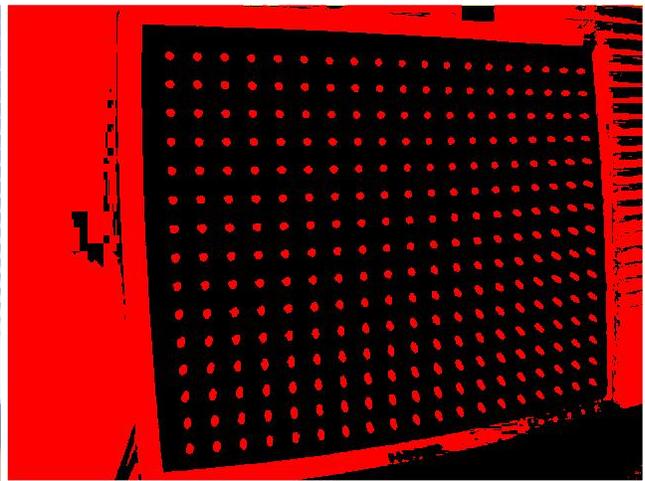

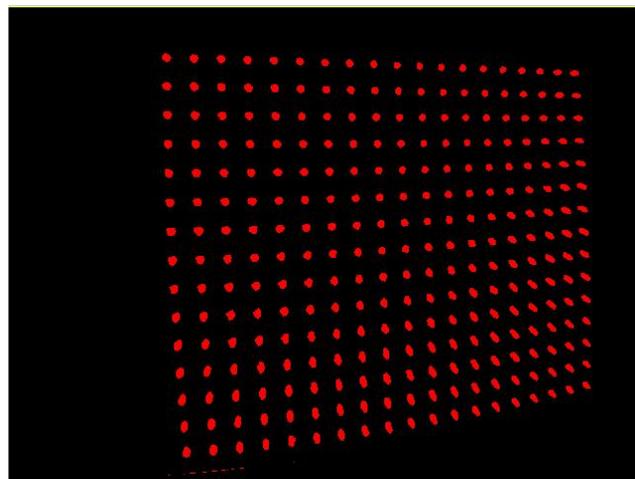Figure 6.2. Result of thresholding.

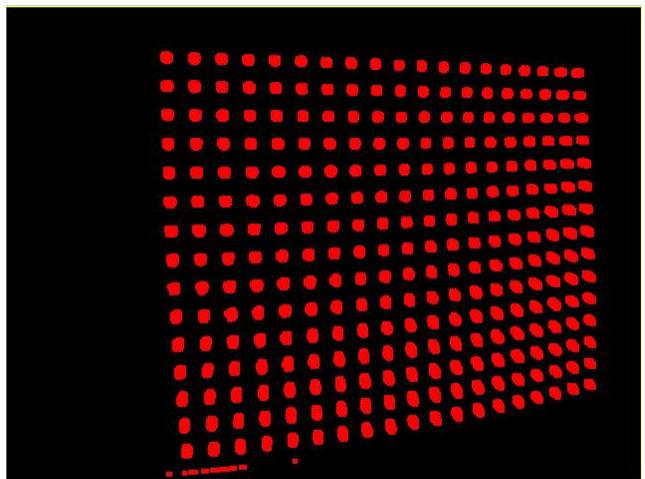
Figure 6.3.   Result of removing background.

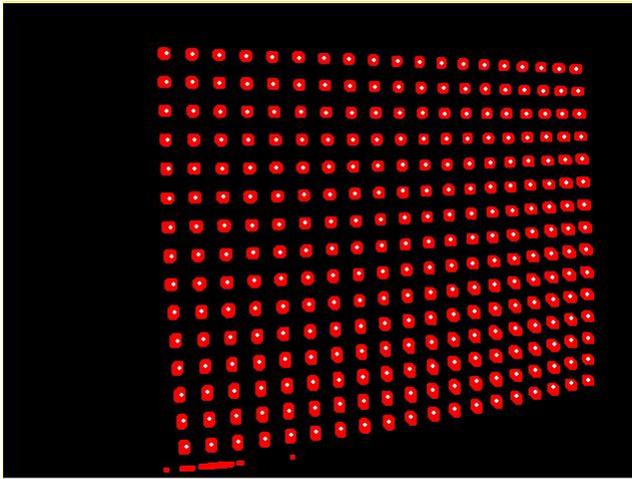Figure 6.4. Result after filling gaps.

Figure 6.5. Detected dots.

