

به نام خدا



دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

برنامه سازی موبایل - امید جعفرنژاد

نیمسال دوم ۹۸-۹۹

پروژه کدخوانی

برنامه تشخیص مرز تصویر

نام اعضاي گروه:

• امیرحسین رستمی (۹۶۱۰۱۶۳۵)

• امید شرفی (۹۶۱۰۱۸۳۸)

سرفصل مطالب :

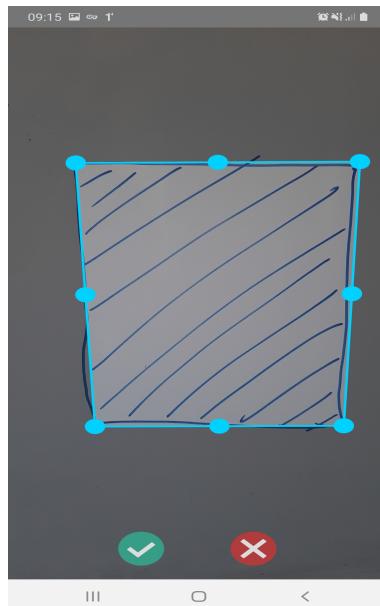
• مرور قابلیت ها و کاربرد برنامه

• بررسی کد و ساختار برنامه

• ترسعه ی پروژه

۱ مرور قابلیت ها و کاربرد برنامه

این پروژه به منظور پیدا کردن مرز یک تصویر مستطیلی و برش آن استفاده میشود. این تشخیص مرز تصویر با استفاده از OpenCV و به صورت لایو با استفاده از دوربین گوشی فرد انجام میشود.



شکل ۱: نمونه تست برنامه

از ویژگی های برنامه می توان به موارد زیر اشاره کرد :

- تشخیص مرز تصویر به صورت لایو با استفاده از دوربین گوشی
- دادن فیدبک به کاربر در صورت دور بودن، نزدیک بودن، یا زاویه‌ی نامناسب دوربین
- امکان تغییر مرزبندی تصویر پس از برش آن توسط برنامه
- کارکرد بهتر برنامه بر روی پس زمینه تیره

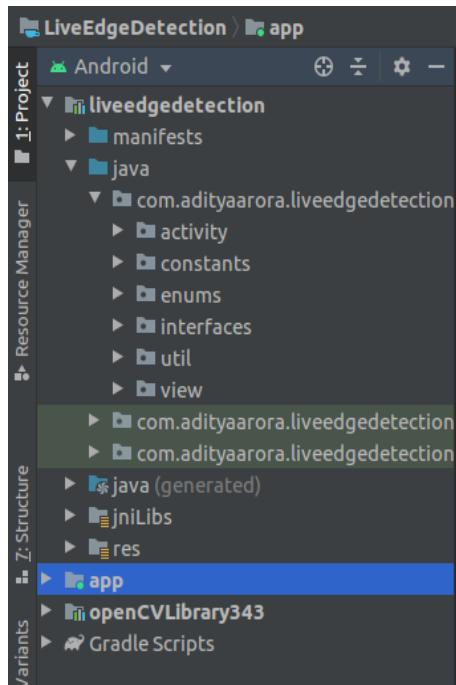


شکل ۲: هشدار نامناسب بودن زاویه دوربین

پس از اجرای برنامه و برش تصویر، این پروژه قابلیتی فراتر از این موضوع مانند ذخیره سازی تصویر یا ادیت تصویر را ندارد. در حقیقت امر، هدف توسعه دهنده‌گان این پروژه، ارائه یک ریپازیتوری قوی برای تشخیص لبه‌های تصویر به منظور استفاده در دیگر برنامه‌ها مانند برنامه‌های اسکن متن و ... میباشد.

۲ برسی کد و ساختار برنامه

در این بخش، به بررسی و توضیح اجمالی فایل های مختلف موجود در پروژه میپردازیم.



شکل ۳: ساختار درختی پروژه

۱.۲ MainActivity

در اکتیویتی اصلی برنامه، در ابتدا لیوت activity-main است میشود. داخل این لیوت همون طور که در تصویر زیر مشاهده میشود، از یک ImageView موجود در liveedgedetection استفاده شده تا تصویر بربار شده در آن نمایش داده شود. بعد از است کردن این کامپوننت در اکتیویتی اصلی، متد startScan صدای زده میشود که ScanActivity را صدا می کند. همچنین در آخرین قسمت اکتیویتی اصلی، متد onActivityResult میشود که پس از برگشتندن به این اکتیویتی اصلی ابتدایا چک کردن وضعیت requestCode، در صورتی که برابر با REQUEST-CODE بود یعنی ما درخواست اسکن عکس را داده بودیم و همچنینresultCode برابر با Activity.RESULT-OK بود، یعنی عکس به درستی اسکن شده بود، در صورتی که دیتاایی موجود بود، این عکس برش داده شده داخل کامپوننت scannedImageView لیوت قرار داده می شود و کاربر عکس برش داده شده را در انتهای مشاهده میکند.

```

public class MainActivity extends AppCompatActivity {

    private static final int REQUEST_CODE = 101;
    private ImageView scannedImageView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        scannedImageView = findViewById(R.id.scanned_image);
        startScan();
    }

    private void startScan() {
        Intent intent = new Intent( packageContext: this, ScanActivity.class);
        startActivityForResult(intent, REQUEST_CODE);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == REQUEST_CODE) {
            if(resultCode == Activity.RESULT_OK) {
                if(null != data && null != data.getExtras()) {
                    String filePath = data.getExtras().getString(ScanConstants.SCANNED_RESULT);
                    Bitmap baseBitmap = ScanUtils.decodeBitmapFromFile(filePath, ScanConstants.IMAGE_NAME);
                    scannedImageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
                    scannedImageView.setImageBitmap(baseBitmap);
                }
            } else if(resultCode == Activity.RESULT_CANCELED) {
                finish();
            }
        }
    }
}

```

شکل ۴ : MainActivity

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/black"
    tools:context="com.adityaarora.liveedgedetection.MainActivity">

    <com.adityaarora.liveedgedetection.view.TouchImageView
        android:id="@+id/scanned_image"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="center" />

</android.support.constraint.ConstraintLayout>

```

شکل ۵ : activity-main.xml

در اینجا یک نکته نیز وجود دارد، آن که همانطور که در کد MainActivity نیز مشاهده میشود، پس از اسکن شدن عکس، اگر عکس به درستی اسکن شده باشد صرفا این عکس نمایش داده میشود و دیگر کاربر امکان دیگری برای اسکن مجدد نخواهد داشت. در حقیقت چون پس از اسکن شدن عکس به اکتیویتی اصلی برمیگردیم و عکس نمایش داده می شود، اگر کاربر دکمه بک را بزنند، چون هیچ نمایش مثلا بر روی فرگمتنی نبوده که به بک استک ادد شده باشد، الان صرفا با زدن بک از اکتیویتی اصلی و از کل برنامه خارج خواهیم شد. این موضوع در صورتی که به این پروژه به صورت یک پروژه مستقل (نه به عنوان پروژه ای که داخل دیگر پروژه ها استفاده شود) نگاه کنیم، یک ایراد بزرگ محسوب خواهد شد. نمونه خروجی تست شده برای این موضوع در تصویر زیر مشاهده می شود.



شکل ۶: امکان اسکن مجدد برای کاربر وجود ندارد

ScanActivity ۲.۲

در ادامه، درخواست اسکن عکس از مین اکتیویتی برنامه به این اکتیویتی داده میشود. در ابتدا بد نیست نگاهی به ماژول liveedgedetection Mainfest بیندازیم.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.adityaarora.liveedgedetection">

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <uses-feature android:name="android.hardware.camera" />

    <application android:supportsRtl="true">
        <activity
            android:name="com.adityaarora.liveedgedetection.activity.ScanActivity"
            android:theme="@style/Theme.AppCompat.Light.NoActionBar" />

    </application>

</manifest>
```

permissions : ۷

همانطور که مشاهده میشود هم پرمیشن دسترسی به دوربین برای اسکن تصویر احتیاج هست، و هم دسترسی به storage برای ذخیره سازی تصویر اسکن شده و نمایش آن در اکتیویتی اصلی که در بخش قبل بررسی شد.

