Lab Report for EI447 - Mobile Internet

Lab 3: QR Code Scanner

Yiwen Song

2020-5-1

Preliminaries

Development Environment

- Operating system: Windows 10, function version 1909.
- IDE: Android Studio 3.6.2 (latest version).
- Build: Gradle 5.6.4.
- JDK: Java 1.8.0_212 for AS Runtime Environment. Java > 8 not suggested for the lab project.
- Android Simulator: Pixel 2 API Q (1080*1920; Android 10.0/Pure Android; x86).
- Android Tester: OXF-AN00/Honor V30 (1080*2400; Android 10.0/MagicUI 3.0.1; ARM).

Software Information

- Support Android > 5.0 devices.
- Open-source under GNU/GPL License v3.0. Any modifications/redistributions of the project software must be restricted by the GPL license. For more information, check the *open source* of the program.
- Copyright information: During the development of the project, the following external references have been used:
  - Template for the report is *eisvogel*.
  - Based on *QR Code Scanner* given by Intelligent Internet of Things (IIoT) Lab, Shanghai Jiao Tong University.
  - QR Code Encoder/Decoder Technically supported by *zxing*.

MainActivity & Front-end

MainActivity

The *MainActivity* class controls the main panel of the app. It contains two functions, *onActivityResult()* which displays the result of the decoder, and *onCreate()* which initializes two buttons guiding to the encoder or decoder page.

In function *onActivityResult()* we display the result of the decoder by an AlertDialog box. If the result code returned by the decoder is *RESULT_OK*, which means the decoder has decoded the result, there

will be an alert box displaying the decode result. Meanwhile, there will be a button “Copy to Clipboard”. By clicking this button, the decoded result will be copied to clipboard, which will be convenient to the user to apply the decoded result.

```
if (resultCode == RESULT_OK) {
    Bundle bundle = data.getExtras();
    final String scanResult = bundle != null ? bundle.getString("result
        ") : null;
    new AlertDialog.Builder(MainActivity.this)
            .setTitle("Decode Result")
            .setMessage(scanResult)
            .setNegativeButton("Confirm", new DialogInterface.
                OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which)
                    {
                    dialog.dismiss();
                }
            })
            .setPositiveButton("Copy to Clipboard", new DialogInterface
                .OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which)
                    {
                    ClipboardManager clipboard = (ClipboardManager)
                        getSystemService(CLIPBOARD_SERVICE);
                    ClipData clipData = ClipData.newPlainText("result",
                         scanResult);
                    clipboard.setPrimaryClip(clipData);
                    Toast.makeText(MainActivity.this, "Copied to
                        Clipboard", Toast.LENGTH_SHORT).show();
                }
            }).show();
}
```

Otherwise, if the result code is negative or returns error, an alert box will be displayed informing the user that no QR code is recognized.

```
else {
    new AlertDialog.Builder(MainActivity.this)
            .setTitle("Decode Result")
            .setMessage("Can't recognize QR Code.")
            .setPositiveButton("Confirm", new DialogInterface.
                OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which)
                    {
                    dialog.dismiss();
                }
            }).show();
```

```
}
```

In the *onCreate()* function, we link two buttons corresponding to the encoder and decoder with the button at the front-end, and direct these buttons to the sub-pages defined by the encoder and decoder.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Button encoder = findViewById(R.id.btn_encoder);
    Button decoder = findViewById(R.id.btn_decoder);

    View.OnClickListener mainListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent;
            switch (v.getId()) {
                case R.id.btn_encoder:
                    intent = new Intent(MainActivity.this, QREncoder.
                        class);
                    startActivity(intent);
                    break;
                case R.id.btn_decoder:
                    intent = new Intent(MainActivity.this, QRDecoder.
                        class);
                    startActivityForResult(intent, FragmentActivity.
                        RESULT_CANCELED);
                    break;
                default:
                    break;
            }
        }
    };