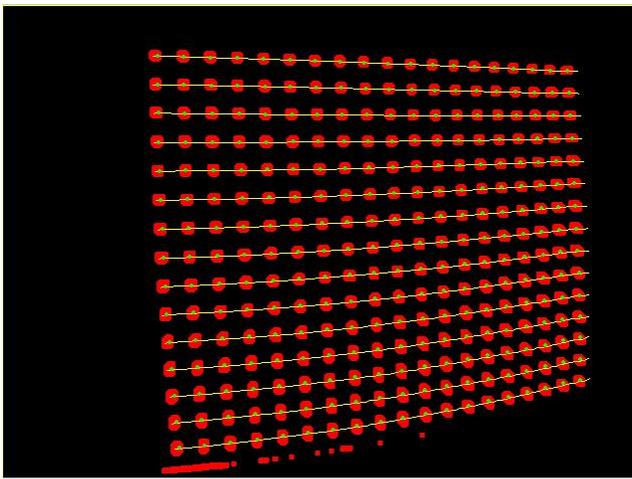Figure 6.6. Detected dots and original image.

Figure 6.7. Detecting dots using detected lines.

8. Using LCD as touch screen

Now that initialization is done we can start using LCD as touch screen. By using finger we set cursor position. While finger is moving around nothing will happened. When finger becomes stationary left mouse click will be initiated. If finger is not moved after left click was done double left click will be initiated.

Web Cam constantly takes images of LCD. Example of such image is shown on figure 8.1.

Such image is then transformed into normalized RGB color space shown on figure 8.2. Normalized RGB coordinates contain only color information, unlike RGB coordinates which also contain intensity information.

Such image is thresholded using skin color as threshold parameter. Result is shown in image 8.3.

After that isolated pixels are removed from the image as shown in figure 8.4.

Small gaps in the image are then filled. Result is shown in figure 8.5.

Algorithm now searches for different cluster and takes only those that have more pixels then some predefined value. In this particular case this will result in two big clusters. One belongs to the hand and the other to the brown wall on the left and behind the LCD. Each of these two clusters is now taken separately and background is erased. In the case of wall cluster result is empty image. In the case of hand cluster result is shown in figure 8.6.

Resulting image is now analyzed line by line from top to bottom going left to right at each line until first non zero pixel is reached. Coordinates of such pixel are taken as finger coordinates.

Such coordinates are being memorized and if speed of the finger drops below predefined threshold clicking on left mouse button will be emulated.
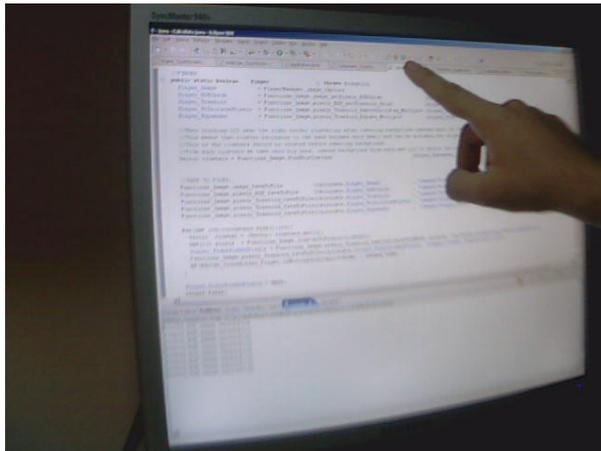

Figure 8.1. Image taken by Web Cam.

Figure 8.2. Normalized RGB image.

Figure 8.3. Threshold image.

Figure 8.4. Isolated pixels are removed.

Figure 8.5. Result after filling gaps.

Figure 8.6. Isolated hand cluster without background.

9. Algorithms

9.1. pixels_Treshold_FillGaps

This function will set each empty pixel if number of set pixels in its neighborhood is greater o equal to minNeighbours.

```
static public int[][] pixels_Treshold_FillGaps (int[][] pixels, int minNeighbours) {

  int    filledNeighbours = 0;
  int    lines            = pixels   .length;
  int    rows             = pixels[0].length;
  int[][] pixelsFilled    = new int[lines][rows];

  for(int l=1;l<lines-1;l++) {
    for(int r=1;r<rows-1;r++) {
      if    (pixels[l][r]==1) { pixelsFilled[l][r] = 1; }
      else {
        filledNeighbours=0;
        if(pixels[l-1][r-1]==1) { filledNeighbours++; }
        if(pixels[l-1][r  ]==1) { filledNeighbours++; }
        if(pixels[l-1][r+1]==1) { filledNeighbours++; }
        if(pixels[l  ][r-1]==1) { filledNeighbours++; }
        if(pixels[l  ][r+1]==1) { filledNeighbours++; }
        if(pixels[l+1][r-1]==1) { filledNeighbours++; }
        if(pixels[l+1][r  ]==1) { filledNeighbours++; }
        if(pixels[l+1][r+1]==1) { filledNeighbours++; }
        if(filledNeighbours>=minNeighbours) { pixelsFilled[l][r] = 1; }
        else                                { pixelsFilled[l][r] = 0; }
      }
    }
  }

  return pixelsFilled;
}
```