حال اگر به خود ScanActivity برگردیم، این کلاس شامل متدها مختلفی هست که همگی را در ادامه بررسی خواهیم کرد. در ابتدا، اولین مرحله که در onCreate و ابتدای متدها init() اتفاق میافتد، سنت کردن لیوت و کامپونت های آن هست. در نتیجه ابتدا بررسی کامپونت های لیوت activity-scan.xml کاربرد هر کدام میپردازیم.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_scan);
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);

    init();
}

private void init() {
    containerScan = findViewById(R.id.container_scan);
    cameraPreviewLayout = findViewById(R.id.camera_preview);
    captureHintLayout = findViewById(R.id.capture_hint_layout);
    captureHintText = findViewById(R.id.capture_hint_text);
    polygonView = findViewById(R.id.polygon_view);
    cropImageView = findViewById(R.id.crop_image_view);
    cropAcceptBtn = findViewById(R.id.crop_accept_btn);
    cropRejectBtn = findViewById(R.id.crop_reject_btn);
    cropLayout = findViewById(R.id.crop_layout);

    cropAcceptBtn.setOnClickListener(this);
    cropRejectBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT)
                TransitionManager.beginDelayedTransition(containerScan);
            cropLayout.setVisibility(View.GONE);
            mImageSurfaceView.setPreviewCallback();
        }
    });
    checkCameraPermissions();
}
```

شکل ۸: ScanActivity

همانطور که در تصویر زیر نیز مشاهده میشود، در activity-scan.xml دو تا فریم لیوت اصلی هست. یکی camera-preview هست که در ابتدا موجود هست و تصویر گرفته شده از دوربین را به صورت لايو نمایش می دهد. همچنین یک فریم لیوت دیگر با آیدی crop-layout موجود هست که در ابتدا visibility آن gone است شده است و هنگامی که مرز های تصویر تشخیص داده شود، تصویر برش خورده شده در این در این فرم لیوت نمایش داده میشود.

```
<FrameLayout  
    android:id="@+id/camera_preview"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" />  
  
<FrameLayout  
    android:id="@+id/crop_layout"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:layout_gravity="center"  
    android:visibility="gone">  
  
    <ImageView  
        android:id="@+id/crop_image_view"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_gravity="center" />  
  
    <com.adityaarora.liveedgedetection.view.PolygonView  
        android:id="@+id/polygon_view"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_gravity="center" />  
  
    <LinearLayout  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:layout_gravity="bottom|center"  
        android:layout_marginBottom="20dp"  
        android:gravity="center">
```

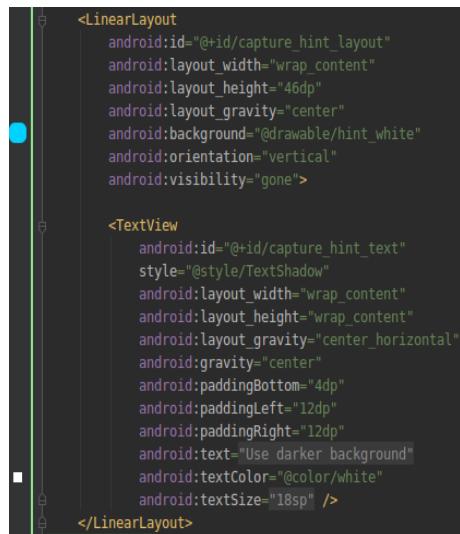
شکل ۹ activity-scan.xml

فریم لیوت crop-layout هست که یک لیوت تعریف شده در خود مازول هست و در حقیقت به صورتی هست که پس از اسکن تصویر امکان جا به جا کردن مرزهای بخش برش داده شده را به کاربر میدهد. در عین حال دو دکمه‌ی crop-reject-btn و crop-accept-btn در یک LinearLayout در پایین صفحه وجود دارند که لیسنر هرکدام در ScanActivity ست شده است. لیسنر crop-reject-btn به این صورت عمل میکند که با فشار آن یعنی ما تصویر اسکن شده را نمیخواهیم و دوباره به اسکن کردن ادامه دهد که در حقیقت در لیسنر این دکمه در ScanActivity ما cropLayout را دوباره پنهان کرده و mImageSurfaceView.setPreviewCallback() را صدا میزنیم که حالا در جلوتر به بحث ScanSurfaceView خواهیم پرداخت. onClick متناسب با فشرده شدن دکمه‌ی crop-accept-btn که یعنی کار من با تصویر تمام شده و تصویر برش داده شده را در اکتیویتی اصلی نمایش بده را نیز در جلوتر بررسی خواهیم کرد.



شکل ۱۰ activity-scan.xml :

در انتهای این بخش برای تکمیل آنالیز لیوت activity-scan.xml صرفا یک بخش باقی میماند که آن capture-hint-layout است. در حقیقت این بخش activity-scan.xml یک LinearLayout هست که وظیفه‌ی نمایش hint و در حقیقت اون فیدبکی هست که کاربر حین اسکن کردن مشاهده میکند. نوشته‌ی آن میتواند هر کدام از مقادیر موجود در ScanHint باشد. این که این بخش چگونه و با چه منطقی و در کدام متد هر کدام از مقادیر موجود در ScanActivity سمت میشود در ادامه توضیح داده خواهد شد.



شکل ۱۱ activity-scan.xml :

```

package com.adityaarora.livedgedetection.enums;

/**
 * Enum that defines receipt detection messages
 */
public enum ScanHint {
    MOVE_AWAY,
    MOVE_CLOSER,
    FIND_RECT,
    ADJUST_ANGLE,
    CAPTURING_IMAGE,
    NO_MESSAGE
}

```

شکل :۱۲ ScanHint

در ادامه، در انتهای متد init() متد checkCameraPermissions() صدا زده میشود. همانطور که در شکل زیر میبینیم، این بخش مسئول گرفتن پرمیشن دوربین از کاربر هست. روند کد این بخش به این صورت است. ابتدا چک میکند، اگر پرمیشن دوربین را نداشتیم، در صورتی که اولین بار هست که از کاربر پرمیشن میخواهیم، onRequestPermissionsResult را صدا میزنیم که داخل requestPermissionsResult میخواهیم، onRequestPermissionsResult را صدا میزنیم. همچنین اگر پرمیشن وجود نداشت و قبل از نیز درخواست پرمیشن را داده بودیم و کاربر قبول نکرده بود به تست پیغام "Enable camera" از دوربین کار میکند و بدون این پرمیشن اکتفا میکنیم. طبیعتاً چون این پروژه تنها با استفاده از دوربین کار میکند و بدون این پرمیشن به صورت کلی کار نمیکند، اگر به پروژه به دید یک پروژه‌ی مستقل نگاه کنیم، میتوانستیم در این بخش یکم بیشتر کاربر را مجاب به دادن پرمیشن به برنامه کنیم. مثلاً در این بخش به جای این تست یک مسج به کاربر نشان بدهیم و توضیح دهیم که چرا به پرمیشن دوربین احتیاج داریم و این که بدون این پرمیشن برنامه کار نمیکند و سعی کنیم در همان جا پرمیشن دوربین را از کاربر بگیریم!

```

private void checkCameraPermissions() {
    if (ContextCompat.checkSelfPermission(context, Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
        isPermissionNotGranted = true;
        if (ActivityCompat.shouldShowRequestPermissionRationale(activity, this,
                Manifest.permission.CAMERA)) {
            Toast.makeText(context, "Enable camera permission from settings", Toast.LENGTH_SHORT).show();
        } else {
            ActivityCompat.requestPermissions(activity, new String[]{Manifest.permission.CAMERA},
                    MY_PERMISSIONS_REQUEST_CAMERA);
        }
    } else {
        if (!isPermissionNotGranted) {
            nImageSurfaceView = new ScanSurfaceView(context, ScanActivity.this, @Scanner this);
            cameraPreviewLayout.addView(nImageSurfaceView);
        } else {
            isPermissionNotGranted = false;
        }
    }
}

@Override
public void onRequestPermissionsResult(int requestCode,
        @NotNull String permissions[], @NotNull int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_CAMERA:
            onRequestCamera(grantResults);
            break;
        default:
            break;
    }
}

```

شکل :۱۳ permission Check

همچنین در صورتی که پرمیشن دوربین را داشته باشیم، یک ScanSurfaceView ایجاد کرده و آن را به عنوان View فریم cameraPreviewLayout ست میکنیم. (کاربرد این فریم نمایش لایو تصویر دوربین هست که در بخش قبل توضیح داده شد)

خب اکنون قبل از این که به تحلیل بخش هسته‌ی اصلی کد یعنی ScanSurfaceView و ابزارهای استفاده شده داخل آن بپردازیم، صرفاً بخش onRequestCamera را در ScanActivity توضیح میدهیم. در این بخش صرفاً در صورت دادن پرمیشن دوربین توسط کاربر یک ترد ست میشود که در متد ران آن همان ایجاد شدن یک ScanSurfaceView و ست شدن آن به عنوان cameraPreviewLayout فریم اتفاق می‌افتد تا کاربر به صورت لایو تصویر دوربین، hint مربوط به آن و برش تصویر در صورت تشخیص مستطیل مشاهده کند. در صورتی هم که کاربر پرمیشن را نداده باشد، صرفاً یک تست نمایش داده میشود.

