Software Code Review Document

Phool Chandra : 160101051

Savinay :160101062

Savsani kevin :160101063

INDEX

1. INTRODUCTION

This document contains the complete software review of our app Food nutrients visualization . Code review for a model is carried out after the module is successfully compiled and the all the syntax errors have been eliminated .Code review are extremely cost-effective strategies for reduction in coding errors and to produce high quality code. Normally , two types of reviews are carried out on the code of a module .These two types of code review techniques are code inspection and code walk through .The code testing team in isolation tests different units and modules of the system . Different members of the code testing team have submitted their reports . Although , we are performing the code inspection part wherein each member goes through code to discover some common types of errors caused due to oversight and improper programming .

The main objective of the walk through was to discover the algorithmic and logical errors in the code . The members noted down their findings , which were discussed in a walk through meeting where the coder of the module were also present.

2. CODE

There are 2 modules in code : MainActivity and CameramodeActivity

2.1 MAIN ACTIVITY

```java
/**
Name of the module : MainActivity.java

Date on which the module was created : 20/04/2018

Author's name : Phoolchandra

Modification history : by Savinay  21/04/2018
                     : Savsani kevin 22/04/2018

Synopsis of the module : Main File which is executed when the app is started

Different functions supported along with their input/Output parameters.

Globel variables accessed/modified by the module.

   classes : MainActivity{}

   functions : onCreate(Bundle savedInstanceState )
               CameraMode(View view)



 */

package vrlab.foodui;   // package name

/*************** imported modules   *************************/
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
/***************** End importing modules *************/

 /********** MainActivity class **********/
 public class MainActivity extends AppCompatActivity{

    // inetnt variable used to call new activity
    private Intent callonGetImageListener; // callonGetImageListener used to call onGetImageListener

    // onCreate runs when activity is created
    @Override
    public void onCreate(Bundle savedInstanceState ){
        /**
        * It loads the saved instances to app and link listview to backend
        */
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main); // take the view from activity_main.xml file
    }

    // Function to call onGetImageListener activity
    // It is executed when user click the button CamerMode
    // view is an argument generated when a button is pressed
    public void onGetImageListener(View view){

        // variable callCameraModeActivity get the result from Intent Class
        callonGetImageListener = new Intent(this,onGetImageListener.class);

        //starting new activity
        startActivity(callonGetImageListener);
    }

 }

