MACHINE VISION
MINI PROJECT
RICE QUALITY TESTING
MEX5271

GROUP MEMBERS

P. A. HEWAPATHIRANA 613344004

W. I. A. PERERA 613345350

N. DADALLAGE 713344839

SUBMITTED DATE : 18/10/2017

CENTER : COLOMBO

TABLE OF CONTENT

LIST OF ILLUSTRATIONS

1. INTRODUCTION

Rice is the single most important food in the Asian countries, especially in Sri Lanka. Averagely rice cultivation occupies 37% from the total cultivation in Sri Lanka. For the global demand, from our country, rice is exported to the other countries (approximately 1000 MT) and global rice demand is increase by 1.95% annually. More people consume rice as food, so demand is getting increased day by day. Due to this the quality of the exported and even the locally circulating rice should have a good standard. Else the local demand will be reduced and more will go after importing rice which will not so great for the economy of the country with the current economic situation. So doing a thorough quality testing has become a must.

For the purpose of testing the quality of the rice grain the image processing techniques can be used. The quality of the rice grain is based on the several parameters. Such as,

- Grain color
- Shape
- Size.

Machine vision systems are used to identify the quality of the grain. Digital images and videos are the key sources of the machine vision systems.

The task to advance through this mini project is to develop a vision based system which capable to predict broken rice percentage in a given rice sample. In addition to that the application develop to a mobile application which fulfils the modern requirements in mobile platforms and to be flexible to test out the quality of the rice anywhere you are, other than using a bulky setup to identify the quality of the rice sample at the palm of your hand.

2. AIM

To develop a vision based system and mobile application to identify broken percentage of a given rice sample.

3. OBJECTIVES

- To develop a vision based system.
- To develop a mobile application for the task.