```
private void onRequestCamera(int[] grantResults) {
    if (grantResults.length > 0
        && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
        new Handler().postDelayed(new Runnable() {
            @Override
            public void run() {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        mImageSurfaceView = new ScanSurfaceView(context, iScanner);
                        cameraPreviewLayout.addView(mImageSurfaceView);
                    }
                });
            }
        }, delayMillis);
    } else {
        Toast.makeText(context, "Please go to app settings and grant camera permission", Toast.LENGTH_SHORT).show();
        this.finish();
    }
}
```

شکل ۱۴: onRequestCamera

توجه کنید که کانسٹراکتور ScanActivity.this شامل یک کانتکست هست که خود canSurfaceView را به عنوان کانتکست پاس میدهیم و یک iScanner ایترفیس. اگر داخل این ایترفیس را مشاهده کنیم شامل دو متد onPictureClicked و displayHint میباشد.

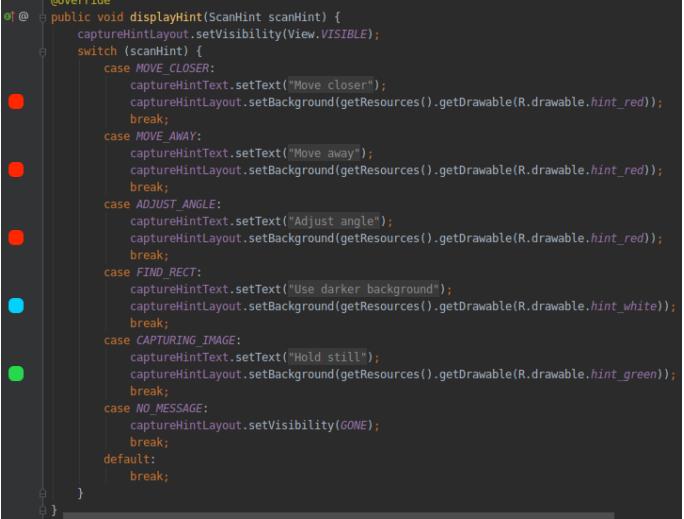
```
package com.adityaarora.liveedgedetection.interfaces;

import ...

/**
 * Interface between activity and surface view
 */
public interface IScanner {
    void displayHint(ScanHint scanHint);
    void onPictureClicked(Bitmap bitmap);
}
```

شکل ۱۵: interface iScanner

متد displayHint برای نمایش hint روی صفحه به کاربر و onPictureClicked برای تغییر مرزهای تصویر انتخاب شده توسط کاربر استفاده می‌شود. در نتیجه وقتی ما ScanActivity.this را به عنوان ScanActivity ساخت می‌کنیم، باید این دو متدهای داخل ScanActivity پیاده سازی شده باشند که همانطور که در تصاویر زیر مشاهده می‌کنید، هر دو داخل ScanActivity تعریف شده‌اند.



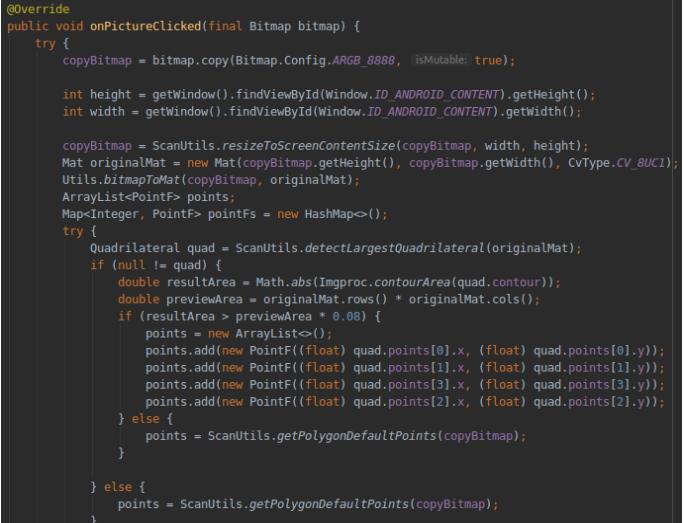
```

@Override
public void displayHint(ScanHint scanHint) {
    captureHintLayout.setVisibility(View.VISIBLE);
    switch (scanHint) {
        case MOVE_CLOSER:
            captureHintText.setText("Move closer");
            captureHintLayout.setBackground(getResources().getDrawable(R.drawable.hint_red));
            break;
        case MOVE_AWAY:
            captureHintText.setText("Move away");
            captureHintLayout.setBackground(getResources().getDrawable(R.drawable.hint_red));
            break;
        case ADJUST_ANGLE:
            captureHintText.setText("Adjust angle");
            captureHintLayout.setBackground(getResources().getDrawable(R.drawable.hint_red));
            break;
        case FIND_RECT:
            captureHintText.setText("Use darker background");
            captureHintLayout.setBackground(getResources().getDrawable(R.drawable.hint_white));
            break;
        case CAPTURING_IMAGE:
            captureHintText.setText("Hold still");
            captureHintLayout.setBackground(getResources().getDrawable(R.drawable.hint_green));
            break;
        case NO_MESSAGE:
            captureHintLayout.setVisibility(GONE);
            break;
        default:
            break;
    }
}

```

شکل ۱۶ : displayHint

در متدهای displayHint و scanHint، صرفاً متناسب با hint‌ها نمایش داده می‌شود. در حقیقت تکست تکستویو captureHintText را سمت می‌کنیم و رنگ متناسب را نیز برای بک‌گراند – Layout سمت می‌کنیم. (دو کامپونت نام بردۀ شده داخل توضیحات بخش activity-scan.xml توضیح داده شده‌اند) چگونگی سمت شدن مقدار scanHint در بخش‌های بعد توضیح داده خواهد شد.



```

@Override
public void onPictureClicked(Bitmap bitmap) {
    try {
        copyBitmap = bitmap.copy(Bitmap.Config.ARGB_8888, true);

        int height = getWindow().findViewById(Window.ID_ANDROID_CONTENT).getHeight();
        int width = getWindow().findViewById(Window.ID_ANDROID_CONTENT).getWidth();

        copyBitmap = ScanUtils.resizeToScreenContentSize(copyBitmap, width, height);
        Mat originalMat = new Mat(copyBitmap.getHeight(), copyBitmap.getWidth(), CvType.CV_8UC1);
        Utils.bitmapToMat(copyBitmap, originalMat);
        ArrayList<Point> points;
        Map<Integer, Point> pointFs = new HashMap<>();
        try {
            Quadrilateral quad = ScanUtils.detectLargestQuadrilateral(originalMat);
            if (quad != null) {
                double resultArea = Math.abs(Imgproc.contourArea(quad.contour));
                double previewArea = originalMat.rows() * originalMat.cols();
                if (resultArea > previewArea * 0.08) {
                    points = new ArrayList<>();
                    points.add(new PointF((float) quad.points[0].x, (float) quad.points[0].y));
                    points.add(new PointF((float) quad.points[1].x, (float) quad.points[1].y));
                    points.add(new PointF((float) quad.points[3].x, (float) quad.points[3].y));
                    points.add(new PointF((float) quad.points[2].x, (float) quad.points[2].y));
                } else {
                    points = ScanUtils.getPolygonDefaultPoints(copyBitmap);
                }
            } else {
                points = ScanUtils.getPolygonDefaultPoints(copyBitmap);
            }
        } catch (Exception e) {
            Log.e("ScanActivity", "Error processing image: " + e.getMessage());
        }
    } catch (Exception e) {
        Log.e("ScanActivity", "Error processing image: " + e.getMessage());
    }
}

```

شکل ۱۷ : onPictureClicked

داخل `onPictureClicked` در حقیقت مرز های برش تصویر و `polygonView` که روی تصویر ترسیم میشود را ست میکند که تنظیم نقاط را با استفاده از توابع موجود در `ScanUtils` انجام داده و نهایتاً در `cropLayout` نمایش داده میشود. نحوه ای عملکرد توابع موجود در این بخش در ادامه بهتر مشخص خواهد شد.