```

2.2 onGetImageListener

```
/**
Name of the module : onGetImageListener.java

Date on which the module was created : 20/04/2018

Author's name : Phoolchandra

Modification history : by Savinay  21/04/2018
                     : Savsani kevin 22/04/2018

Synopsis of the module : onGetImageListener file is executed when button named camera is pressed/clicked

Different functions supported along with their input/Output parameters.

Globel variables accessed/modified by the module.

 */


 /******************** START IMPORT *************************************/

package ssl.frnutrition;

import org.opencv.android.CameraBridgeViewBase.CvCameraViewFrame;
import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.android.CameraBridgeViewBase;
import org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2;
import org.opencv.core.MatOfFloat;
import org.opencv.core.MatOfInt;
import org.opencv.core.MatOfPoint;
import org.opencv.core.MatOfPoint2f;
import org.opencv.core.Point;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.imgproc.Imgproc;

import android.Manifest;
import android.animation.ObjectAnimator;
import android.animation.PropertyValuesHolder;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.view.WindowManager;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.Toast;

import com.oguzdev.circularfloatingactionmenu.library.FloatingActionButton;
import com.oguzdev.circularfloatingactionmenu.library.FloatingActionMenu;
import com.oguzdev.circularfloatingactionmenu.library.SubActionButton;
import com.yahoo.mobile.client.android.util.rangeseekbar.RangeSeekBar;

import java.util.ArrayList;
```

```java
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;
import java.util.Locale;

import ssl.ARNutrition.database.DbAccessFoodTypes;
import ssl.ARNutrition.detectableObjects.Food;
import ssl.ARNutrition.detectableObjects.FoodType;
import ssl.ARNutrition.information.NutritionalValue;


/***************************** END IMPORT *****************************************************/



public class onGetImageListener
        extends Activity
        implements CvCameraViewListener2,
        View.OnClickListener,
        View.OnTouchListener {

    // load opencv library
    static {
        System.loadLibrary("opencv_java3");
        System.loadLibrary("ARNutrition");
    }



    //// global constant section
    // constants for permissions
    private static final int CAMERA_PERMISSIONS = 0;

    // constants for development purposes only
    private static final boolean DEACTIVATE_HSV_MODE = true; //deactives the long hsv mode information toast
    private static final boolean ACTIVATE_COMPARISON = false;
    //activates use of testHistogramComparisonMethods

    // constants for clearer mode operations
    private static final int COLOUR_DETECTION_VIEW = 0;
    private static final int BORDER_DETECTION_VIEW = 1;
    private static final int KEYPOINTS_DETECTION_VIEW = 2;
    private static final int HSV_FILTER_VIEW = 3;

    // constants regarding tags for debugging purposes
    private static final String LABEL = "ARNutrition::ARNutritionActivity";
    private static final String DETECTION_MODE = "modeDetection";
    private static final String EDGE_DETECTION = "modeEdge";
    private static final String LABEL_FEATURES = "modeFeatures";
    private static final String HSV_FILTER_MODE = "modeHsvFilter";
    private static final String HIST_COMP_TESTS = "HistCompTests";
    private static final String DEBUG = "Debug";

    // constants regarding menu operations
    private static final int MAX_DETECTOR_MODE = 3;

    // histogram related settings
    private static final int HIST_SIZE = 256;  // size of histogram (value pairs) [max=256]
    private static final double HIST_BORDER_MIN = 0.07; // min value for autofit borders
    private static final int HIST_COMPARISON_METHOD = Imgproc.CV_COMP_CORREL;
    // used method for histogram comparison
    // histogram drawing related settings
    final int HIST_SETTING = 20;
    final int HIST_AMOUNT = 2 + 1;                // one more to save space for the buttons
```

```
133
134     // defines, which histograms should be compared - choose ONE
135     private static final boolean COMPARE_HUE = false;
136     private static final boolean COMPARE_SATURATION = false;
137     private static final boolean COMPARE_VALUE = true;
138
139     // threshold for comparison results
140     private static final double HUE_MIN = 0.85;
141     private static final double HUE_MAX = 1;
142     private static final double SATURATION_MIN = 0.75;
143     private static final double SATURATION_MAX = 1;
144
145     // constants for contour relating settings
146     private static final boolean DETECTED_DESIRED_CONTOUR = true;
147     private static final int CONTOUR_LEASTVALUE = 15000;   // threshold for found contours
148     private static final int CONTOUR_MaxValue = 300000;     // threshold for found contours
149     private static final double CONTOUR_DISTANCE = 5;     // distance between points of contour line
150
151     // constants relating tracking operations
152     private static final int FRAMES_TRACKING_RANGE = 10;
153     // amount of frames to consider in tracking
154
155     // constants related to the dynamically evenly generated hue scalar
156     private static final int HUE_PHASES = 5;
157     private static final int STEPS_PER_PHASE = HIST_SIZE /HUE_PHASES;
158
159     // constants related to morphing operations
160     private static final Size CV_ERODE = new Size(3, 3);
161     private static final Size CV_DILATE = new Size(5, 5);
162
163     // food label and information display related settings
164     private static final int INFORMATION_DISPLAY_VALUE = 5;
165     private static final int BUFFER_TEXT = INFORMATION_DISPLAY_VALUE;
166     private static final int INFORMATION_HEIGHT = 30;
167     private static final int INFORMATION_WIDTH = 30;
168     private static final int INFORMATION_THICKNESS = 5;
169     private static final int LABEL_THICKNESS = 3;
170     private static final int INFORMATION_X = 8;  // dig line into object, if no contour is applied (x)
171     private static final int INFORMATION_Y = 0;  // as above, (y)
172     private static final int TEXT_FONT_STYLE = 3;
173     private static final int TEXT_FONT_SCALE = 1;
174     private static final int TEXT_THICKNESS = 2;
175     private static final int TEXT_LINE = 0;
176     private static final int TEXT_MARGIN = 3 + LABEL_THICKNESS;
177
178
179
180
181     //// global variable section
182     // opencv specific variables
183     private CameraBridgeViewBase OpenCVCameraView;
184     private int OpenCVViewMode;
185
186
187     // various general mats
188     private Mat color;
189     private Mat HsvValue;
190     private Mat BorderMap;
191     private Mat IntermediateMat;
192     private Mat Maskcolor;
193
194     // variables for hsv mode (including histogram operations)
195     private Mat Hsv;
196     private Mat HsvRange;
197     private List<Mat> hsvHistograms;          // saves histograms for hue and saturation
198     private Mat OriginalMat;
```

```
199     private MatOfInt Channels[];
200     private MatOfInt HistSize;
201     private MatOfFloat Ranges;
202     private Scalar HueColor[];
203     private Scalar White;
204
205     // control variables
206     private int detectorChoice = MAX_DETECTOR_MODE;
207     private boolean Pause;    // freeze camera, e.g. during input operations
208
209     //// Tracking Targets
210     // vector for user defined filter values: hMin, sMin, vMin, hMax, sMax, vMax
211     // initialized with values for a good detection of a banana
212     private int[] HsvFilterValues = new int[]{0, 36, 141, 44, 188, 255};
213
214     // objects for food and food type handling
215     private List<FoodType> foodTypes = new ArrayList<>();
216     private List<Food> detectedFoods = new ArrayList<>();
217     private Food detectedFood;
218
219     // objects for tracking of detected foods --> using for remembering past states
220     private List<LinkedList<Food>> foodTracker = new ArrayList<LinkedList<Food>>();
221     private DbAccessFoodTypes dbFoodTypes;
222
223
224     ////////////////////////////////// END INITIALIZATION ////////////////////////////////////////
225
226
227
228     /**
229      * empty standard constructor
230      *
231      */
```

```
232     public onGetImageListener() {}
233
234
235
236     // executes various initialization operations
237     // initializes camera view from opencv
238     // initializes food type list with data from database
239     // initializes user interface
240
241     //onCreate is runs when activity is created
242
243     @Override
244     public void onCreate(Bundle savedInstanceState) {
245
246         // It load the saved instance to app and link listview to backend
247
248         Log.i(LABEL, "called onCreate");
249         super.onCreate(savedInstanceState);
250         getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
251         setContentView(R.layout.activity_main);
252
253         // check for permissions if API is lvl 23 or higher
254         // for API 23+ we need to request the read/write permissions even if they are already in manifest
255
256         if (android.os.Build.VERSION.SDK_INT >= 23) {
257             if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA)
258                     != PackageManager.PERMISSION_GRANTED) {
259                 if (ActivityCompat.shouldShowRequestPermissionRationale(this,
260                         Manifest.permission.CAMERA)) {
261                     // permission available, no further steps needed
262                 } else {
263                     // ask for permission of camera
264                     ActivityCompat.requestPermissions(this,
265                             new String[]{Manifest.permission.CAMERA},
266                             CAMERA_PERMISSIONS);
267                 }
268             }
269         }
270
271         //// initialize variables
272         // initialize opencv variables
273
274         OpenCVCameraView = (CameraBridgeViewBase) findViewById(R.id.activity_main_surface_view);
275         OpenCVCameraView.setCvCameraViewListener(this);
276         OpenCVCameraView.enableFpsMeter();
277
278         // initialize existing foodTypes from database
279         initializeFromDatabase();
280
281         // build seekbars and disable  them, since hsv mode is not default
282         buildRangeSeekBars();
283
284         // build floating button and menu
285         buildFab();
286     }
287
288
289
290     //cleanup operations, if app is paused
291
292     @Override
293     public void onPause() {
294         super.onPause();
295         if (OpenCVCameraView != null)
296             OpenCVCameraView.disableView();
297     }
```

```
298
299
300
301      // cleanup operations, if app is resumed
302
303
304     @Override
305     public void onResume() {
306         super.onResume();
307         OpenCVCameraView.enableView();
308         OpenCVCameraView.setOnTouchListener(onGetImageListener.this);
309     }
310
311
312
313      // cleanup operations, if app is resumed
314
315
316     public void onDestroy() {
317         super.onDestroy();
318         if (OpenCVCameraView != null)
319             OpenCVCameraView.disableView();
320     }
321
322
323     // function getting the result of requests
324     @Override
325     public void onRequestPermissionsResult(int requestCode,
326                                            String permissions[], int[] grantResults) {
327         switch (requestCode) {
328             case CAMERA_PERMISSIONS: {
329                 // If request is cancelled, the result arrays are empty.
330                 if (grantResults.length > 0
331                         && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
332
333                     // permission was granted, yay! Do the
334                     // contacts-related task you need to do.
335
336                 } else {
337                     // feedback to user for not allowing camera
338                     String text = "Sorry, without a camera this app does not work.";
339                     Toast toast = Toast.makeText(getApplicationContext(), text, Toast.LENGTH_LONG);
340                     toast.show();
341                 }
342                 return;
343             }
344
345             // other 'case' lines to check for other
346             // permissions this app might request
347         }
348     }
349
350     /**
351      * initializes all mats, scalars and other variables, which are needed for the frame processing
352      *
353      * @param width     the width of the frames that will be delivered
354      * @param height    the height of the frames that will be delivered
355      */
356
357     public void onCameraViewStarted(int width, int height) {
358         // initialize main mats
