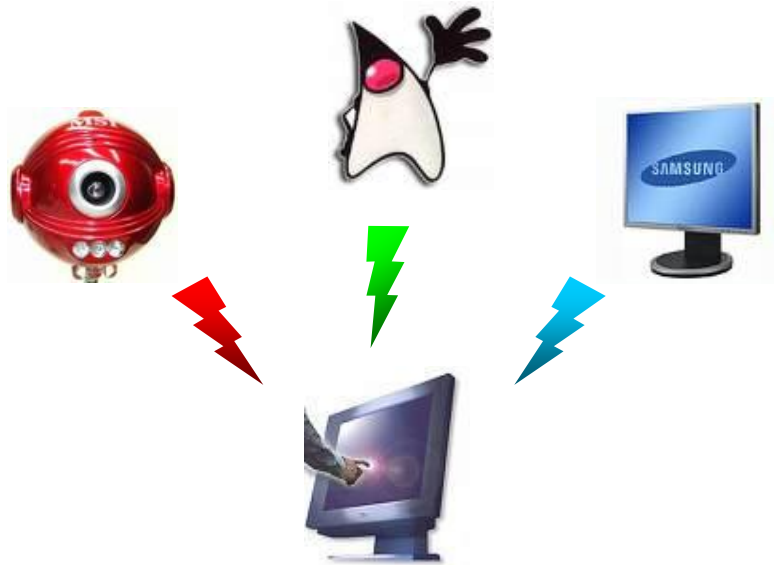


Adding touch screen ability for specialized applications
to normal monitor using Web Cam



Contents

1. [Introduction](#)
2. [Displaying Initialization Screens](#)
3. [Capturing Initialization Screens](#)
4. [Analyzing Screen Containing Background](#)
5. [Analyzing Screen Containing Lines](#)
6. [Analyzing Screen Containing Dots](#)
7. [Analyzing Screen Containing GUI](#)
8. [Using GUI](#)
9. [Algorithms](#)
10. [Literature](#)

1. Introduction

Goal of this project was to transform part of Monitor into region that reacts when touched. This will be done by using single Web Cam which will be filming that part of Monitor detecting motion which will then be transformed into adequate action like pressing a button. To demonstrate such behavior small calculator application was created as shown in figure 1.1.

Figure 1.2 shows an image taken from Web Cam while using the application.

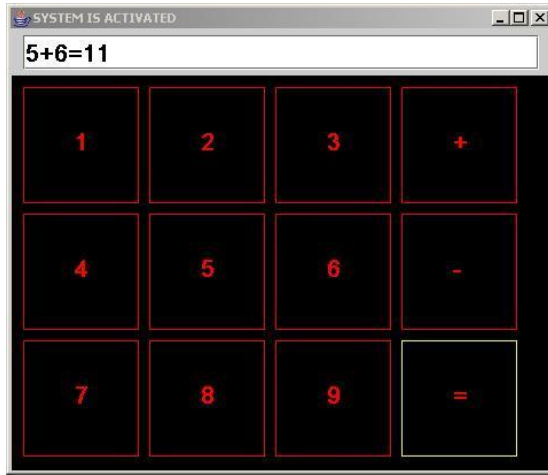


Figure 1.1
Calculator GUI displayed on LCD.

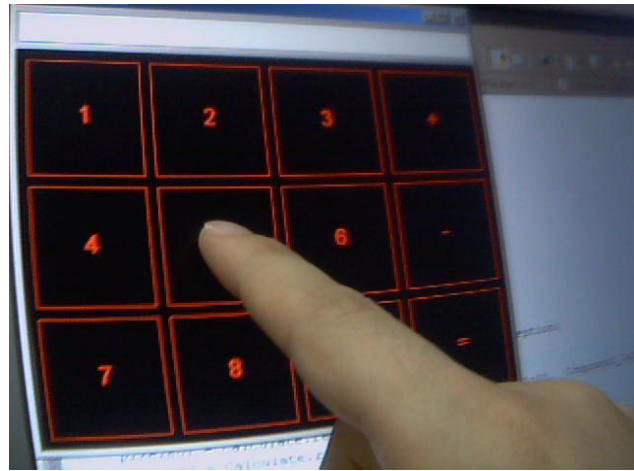


Figure 1.2.
Image taken by Web Cam while using GUI.

During tests on which this seminar is based, application was displayed in the top left corner of monitor as shown in figure 1.2.. Web Cam was positioned left from monitor on an elevated platform. There are no restrictions as to where the application can be displayed as long as Web Cam is mounted in the way to be able to see the application.