```
        int index = -1;
        for (PointF pointF : points) {
            pointFs.put(++index, pointF);
        }

        polygonView.setPoints(pointFs);
        int padding = (int) 18dp;
        FrameLayout.LayoutParams layoutParams = new FrameLayout.LayoutParams( width: copyBitmap.getWidth()
                + 2 * padding, height: copyBitmap.getHeight() + 2 * padding );
        layoutParams.gravity = Gravity.CENTER;
        polygonView.setLayoutParams(layoutParams);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT)
            TransitionManager.beginDelayedTransition(containerScan);
        cropLayout.setVisibility(View.VISIBLE);

        cropImageView.setImageBitmap(copyBitmap);
        cropImageView.setScaleType(ImageView.ScaleType.FIT_XY);
    } catch (Exception e) {
        Log.e(TAG, e.getMessage(), e);
    }
} catch (Exception e) {
    Log.e(TAG, e.getMessage(), e);
}
}
```

onPictureClicke : ۱۸ شکل

صرفا از ScanActivity یک بخش باقی مانده و آن هم لیستنر cropAcceptBtn هست که داخل آن هم همانطور که مشاهده میشود، بیتیمپ تصویر بریده شده را گرفته و داخل مموری اینترنال ذخیره میکنیم که بعدا در اکتیویتی اصلی و برای نمایش به کاربر استفاده شود. Intent resultCode و مناسب برای دسترسی به تصویر بریده شده نیز سمت میکنیم و به اکتیویتی اصلی بر میگردیم. اتفاقاتی که در ادامه داخل onActivityResult اکتیویتی اصلی رخ میدهد قبلا توضیح داده شده است.

```
    @Override
    public void onClick(View view) {
        Map<Integer, PointF> points = polygonView.getPoints();
        Bitmap croppedBitmap;

        if (ScanUtils.isScanPointsValid(points)) {
            Point point1 = new Point(points.get(0).x, points.get(0).y);
            Point point2 = new Point(points.get(1).x, points.get(1).y);
            Point point3 = new Point(points.get(2).x, points.get(2).y);
            Point point4 = new Point(points.get(3).x, points.get(3).y);
            croppedBitmap = ScanUtils.enhanceReceipt(copy@Bitmap, point1, point2, point3, point4);
        } else {
            croppedBitmap = copyBitmap;
        }

        String path = ScanUtils.saveToInternalMemory(croppedBitmap, ScanConstants.IMAGE_DIR,
                ScanConstants.IMAGE_NAME, mContent: ScanActivity.this, mQuality: 90)[0];
        setResult(Activity.RESULT_OK, new Intent().putExtra(ScanConstants.SCANNED_RESULT, path));
        //bitmap.recycle();
        System.gc();
        finish();
    }
}
```

شکل ۱۹: OnClickListener : cropAcceptBtn

ScanSurfaceView ۳.۲

کلاس ScanSurfaceView اصلی ترین کلاس برنامه و هسته‌ی اصلی برنامه محسوب می‌شود. در ابتدا متغیر‌های کلاس و کانسٹراکتور آن در شکل زیر قابل رویت هست که نکته‌ی خاصی ندارد.

```
public class ScanSurfaceView extends FrameLayout implements SurfaceHolder.Callback {
    private static final String TAG = ScanSurfaceView.class.getSimpleName();
    SurfaceView mSurfaceView;
    private final ScanCanvasView scanCanvasView;
    private int width = 0;
    private int height = 0;

    private final Context context;
    private Camera camera;

    private final IScanner iScanner;
    private CountdownTimer autoCaptureTimer;
    private int secondsLeft;
    private boolean isAutoCaptureScheduled;
    private Camera.Size previewSize;
    private boolean isCapturing = false;

    public ScanSurfaceView(Context context, IScanner iScanner) {
        super(context);
        mSurfaceView = new SurfaceView(context);
        addView(mSurfaceView);
        this.context = context;
        this.scanCanvasView = new ScanCanvasView(context);
        addView(scanCanvasView);
        SurfaceHolder surfaceHolder = mSurfaceView.getHolder();
        surfaceHolder.addCallback(this);
        this.iScanner = iScanner;
    }
}
```

شکل :۲۰ ScanSurfaceView

طبعاً باید توابع surfaceCreated، surfaceChanged و surfaceDestroyed پیاده سازی شوند. برای تابع surfaceCreated همانطور که در شکل زیر مشاهده می‌شود درخواست دوربین میدهیم و آن را سمت میکنیم.

```
@Override
public void surfaceCreated(SurfaceHolder holder) {
    try {
        requestLayout();
        openCamera();
        this.camera.setPreviewDisplay(holder);
    } catch (IOException e) {
        Log.e(TAG, e.getMessage(), e);
    }
}

public void clearAndInvalidateCanvas() {...}

public void invalidateCanvas() { scanCanvasView.invalidate(); }

private void openCamera() {...}
```

شکل :۲۱ ScanSurfaceView

در ادامه در surfaceChanged با استفاده از متدهای getOptimalPreviewSize و setPreviewCallback سایز مناسب برای دوربین گرفته می‌شود و داخل این تابع سمت می‌شود. در انتها نیز stopPreviewAndFreeCamera() را صدا می‌زنیم که در ادامه به تشزیح نحوه‌ی کارکرد این دو متدهای خواهیم پرداخت. همچنین در surfaceDestroyed نیز متدهای stopPreviewAndFreeCamera() را صدا می‌زنیم که داخل این متدهای صرفاً چون کار ما با دوربین تمام شده است دوربین را آزاد می‌کنیم.

```

public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
    if (vWidth == vHeight) {
        return;
    }
    if (previewSize == null)
        previewSize = ScanUtils.getOptimalPreviewSize(camera, vWidth, vHeight);

    Camera.Parameters parameters = camera.getParameters();
    camera.setDisplayOrientation(ScanUtils.configureCameraAngle((Activity) context));
    parameters.setPreviewSize(previewSize.width, previewSize.height);
    if (parameters.getSupportedFocusModes() != null
        && parameters.getSupportedFocusModes().contains(Camera.Parameters.FOCUS_MODE_CONTINUOUS_PICTURE)) {
        parameters.setFocusMode(Camera.Parameters.FOCUS_MODE_CONTINUOUS_PICTURE);
    } else if (parameters.getSupportedFocusModes() != null
        && parameters.getSupportedFocusModes().contains(Camera.Parameters.FOCUS_MODE_AUTO)) {
        parameters.setFocusMode(Camera.Parameters.FOCUS_MODE_AUTO);
    }

    Camera.Size size = ScanUtils.determinePictureSize(camera, parameters.getPreviewSize());
    parameters.setPictureSize(size.width, size.height);
    parameters.setPictureFormat(ImageFormat.JPEG);

    camera.setParameters(parameters);
    requestLayout();
    setPreviewCallback();
}

@Override
public void surfaceDestroyed(SurfaceHolder holder) { stopPreviewAndFreeCamera(); }

private void stopPreviewAndFreeCamera() {...}

```

شکل :۲۲ ScanSurfaceView

دو متده در بخش قبل ذکر شد که به تشریح نحوه‌ی کارکرد آن‌ها میپردازیم. ابتدا متده `getOptimalPreviewSize` موجود در کلاس `ScanUtils` را بررسی میکنیم. این متده یک دوربین و دو متغیر به عنوان طول و عرض گرفته و سایز مناسب را خروجی میدهد. کارکرد این تابع به این صورت هست که ابتدا نسبت طول و عرض مورد نظر ما را حساب کرده، در ادامه از دوربین طول و عرض هایی که ساپورت میکند گرفته و با توجه به اندازه‌ی اختلاف نسبت ها سورت کرده و کوچکترین آن‌ها که کمترین اختلاف و بهترین سایز هست به عنوان خروجی برミگردانیم.