359         color = new Mat(height, width, CvType.CV_8UC4);
360         BorderMap = new Mat(height, width, CvType.CV_8UC1);
361         IntermediateMat = new Mat(height, width, CvType.CV_8UC4);
362         Hsv = new Mat(height, width, CvType.CV_8UC4);
363         HsvValue = new Mat(height, width, CvType.CV_8UC4);
```

```
364         HsvRange = new Mat(height, width, CvType.CV_8UC1);
365         Maskcolor = new Mat(height, width, CvType.CV_8UC3);
366
367         // initialize variables for hsv histograms
368         IntermediateMat = new Mat();
369         hsvHistograms = new ArrayList<>(Arrays.asList(
370                 new Mat(HIST_SIZE, 1, CvType.CV_32F),
371                 new Mat(HIST_SIZE, 1, CvType.CV_32F)));
372         Channels = new MatOfInt[]{new MatOfInt(0), new MatOfInt(1), new MatOfInt(2)};
373         HistSize = new MatOfInt(HIST_SIZE);
374         Ranges = new MatOfFloat(0f, 256f);
375         OriginalMat = new Mat();
376         White = Scalar.all(255);
377
378         // fill hue scalar with as much values as histogram bars (HIST_SIZE)
379         HueColor = createEvenHueScalar(HIST_SIZE);
380
381         // initialize other variables
382         Pause = false;
383     }
384
385
386
387     // cleanup operations, when camera view stops
388     // it releases the camera for use of next time
389     // if it does not release then this app or another use can not use the camera
390
391     public void onCameraViewStopped() {
392         color.release();
393         BorderMap.release();
394         IntermediateMat.release();
395
396         Hsv.release();
397         HsvValue.release();
398         HsvRange.release();
399         Maskcolor.release();
400     }
401
402
403     /**
404      * starting point for every frame, arranges steps depending on the active mode
405      *
406      * @param inputFrame        contains data from the live frame of the camera
407      * @return                  returns rgba frame as a result of the frame handling processes
408      *                          to display it in the next step
409      */
410
411     public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
412         final int VIEW_MODE = OpenCVViewMode;
413
414         if (!Pause) {
415             switch (VIEW_MODE) {
416                 case COLOUR_DETECTION_VIEW:
417                     // input frame has RBGA format
418                     color = inputFrame.rgba();
419                     Imgproc.cvtColor(color, HsvValue, Imgproc.COLOR_RGB2HSV_FULL);
420
421                     // execute food detection, if at least one food type is saved
422                     if (foodTypes.size() > 0)
423                         for (FoodType ft : foodTypes)
424                             detectFoodType(ft);
425
426                     // handle detected foods and clear the list afterwards
427                     if (detectedFoods.size() > 0)
428                         for (Food f : detectedFoods)
429                             handleDetectedFood(f, false);
```

```
                detectedFoods.clear();

                break;
            case BORDER_DETECTION_VIEW:
                // initialize needed mats
                BorderMap = inputFrame.gray();
                color = inputFrame.rgba();
                Imgproc.Canny(inputFrame.gray(), IntermediateMat, 80, 100);
                Imgproc.cvtColor(IntermediateMat, color, Imgproc.COLOR_GRAY2RGBA, 4);

                break;
            case KEYPOINTS_DETECTION_VIEW:
                // initialize needed mats
                BorderMap = inputFrame.gray();
                color = inputFrame.rgba();
                ARNutritionJNI.findFeatures(BorderMap.getNativeObjAddr(),
                        color.getNativeObjAddr(), detectorChoice);

                break;
            case HSV_FILTER_VIEW:
                // initialize needed mats -> backup mats to be able to reset in loops
                color = inputFrame.rgba();
                Imgproc.cvtColor(color, HsvValue, Imgproc.COLOR_RGB2HSV_FULL);

                // apply hsv filter from range seek bar settings as a mask to the screen
                // also show color histograms of hue and saturation
                handleHsvMode();

                // execute food detection, if at least one food type is saved
                if (foodTypes.size() > 0)
                    for (FoodType ft : foodTypes)
                        detectFoodType(ft);

                // handle detected foods and clear the list afterwards
                if (detectedFoods.size() > 0)
                    for (Food f : detectedFoods)
                        handleDetectedFood(f, true);
                detectedFoods.clear();

                break;
            }
        }
        return color;
    }


    /**
     * handle onClick events, such as the manipulation of hsv filter values
     *
     * @param v
     * this function is called when crrospending button is pressed
     */


    @Override
    public void onClick(View v) {
        // variable declaration
        Context context = getApplicationContext();
        String text = "";
        int length = Toast.LENGTH_SHORT;

        // find out, which button was pressed and execute corresponding actions
        if (v.getTag().equals(HSV_FILTER_MODE)) {
            // handle the option 'hsv mode'
            Log.i(LABEL, "Activated hsv filter");
            setHsvModeGuiVisibility(true);
```

```
496
497             // if it's already activated, ask for input to save current hsv filter values
498             if (OpenCVViewMode != HSV_FILTER_VIEW) {
499                 OpenCVViewMode = HSV_FILTER_VIEW;
500
501                 // prepare tutorial text and set toast length to long,
502                 // since it's fairly a bit of text
503                 if (!DEACTIVATE_HSV_MODE) {        // skip the introduction, when debugging, since it's long
504                     length = Toast.LENGTH_LONG;
505                     int lengthHsvIntroduction = 3;
506                     text = "HSV filter mode for food type storing" +
507                             "\nTop left: HSV range filter" +
508                             "\nBottom left: Color histograms (hue, saturation)" +
509                             "\nPress again the 'hsv filter mode' button to save the food!";
510
511                     // show toast multiple times, since TOAST_LONG is still too short
512                     for (int i = 0; i < lengthHsvIntroduction; i++) {
513                         Toast toast = Toast.makeText(context, text, length);
514                         toast.show();
515                     }
516                 }
517             } else {
518                 dialogInputSaveFoodType();
519             }
520         } else {
521             // disable seekbars
522             Log.i(LABEL, "Activated another mode");
523             setHsvModeGuiVisibility(false);
524
525             // reset detectorChoice when another mode is chosen
526             // -> initialization with last mode so it starts with first one
527             if (!(v.getTag().equals(LABEL_FEATURES))) detectorChoice = MAX_DETECTOR_MODE;
528
529             // handle the selection of all modes besides the hsv mode
530             if (v.getTag().equals(DETECTION_MODE)) {
531                 // handle the option 'normal mode'
532                 Log.i(LABEL, "activated rgba mode");
533                 OpenCVViewMode = COLOUR_DETECTION_VIEW;
534
535                 // prepare introduction text
536                 text = "RGB camera mode with active food detection.";
537             } else if (v.getTag().equals(EDGE_DETECTION)) {
538                 // handle the option 'edge mode'
539                 Log.i(LABEL, "activated canny mode");
540                 OpenCVViewMode = BORDER_DETECTION_VIEW;
541
542                 // prepare introduction text
543                 text = "Explore the edges in your environment!";
544             } else if (v.getTag().equals(LABEL_FEATURES)) {
545                 // handle the option 'feature mode'
546                 Log.i(LABEL, "activated feature mode");
547                 OpenCVViewMode = KEYPOINTS_DETECTION_VIEW;
548                 detectorChoice = (detectorChoice + 1) % (MAX_DETECTOR_MODE + 1);
549
550                 // prepare introduction text and increase length, since it's longer
551                 text = "Explore features with different detectors!";
552                 length = Toast.LENGTH_LONG;
553
554                 // give feedback to chosen mode
555                 switch (detectorChoice) {
556                     case 0:
557                         text += "\nFeature detector: FAST";
558                         break;
559                     case 1:
560                         text += "\nFeature detector: AGAST";
561                         break;
```

```
562                 case 2:
563                     text += "\nFeature detector: GFTT";
564                     break;
565                 case 3:
566                     text += "\nFeature detector: SimpleBlobDetector";
567                     break;
568                 default:
569                     text += "\nError in detectorChoice feedback";
570                     Log.i(LABEL, "Error in detectorChoice feedback");
571                     break;
572             }
573             text += "\nPress again for another detector!";
574         } else {
575             // handle exceptions
576             Log.i(LABEL, "error in mode activation");
577             OpenCVViewMode = COLOUR_DETECTION_VIEW;
578         }
579         // show prepared toast
580         Toast toast = Toast.makeText(context, text, length);
581         toast.show();
582     }
583 }
584
585
586 /**
587  * handle onTouch events (such as displaying nutritional values to a touched food)
588  *
589  * @param arg0      standard transfer variable
590  * @param event     standard transfer variable, which contains information about the touch event
591  * @return
592  */
593
594
595 @Override
596 public boolean onTouch(View arg0, MotionEvent event) {
597     // variable declaration
598     double[] coordinates = new double[2];
599     Point touchedPoint;
600
601     // define size of color
602     double columns = color.columns();// color is your image frame
603     double rows = color.rows();
604
605     // determine size of used screen and define offset of color frame
606     int width = OpenCVCameraView.getWidth();
607     int height = OpenCVCameraView.getHeight();
608     double scaleFactor = columns / width;
609     double xOffset = (width * scaleFactor - columns) / 2;
610     double yOffset = (height * scaleFactor - rows) / 2;
611
612     // temporary variable for conversion tasks
613     MatOfPoint2f TempMat = new MatOfPoint2f();
614
615     // determine coordinates of touched point in color frame and create a Point for the result
616     coordinates[0] = (event).getX() * scaleFactor - xOffset;
617     coordinates[1] = (event).getY() * scaleFactor - yOffset;
618     touchedPoint = new Point(coordinates[0], coordinates[1]);
619
620     //// check depending on detection scale if a touch occured within the contour of a food
621     // convert contour to needed format
622     for(LinkedList<Food> currentList : foodTracker)
623         if (currentList.size() > 0) {
624             if (currentList.getLast() != null) {
625                 currentList.getLast().getContour().convertTo(TempMat, CvType.CV_32FC2);
626
627                 // toggle, that information should be displayed, since a touch
```

```
                    // occured within the contour
                    if (Imgproc.pointPolygonTest(TempMat, touchedPoint, false) > 0) {
                        // set, that information is desired, if it's not set - reset, if it's set
                        if (!currentList.getLast().getIsInformationDisplayed())
                            currentList.getLast().setIsInformationDisplayed(true);
                        else
                            currentList.clear();
                    }
                }
            }

        return false;       //false: no subsequent events ; true: subsequent events
    }


    /******************** END ACTIVITY CYCLE METHODS *******************************/



    // initialize foodTypes with saved values from database

    public void initializeFromDatabase() {
        // get database
        dbFoodTypes = new DbAccessFoodTypes(this);

        // get data if table already exists
        if (dbFoodTypes.isTableExisting(dbFoodTypes.TABLE_NAME_FOOD_TYPES)) {
            // process data
            foodTypes = dbFoodTypes.getAllData();
        }

        // update food tracker
        for(FoodType ft : foodTypes)
            foodTracker.add(new LinkedList<Food>());
    }



    //create floating main button and floating menu, which is activated through a click on the
    //main button


    public void buildFab() {