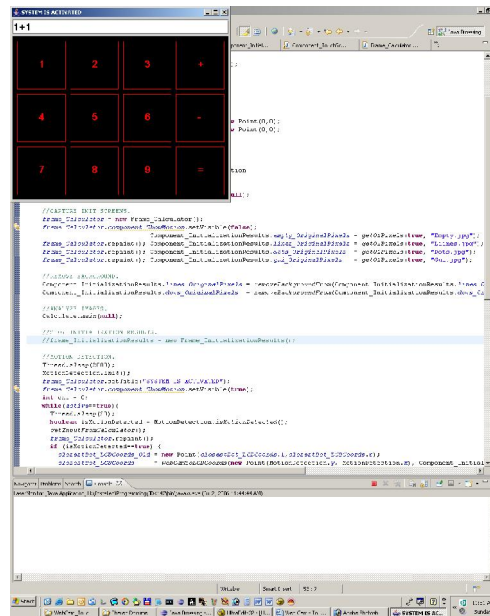


Figure 1.3
Contents of LCD while using GUI.

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.

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 four pictures on the part of LCD which we want to transform into touch screen.

Area is first filled with red 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.

Then area is filled with lines as shown in figure 2.2. These 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.

At the end GUI is displayed as shown in figure 2.4. This picture together with the one shown in figure 2.1. will be used to filter out background from images taken from Web Cam.



Figure 2.1.
Area is filled with red color.



Figure 2.2.
Area is filled with lines.

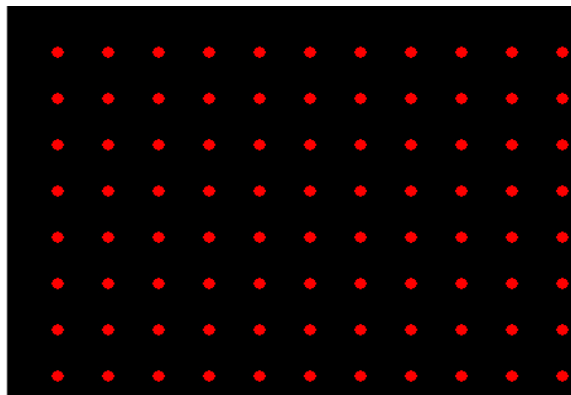


Figure 2.3.
Area is filled with dots.



Figure 2.4.
GUI is displayed.

3. Capturing Initialization Screens

Each of these pictures are captured by Web Cam. Web Cam captures not only the region which will be Touch Screen enabled but also the surrounding pixels and even parts of the environment which are outside LCD. On following figures are shown images captured by Web Cam at the moments when each of the above initialization pictures were 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 as collection of tabs as shown in figure 3.5.

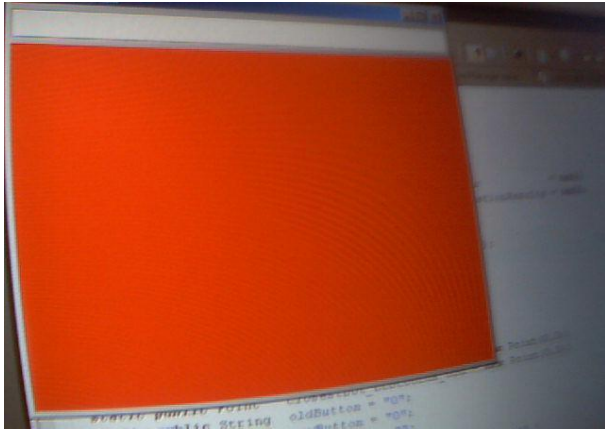


Figure 3.1.
Image taken from Web Cam when the whole touch screen area was red.

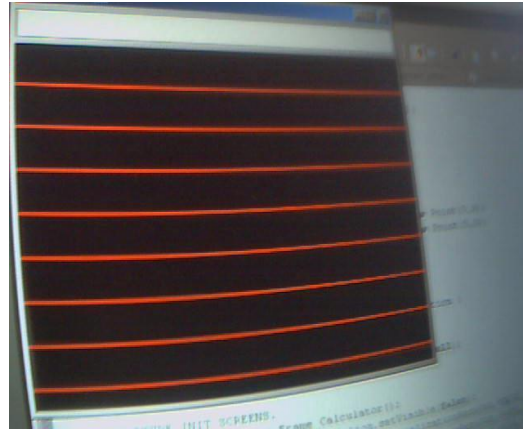


Figure 3.2.
Image taken by the Web Cam while the area was filled with lines.

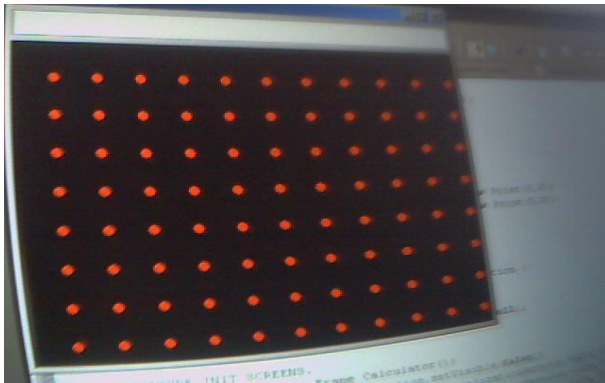


Figure 3.3.
Image taken by the Web Cam while the area was filled with dots.