```

public static Camera.Size getOptimalPreviewSize(Camera camera, int w, int h) {
    if (camera == null) return null;
    final double targetRatio = (double) h / w;
    Camera.Parameters cameraParams = camera.getParameters();
    List<Camera.Size> previewSizeList = cameraParams.getSupportedPreviewSizes();
    Collections.sort(previewSizeList, (Comparator<Size>) (size1, size2) - {
        double ratio1 = (double) size1.width / size1.height;
        double ratio2 = (double) size2.width / size2.height;
        Double ratioDiff1 = Math.abs(ratio1 - targetRatio);
        Double ratioDiff2 = Math.abs(ratio2 - targetRatio);
        if (ScanUtils.compareFloats(ratioDiff1, ratioDiff2)) {
            Double h1 = Math.sqrt(size1.width * size1.width + size1.height * size1.height);
            Double h2 = Math.sqrt(size2.width * size2.width + size2.height * size2.height);
            return h2.compareTo(h1);
        }
        return ratioDiff1.compareTo(ratioDiff2);
    });
    return previewSizeList.get(0);
}

```

شکل :۲۳ getOptimalPreviewSize

حال داخل متده `startPreview()` ابتدا `setPreviewCallback` برای دوربین صدا زده میشود و سپس `setPreviewCallback(previewCallback)` که در اینجا به جز محاسبه‌ی ابعاد تصویر، چند متده مهم

که متدهای اصلی تشخیص تصویر و برش آن هستند صدا زده میشوند که به ترتیب به معرفی هر کدام خواهیم پرداخت.

```
public void setPreviewCallback() {...}

private final Camera.PreviewCallback previewCallback = (data, camera) -> {
    if (null != camera) {
        try {
            Camera.Size pictureSize = camera.getParameters().getPreviewSize();
            Log.d(TAG, msg: "onPreviewFrame - received image " + pictureSize.width + "x" + pictureSize.height);

            Mat yuv = new Mat(new Size(pictureSize.width, height: pictureSize.height * 1.5), CvType.CV_8UC1);
            yuv.put( row: 0, col: 0, data);

            Mat mat = new Mat(new Size(pictureSize.width, pictureSize.height), CvType.CV_8UC4);
            Imgproc.cvtColor(yuv, mat, Imgproc.COLOR_YUV2BGR_NV21, dstCn: 4);
            yuv.release();

            Size originalPreviewSize = mat.size();
            int originalPreviewArea = mat.rows() * mat.cols();

            Quadrilateral largestQuad = ScanUtils.detectLargestQuadrilateral(mat);
            clearAndInvalidateCanvas();
        } finally {
            mat.release();
        }
    }
    if (null != largestQuad) {
        drawLargestRect(largestQuad.contour, largestQuad.points, originalPreviewSize, originalPreviewArea);
    } else {
        showFindingReceiptHint();
    }
} catch (Exception e) {
    showFindingReceiptHint();
}
```

شکل ۲۴ : setPreviewCallback

از اینجا به بعد در حقیقت به منظور پردازش تصویر و تشخیص مرزهای مستطیل از کتابخانه `openCV` نیز کمک گرفته شده است. به طور مثال خروجی متد `detectLargestQuadrilateral` یک چندضلعی هست که کلاس چندضلعی به صورت زیر تعریف شده است که در حقیقت شامل نقاط چندضلعی و کانتور آنها میباشد.

```
package com.adityaarora.livedgedetection.view;

import org.opencv.core.MatOfPoint2f;
import org.opencv.core.Point;

/**
 * This class defines detected quadrilateral
 */
public class Quadrilateral {
    public final MatOfPoint2f contour;
    public final Point[] points;

    public Quadrilateral(MatOfPoint2f contour, Point[] points) {
        this.contour = contour;
        this.points = points;
    }
}
```

شکل ۲۵ : Quadrilateral

حال به بررسی نحوه کارکرد تابع `detectLargestQuadrilateral` میپردازیم. این متد هسته ای اصلی پردازش تصویر هست. ابتدا با کمک `Imgproc` از کتابخانه `openCV` در دو مرحله تصویر را نورمالایز کرده و آماده پردازش میکنیم. سپس از الگوریتم `canny` برای تشخیص لبه ها استفاده کرده، سپس یک ترشولد بر روی تصویر اعمال میکنیم که لبه های ضعیف تر تصویر که ممکن است صرفا نویز باشند حذف شوند. حال یک مجموعه ای از لبه ها داریم، همانطور که میدانیم، لبه های تصویر با توجهی اختلاف گرادینان شدید تصویر

موجود در دو طرف آنها به عنوان لبه تشخیص داده می شوند ولی طبیعتاً لبه ای که مد نظر ما هست لبه‌ی دور تصویر نیست. پس باید شدت این لبه‌ها را کم کرده و در عین حال احیاناً اگر جاهايی که لبه تشخیص داده نشدنند و با توجه به بقیه‌ی لبه‌های تشخیص داده شده، احتمال زیادی باید لبه باشد را نیز لبه در نظر بگیریم که این کار را morphologyEx برای ما انجام میدهد. نهایاتا باید بزرگترین کانتور‌های تشکیل شده توسط لبه‌ها را پیدا کرده و سپس از آن‌ها سعی کنیم بزرگترین مستطیل تصویر را استخراج کنیم. این دو بخش به ترتیب توسط دو متده findLargestContours و findQuadrilateral موجودند که در ادامه به اختصار آن‌ها را بررسی می‌کنیم. داک الگوریتم لبه یابی canny در [اینجا](#) موجود است.

```
// step 1.
Imgproc.blur(originalMat, originalMat, new Size(ScanConstants.KSIZE_BLUR, ScanConstants.KSIZE_BLUR));
Core.normalize(originalMat, originalMat, alpha: 0, beta: 255, Core.NORM_MINMAX);
// step 2.
// As most papers are bright in color, we can use truncation to make it uniformly bright.
Imgproc.threshold(originalMat, originalMat, ScanConstants.TRUNC_THRESH, maxval: 255, Imgproc.THRESH_TRUNC);
Core.normalize(originalMat, originalMat, alpha: 0, beta: 255, Core.NORM_MINMAX);
// step 3.
// After above preprocessing, canny edge detection can now work much better.
Imgproc.Canny(originalMat, originalMat, ScanConstants.CANNY_THRESH_U, ScanConstants.CANNY_THRESH_L);
// step 4.
// Cutoff the remaining weak edges
Imgproc.threshold(originalMat, originalMat, ScanConstants.CUTOFF_THRESH, maxval: 255, Imgproc.THRESH_TOZERO);
// step 5.
// Closing - closes small gaps. Completes the edges on canny image; AND also reduces stringy lines near edge of paper.
Imgproc.morphologyEx(originalMat, originalMat, Imgproc.MORPH_CLOSE, morph_kernel, new Point(x: -1, y: -1), iterations: 1);

// Get only the 10 largest contours (each approximated to their convex hulls)
List<MatOfPoint> largestContour = findLargestContours(originalMat, NUM_TOP_CONTOURS: 10);
if (null != largestContour) {
    Quadrilateral mLargestRect = findQuadrilateral(largestContour);
    if (mLargestRect != null)
        return mLargestRect;
}
return null;
```

شکل ۲۶ : detectLargestQuadrilateral

حال در متده findLargestContours ابتدا با استفاده از دستور findContours کانتور‌های تشکیل شده توسط لبه‌ها را پیدا می‌کنیم، سپس توسط دستور convexHull برای هر کدام از کانتورها، پوش محدب هر کانتور را به دست می‌آوریم. این کار باعث رفع لرزه‌ها موجود در لبه‌های کانتورها می‌شود. در نهایت با استفاده از دستور contourArea برای هر کدام از پوش محدب‌ها مساحت آن را محاسبه و به اندازه‌ی NUM-TOP-CONTOURS که در اینجا ۱۰ است شده بود، پوش محدب‌ها را خروجی میدهیم. نهایتاً در تابع findQuadrilateral گوشه‌های شکل پیدا شده و در صورت چهارگوشی بودن شکل یک Quadrilateral خروجی میدهد که میتوان از آن وضعيت شکل و مستطيل بودن آن را تشخیص داد و در نتیجه تشخیص مستطيل حاصل از تشخیص لبه‌های تصویر در اینجا به پایان میرسد و در ادامه به نحوه‌ی ترسیم بر روی تصویر میپردازیم. در اینجا دو نکته حائز اهمیت هست. اولاً که تعداد دیگری توابع محاسباتی در ScanUtils openCV در کد موجود هستند که بررسی دقیق محاسبات ریاضی هر کدام خارج از حوصله بحث هست و صرفا در اینجا همانطور که اشاره شد به معروفی روند کلی عملیات‌ها و توابع اجرا شده برای پردازش تصویر اکتفا شده است.

همچنین در اینجا اشاره به یک نکته‌ی دیگر نیز خالی از لطف نیست. فرض کنید ما این پروژه در در دسترس کاربرانمان قرار دادیم و با آن کار کرده و آن را تست کردند. حال این سناریو ممکن است رخ دهد که کاربران به ما فیدبک دهند که برنامه‌ی شما بسیار در پیدا کردن لبه و اسکن به طور مثال ورقه‌ای که جلوی

دوربین قرار میدهیم و قصد اسکن آن را داریم، سختگیر هست. در همچین شرایطی، با کم کردن ترشولد لبهایی که حذف میکنیم و یا افزایش تعداد کانتورهایی که لبه های آنها را برری میکنیم امکان تنظیم میکنیم. در نتیجه خواستم در اینجا اشاره کنم که اعداد و ثوابتی که در کد وجود دارند حاصل تجربه و فیدبک مصرف کنندگان کد هست و میتوان آن ها را متناسب با نیاز کد و برنامه بهینه سازی کرد.