        // create floating button
        final ImageView mainFab = new ImageView(this);
        mainFab.setImageResource(R.drawable.ic_plus);
        final FloatingActionButton actionButton = new FloatingActionButton.Builder(this)
                .setContentView(mainFab)
                .setBackgroundDrawable(getResources()
                        .getDrawable(R.drawable.selector_button_gold, getTheme()))
                .build();

        // create menu items
        final ImageView fabModeNormal = new ImageView(this);
        final ImageView fabModeEdge = new ImageView(this);
        final ImageView fabModeFeatures = new ImageView(this);
        final ImageView fabModeHsvFilter = new ImageView(this);
        fabModeNormal.setImageResource(R.drawable.ic_camera);
        fabModeEdge.setImageResource(R.drawable.ic_edge);
        fabModeFeatures.setImageResource(R.drawable.ic_feature);
        fabModeHsvFilter.setImageResource(R.drawable.ic_filter);

        // create menu buttons
        SubActionButton.Builder itemBuilder = new SubActionButton.Builder(this);
        itemBuilder.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.selector_button_green, getTheme()));
```

```
694         SubActionButton buttonFabModeNormal = itemBuilder.setContentView(fabModeNormal).build();
695         itemBuilder.setBackgroundDrawable(getResources()
696                 .getDrawable(R.drawable.selector_button_blue, getTheme()));
697         SubActionButton buttonFabModeEdge = itemBuilder.setContentView(fabModeEdge).build();
698         itemBuilder.setBackgroundDrawable(getResources()
699                 .getDrawable(R.drawable.selector_button_red, getTheme()));
700         SubActionButton buttonFabModeFeatures = itemBuilder.setContentView(fabModeFeatures).build();
701         itemBuilder.setBackgroundDrawable(getResources()
702                 .getDrawable(R.drawable.selector_button_purple, getTheme()));
703         SubActionButton buttonFabModeHsvFilter =
704                 itemBuilder.setContentView(fabModeHsvFilter).build();
705
706         // set tags for different floating buttons
707         buttonFabModeNormal.setTag(DETECTION_MODE);
708         buttonFabModeEdge.setTag(EDGE_DETECTION);
709         buttonFabModeFeatures.setTag(LABEL_FEATURES);
710         buttonFabModeHsvFilter.setTag(HSV_FILTER_MODE);
711
712         //  set onClickListener
713         buttonFabModeNormal.setOnClickListener(this);
714         buttonFabModeEdge.setOnClickListener(this);
715         buttonFabModeFeatures.setOnClickListener(this);
716         buttonFabModeHsvFilter.setOnClickListener(this);
717
718
719         // create floating menu
720         FloatingActionMenu actionMenu = new FloatingActionMenu.Builder(this)
721                 .addSubActionView(buttonFabModeNormal)
722                 .addSubActionView(buttonFabModeEdge)
723                 .addSubActionView(buttonFabModeFeatures)
724                 .addSubActionView(buttonFabModeHsvFilter)
725                 .attachTo(actionButton)
726                 .build();
727
728
729                 // create animation for the main fab, so it becomes an x (for exit) when active
730         actionMenu.setStateChangeListener(new FloatingActionMenu.MenuStateChangeListener() {
731             @Override
732             public void onMenuOpened(FloatingActionMenu menu) {
733                 // rotate the icon of rightLowerButton 45 degrees clockwise
734                 mainFab.setRotation(0);
735                 PropertyValuesHolder propertyValuesHolder =
736                         PropertyValuesHolder.ofFloat(View.ROTATION, 45);
737                 ObjectAnimator animation =
738                         ObjectAnimator.ofPropertyValuesHolder(mainFab, propertyValuesHolder);
739                 animation.start();
740             }
741
742             @Override
743             public void onMenuClosed(FloatingActionMenu menu) {
744                 // rotate the icon of rightLowerButton 45 degrees counter-clockwise
745                 mainFab.setRotation(45);
746                 PropertyValuesHolder propertyValuesHolder =
747                         PropertyValuesHolder.ofFloat(View.ROTATION, 0);
748                 ObjectAnimator animation =
749                         ObjectAnimator.ofPropertyValuesHolder(mainFab, propertyValuesHolder);
750                 animation.start();
751             }
752         });
753     }
754
755
756
757     //create RangeSeekBars for setting HSV ranges in HSV Filter mode
758
759     public void buildRangeSeekBars() {
```

```
760         // setup the new range seek bars
761         final RangeSeekBar<Integer> FilterH = new RangeSeekBar<>(this);
762         final RangeSeekBar<Integer> FilterS = new RangeSeekBar<>(this);
763         final RangeSeekBar<Integer> FilterV = new RangeSeekBar<>(this);
764
765         // set the range
766         FilterH.setRangeValues(0, 255);
767         FilterH.setSelectedMinValue(HsvFilterValues[0]);
768         FilterH.setSelectedMaxValue(HsvFilterValues[3]);
769         FilterS.setRangeValues(0, 255);
770         FilterS.setSelectedMinValue(HsvFilterValues[1]);
771         FilterS.setSelectedMaxValue(HsvFilterValues[4]);
772         FilterV.setRangeValues(0, 255);
773         FilterV.setSelectedMinValue(HsvFilterValues[2]);
774         FilterV.setSelectedMaxValue(HsvFilterValues[5]);
775
776
777         // set onChangeListeners for all three range seek bars
778         FilterH.setOnRangeSeekBarChangeListener(
779                 new RangeSeekBar.OnRangeSeekBarChangeListener<Integer>() {
780                     @Override
781                     public void onRangeSeekBarValuesChanged(RangeSeekBar<?> bar, Integer miAugmentValue,
782                                                             Integer maxValue) {
783                         // handle changed range values
784                         HsvFilterValues[0] = miAugmentValue;
785                         HsvFilterValues[3] = maxValue;
786                     }
787                 });
788
789         FilterS.setOnRangeSeekBarChangeListener(
790                 new RangeSeekBar.OnRangeSeekBarChangeListener<Integer>() {
791                     @Override
792                     public void onRangeSeekBarValuesChanged(RangeSeekBar<?> bar, Integer miAugmentValue,
793                                                             Integer maxValue) {
794                         // handle changed range values
795                         HsvFilterValues[1] = miAugmentValue;
796                         HsvFilterValues[4] = maxValue;
797                     }
798                 });
799
800         FilterV.setOnRangeSeekBarChangeListener(
801                 new RangeSeekBar.OnRangeSeekBarChangeListener<Integer>() {
802                     @Override
803                     public void onRangeSeekBarValuesChanged(RangeSeekBar<?> bar, Integer miAugmentValue,
804                                                             Integer maxValue) {
805                         // handle changed range values
806                         HsvFilterValues[2] = miAugmentValue;
807                         HsvFilterValues[5] = maxValue;
808                     }
809
810                 });
811
812         // add RangeSeekBars to layout
813         LinearLayout layout = (LinearLayout) findViewById(R.id.seekbar_placeholder);
814         layout.addView(FilterH);
815         layout.addView(FilterS);
816         layout.addView(FilterV);
817
818         setLinearLayoutVisibility(false, layout);
819     }
820
821
822     /**
823      * updates range bar values to the ones defines in the transfer variable
824      *
825      * @param values        defines values, with which the range bars need to be updated
```

```
826     */
827
828    public void setRangeBarValues(int[] values) {
829        // variable declaration and initialization
830        RangeSeekBar<Integer> RangeSeekBar;
831        LinearLayout layout = (LinearLayout) findViewById(R.id.seekbar_placeholder);
832
833        // set values of all rangeseekbars according to given values
834        for(int i=0; i < 3; i++) {
835            // get a range seek bar
836            RangeSeekBar = (RangeSeekBar<Integer>) layout.getChildAt(i);
837
838            // set the range bar values
839            RangeSeekBar.setSelectedMinValue(values[i]);
840            RangeSeekBar.setSelectedMaxValue(values[i + 3]);
841
842            // update the HsvFilterValues
843            HsvFilterValues[i] = values[i];
844            HsvFilterValues[i+3] = values[i+3];
845        }
846    }
847
848
849    /**
850     * function for visibility setting of all seekbars in defined LinearLayout
851     *
852     * @param visible       determines, if items are made visible or gone (invisible-like)
853     * @param layoutParent  determines layout parent item, whose children are made (in-)visible
854     */
855    public void setLinearLayoutVisibility(boolean visible, LinearLayout layoutParent) {
856        int childCount = layoutParent.getChildCount();
857
858        Animation animFadeOut = AnimationUtils.loadAnimation(getApplicationContext(),
859                android.R.anim.fade_out);
860        Animation animFadeIn = AnimationUtils.loadAnimation(getApplicationContext(),
861                android.R.anim.fade_in);
862
863
864        if (visible) {
865            layoutParent.setAnimation(animFadeIn);
866            for (int i = 0; i < childCount; i++) {
867                View v = layoutParent.getChildAt(i);
868                v.setVisibility(View.VISIBLE);
869            }
870        } else {
871            layoutParent.setAnimation(animFadeOut);
872            for (int i = 0; i < childCount; i++) {
873                View v = layoutParent.getChildAt(i);
874                v.setVisibility(View.GONE);
875            }
876        }
877    }
878
879
880
881    //creates an array of scalars, according to defined size, with an even color range
882    // for the visualization of hue vector values
883
884    /**
885     * @param size      specifies the size of the scalar vector
886     */
887    public Scalar[] createEvenHueScalar(int size) {
888        // variable declaration
889        Scalar[] HueColor = new Scalar[size];
890        double IntermediateSize = 255/(STEPS_PER_PHASE - 1);
891        double r = 255, g = 0, b = 0;                   // starting values for the scalar
```

```
892         double progress = 0;
893         double channelValue;
894
895
896         /*
897             * create steady scalar interpolation between
898             * How to create a hue scalar:
899             * start with 255,0,0 (rgb)
900             * phase 1: increase g to max (255)
901             * phase 2: decrease r to min (0)
902             * phase 3: increase b to max
903             * phase 4: increase r to max
904             * phase 5: decrease b to min
905             * progress: 0...5x255
906         */
907         for (int i=0; i < size; i++) {
908
909             channelValue = progress % 255;
910
911             if (progress < 255) {
912                 g = channelValue;
913             }
914             else if (progress < 255*2 && progress >= 255) {
915                 if (g < 255) g = 255;
916                 r -= channelValue;
917             }
918             else if (progress < 255*3 && progress >= 255*2) {
919                 if (r > 0) r = 0;
920                 b = channelValue;
921             }
922             else if (progress < 255*4 && progress >= 255*3) {
923                 if (b < 255) b = 255;
924                 r = channelValue;
925             }
926             else if (progress < 255*5 && progress >= 255*4) {
927                 if (r < 255) r = 255;
928                 b -= channelValue;
929                 if (i == (size -1)) b = 0;      // safety measures for the last iteration
930             }
931
932             // create new scalar with calculated values
933             HueColor[i] = new Scalar(r, g, b, 255);
934
935             // progress 1 step further
936             progress += IntermediateSize;
937         }
938         return HueColor;
939     }
940
941
942
943     // handles the hsv mode: applies mask to live screen, depending on the hsv filter values
944     //of the range seek bars, also show color histogram of the filtered image
945
946
947     public void handleHsvMode() {
948         // variable declaration
949         List<MatOfPoint> contours;
950         MatOfPoint LargestContour;
951
952         // copy rgb frame to temporary mat
953         color.copyTo(IntermediateMat);
954
955         // apply hsv filter values to screen
956         getHsvFilterMask(HsvFilterValues);
957
```

```
958         // find biggest contour in binary mask from hsv filter mask
959         contours = findContoursAboveThreshold(DETECTED_DESIRED_CONTOUR);
960
961         // apply biggest contour as a mask to frame, if a contour was found
962         if (contours.size() > 0) {