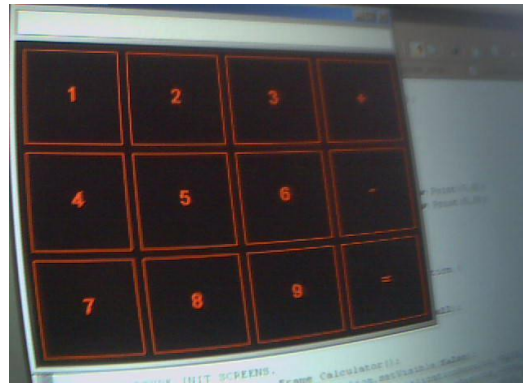


Figure 3.4.
Image taken by the Web Cam while GUI was displayed.

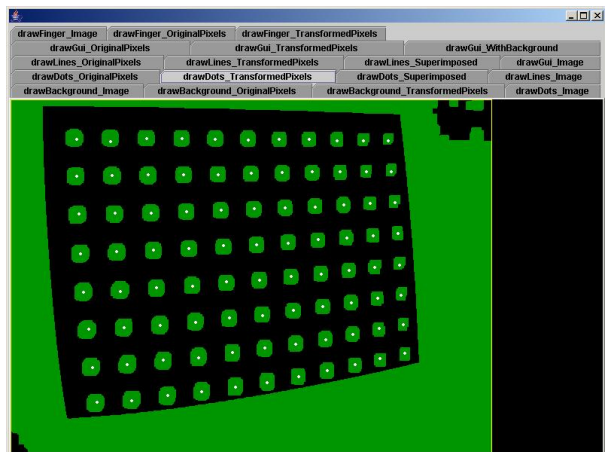


Figure 3.5.
Frame containing different stages of image processing.

4. Analyzing Screen Containing Background

Image taken when the whole touch screen area was red is once 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 1 only if equivalent pixel in original image has red component which is greater than 90 and at the same time 2 or more times higher from blue and green component. This results in an image shown on figure 4.2.

After that removeIsolatedPixels algorithm is used to remove noise.

Then image is inverted after which parts of image which are background are set to 1. This is shown on figure 4.3.

At the end fillGaps algorithm is used to fill possible gaps in background region. Result is shown in figure 4.4. Resulting image will be used to eliminate background from GUI calculator.

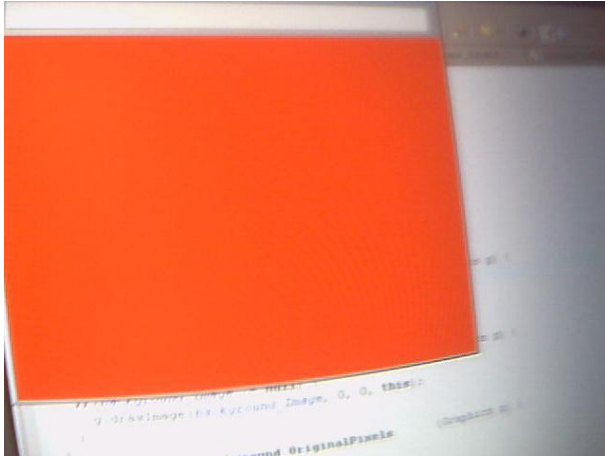


Figure 4.1.

Image taken from Web Cam when the whole touch screen area was red.

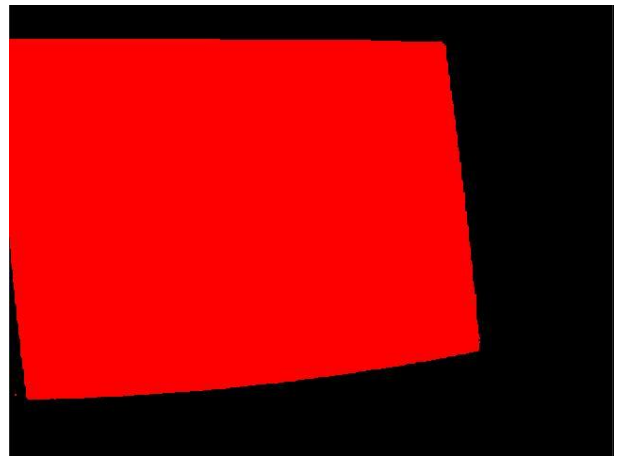


Figure 4.2

Pixels with adequate values of their color components are set to 1.

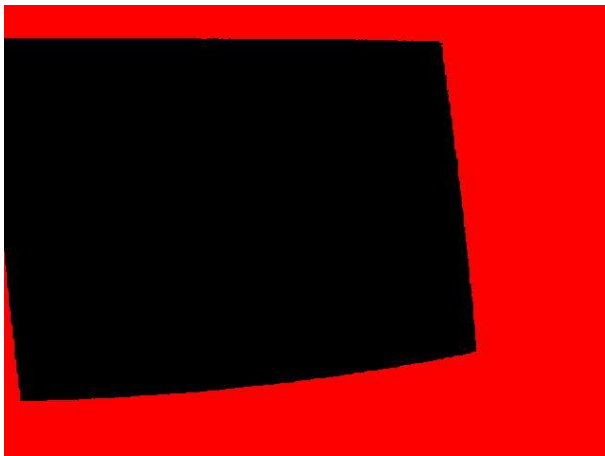


Figure 4.3.