```
private static List<MatOfPoint> findLargestContours(Mat inputMat, int NUM_TOP_CONTOURS) {
    Mat mHierarchy = new Mat();
    List<MatOfPoint> mContourList = new ArrayList<>();
    //finding contours - as we are sorting by area anyway, we can use RETR_LIST - faster than RETR_EXTERNAL,
    Imgproc.findContours(inputMat, mContourList, mHierarchy, Imgproc.RETR_LIST, Imgproc.CHAIN_APPROX_SIMPLE);

    // Convert the contours to their Convex Hulls i.e. removes minor nuances in the contour
    List<MatOfPoint> mHullList = new ArrayList<>();
    MatOfInt tempHullIndices = new MatOfInt();
    for (int i = 0; i < mContourList.size(); i++) {
        Imgproc.convexHull(mContourList.get(i), tempHullIndices);
        mHullList.add(hull2Points(tempHullIndices, mContourList.get(i)));
    }
    // Release mContourList as its job is done
    for (MatOfPoint c : mContourList)
        c.release();
    tempHullIndices.release();
    mHierarchy.release();

    if (mHullList.size() != 0) {
        Collections.sort(mHullList, (Comparator) (lhs, rhs) -> {
            return Double.compare(Imgproc.contourArea(rhs), Imgproc.contourArea(lhs));
        });
        return mHullList.subList(0, Math.min(mHullList.size(), NUM_TOP_CONTOURS));
    }
    return null;
}

private static Quadrilateral findQuadrilateral(List<MatOfPoint> mContourList) [...]
```

شکل ۲۷: findLargestContours

از متدهای دیگر ScanSurfaceView drawLargestRect که باقی مونده متدهای drawLargestRect هست که یکی از ورودی های آن مجموعه نقاط کانتوری هست که پیدا کردیم و آن را ترسیم میکند. در حقیقت داخل این متدها ابتدا با استفاده از Paint برای ترسیم شکل حاصله استفاده میشود.

```
private void drawLargestRect(MatOfPoint2f approx, Point[] points, Size stdSize, int previewArea) {
    Path path = new Path();
    // ATTENTION: axis are swapped
    float previewWidth = (float) stdSize.height;
    float previewHeight = (float) stdSize.width;

    Log.i(TAG, msg: "previewWidth: " + String.valueOf(previewWidth));
    Log.i(TAG, msg: "previewHeight: " + String.valueOf(previewHeight));

    //Points are drawn in anticlockwise direction
    path.moveTo(x: previewWidth - (float) points[0].y, y: (float) points[0].x);
    path.lineTo(x: previewWidth - (float) points[1].y, y: (float) points[1].x);
    path.lineTo(x: previewWidth - (float) points[2].y, y: (float) points[2].x);
    path.lineTo(x: previewWidth - (float) points[3].y, y: (float) points[3].x);
    path.close();

    double area = Math.abs(Imgproc.contourArea(approx));
    Log.i(TAG, msg: "Contour Area: " + String.valueOf(area));

    PathShape newBox = new PathShape(path, previewWidth, previewHeight);
    Paint paint = new Paint();
    Paint border = new Paint();

    //Height calculated on Y axis
    double resultHeight = points[1].x - points[0].x;
    double bottomHeight = points[2].x - points[3].x;
    if (bottomHeight > resultHeight)
        resultHeight = bottomHeight;
```

شکل ۲۸: drawLargestRect

متدهای drawLargestRect از این جهت اهمیت دارد که متناسب با ابعاد تصویر و چهار نقطه‌ی پیدا شده ارزیابی وضعیت شکل را با استفاده از کلاس ImageDetectionProperties انجام داده و متناسب با پاسخ این کلاس مقدار scanHint و بحث AutoCapture را برای خواندن داده دوربین تنظیم می‌کند.

```
ImageDetectionProperties imgDetectionPropsObj
    = new ImageDetectionProperties(previewWidth, previewHeight, resultWidth, resultHeight,
    previewArea, area, points[0], points[1], points[2], points[3]);

final ScanHint scanHint;

if (imgDetectionPropsObj.isDetectedAreaBeyondLimits()) {...} else if (imgDetectionPropsObj.isDe
Log.i(TAG, msg, "Preview Area 95%: " + 0.95 * previewArea +
    " Preview Area 20%: " + 0.20 * previewArea +
    " Area: " + String.valueOf(area) +
    " Label: " + scanHint.toString());

border.setStrokeWidth(12);
iScanner.displayHint(scanHint);
setPaintAndBorder(scanHint, paint, border);
scanCanvasView.clear();
scanCanvasView.addShape(newBox, paint, border);
invalidateCanvas();
```

شكل ۲۹ : drawLargestRect

این شرطی که در بالا گفته شد در حقیقت در کلاس ImageDetectionProperties با ساخت یک آبجکت از این کلاس و پاس دادن مقادیر مناسب اتفاق می‌افتد. این شرط در حقیقت یک سری شرط‌های ساده‌ی ریاضی هستن که در مجموعه‌ای از متدها در این کلاس جمع آوری شده‌اند.

```
public boolean isDetectedAreaBeyondLimits() {...}

public boolean isDetectedWidthAboveLimit() { return resultWidth / previewWidth > 0.9; }

public boolean isDetectedHeightAboveLimit() { return resultHeight / previewHeight > 0.9; }

public boolean isDetectedHeightAboveNinetySeven() {...}

public boolean isDetectedHeightAboveEightyFive() { return resultHeight / previewHeight > 0.85; }

public boolean isDetectedAreaAboveLimit() { return resultArea > previewArea * 0.75; }

public boolean isDetectedImageDisProportionate() { return resultHeight / resultWidth > 4; }

public boolean isDetectedAreaBelowLimits() { return resultArea < previewArea * 0.25; }

public boolean isDetectedAreaBelowRatioCheck() { return resultArea < previewArea * 0.35; }

public boolean isAngleNotCorrect(MatOfPoint2f approx) {...}

private boolean isRightEdgeDistorted() {...}

private boolean isLeftEdgeDistorted() {...}

private boolean getMaxCosine(MatOfPoint2f approx) {...}

public boolean isEdgeTouching() {...}
```

شكل ۳۰ : ImageDetectionProperties

در حقیقت مجموعه ای از شروط متناسب با پاسخ هر کدام از متدهای صدا زده شده برای آبجکت drawLargestRect در کلاس imgDetectionPropsObj وجود دارد. مثلاً اگر ارتفاع مستطیل بیشتر از لیمیت باشد، مقدار scanHint برابر MOVE-AWAY میگذاریم که به کاربر بگوییم دوربین را دورتر بگیرد و خب متدها cancelAutoCapture() برای لغو گرفتن اسکن تصویر از دوربین را صدا میزنیم. و به همین ترتیب این موضوع را برای شروط مختلف چک می کنیم.