963             // get the biggest contour
964             LargestContour = contours.get(0);
965             Log.e(DEBUG, "Biggest Contour Area: " + Imgproc.contourArea(LargestContour));
966
967             // apply biggest contour as a new mask (with reseted HsvRange) to rgb frame
968             Core.setIdentity(HsvRange, new Scalar(0, 0, 0));
969             Imgproc.drawContours(HsvRange,
970                     new ArrayList<>(Arrays.asList(LargestContour)), 0, White, -1);
971             Imgproc.cvtColor(HsvRange, Maskcolor, Imgproc.COLOR_GRAY2RGBA, 4);
972             Core.bitwise_and(IntermediateMat, Maskcolor, IntermediateMat);
973             Imgproc.drawContours(IntermediateMat,
974                     new ArrayList<>(Arrays.asList(LargestContour)), 0, White, 1);
975         }
976         else {
977             // apply mask to rgba frame without using any contour information
978             Imgproc.cvtColor(HsvRange, Maskcolor, Imgproc.COLOR_GRAY2RGBA, 4);
979             Core.bitwise_and(IntermediateMat, Maskcolor, IntermediateMat);
980         }
981
982         // give feedback about histograms of picture in hsv mode (calc them and then draw them)
983         drawHistograms(calcHsvHistograms());
984         IntermediateMat.copyTo(color);
985     }
986
987
988     /**
989      * executes the whole food detection process, shows histograms and mask in hsv mode
990      *
991      * @param ft         defines food type, for which the check occures
992      */
993     public void detectFoodType(FoodType ft) {
994         // variable declaration for temporary saving
995         Food detectedFood = null;
996         List<MatOfPoint> contours;
997         MatOfPoint LargestContour;
998         List<Mat> PresentHist;
999         LinkedList<Food> currentList = foodTracker.get(foodTypes.indexOf(ft));
1000
1001
1002        //// handle single detection --> use largest contour of food candidates
1003        // apply hsv filter from currently checked food type to screen
1004        getHsvFilterMask(ft.getHsvFilterValues());
1005
1006        // find biggest contour in binary mask from hsv filtered frame
1007        // (uses: HsvRange for contour searching, without modifying it (uses clone))
1008        contours = findContoursAboveThreshold(DETECTED_DESIRED_CONTOUR);
1009
1010        // reject further actions if there's no contour within the bottom and top threshold
1011        if (contours.size() > 0) {
1012            // get the biggest contour
1013            LargestContour = contours.get(0);
1014
1015            //// apply the biggest contour as a new mask to reseted mask
1016            // reset needed mats
1017            Core.setIdentity(HsvRange, new Scalar(0, 0, 0));
1018            color.copyTo(IntermediateMat);
1019
1020            // draw contour
1021            Imgproc.drawContours(HsvRange,
1022                    new ArrayList<>(Arrays.asList(LargestContour)), 0, White, -1);
1023
```

```
1024            // convert to rgba mask and apply mask to rgb frame
1025            Imgproc.cvtColor(HsvRange, Maskcolor, Imgproc.COLOR_GRAY2RGBA, 4);
1026            Core.bitwise_and(IntermediateMat, Maskcolor, IntermediateMat);
1027
1028            // calculate the histograms from IntermediateMat
1029            PresentHist = calcHsvHistograms();
1030
1031            // create new food with detected type and accompanying contour and handle it
1032            if (compareHsHistograms(ft, PresentHist)) {
1033                detectedFood = new Food(ft, LargestContour);
1034
1035                // information should be displayed, when the detected type appeared in the last
1036                // few frames (according to FRAMES_TRACKING_RANGE)
1037                if (currentList.size() > 0)
1038                    for (Food f : currentList)
1039                        if (f != null)
1040                            if (f.getIsInformationDisplayed() && (f.getType().getName()
1041                                    .compareToIgnoreCase(ft.getName()) == 0))
1042                                detectedFood.setIsInformationDisplayed(true);
1043
1044                // save found object in tracker to remember the state (fifo list)
1045                if (currentList.size() < FRAMES_TRACKING_RANGE)
1046                    currentList.add(detectedFood);
1047                else {
1048                    currentList.removeFirst();
1049                    currentList.add(detectedFood);
1050                }
1051
1052                //save detected food
1053                detectedFoods.add(detectedFood);
1054
1055                // clear object of detected food, so it's clean for the next frame
1056                detectedFood = null;
1057            } else {
1058                // put null as an object into the list, so it can act as a counter for frames
1059                // with abandoned candidates
1060                if (currentList.size() < FRAMES_TRACKING_RANGE)
1061                    currentList.add(null);
1062                else {
1063                    currentList.removeFirst();
1064                    currentList.add(null);
1065                }
1066            }
1067        } else {
1068            // put null as an object into the list, so it can act as a counter for frames
1069            // without detected objects of this food type
1070            if (currentList.size() < FRAMES_TRACKING_RANGE)
1071                currentList.add(null);
1072            else {
1073                currentList.removeFirst();
1074                currentList.add(null);
1075            }
1076        }
1077    }
1078
1079
1080
1081    //applies defined filter values to hsv image, copies result to intermediate mat
1082    //also be done in the background - another result is a binary mask
1083    //(mask is saved in HsvRange)
1084    //(modified global variable: IntermediateMat gets original rgba values)
1085    //(modified global variable: Hsv gets hsv values from original rgba values)
1086    //(modified global variable: HsvRange gets mask from hsv image with
1087    //     applied hsv filter values)
1088
1089    /**
```

```
1090      * @param hsvFilterValues       defines the min/max values for h, s and v filtering
1091      */
1092
1093     public void getHsvFilterMask(int[] hsvFilterValues) {
1094         // create elements for morphing operations
1095         Mat Eroding = Imgproc.getStructuringElement(Imgproc.MORPH_RECT, CV_ERODE);
1096         Mat Dilating = Imgproc.getStructuringElement(Imgproc.MORPH_RECT, CV_DILATE);
1097
1098         // save color value (live screen) temporary in IntermediateMat, to use the latter here
1099         HsvValue.copyTo(Hsv);
1100
1101         // apply hsv filter values to hsv frame
1102         Core.inRange(Hsv,
1103                 new Scalar(hsvFilterValues[0], hsvFilterValues[1], hsvFilterValues[2]),
1104                 new Scalar(hsvFilterValues[3], hsvFilterValues[4], hsvFilterValues[5]),
1105                 HsvRange);
1106
1107         //// erode, dilate and blur frame to get rid of interference
1108         // erode and dilate in each two iterations
1109         Imgproc.erode(HsvRange, HsvRange, Eroding);
1110         Imgproc.erode(HsvRange, HsvRange, Eroding);
1111         Imgproc.dilate(HsvRange, HsvRange, Dilating);
1112         Imgproc.dilate(HsvRange, HsvRange, Dilating);
1113     }
1114
1115
1116
1117     // calculates hsv histograms and feedback graph and puts the feedback graph of the color
1118     // histograms (hue, saturation) into a copy of color (IntermediateMat), so it can also
1119     // be done in the background without affecting the live frame
1120     // (needs rgba copy in IntermediateMat, which is converted to Hsv)
1121
1122
1123     public List<Mat> calcHsvHistograms() {
1124         // declaration and initialization of variables
1125         float MaxValue;
1126         float Buffer[] = new float[HIST_SIZE];
1127         Mat hist = new Mat(HIST_SIZE, 1, CvType.CV_32F);
1128         List<Mat> resultHistograms = new ArrayList<>();
1129
1130         // get hsv mat from mat buffer (intermediate mat --> frame with applied mask)
1131         Imgproc.cvtColor(IntermediateMat, Hsv, Imgproc.COLOR_RGB2HSV_FULL);
1132
1133         //// calculate and display histogram for hue
1134         Imgproc.calcHist(Arrays.asList(Hsv), Channels[0], OriginalMat, hist, HistSize, Ranges);
1135
1136         //// normalize, so the values are [0..1] and reset the first value, when it's at maximum
1137         //// (this happens, when the mask is applied, since a lot of area is black)
1138         Core.normalize(hist, hsvHistograms.get(0), 1, 0, Core.NORM_INF);
1139         hsvHistograms.get(0).get(0, 0, Buffer);
1140
1141         // find out highest value
1142         MaxValue = 0;
1143         for (float d : Buffer)
1144             if (d > MaxValue) MaxValue = d;
1145
1146         // reset first value to zero, if it's the highest value, or trim it to zero
1147         if (Buffer[0] == MaxValue) {
1148             // reset the freak value (comes from the mask probably)
1149             Buffer[0] = 0;
1150
1151             // update histogram and normalize it again
1152             hsvHistograms.get(0).put(0, 0, Buffer);
1153             Core.normalize(hsvHistograms.get(0), hsvHistograms.get(0), 1, 0, Core.NORM_INF);
1154         }
1155
```

```
1156        //// calculate and display histogram for saturation
1157        Imgproc.calcHist(Arrays.asList(Hsv), Channels[1], OriginalMat, hist, HistSize, Ranges);
1158
1159        // normalize, so the values are [0..1] and reset the first value, when it's at maximum
1160        // (this happens, when the mask is applied, since a lot of area is black)
1161        Core.normalize(hist, hsvHistograms.get(1), 1, 0, Core.NORM_INF);
1162        hsvHistograms.get(1).get(0, 0, Buffer);
1163
1164        // find out highest value
1165        MaxValue = 0;
1166        for (float d : Buffer)
1167            if (d > MaxValue) MaxValue = d;
1168
1169        // reset first value to zero, if it's the highest value, or trim it to zero
1170        if (Buffer[0] == MaxValue) {
1171            Buffer[0] = 0;       //reset
1172
1173            // update histogram and normalize it again
1174            hsvHistograms.get(1).put(0, 0, Buffer);
1175            Core.normalize(hsvHistograms.get(1), hsvHistograms.get(1), 1, 0, Core.NORM_INF);
1176        }
1177
1178        // clone calculated histograms and return them as a result
1179        for (Mat m : hsvHistograms)
1180            resultHistograms.add(m.clone());
1181
1182        return resultHistograms;
1183    }
1184
1185
1186    /**
1187     * draws the two first histograms from a given list
1188     *
1189     * @param histograms    defines list containing histograms
1190     */
1191    public void drawHistograms(List<Mat> histograms) {
1192        // constant declaration
1193        final double BAR_HEIGHT = color.size().height / 16 * 7;
1194
1195        // declaration and initialization of variables
1196        int thickness = (color.width() - HIST_SETTING * (HIST_AMOUNT + 1)) /
1197                (HIST_SIZE * HIST_AMOUNT);
1198        if (thickness > 7) thickness = 7;               // limit thickness
1199        float Buffer[] = new float[HIST_SIZE];
1200        Point mPoint1 = new Point();
1201        Point mPoint2 = new Point();
1202        Mat hist = new Mat(HIST_SIZE, 1, CvType.CV_32F);
1203
1204        // normalize to the desired height of the color histogram bars
1205        Core.normalize(hsvHistograms.get(0), hist, BAR_HEIGHT, 0, Core.NORM_INF);
1206        hist.get(0, 0, Buffer);
1207        for (int h = 0; h < HIST_SIZE; h++) {
1208            mPoint1.x = mPoint2.x = HIST_SETTING + h * thickness;
1209            mPoint1.y = color.height() - 1;
1210            mPoint2.y = mPoint1.y - (int) Buffer[h];
1211            Imgproc.line(IntermediateMat, mPoint1, mPoint2, HueColor[h], thickness);
1212        }
1213
1214        // normalize to the desired height of the color histogram bars
1215        Core.normalize(histograms.get(1), hist, BAR_HEIGHT, 0, Core.NORM_INF);
1216        hist.get(0, 0, Buffer);
1217        for (int h = 0; h < HIST_SIZE; h++) {
1218            mPoint1.x = mPoint2.x = 2 * HIST_SETTING + (HIST_SIZE + h) * thickness;
1219            mPoint1.y = color.height() - 1;
1220            mPoint2.y = mPoint1.y - (int) Buffer[h];
1221            Imgproc.line(IntermediateMat, mPoint1, mPoint2, White, thickness);
```

```
1222        }
1223    }
1224
1225
1226    /**
1227     * applies HIST_BORDER_MIN to given histogram
1228     *
1229     * @param histograms    histograms, to which autofit should be applied
1230     * @return              returns a vector of new filter values
1231     */
1232    public int[] autofitHistograms(List<Mat> histograms) {
1233        // declaration and initialization of variables
1234        float Buffer[] = new float[HIST_SIZE];
1235        int[] hsvFilterValuesAutofit =