Inverted background.

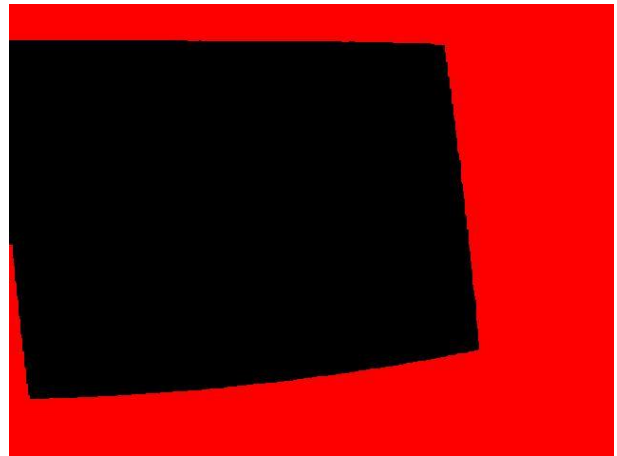


Figure 4.4.

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

This image is then used to create new image, using 2D array, in the way that pixels in the new image are set to 1 only if equivalent pixel in original image has red component greater than 90 and at the same time 2 or more times higher from blue and green component. Result is shown on figure 5.2.

Resulting image is then transformed by dialating few times usigng fillGaps algorithm in order to fill the existing gaps in each line. Result is shown on figure 5.3.

Then the lines are trimmed from above and below until their height is reduced to one pixle. 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 hte middle of the lines. Now an algorithm is started which goes from that point to the left and right folowing line pixels until left and right line end are reached which are hihglighted with yellow dots at the end of the lines. Result is shown on figure 5.4.

Figure 5.5. shows how detected lines corespond to the original lines in the image taken by Web Cam.

Figure 5.6. additionally shows effects of dialteing lines to remove small gaps which might negativly influence trimming of the lines and therefore detecting the lines.

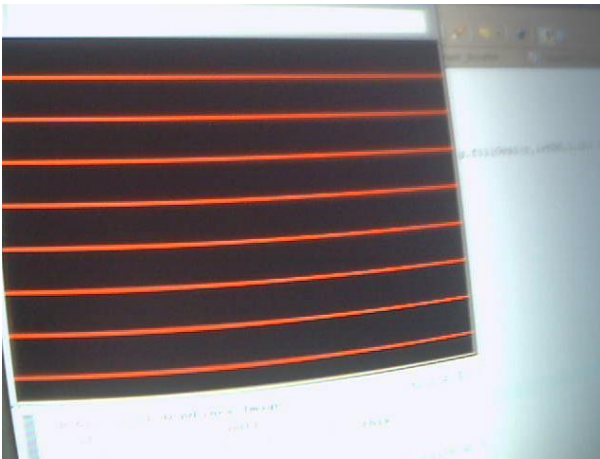


Figure 5.1.
Image taken by the Web Cam while the area was filled with lines.

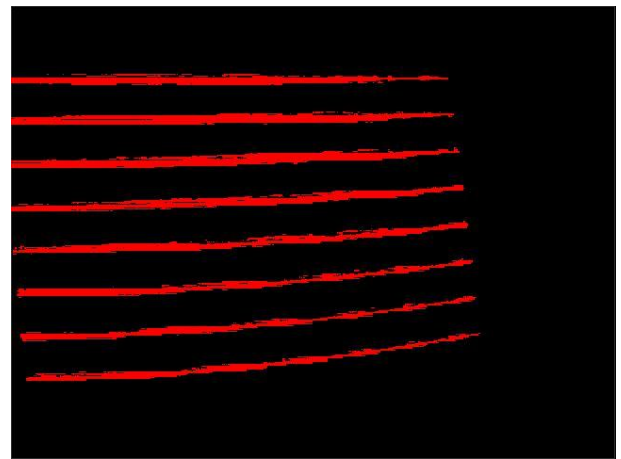


Figure 5.2.
Pixels with red component greater the 50 are set to 1.



Figure 5.3.
Dialated lines to eliminate gaps.

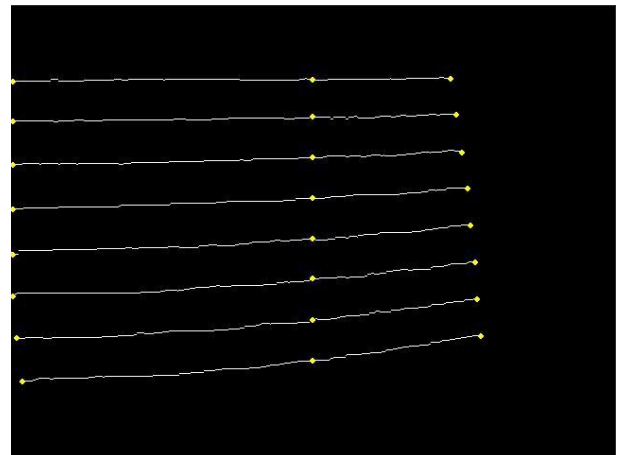


Figure 5.4.
Detected lines.