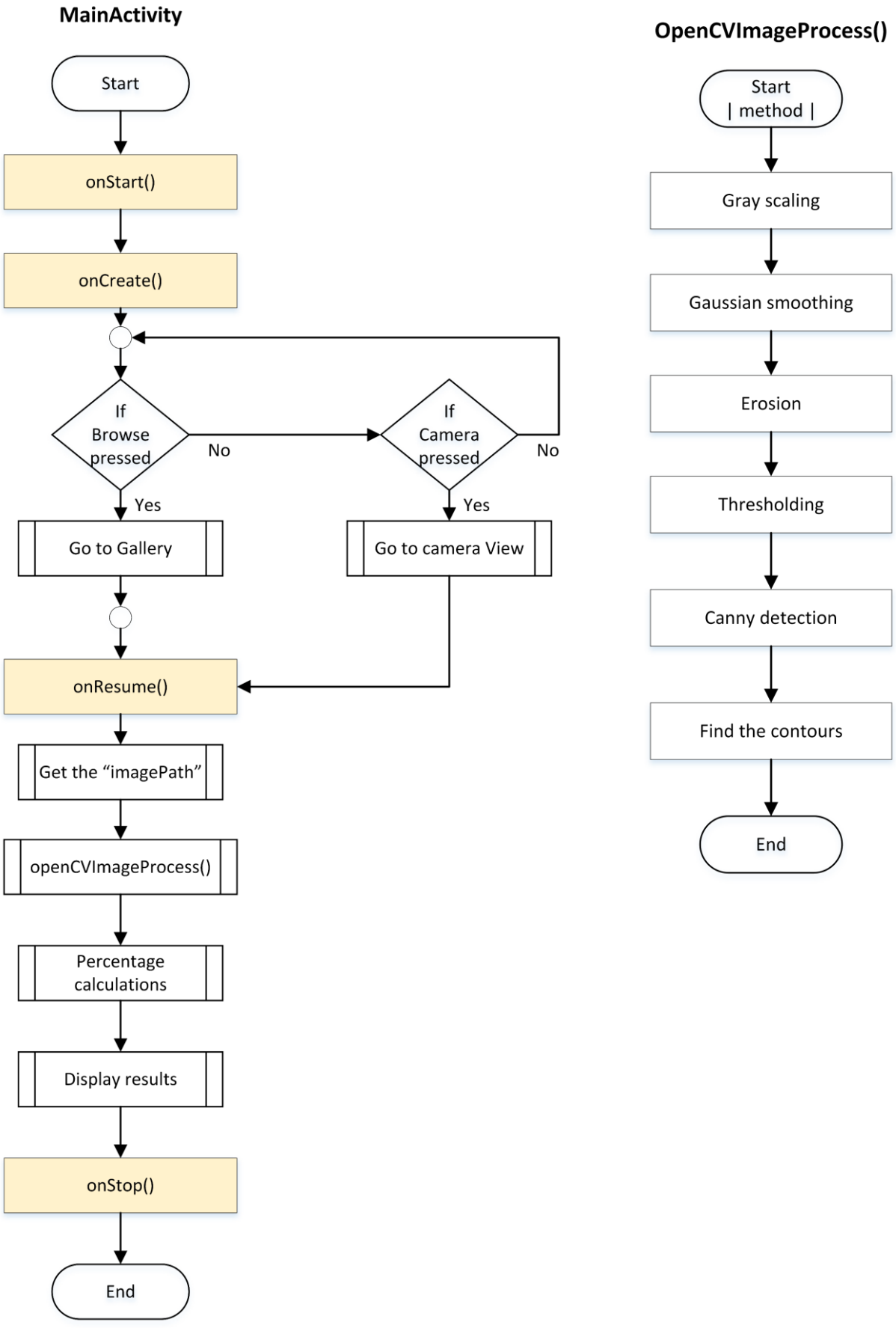
4. GOALS

- To develop a mobile application.
- To develop an algorithm which occupies the decision making from final output.

5. TECHNOLOGIES USED

- Development Environments,
    - Visual studio [5.1]- for debugging and developing.
    - Android studio [5.2] - for the mobile development.
- Libraries used,
    - OpenCV [5.3]
- Testing Environments
    - MS Core.NET environment – for debugging and testing.
    - Genymotion Emulator [5.4]– for mobile testing.
    - Android 6.0 (API 23) mobile device – for testing.

6. FLOW CHART

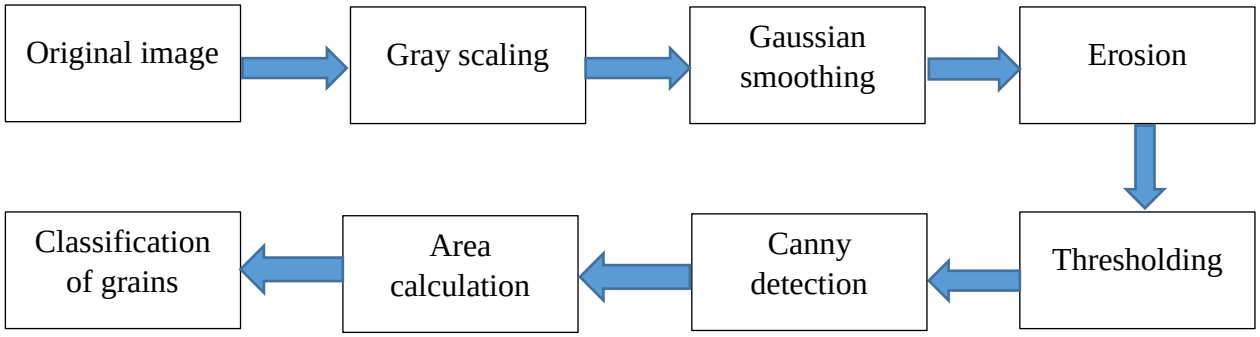

MainActivity
Start
onStart()
onCreate()
If Browse pressed
No
Yes
Go to Gallery
If Camera pressed
No
Yes
Go to camera View
onResume()
Get the “imagePath”
openCVImageProcess()
Percentage calculations
Display results
onStop()
End
OpenCVImageProcess()
Start | method |
Gray scaling
Gaussian smoothing
Erosion
Thresholding
Canny detection
Find the contours
End

7. METHODOLOGY

➢ Assumptions made out.

- There should be a considerable difference between the grain colors and the background.
- There should be at least one proper grain.
- There should be a proper light condition during the task.
- Grains must not be overlapped with one another.

The system development process



7.1. Vision based system development.

- **Gray scaling**

  Grayscale is range of monochromatic shades from black to white. Therefore, a grayscale image contains only shades of gray and no color. The process of gray scaling removes all color information, leaving only the luminance of each pixel.