1236                new int[]{0,0,HsvFilterValues[2],255,255,HsvFilterValues[5]};
1237        int i = 0,j = 0, k = 0;
1238
1239        // handle all available histograms
1240        for (Mat hist : histograms) {
1241            // get data from histogram
1242            hist.get(0, 0, Buffer);
1243
1244            // apply left border --> set all histogram values to zero,
1245            // if they're beneath the threshold
1246            for (i = 0; i < Buffer.length && Buffer[i] < HIST_BORDER_MIN; i++)
1247                if (Buffer[i] < HIST_BORDER_MIN)
1248                    Buffer[i] = 0;
1249
1250            // apply right border --> set all histogram values to zero,
1251            // if they're beneath the threshold
1252            for (j = Buffer.length-1; j > 0 && Buffer[j] < HIST_BORDER_MIN; j--)
1253                if (Buffer[j] < HIST_BORDER_MIN)
1254                    Buffer[j] = 0;
1255
1256            // save data to histogram
1257            hist.put(0, 0, Buffer);
1258
1259            // save settings, if the histogram consists of more than just zeros
1260            if (j > 0) {
1261                hsvFilterValuesAutofit[k] = i;
1262                hsvFilterValuesAutofit[k + 3] = j;
1263            }
1264            else if (j == 0) {
1265                hsvFilterValuesAutofit[k] = i;
1266                hsvFilterValuesAutofit[k + 3] = 255;
1267            }
1268
1269            k++;
1270        }
1271
1272        return hsvFilterValuesAutofit;
1273    }
1274
1275
1276    /**
1277     * find closest match to picture --> hs color histogram comparison with specified method
1278     *
1279     * @return      returns the detected FoodType, which is most likely the captured food
1280     */
1281    public boolean compareHsHistograms(FoodType ft, List<Mat> PresentHist) {
1282        double Highest_H_Value = 0;
1283        double Highest_S_Value = 0;
1284        boolean MatchFound = false;
1285
1286        // find closest match to picture --> hs color histogram comparison with chi square method
1287        // histogram comparison for the currently checked food type
```

```
1288        double Comparison_H_Value = 0;
1289        double Comparison_S_Value = 0;
1290
1291        // check if all histograms are available
1292        if (ft.getHsvHistogramH() != null && ft.getHsvHistogramS() != null
1293                && PresentHist.get(0) != null && PresentHist.get(1) != null) {
1294
1295            // <debugSection>
1296            // compare histograms with all available methods and print results in the log
1297            if (ACTIVATE_COMPARISON) {
1298                String results;
1299                results = testHistogramComparisonMethods(ft, PresentHist);
1300                Log.e(HIST_COMP_TESTS, results);
1301            }
1302            // </debugSection>
1303
1304            // compare histograms with intersect method
1305            Comparison_H_Value = Imgproc.compareHist(ft.getHsvHistogramH(),
1306                    PresentHist.get(0), HIST_COMPARISON_METHOD);
1307            Comparison_S_Value = Imgproc.compareHist(ft.getHsvHistogramS(),
1308                    PresentHist.get(1), HIST_COMPARISON_METHOD);
1309
1310            if (Comparison_H_Value > Highest_H_Value) {
1311                Highest_H_Value = Comparison_H_Value;
1312            }
1313            if (Comparison_S_Value > Highest_S_Value) {
1314                Highest_S_Value = Comparison_S_Value;
1315            }
1316
1317            Log.e(HIST_COMP_TESTS, ft.getName() + " - H: " + Comparison_H_Value +
1318                    " ; S:" + Comparison_S_Value);
1319        }
1320
```

```
1321        // apply threshold, depending on the defined scope
1322        if (COMPARE_HUE) {
1323            if (Highest_H_Value >= HUE_MIN
1324                    && Highest_H_Value <= HUE_MAX) {
1325                MatchFound = true;
1326            }
1327        }
1328        else if (COMPARE_SATURATION) {
1329            if (Highest_S_Value >= SATURATION_MIN
1330                    && Highest_S_Value <= SATURATION_MAX) {
1331                MatchFound = true;
1332            }
1333        }
1334        else if (COMPARE_VALUE) {
1335            if (Highest_H_Value >= HUE_MIN
1336                    && Highest_H_Value <= HUE_MAX
1337                    && Highest_S_Value >= SATURATION_MIN
1338                    && Highest_S_Value <= SATURATION_MAX) {
1339                MatchFound = true;
1340            }
1341        }
1342
1343        return MatchFound;
1344    }
1345
1346
1347    /**
1348     * finds biggest contour in HsvRange (binary mask from hsv threshold filtering) above
1349     * the defined threshold, or even all contours above this threshold,
1350     * depending on the parameter isOnlyBiggestDesired
1351     *
1352     * @param isOnlyBiggestDesired      defines, if the task is to find the biggest contour or all
1353     *                                  above the threshold
```

```
1354     * @return                          list of contours with biggest contour only are all above
1355     *                                  the threshold
1356     */
1357    public List<MatOfPoint> findContoursAboveThreshold(boolean isOnlyBiggestDesired) {
1358        //// prepare variables for getting the contours
1359        //// out of the binary image (filter threshold result)
1360        // general variables
1361        List<MatOfPoint> results = new ArrayList<>();
1362        Mat temp = new Mat();       // needed, since original mat is modified
1363
1364        // contour specific variables
1365        List<MatOfPoint> contours = new ArrayList<>();
1366        MatOfPoint LargestContour = new MatOfPoint();
1367
1368        // temporary variable for conversion tasks
1369        MatOfPoint2f TempMat1 = new MatOfPoint2f();
1370        MatOfPoint2f TempMat2 = new MatOfPoint2f();
1371
1372        // apply hsv filter values from each registered food type and clone the resulting
1373        // binary mask into temporary Mat for further processing
1374        HsvRange.copyTo(temp);
1375
1376        // find all contours in clone of binary mask
1377        Imgproc.findContours(temp, contours, new Mat(), Imgproc.RETR_EXTERNAL,
1378                Imgproc.CHAIN_APPROX_SIMPLE);
1379
1380        // security measures for the case of no found contours
1381        if (contours.size() > 0) {
1382            // walk through all contours
1383            for (int i = 0; i < contours.size(); i++) {
1384                // operations, if only the biggest contour is desired
1385                if (isOnlyBiggestDesired) {
1386                    // save index, if the contour is the biggest yet
1387                    if (i == 0)
1388                        LargestContour = contours.get(i);
1389                    else if (Imgproc.contourArea(contours.get(i))
1390                            > Imgproc.contourArea(LargestContour)
1391                            && Imgproc.contourArea(contours.get(i)) > CONTOUR_LEASTVALUE
1392                            && Imgproc.contourArea(contours.get(i)) < CONTOUR_MaxValue)
1393                        LargestContour = contours.get(i);
1394                }
1395                // operations, if multiple contours are desired
1396                else {
1397                    if (Imgproc.contourArea(contours.get(i)) > CONTOUR_LEASTVALUE
1398                            && Imgproc.contourArea(contours.get(i)) < CONTOUR_MaxValue)
1399                        results.add(contours.get(i));
1400                }
1401            }
1402
1403            //// handle found contours
1404            if (isOnlyBiggestDesired) {
1405                // if there was found one above the threshold size: smooth
1406                // and save the found contour
1407                if (Imgproc.contourArea(LargestContour) > CONTOUR_LEASTVALUE
1408                        && Imgproc.contourArea(LargestContour) < CONTOUR_MaxValue) {
1409                    // convert mats of contour (forth and back) and smooth contour
1410                    LargestContour.convertTo(TempMat1, CvType.CV_32FC2);
1411                    Imgproc.approxPolyDP(TempMat1, TempMat2, CONTOUR_DISTANCE, true);
1412                    TempMat2.convertTo(LargestContour, CvType.CV_32S);
1413                    results.add(LargestContour);
1414                }
1415            }
1416            else {
1417                if (results.size()!=0) {
1418                    //smooth each found contours, if any are found
1419                    for (MatOfPoint mop : results) {
```

```
                    mop.convertTo(TempMat1, CvType.CV_32FC2);
                    Imgproc.approxPolyDP(TempMat1, TempMat2, CONTOUR_DISTANCE, true);
                    TempMat2.convertTo(mop, CvType.CV_32S);
                }
            }
            else {
                // no contour found
                results = null;
            }
        }
    }

    return results;
}


/**
 * handle the detected food: draw contour, line and label
 *
 * @param f                     food, which should be handled
 * @param isContourDesired      determines, if contour should be drawn
 */
public void handleDetectedFood(Food f, boolean isContourDesired) {
    // variable declaration
    Size textSize;
    int labelLine2Width;

    Point start;
    Point Point1;
    Point Point2;
    Point locationName;
    Point locationAddInformation;

    List<String> AddInformation = new ArrayList<>();
    String format = "%3.1f";  // width = 3 and 2 digits after the dot
    NutritionalValue AugmentValue;

    // safety measurement
    if (f != null) {
        // variable initialization
        textSize = Imgproc.getTextSize(f.getType().getName(), TEXT_FONT_STYLE, TEXT_FONT_SCALE,
                TEXT_THICKNESS, null);
        labelLine2Width = (int) (textSize.width
                + INFORMATION_DISPLAY_VALUE + BUFFER_TEXT);
        start =  f.getLocationLabel().clone();

        // draw contour if wished with transfer variable
        if (isContourDesired) {
            List<MatOfPoint> contours = new ArrayList<>(Arrays.asList(f.getContour()));
            Imgproc.drawContours(color, contours, 0, f.getType().getMarkerColor(),
                    2, 8, new Mat(), 0, new Point());
        }
        else {
            // dig line of label into object (otherwise it could float)
            start.x -= INFORMATION_X;
            start.y -= INFORMATION_Y;
        }

        // calculate points for the label lines and the text
        Point1 = new Point(start.x + INFORMATION_WIDTH, start.y - INFORMATION_HEIGHT);
        Point2 = new Point(Point1.x + labelLine2Width, Point1.y);
        locationName = new Point(Point1.x + INFORMATION_DISPLAY_VALUE, Point1.y - TEXT_MARGIN);

        // draw lines of label
        Imgproc.line(color, start, Point1, f.getType().getMarkerColor(),
                INFORMATION_THICKNESS);
```

```
1486            Imgproc.line(color, Point1, Point2, f.getType().getMarkerColor(), LABEL_THICKNESS);
1487
1488            // draw text of label with name of detected food type
1489            Imgproc.putText(color, f.getType().getName(), locationName,
1490                    TEXT_FONT_STYLE, TEXT_FONT_SCALE, f.getType().getMarkerColor(),
1491                    TEXT_THICKNESS, TEXT_LINE, false);
1492
1493            // display further information, if it's desired (state of food)
1494            if (f.getIsInformationDisplayed()) {
1495                // prepare string for further information
1496                AugmentValue = f.getType().getNutritionalValue();
1497
1498                // security measures
1499                if (AugmentValue != null) {
1500                    AddInformation.add("" + (int) (AugmentValue.getCalorie() *
1501                            AugmentValue.getMeanWeight()/100) + " kcal");
1502                    AddInformation.add("Protein: " + String.format(Locale.ENGLISH, format,
1503                            (AugmentValue.getProtien() *
1504                                    AugmentValue.getMeanWeight()/100)) + "g");
1505                    AddInformation.add("Carbs: " + String.format(Locale.ENGLISH, format,
1506                            (AugmentValue.getCarbohydrate() *
1507                                    AugmentValue.getMeanWeight()/100)) + "g");
1508                    AddInformation.add("Fat: " + String.format(Locale.ENGLISH, format,
1509                            (AugmentValue.getFat() *
1510                                    AugmentValue.getMeanWeight()/100)) + "g");
1511                    AddInformation.add("Weight: "
1512                            + (int) AugmentValue.getMeanWeight() + "g");
1513                }
1514
1515                // prepare location
1516                locationAddInformation = new Point(Point1.x, Point1.y + TEXT_MARGIN + textSize.height);
1517
1518                // check if any text was prepared yet before drawing it