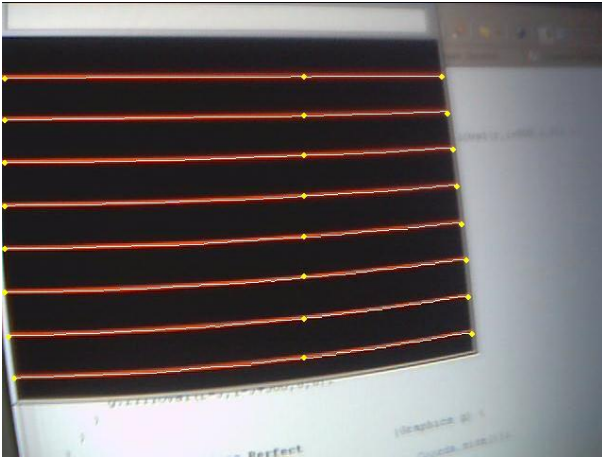


Figure 5.5.
Detected lines and original image taken by Web Cam.

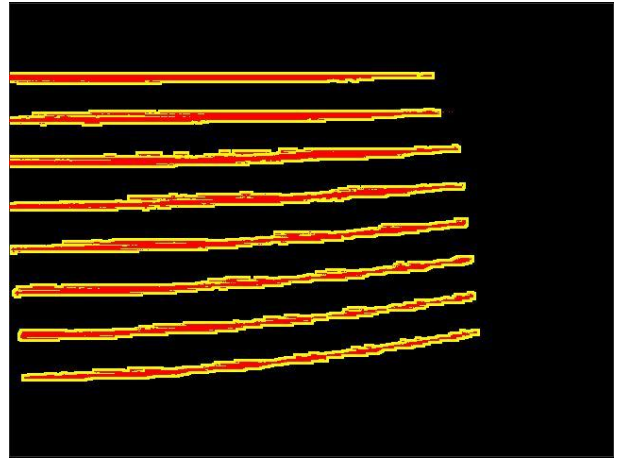


Figure 5.6.
Effects of dialating lines.

6. Analyzing Screen Containing Dots

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

This image is then used to create new image, using 2D array, in the way that pixels in the new image are set to 1 only if equivalent pixel in original image has grayscale value greater than 150. Result is shown on figure 6.2.

Resulting image is transformed by removing isolated pixels using removeIsolatedPixels algorithm. Then image is dilated few times, using fillGaps algorithm, in order to fill the existing gaps in each dot. Result is shown on figure 6.3.

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 these two values to approximate center of the dot. This is how the white dots are calculated.

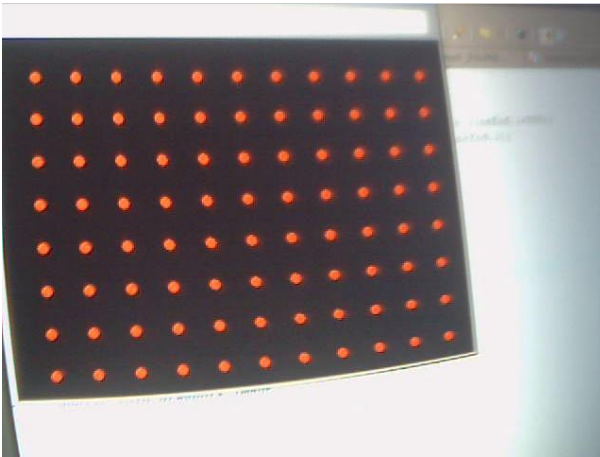


Figure 6.1.
Image taken by the Web Cam while the area was filled with dots.

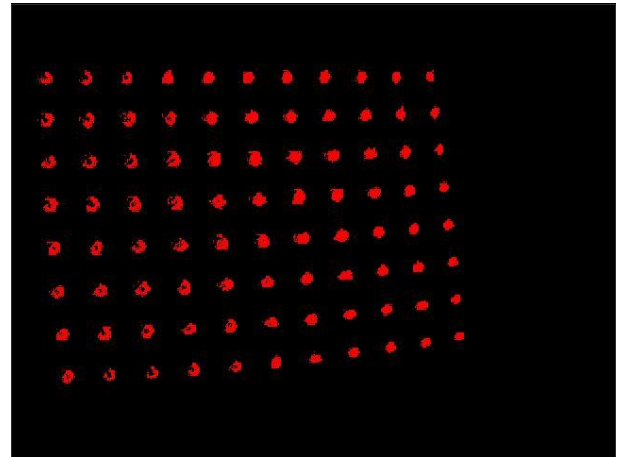


Figure 6.2.
Pixels with red component greater than 50 are set to 1.