```

if (imgDetectionPropsObj.isDetectedAreaBeyondLimits()) {
    scanHint = ScanHint.FIND_RECT;
    cancelAutoCapture();
} else if (imgDetectionPropsObj.isDetectedAreaBelowLimits()) {
    cancelAutoCapture();
    if (imgDetectionPropsObj.isEdgeTouching()) {...} else {...}
} else if (imgDetectionPropsObj.isDetectedHeightAboveLimit()) {...} else if (imgDetectionPropsObj.isDetectedWidthAboveLimit())
    cancelAutoCapture();
    scanHint = ScanHint.MOVE_AWAY;
} else {
    if (imgDetectionPropsObj.isEdgeTouching()) {
        cancelAutoCapture();
        scanHint = ScanHint.MOVE_AWAY;
    } else if (imgDetectionPropsObj.isAngleNotCorrect(approx)) {
        cancelAutoCapture();
        scanHint = ScanHint.ADJUST_ANGLE;
    } else {
        Log.i(TAG, msg: "GREEN" + "(resultWidth/resultHeight) > 4: " + (resultWidth / resultHeight) +
            " points[0].x == 0 && points[3].x == 0; " + points[0].x + ":" + points[3].x +
            " points[2].x == previewHeight && points[1].x == previewHeight: " + points[2].x + ":" + points[1].x +
            " previewHeight: " + previewHeight);
        scanHint = ScanHint.CAPTURING_IMAGE;
        clearAndInvalidateCanvas();

        if (!isAutoCaptureScheduled) {
            scheduleAutoCapture(scanHint);
        }
    }
}

```

شكل ۳۱: drawLargestRect

در نهایت همانطور که در شکل بالا هم مشاهده میشود، در صورتی که مستطیل آماده برش باشد،- Hint را برابر با CAPTURING-IMAGE سنت کرده و scanCanvasView را توسط متدهای clearAnd-InvalidatesCanvas کلیر کرده و با استفاده از متدهای scheduleAutoCapture با سنت کردن یک تایمر، تصویر را دریافت میکنیم. داخل scheduleAutoCapture میکنیم. داخل shutterCallBack برای drawLargestRect پخش صداست شده است تا هنگام گرفتن عکس و آماده شدن عکس صدا هم همزمان پخش شود. نهایتاً در انتهای متدهای drawLargestRect پس از صدا کردن متدهای displayHint برای نمایش hint به کاربر، شکل را به scanCanvasView اضافه میکنیم. وظیفه ای آن ترسیم اشکال مختلف بر روی تصویر حین اسکن کردن عکس می باشد.

در اینجا بررسی متدها و کلاس های پروژه به پایان میرسد. صرفاً دو کلاس PolygonView که از FrameLayout ارث میبرد و وظیفه ای نمایش صفحه ای برش را دارد و مرزهای تصویر را میتوان با دست در آن تغییر داد و همچنین کلاس TouchImageView که از ImageView ارث میبرد و وظیفه ای تبدیل تصویر بریده شده به مستطیل در ابعاد تصویر گوشی و نمایش آن را دارد باقی مانده اند که آن ها را نیز در ادامه به اختصار توضیح میدهیم.

PolygonView ۴.۲

این ویو در حقیقت از FrameLayout ارث میبرد و برای هندل کردن ۴ ضلعی و ۴ نقطه‌ی بین هرکدام هست که با توجه با این که کاربر کدوم نقطه را جا به جا میکند، متناسب با نقطه‌ای که کاربر صفحه را فشار میدهد و جایی که آن را رها میکند، در صورتی که شکل حاصل همچنان یک چندضلعی محدب باقی بماند، آن شکل را اعمال می‌کند. متغیرهای اصلی این کلاس به شرح زیر هست.

```
/*
 * This class defines polygon for cropping
 */
public class PolygonView extends FrameLayout {

    private static final String TAG = PolygonView.class.getSimpleName();
    private final Context context;
    private Paint paint;
    private ImageView pointer1;
    private ImageView pointer2;
    private ImageView pointer3;
    private ImageView pointer4;
    private ImageView midPointer13;
    private ImageView midPointer12;
    private ImageView midPointer34;
    private ImageView midPointer24;
    private PolygonView polygonView;
    private Paint circleFillPaint;
```

شکل ۳۲: PolygonView

در ادامه صرفا مجموعه‌ای از توابع ریاضیاتی و ترسیمگرها برای هندل کردن این ویو هستن و نکته‌ی خاصی ندارد. مثلا در init() نقاط رو در ابتدا در چهار گوشه‌ی ImageView سمت میکند که کل تصویر رو شامل شود و نقاط میانی و OnTouchListener‌های هرکدام را سمت میکند و در ادامه نقاط را سمت کده و ترسیم را آغاز میکند.

```
private void init() {
    polygonView = this;
    pointer1 = getImageView(x: 0, y: 0);
    pointer2 = getImageView(getWidth(), y: 0);
    pointer3 = getImageView(x: 0, y: getHeight());
    pointer4 = getImageView(getWidth(), getHeight());
    midPointer13 = getImageViewTransparent(x: 0, y: getHeight() / 2);
    midPointer13.setOnTouchListener(new MidPointTouchListenerImpl(pointer1, pointer3));
    midPointer12 = getImageViewTransparent(x: 0, y: getWidth() / 2);
    midPointer12.setOnTouchListener(new MidPointTouchListenerImpl(pointer1, pointer2));
    midPointer34 = getImageViewTransparent(x: 0, y: getWidth() / 2);
    midPointer34.setOnTouchListener(new MidPointTouchListenerImpl(pointer3, pointer4));
    midPointer24 = getImageViewTransparent(x: 0, y: getHeight() / 2);
    midPointer24.setOnTouchListener(new MidPointTouchListenerImpl(pointer2, pointer4));

    addView(pointer1);
    addView(pointer2);
    addView(midPointer13);
    addView(midPointer12);
    addView(midPointer34);
    addView(midPointer24);
    addView(pointer3);
    addView(pointer4);

    initPaint();
}
```

شکل ۳۳: PolygonView

یا مثلا برای راحت تر شدن هندل نقاط آن‌ها را داخل یک مپ ذخیره می‌کند تا راحت و به ترتیب با نقاط کار کند. و چک کردن استاندارد بودن شکل حاصل را متناسب با مختصات نقاط و وضعیت آن‌ها نسبت به یکدیگر به صورت زیر چک میکند.

```

@Override
public boolean onTouchEvent(MotionEvent event) { return super.onTouchEvent(event); }

private boolean isValidShape(Map<Integer, PointF> pointFMap) { return pointFMap.size() == 4; }

private boolean isValidPointer4() {
    return pointer4.getY() > pointer2.getY() && pointer4.getX() > pointer3.getX();
}

private boolean isValidPointer3() {
    return pointer3.getY() > pointer1.getY() && pointer3.getX() < pointer4.getX();
}

private boolean isValidPointer2() {
    return pointer2.getY() < pointer4.getY() && pointer2.getX() > pointer1.getX();
}

private boolean isValidPointer1() {
    return pointer1.getY() < pointer3.getY() && pointer1.getX() < pointer2.getX();
}

```

شکل ۳۴ PolygonView :

و خب گرفتن اکشن کاربر و نشون دادن ریکشن متناسب با اون رفتار و در حقیقت هسته‌ی اصلی این ویو قسمت زیر است که متناسب اکشن کاربر و وضعیت قبلی و جدید نقاط رفتار مناسب را نشان می‌دهد و پس تنظیم چک کردن درست بودن شکل با مجموعه متدهای isvalid می‌نماید. دستور ترسیم دوباره polygonView داده میشود.

```

@Override
public boolean onTouch(View v, MotionEvent event) {
    int eid = event.getAction();
    switch (eid) {
        case MotionEvent.ACTION_MOVE:
            PointF mv = new PointF((float) event.getX() - DownPT.x, (float) event.getY() - DownPT.y);
            if ((StartPT.x + mv.x + v.getWidth()) < polygonView.getWidth() && (StartPT.y + mv.y + v.getHeight()) < polygonView.getHeight())
                break;
        case MotionEvent.ACTION_DOWN:
            ScanActivity.allDraggedPointsStack.push(new PolygonPoints(new PointF(pointer1.getX(), pointer1.getY()),
                    new PointF(pointer2.getX(), pointer2.getY()),
                    new PointF(pointer3.getX(), pointer3.getY()),
                    new PointF(pointer4.getX(), pointer4.getY())));
            DownPT.x = event.getX();
            DownPT.y = event.getY();
            StartPT = new PointF(v.getX(), v.getY());
            latestPoint = new PointF(v.getX(), v.getY());
            break;
        case MotionEvent.ACTION_UP:
            int color = 0;
            if (isValidShape(getPoints()) && isValidPointer4() && isValidPointer3() && isValidPointer2() && isValidPointer1())
                paint.setColor(color);
            break;
        default:
            break;
    }
    polygonView.invalidate();
    return true;
}

```

شکل ۳۵ PolygonView :

TouchImageView ۵.۲

این ویو نیز از ImageView ارث میرد و وظیفه‌ی آن نمایش عکس هست که متناسب با تک انگشتی یا دو انگشتی تاچ کردن کاربر، اگر کاربر دو انگشتی تاچ کند، متناسب با فاصله گرفتن انگشت‌ها از یکدیگر یا نزدیک شدن آنها، وضعیت زوم تصویر را چک کرده و در صورت امکان روم این یا زوم اوت را انجام میدهد. در صورت حرکت تک انگشتی کاربر نیز باز متناسب با وضعیت تصویر، اسکرول در جهت حرکت انگشت کاربر و متناسب با سرعت حرکت دست او انجام می‌شود.