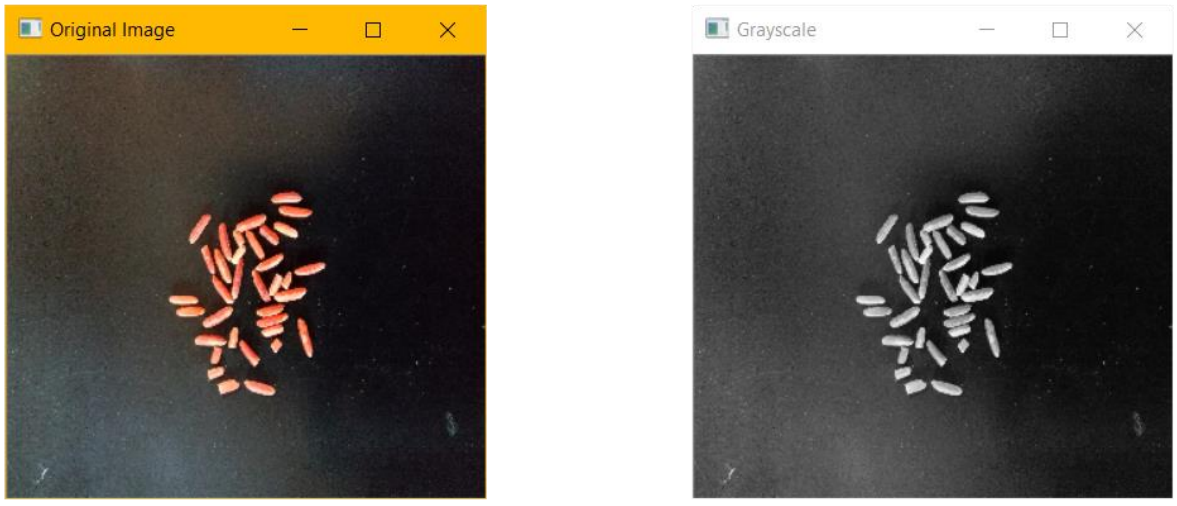


Figure 6.1 – Original Image (Left) and the Gray scaled image (Right)

Code Snippet

```
//Create a Mat image from image path
Mat src = imread(picPath, CvType.CV_8UC4);
Mat srcEdited = new Mat();

//Convert Original src to grayscale
Imgproc.cvtColor(
        src,                        // input image
        src,                        // output image
        Imgproc.COLOR_BGR2GRAY);    //conversion method
```

- **Gaussian smoothing**

Gaussian filtering is used to blur images and remove noise and detail. When working with images we need to use the two dimensional Gaussian function. This is simply the product of two 1D Gaussian functions and is given by,

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-(\frac{x^2+y^2}{2\sigma^2})}$$

Figure 6.2 – Gray scaled Image (Left) and the blurred image (Right)

Code Snippet

```
//openCV Size variable
org.opencv.core.Size s = new Size(9,9);
//Apply Gaussian Blur
Imgproc.GaussianBlur(
        src,                // input image
        srcEdited,          // output image
        s,                  // Mask size
        1.5);               // Sigma value of Gauss mask
```

- **Erosion**

The most basic morphological operation in image processing is erosion which removes pixels on object boundaries.

Figure 6.3 – blurred Image (Left) and the eroded image (Right)

Code Snippet

```
//openCV Size variable
org.opencv.core.Size s = new Size(21,21);
Mat kernal = Imgproc.getStructuringElement(
        Imgproc.MORPH_ELLIPSE,              // shape
        s,                                  // k size
        org.opencv.Point(1,1));             // anchor position within the
                                                element

//Erode
Imgproc.erode(
        srcEdited,          // input image
        srcEdited,          // output image
        kernal);            // kernal for erosion
```

- **Canny detection**

The canny detector was developed by John F. Canny in 1986. Also known as the optimum detector, canny algorithm aims to satisfy three main criteria,

1. Low error rate : Means a good detection of only existing edges.
2. Good localization : The distance between edge pixels detected and real edge pixels have to be minimized.
3. Minimal response : Only one detector response per edge.