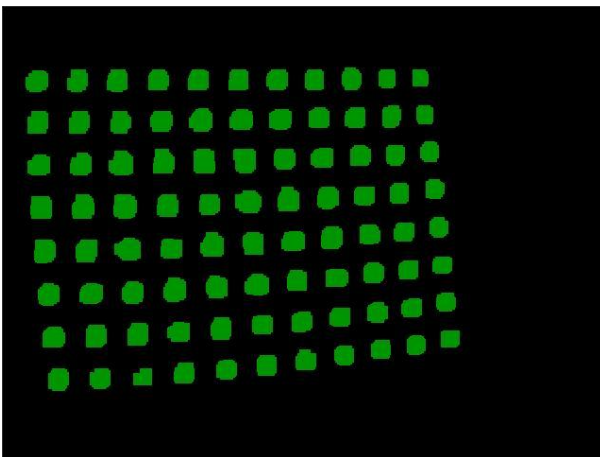


Figure 6.3.
Dilated dots to eliminate gaps.

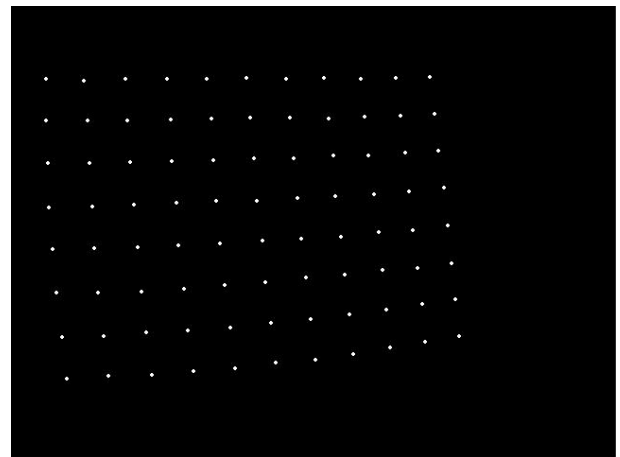


Figure 6.4.
Located dots.

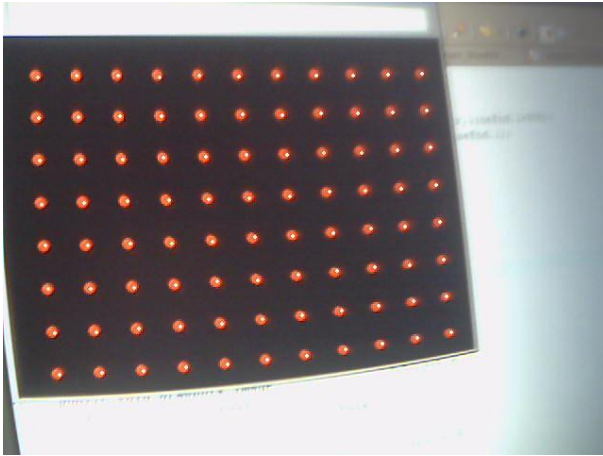


Figure 6.5.
Detected dots and original image taken by Web Cam.

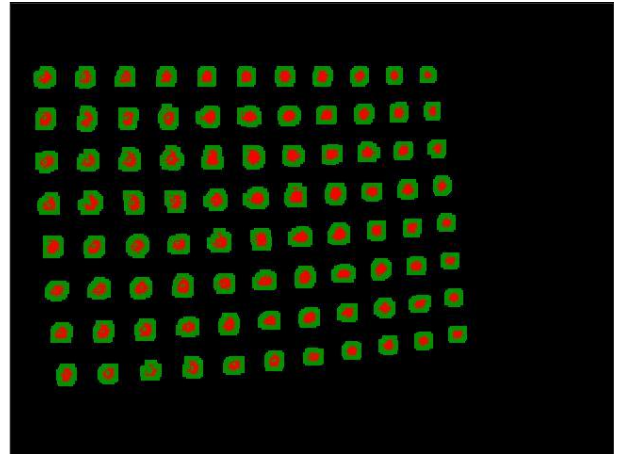


Figure 6.6.
Effects of dialating dots.

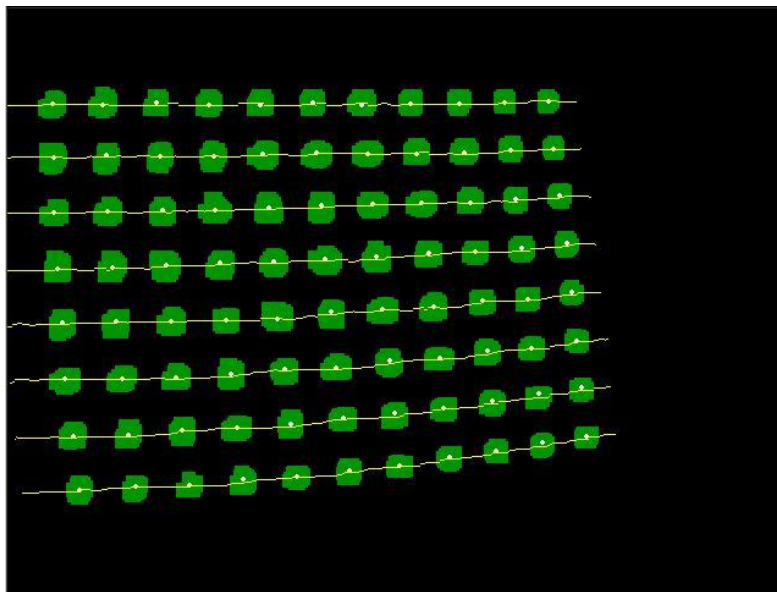


Figure 6.7.
Detecting dots using detected lines.