1519                if (!AddInformation.isEmpty()) {
1520                    for(int i=0; i < AddInformation.size(); i++) {
1521                        Imgproc.putText(color, AddInformation.get(i),
1522                                new Point(locationAddInformation.x,
1523                                        locationAddInformation.y
1524                                                + i*(textSize.height + TEXT_MARGIN)),
1525                                TEXT_FONT_STYLE, TEXT_FONT_SCALE, f.getType().getMarkerColor(),
1526                                TEXT_THICKNESS, TEXT_LINE, false);
1527                    }
1528                }
1529            }
1530        }
1531    }
1532
1533
1534    /**
1535     * method for testing all histogram comparison methods from opencv with given food type on
1536     * live frames
1537     *
1538     * @param ft                    FoodType, which histogram is compared with the live screen
1539     * @return Result    String with all formatted results, can be used for the log
1540     */
1541    public String testHistogramComparisonMethods(FoodType ft, List<Mat> PresentHist) {
1542        // variable declaration for string and format of results
1543        String Result;
1544        String format = "%7.2f";  // width = 7 and 2 digits after the dot
1545
1546        // compare h and s histogram and add results to the corresponding string
1547        Result = "Food type: " + ft.getName() + "\n";
1548        Result += "Method: CV_COMP_CHISQR        ---   Values: H: " +
1549                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramH(),
1550                        PresentHist.get(0), Imgproc.CV_COMP_CHISQR)) + ", S: " +
1551                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramS(),
```

```
                    PresentHist.get(1), Imgproc.CV_COMP_CHISQR)) + "\n";
        Result += "Method: CV_COMP_CHISQR_ALT    ---   Values: H: " +
                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramH(),
                    PresentHist.get(0), Imgproc.CV_COMP_CHISQR_ALT)) + ", S: " +
                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramS(),
                    PresentHist.get(1), Imgproc.CV_COMP_CHISQR_ALT)) + "\n";
        Result += "Method: CV_COMP_INTERSECT     ---   Values: H: " +
                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramH(),
                    PresentHist.get(0), Imgproc.CV_COMP_INTERSECT)) + ", S: " +
                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramS(),
                    PresentHist.get(1), Imgproc.CV_COMP_INTERSECT)) + "\n";
        Result += "Method: CV_COMP_BHATTACHARYYA ---   Values: H: " +
                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramH(),
                    PresentHist.get(0), Imgproc.CV_COMP_BHATTACHARYYA)) + ", S: " +
                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramS(),
                    PresentHist.get(1), Imgproc.CV_COMP_BHATTACHARYYA)) + "\n";
        Result += "Method: CV_COMP_CORREL        ---   Values: H: " +
                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramH(),
                    PresentHist.get(0), Imgproc.CV_COMP_CORREL)) + ", S: " +
                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramS(),
                    PresentHist.get(1), Imgproc.CV_COMP_CORREL)) + "\n";
        Result += "Method: CV_COMP_HELLINGER     ---   Values: H: " +
                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramH(),
                    PresentHist.get(0), Imgproc.CV_COMP_HELLINGER)) + ", S: " +
                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramS(),
                    PresentHist.get(1), Imgproc.CV_COMP_HELLINGER)) + "\n";
        Result += "Method: CV_COMP_KL_DIV        ---   Values: H: " +
                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramH(),
                    PresentHist.get(0), Imgproc.CV_COMP_KL_DIV)) + ", S: " +
                String.format(Locale.ENGLISH, format, Imgproc.compareHist(ft.getHsvHistogramS(),
                    PresentHist.get(1), Imgproc.CV_COMP_KL_DIV)) + "\n";

        return Result;
    }


    /**
     * Creates a dialog box and asks for a name of the currently filtered object
     * afterwards saves the object with calculated histograms as a food type, locally and
     * also as an update in the database
     *
     */
    public void dialogInputSaveFoodType() {
        // initialization of dialog
        String Title = "Name of currently filtered food:";
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle(Title);

        // use an EditText view to get user input.
        final EditText input = new EditText(this);
        builder.setView(input);

        // define actions for left button of dialog
        builder.setPositiveButton("Save", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int whichButton) {
                // variable declaration
                String name = input.getText().toString();

                // save food type
                saveFoodType(name);

                // abolish frame freeze, which was initiated for inputs
                Pause = false;
                return;
            }
```

```
        });

        // define actions for right button of dialog
        builder.setNegativeButton("Clear Database", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                // clear locally saved food types and also the ones in the database
                foodTypes.clear();
                dbFoodTypes.clear(dbFoodTypes.TABLE_NAME_FOOD_TYPES);

                String text = "Registered food types deleted.";
                Toast toast = Toast.makeText(getApplicationContext(), text, Toast.LENGTH_SHORT);
                toast.show();

                // abolish frame freeze, which was initiated for inputs
                Pause = false;
            }
        });

        // resets frozen screen, if user cancels with a touch anywhere outside the dialog box
        builder.setOnCancelListener(new DialogInterface.OnCancelListener() {
            @Override
            public void onCancel(DialogInterface dialog) {
                Pause = false;
            }
        });

        // freeze camera, so the current picture is saved instead of the live picture
        // unfreeze, when a button is pressed (unfreeze action declared in button declarations)
        Pause = true;

        // now show this dialog
        builder.show();
    }


    /**
     * saves food type with given name
     * gives feedback, if name was empty and nothing was saved
     *
     * @param name      defines name of the to be saved food type
     */
    public void saveFoodType(String name) {
        // variable declaration
        FoodType ft;

        if (!name.isEmpty()) {
            // clone hsv histograms since otherwise just references are copied
            List<Mat> clonedHistograms = new ArrayList<>();
            //List<Mat> clonedHistograms = new ArrayList<>(hsvHistograms);

            // clone histograms
            for (Mat m : hsvHistograms)
                clonedHistograms.add(m.clone());

            // apply autofit to the histograms and apply new histogram borders to
            // hsv filter settings
            //HsvFilterValues = autofitHistograms(clonedHistograms);
            setRangeBarValues(autofitHistograms(clonedHistograms));

            // create food type with current data and name
            ft = new FoodType(name, HsvFilterValues.clone(), clonedHistograms.get(0),
                    clonedHistograms.get(1), null, null, getNutritionalValue(name));

            // register the new food type to the local food type list and also save it in the
            // database, and in the tracker (every food type id of the tracker is thus
```

```
            // corresponding with the id in the foodTypes List)
            foodTypes.add(ft);
            dbFoodTypes.add(ft);
            foodTracker.add(new LinkedList<Food>());
        } else {
            String text = "No name was entered, thus nothing could be saved.\n" +
                    "Please enter a name the next time.";
            Toast toast = Toast.makeText(getApplicationContext(), text, Toast.LENGTH_SHORT);
            toast.show();
        }
    }


    /**
     * makes all declared gui elements of the hsv mode visible or invisible, as defined
     *
     * @param isVisible     distinguishes if goal of actions is visibility or invisibility
     */
    public void setHsvModeGuiVisibility(boolean isVisible) {
        // handle seekbars
        LinearLayout layoutSeekbars = (LinearLayout) findViewById(R.id.seekbar_placeholder);
        setLinearLayoutVisibility(isVisible, layoutSeekbars);

        //// handle other objects
        // ... (not needed for the moment)
    }


    /**
     * delivers the nutritional value from a food with the specified name
     * the result is depending on the name of the object and is hard-coded for the demonstration
     * purposes - a crawler would be a better solution
     *
     * @param name      specifies name of the food
     */
    public NutritionalValue getNutritionalValue(String name) {
        // variable declaration
        NutritionalValue result = null;
        float Calorie;
        float Protien;
        float Carbohydrate;
        float Fat;
        float MeanWeight;
        String healthHazardInfo;

        if (name != null) {
            if (!name.isEmpty()) {
                if (name.contains("Banana") || name.contains("banana")) {
                    // get nutritional values for a banana
                    Calorie = 89;
                    Protien = (float) 1.1;
                    Carbohydrate = (float) 22.8;
                    Fat = (float) 0.3;
                    MeanWeight = 118;
                    healthHazardInfo = "Banana can lead to Weight Gain,Migraine,Respiratory Problems,Constipation,Risk Of Type 2
                    result = new NutritionalValue(Calorie, Protien,
                            Carbohydrate, Fat, MeanWeight, healthHazardInfo);

                } else if (name.contains("Lemon") || name.contains("lemon")) {
                    // get nutritional values for a lemon
                    Calorie = 29;
                    Protien = (float) 1.1;
                    Carbohydrate = (float) 9.3;
                    Fat = (float)0.3;
                    MeanWeight = (float) 58;
                    healthHazardInfo = " Lemon can lead to Tooth decay, Ulcers or gastroesphageal reflux disorder \n Lemon is ric
```

```
1750                    result = new NutritionalValue(Calorie, Protien,
1751                            Carbohydrate, Fat, MeanWeight, healthHazardInfo);
1752
1753                } else if (name.contains("Apple") || name.contains("apple")) {
1754                    // get nutritional values for a apple
1755                    Calorie = 52;
1756                    Protien = (float) 0.3;
1757                    Carbohydrate = (float) 13.8;
1758                    Fat = (float) 0.2;
1759                    MeanWeight = 182;
1760                    healthHazardInfo = "The apple seeds contain cyanide and which can cause death. \nApples are used to control di
1761                    result = new NutritionalValue(Calorie, Protien,
1762                            Carbohydrate, Fat, MeanWeight, healthHazardInfo);
1763
1764                } else if (name.contains("Tomato") || name.contains("Tomato")) {
1765                    // get nutritional values for a mandarin
1766                    Calorie = 53;
1767                    Protien = (float) 0.8;
1768                    Carbohydrate = (float) 13.3;
1769                    Fat = (float) 0.3;
1770                    MeanWeight = 76;
1771                    healthHazardInfo = "Tomato may cause Acid Reflux/Heartburn,Kidney Problems,Diarrhea or Body Aches\nTomato is s
1772                    result = new NutritionalValue(Calorie, Protien,
1773                            Carbohydrate, Fat, MeanWeight, healthHazardInfo);
1774
1775                } else if (name.contains("Orange") || name.contains("orange")) {
1776                    // get nutritional values for a mandarin
1777                    Calorie = 49;
1778                    Protien = (float) 0.9;
1779                    Carbohydrate = (float) 12.5;
1780                    Fat = (float) 0.2;
1781                    MeanWeight = 140;
1782                    healthHazardInfo = "Taking large amounts of sweet orange peel can cause colic, convulsions, or death\n Orange
1783                     result = new NutritionalValue(Calorie, Protien,
1784                             Carbohydrate, Fat, MeanWeight, healthHazardInfo);
1785
1786                 }else {
1787                    // no values available, return null
1788                 }
1789             }
1790         }
1791         return result;
1792     }
1793 }
1794
```