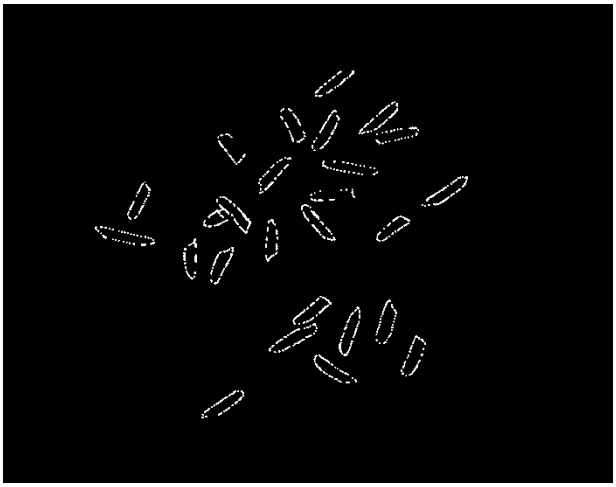


Figure 6.4 – detected contours in the image

Code Snippet

```
Imgproc.Canny(
        srcEdited,              // input image
        srcEdited,              // output image
        50,                     // threshold
        255);                   // max threshold

List<MatOfPoint> contours = new ArrayList<>();
Mat hierarchy = new Mat();      // hierarchy mat
Imgproc.findContours(
        srcEdited,              // input image
        contours,               // output vector
        hierarchy,              // output array of Mat
        Imgproc.RETR_TREE,      // contour retrieval mode
        Imgproc.CHAIN_APPROX_SIMPLE);
```

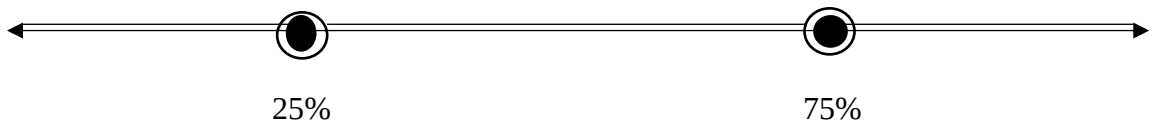
- **Area Calculation**

Code Snippet

```
int counter = 0;
double[] area = new double[200];

for (int contourIdx = 0; contourIdx < contours.size(); contourIdx++) {

    //get the area
    area[counter++] = Imgproc.contourArea(contours.get(contourIdx));
    Imgproc.drawContours(src, contours, contourIdx, new Scalar(255, 0, 0), 5);

    // debug logger
    Log.d(TAG, "area: " + Double.toString(area));

}
```

7.2. Algorithm development for decision making

- Area of "i"th grain = $A_i (1 \leq i \leq n)$
- Maximum grain size in the sample = A_{max}
- Decision parameter $(\beta) = \frac{A_i}{A_{max}} \times 100\%$



- $\beta = \begin{cases} \beta \leq 25\%, proper\ grains \\ 25\% \leq \beta \leq 75\%, Average\ grains \\ 75\% \leq \beta, poor\ grains \end{cases}$

Code snippet

```
int sum = 0;
int counter = 0;
for (int i = 0; i < val; i++) {
	if (area[i] > 0) {
		sum += area[i];
		counter++;
	}
}

sum = sum / counter;

int minVal = 0, midVal = 0, maxVal = 0;
int maxArea = area[counter - 1];

for (int i = 0; i < counter; i++) {
	//float newArea = area[i];
	int grainAreaPara = ((float)area[i] / (float)maxArea) * 100;

	if (grainAreaPara >= 75) { maxVal++; }
	else if (grainAreaPara >= 25) { midVal++; }
	else { minVal++; }
}

if ((maxVal > midVal) && (maxVal > minVal))
	cout << ((float)maxVal / (float)counter) * 100;
if ((midVal > maxVal) && (midVal > minVal))
	cout << ((float)midVal / (float)counter) * 100;
if ((minVal > maxVal) && (minVal > midVal))
	cout << ((float)minVal / (float)counter) * 100;
```