7. Analyzing Screen Containing GUI

Image taken by the Web Cam while GUI was displayed is once again shown on the figure 7.1.

This image is then used to create new image, using 2D array, in the way that pixels in the new image are set to 1 only if equivalent pixel in original image has red component greater than 50. Result is shown on figure 7.2.

Resulting image is transformed by dialating using fillGaps algorithm few times in order to fill the existing gaps. Result is shown on figure 7.3.

Figure 7.4. shows previous image on which background is superimposed. Resulting image presents background which is constant throughout the usage of the application.

Figure 7.5. additionally shows effects of dialating GUI to remove small gaps.



Figure 7.1.

Image taken by the Web Cam while GUI was displayed.

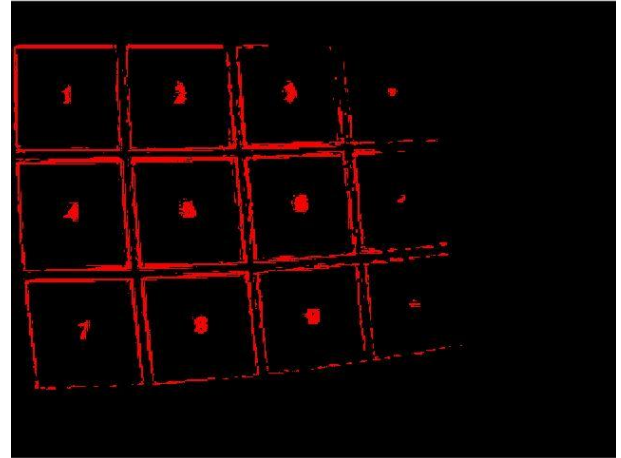


Figure 7.2.

Pixels with red component greater the 50 are set to 1.

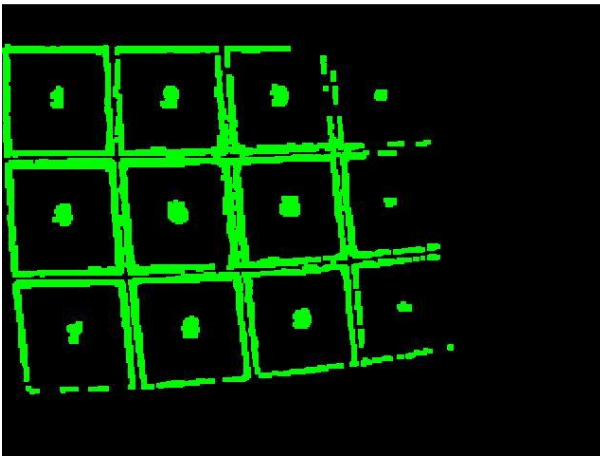


Figure 7.3. Dialated GUI to eliminate gaps.

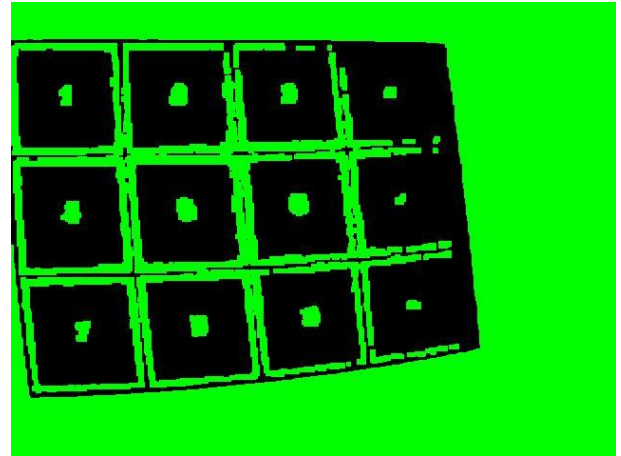


Figure 7.4. GUI with background.

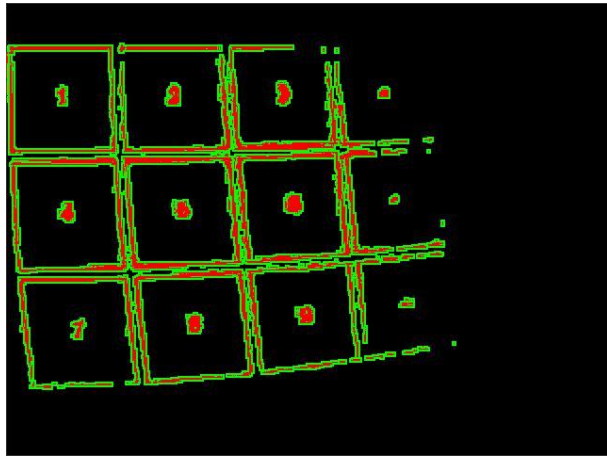


Figure 7.4. Effects of dialating GUI.