3. CODE TESTING TEAM

3.1 TEAM PROFILE

The code testing team comprises of the following members, all of whom are Undergraduates currently pursuing Bachelor of Technology at Indian Institute of Technology Guwahati , India in the Department of Computer Science and Engineering . All of the members are currently in sophomore year .

I. Nikhil
II. Aditya
III. Avinash

All members of the team are proficient in java and have past experience in developing android applications.

1. CODE INSPECTION REPORTS

1.1 MODULES TESTED BY THE TEAM

The different modules tested by the team includes-

FoodUiActivity :-

- void Initializefromdatabase()
- void BuildrangeSeekbars()
- void SetrangeBarValues()
- void SetlinearLayoutVisibility()
- void CreateevenhueScalar()
- void Handlehsvmode()
- void Detectfruitype()
- void Gethsvfiltermask()
- void Calhsvhistograms()
- void Drawhistogram()
- void Comparehistograms()
- void Handledetectedfruit()
- void Drawhistogramcamparisionmethod()
- void Dialoginputsavefruit()
- void Savefruit()
- void Getnutritionalvalue()

In the FoodUiActivity module, while going through the code ,initializefromdata first initialize all the fruits/vegetable and their nutritional content which is saved in database.

Handlehsvmode handle the live screen depending on hsv value .

Buildrangeseekbar set the range of hsv value of fruit/vegetable of live screen .these hsv values passed to createevenhuescalar on

the basis of it Drawhistogram draw the hsvhistogram Comparehistogram compare the presenthistogram to saved histogram and result of is passed to detectfruit which detect the fruit name and name passed to Getnutritionvalue which provide the nutrition value of the recognized fruits

Report by Nikhil

After going through the code , there were some logical errors found in the algorithm used in the code . All these errors are reported in the report.

- The headers of each module had all the details that are required from a good header ,like – Name of the module , Date on which the module was created ,Author's name , Modification history, Synopsis of the module, Different functions supported , along with their input/output parameters.
- Proper indentation is followed while writing the code.
- There is scope of improvement in commenting the code as the functionality of some functions are not explained properly and hence the complete functionality of those functions could not be understood properly .
- In the modules FoodUiActivity , the error handling is minimum. Error are not handled in functions where fruit detection is occurred on the basis of the hsv value .

- In the module FoodUiActivity , all the variables were private to the class and hence common to all the functions of the class only . So none of the variable was global.
- There were none uninitialized variable found in any module.
- No JUMP (go to) statements were used in the modules.
- All the loops terminated according to their condition . no non-terminating loops were found .
- All the array references used in the code were in the bound of array.

Report by Aditya

After going through the code ,there were some logical error found in the algorithm used in the code . All these logical errors are reported in this report.

- Nil uninitialized variable found in the module .
- The headers of each module had all the details that are required from a good header ,like – Name of the module , Date on which the module was created ,Author's name , Modification history, Synopsis of the module, Different functions supported , along with their input/output parameters.
- No JUMP(go to) statements were found in the module .
- No non-terminating loops were found in the module .
- Naming conventions for global variables, local variables, and constant Identifiers is proper
- Many global variables are used.
- handleDetectedFood() is a very big function ,it should have been splitted into smaller functions.
- In the module, the error handling is minimal. Errors are not handled in the functions where hsv filtering occurs.
- HSV uses only one channel to describe color (H), making it very intuitive to specify color.So hue min value specified here is appropriate.

Report by Avinash

After going through the code , there were some logical errors found in the algorithm used in the code . All these logical errors are reported in this report .

- **Uninitialized variable** = nil found in the module .
- **JUMP(go to)** statements were not found in the code.
- Some of the functions are not properly commented because of which it is hard to understand their functionality .Comments in these functions should be more explanatory.
- All the loops will terminate according to their terminating conditions respectively after some finite iterations .
- No index of array is out of bound.
- Errors are not handled properly in some modules like getnutritionvalue, gettypeoffruit.
- while comparison here it exactly checks hsv values detected and largest contour is selected so error in detected object should be too low.
- Proper commenting after every 2-3 lines is done.
- Some variable could have been given better name which could specify its use.
- The length of most functions are below 15 lines.

CONCLUSION

The members of the code review team submitted the reports during their final meeting with the development team. From these submitted reports, we get to know about a few logical errors that were encountered during the execution of the code inspection and were listed down.

These errors will affect the algorithm used to identify fruit/vegetable by the user. The key errors found from the code review are as follows :-

1. There are some unused import statements and should be removed to reduce memory space used by app.
2. Some variable are not understandable in function calcHsvHistograms() as we cannot determine its use.
3. Function handleDetectedFood() should be split into 2 small functions.

Fixing those bugs can improve the accuracy of the application