9.2. collectPixelNeghbours

For current pixel, which coordinates are defined with Point pixel and which belongs to set of pixels int[][] pixels, coordinates of all neighboring set pixels are saved to vector clusterPixels and are erased from int[][] pixels.

```
static public void  collectPixelNeghbours (int[][] pixels, Point pixel, Vector clusterPixels) {

  int l=pixel.l;
  int r=pixel.r;

  int maxl=pixels   .length-1;
  int maxr=pixels[0].length-1;

  if (l-1>=0    && r-1>=0    && pixels[l-1][r-1]==1) { clusterPixels.add( new Point(l-1,r-1) ); pixels[l-1][r-1]=0; }
  if (l-1>=0                 && pixels[l-1][r  ]==1) { clusterPixels.add( new Point(l-1,r  ) ); pixels[l-1][r  ]=0; }
  if (l-1>=0    && r+1<=maxr && pixels[l-1][r+1]==1) { clusterPixels.add( new Point(l-1,r+1) ); pixels[l-1][r+1]=0; }

  if (            r-1>=0    && pixels[l  ][r-1]==1) { clusterPixels.add( new Point(l  ,r-1) ); pixels[l  ][r-1]=0; }
  if (            r+1<=maxr && pixels[l  ][r+1]==1) { clusterPixels.add( new Point(l  ,r+1) ); pixels[l  ][r+1]=0; }

  if (l+1<=maxl && r-1>=0    && pixels[l+1][r-1]==1) { clusterPixels.add( new Point(l+1,r-1) ); pixels[l+1][r-1]=0; }
  if (l+1<=maxl             && pixels[l+1][r  ]==1) { clusterPixels.add( new Point(l+1,r  ) ); pixels[l+1][r  ]=0; }
  if (l+1<=maxl && r+1<=maxr && pixels[l+1][r+1]==1) { clusterPixels.add( new Point(l+1,r+1) ); pixels[l+1][r+1]=0; }
}
```

9.3. collectClusterPixels

This function is used to collect all pixels that belong to the same cluster. Arguments l and r are coordinates of first pixel which belong to the set of pixels in[][] pixels. Coordinates of each pixel belonging to the cluster are stored in Vector clusterPixels using function collectPixelNeghbours.

```
static public Vector  collectClusterPixels (int[][] pixels, int l, int r)                          {
  Vector clusterPixels = new Vector();
        clusterPixels.add(new Point(l,r));
  pixels[l][r]=0;
  int i=0;
  while(1==1){
    if(i>clusterPixels.size()-1) { break; }
    Point pixel = (Point) clusterPixels.get(i);
    Functions_Image.collectPixelNeghbours(pixels, pixel, clusterPixels);
    i++;
  }

  return clusterPixels;
}
```

9.4. findClusters

This function is used to identify all clusters in the image.

```java
static public Vector  findClusters            (int[][] pixels)                                              {

  int     lines     = pixels   .length;
  int     rows      = pixels[0].length;
  int[][] pixelsCopy = pixels_Copy(pixels);
  Vector  clusters   = new Vector();

  for(int l=1;l<lines-1;l++) {
    for(int r=1;r<rows-1;r++) {
      if(pixelsCopy[l][r]==1) { clusters.add(Functions_Image.collectClusterPixels(pixelsCopy,l,r)); }
    }
  }

  return clusters;
}
```

10.   Program Flow

– Development: H:\Installed\Programming\eclipse\Workspaces\Workspace\WebCam_TouchScreen_Finger

– Component_InitiScreen.java:
  – This is simple JComponent class which returns different initialization images of screen dimensions.
  – Class variable 'phase' defines which image should be created: White Screen, Lines or Dots.

– Component_InitializationResults.java:
  – This is simple JComponent class which returns drawing of dimensions 640x480.
  – Which drawing should draw is defined by constructor parameter String _code.
  – This JComponent can be simply added like this:
    ```
    panel.add(new Component_InitializationResults("drawBackground_Image"));
    ```

– Functions_Image.java:
  – This class contains functions that:
    – can get image from WebCam, save image to file or read it from file,
    – implement all algorithms for working with pixels from image or int[][],
    – draw Image or int[][] onto JComponent whose Graphics g is given as second parameter.
  – First parameter of each function is Image or int[][] on which function should work.
  – Such function can be simply called like this:
    ```
    Functions_Image.pixels_Treshold_Draw (Calculate.background_Treshold, g, null); }
    ```

– Calculate.java:
  – This class contains:
    – all images taken from WebCam,
    – all int[][] representing different transforamtons of those images.
    – functions used to analyse transformed initialization images in order to locate reference points.

10. Literature

1. http://www.javaworld.com/javaworld/jw-05-2001/jw-0504-jmf2-p2.html#resources
   Tutorials on using Java Media Framework.

2. http://java.sun.com/products
   Source for downloading JAVA & Java Media Framework.

3. http://www.eclipse.org
   Source for downloading Eclipse Integrated Environment Development.

4. "Fundamentals of Digital Image Processing", Anik K. Jain