8. GUI (GRAPHICAL USER INTERFACE)

GUI – Main Activity

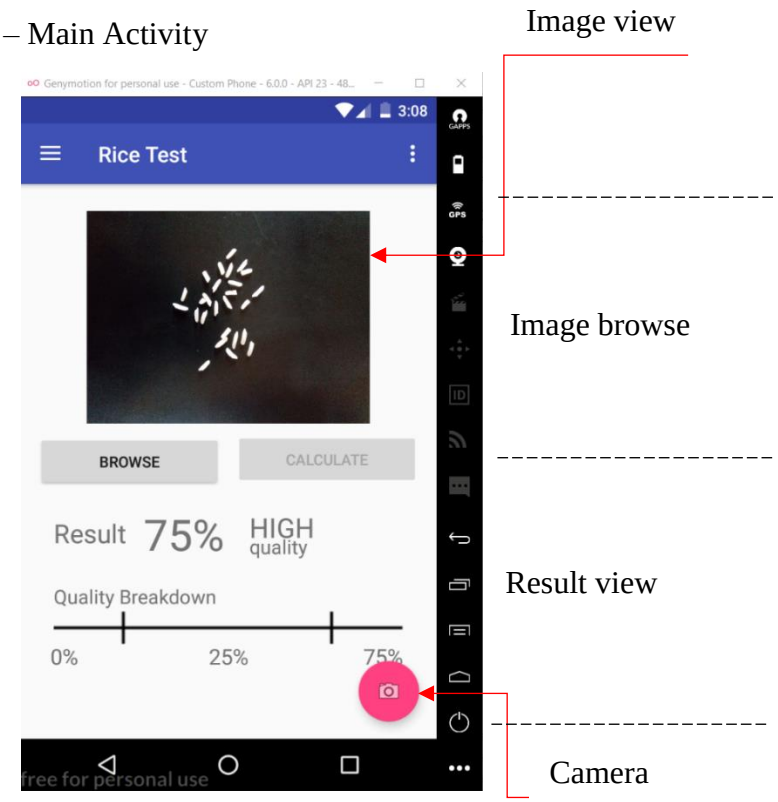


Figure 7.1 – UI of the Main Activity

GUI – Settings Activity

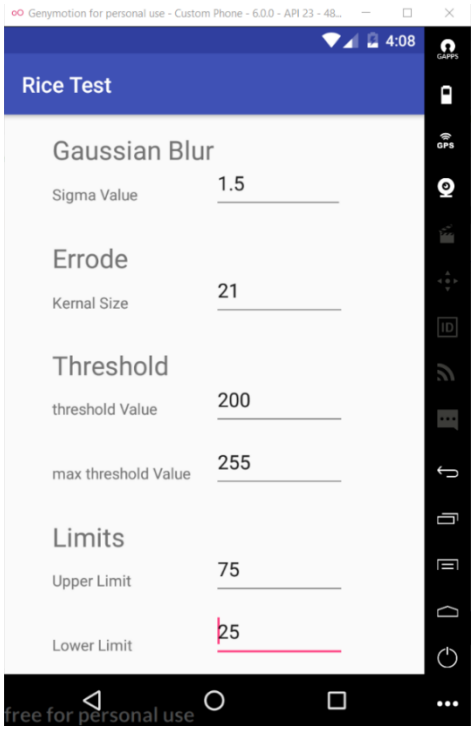


Figure 7.2 – UI of the Settings Activity

9. CONCLUSION

- To avoid the effect of overlapping of grains we used erosion method. But it's not that successful cause it takes much time for processing.

- Results of the system depends on illumination, background color and processing ability of the mobile device.

- If all the grains are broken and if the grains are mixed with another impurity the final result will be false.

- In a real time, application there could be some other grain types which has different colors and some other irregularities so if the system can train using AI based system on supervised learning method then the system may able to provide solution by identifying the different grain types even.

- Today mobile developments are widely used for many applications so developing this kind of applications will demand the quality of the rice as the staple food in Sri lanka.

Figure 8.1 – Before (Left) and after (Right) applying Erosion

10. REFERENCES

5.1. Visual Studio

https://www.visualstudio.com/

5.2. Android Studio

https://developer.android.com/studio/index.html

5.3. OpenCV library

https://opencv.org/

5.4. Genymotion

https://www.genymotion.com/

For the complete Project files,

https://github.com/Poornamith/RiceQualityTesting