8. Using GUI

Now that initialization is done we can start using the application. By using finger we can select buttons to create a mathematical expression. By selecting "=" expression is evaluated. Detection of pressed button is done in the following way.

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

Such images are converted into image where only pixels which red component is greater than 150 are set to 1 as shown in figure 8.2.

After that each pixels belonging to the background defined in figure 7.4. is erased. Noise is reduced by using fillgaps algorithm and the result is shown in figure 8.3.

Resulting image is now analysed line by line from top to bottom going left to right on each line until first non zero pixel is reached. Coordinates of such pixel are then taken as finger coordinates as shown in figure 8.3. as white dot.

Such coordinates are being memorized and if speed of the finger drops below predefined threshold button will be pressed. This way buttons will not get pressed just because the finger was above it while traveling to the destination button. Button which was pressed will become highlighted as shown in figure 1.1.

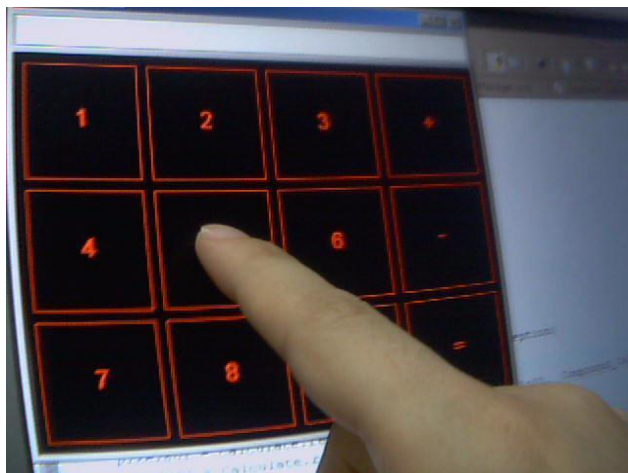


Figure 8.1.

Image taken by the Web Cam while using GUI.



Figure 8.2.

Pixels with value greater than threshold are set to 1.



Figure 8.3.

Thresholded image without background.

9. Algorithms

9.1. lines_TrimFromBelow

Image is analysed line by line going from top to bottom line and from left to right pixel.

If current pixel is 1 and it's lower neighbour is also 1, leave current pixel to 1 since it is not at the lower border of the line.

If current pixel is 1 but both upper & lower neighbours are 0, leave current pixel to 1 since the line would break if we erase it.

In all other cases erase current pixel.

```
public static int[][] lines_TrimFromBelow (int[][] pixels) {

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

    for(int l=1;l<lines-1;l++) {
        for(int r=1;r<rows-1;r++) {
            if (pixels[l+1][r]==1 && pixels[l][r]==1 ) { correctedLines[l][r] = 1; }
            else if (pixels[l-1][r]==0 && pixels[l][r]==1 && pixels[l+1][r]==0 ) { correctedLines[l][r] = 1; }
            else { correctedLines[l][r] = 0; }
        }
    }
}
```

9.2. lines_TrimFromAbove

Image is analysed line by line going from top to bottom line and from left to right pixel.

If current pixel is 1 and it's upper neighbour is also 1, leave current pixel to 1 since it is not at the upper border of the line.

If current pixel is 1 but both upper & lower neighbours are 0, leave current pixel to 1 since the line would break if we erase it.

In all other cases erase current pixel.

```
public static int[][] lines_TrimLineAbove (int[][] pixels) {

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

    for(int l=1;l<lines-1;l++) {
        for(int r=1;r<rows-1;r++) {
            if (pixels[l-1][r]==1 && pixels[l][r]==1 ) { correctedLines[l][r] = 1; }
            else if (pixels[l-1][r]==0 && pixels[l][r]==1 && pixels[l+1][r]==0 ) { correctedLines[l][r] = 1; }
            else { correctedLines[l][r] = 0; }
        }
    }

    return correctedLines;
}
```

9.3. fillGaps

If current pixel or any of it's neighbours is 1, set current pixel to 1.

```
static public int[][] fillGaps(int[][] pixels){

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

    for(int l=1;l<lines-1;l++) {
        for(int r=1;r<rows-1;r++) {
            if (pixels[l-1][r-1]==1 || pixels[l-1][r]==1 || pixels[l-1][r+1]==1 ||
                pixels[l ][r-1]==1 || pixels[l ][r]==1 || pixels[l ][r+1]==1 ||
                pixels[l+1][r-1]==1 || pixels[l+1][r]==1 || pixels[l+1][r+1]==1 ) {correctedLines[l][r]=1;}
            else { correctedLines[l][r] = 0; }
        }
    }

    return correctedLines;
}
```

9.4. removeIsolatedPixels

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

    int cnt = 0;
    int lines = pixels.length;
    int rows = pixels[0].length;
    int[][] correctedLines = new int[lines][rows];

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

    return correctedLines;
}
```

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