```
private Context context;
private Fling fling;

private ScaleType mScaleType;

private boolean imageRenderedAtLeastOnce;
private boolean onDrawReady;

private ZoomVariables delayedZoomVariables;

//...
private int viewWidth, viewHeight, prevViewWidth, prevViewHeight;

//...
private float matchViewWidth, matchViewHeight, prevMatchViewWidth, prevMatchViewHeight;

private ScaleGestureDetector mScaleDetector;
private GestureDetector mGestureDetector;
private GestureDetector.OnDoubleTapListener doubleTapListener = null;
private OnTouchListener userTouchListener = null;
private OnTouchImageViewListener touchImageViewListener = null;
```

شکل :۳۶ PolygonView

در حقیقت همانطور که مشاهده می‌شود از GestureDetector استفاده شده است. با توجه به حجم بالای توابع امکان معرفی تمام توابع وجود ندارد. صرفاً به عنوان آخرین صحبت این بخش گزارش به این نکته اشاره می‌کنیم که توابع این کلاس را میتوان به سه دسته تابع تقسیم کرد. یک سری توابع مانند setImageBitmap و ... مسئول تنظیم محتوای این ویو هستند. یک دسته‌ی دیگر از توابع بررسی وضعیت زوم تصویر و تنظیم و ترسیم را بر عهده دارند مانند توابع getcurrentZoom setZoom و ... و دسته‌ی سوم توابع این کلاس نیز عهده دار بررسی وضعیت ارتباط کاربر با این ویو و انتخاب ریکشن صحیح پاسخ به رفتارهای کاربر هستند که در حقیقت در حقیقت در GestureListener با canScrollVertically و توابع کمکی مانند

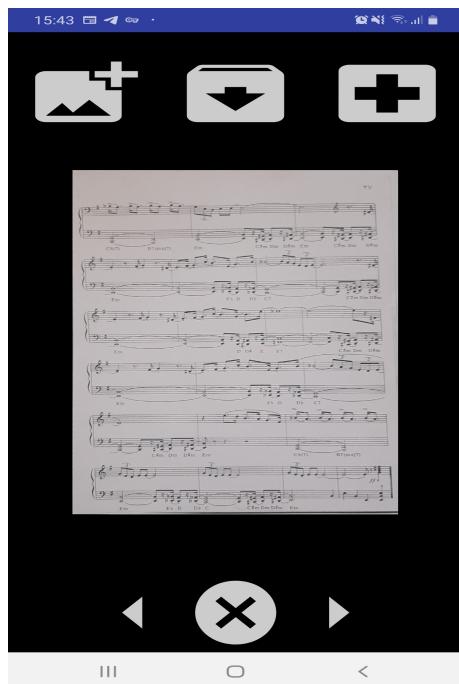
شده‌اند.

در اینجا بررسی کد به پایان میرسد. درادامه به معرفی تغییرات اعمال شده توسط خودمان بر روی این پروژه میپردازیم.

۳ توسعه‌ی پروژه

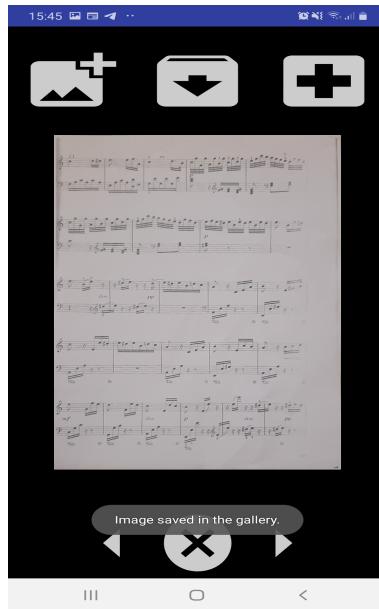
برای توسعه‌ی این پروژه، خود پروژه که صرفاً لبه تشخیص میدهد و کاربر میتواند آن را برش بزند و پس از بریدن تصویر دیگر امکانات دیگری در دسترس کاربر قرار نمی‌دهد. همچنین پروژه‌ی موجود در گیت ایشو خاصی نداشت که حل شود. در نتیجه ما تصمیم بر آن گرفتیم که با استفاده از این پروژه و توسعه‌ی آن، یک اسکنر کامل بسازیم.

در شکل زیر نمونه خروجی اسکنر ما موجود است. همان طور که مشاهده می‌شود شامل دو ردیف دکمه کنترلی می‌باشد. در ردیف بالا به ترتیب از چپ به راست دکمه‌های دخیره کردن عکس جاری در گالری کاربر، ایجاد فایل پی‌دی‌اف از عکس‌های موجود برنامه و دکمه‌ی اضافه کردن عکس جدید به عکس‌های موجود می‌باشد.



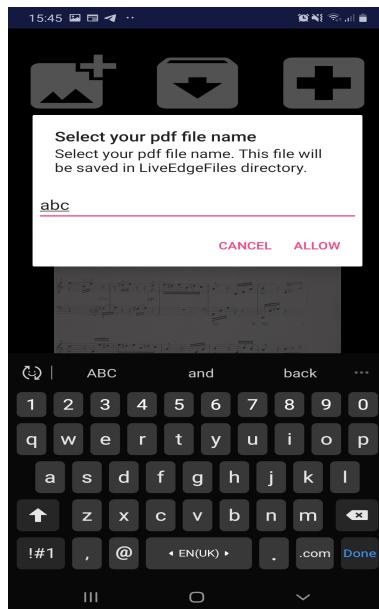
شکل ۳۷: برنامه‌ی اسکنر

در هر کدام از امکانات برنامه، متناسب با وضعیت پرمیشنی که کاربر به ما داده است، عکس العمل مناسب نشان داده می‌شود تا هم از کاربر پرمیشن گرفته شود و هم در عین حال برنامه در هیچ شرایطی کوش نکند.

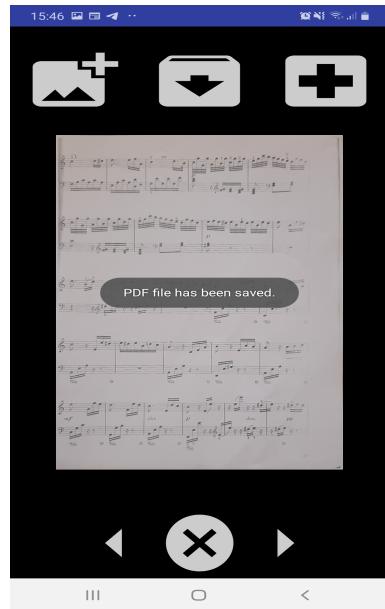


شکل ۳۸: ذخیره عکس فعلی در گالری

همچنین در صورتی که کاربر بخواهد عکس های خود را به صورت یک فایل پی دی اف ذخیره کند نیز با گرفتن نام مورد نظر برای فایل، فایل در مکان مناسب ذخیره می شود و کاربر میتواند از این فایل پی دی اف شده عکس های اسکن شده، استفاده کند.



شکل ۳۹: انتخاب نام مورد نظر برای ذخیره فایل توسط کاربر



شکل ۴۰: ذخیره پی دی اف

در نهایت، ردیف پایین کنترلی نیز به ما امکان جا به جا شدن بین عکس های اسکن شده و حذف هر کدام از عکس های مورد نظر را به ما می دهد و در حقیقت مانند یک آلبوم عمل میکند.