    encoder.setOnClickListener(mainListener);
    decoder.setOnClickListener(mainListener);
}
```

MainActivity Front-end

The front-end of main activity contains three objects, a text suggesting the name of the app, and two buttons directing to encoder and decoder. Codes are not demonstrated here. The demonstration of the main page is shown in the figure below.

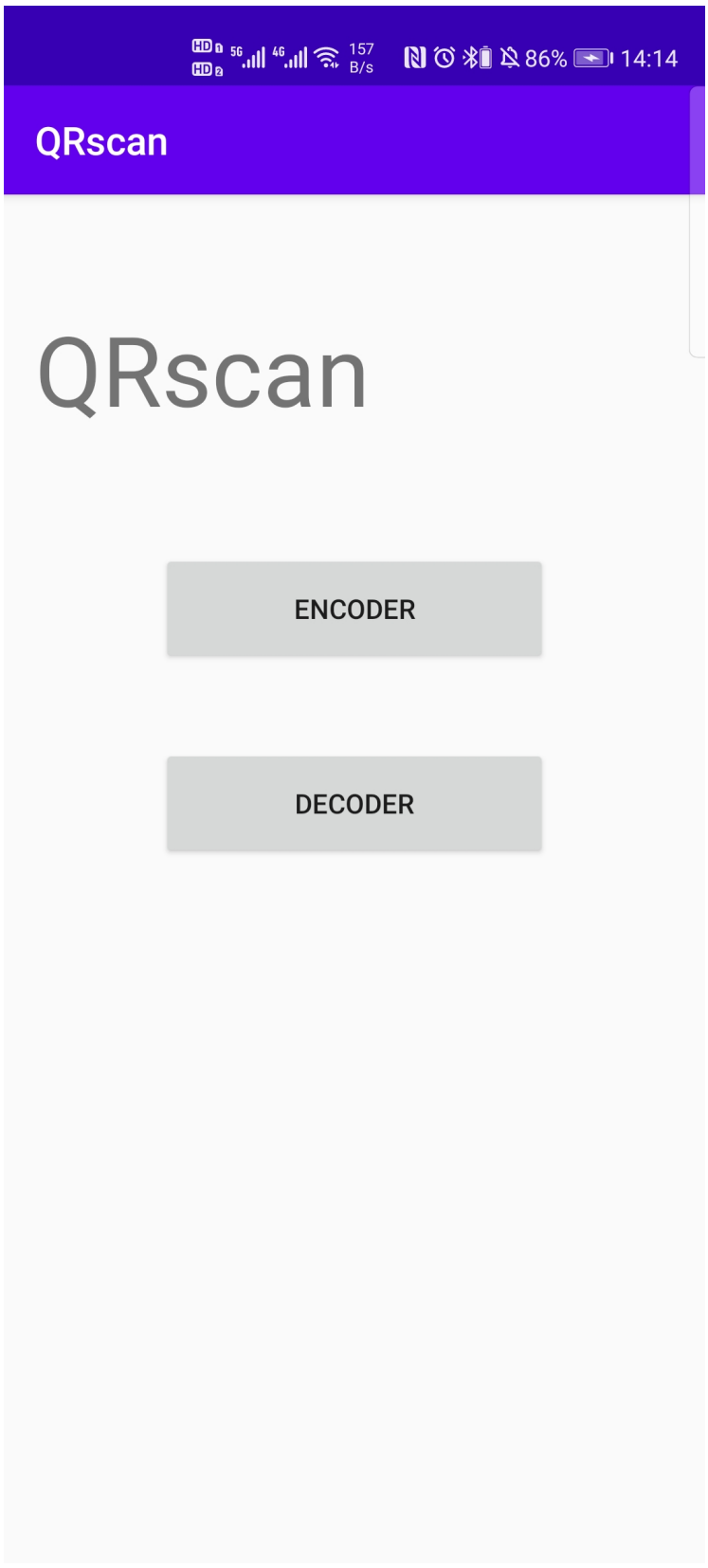

Figure 1: Demo for MainActivity Front Panel.

Encoder Front-end

The encoder panel also contains three object, a text suggesting the name of the app, a text input box, and a button to generate the result. The demonstration is shown below.

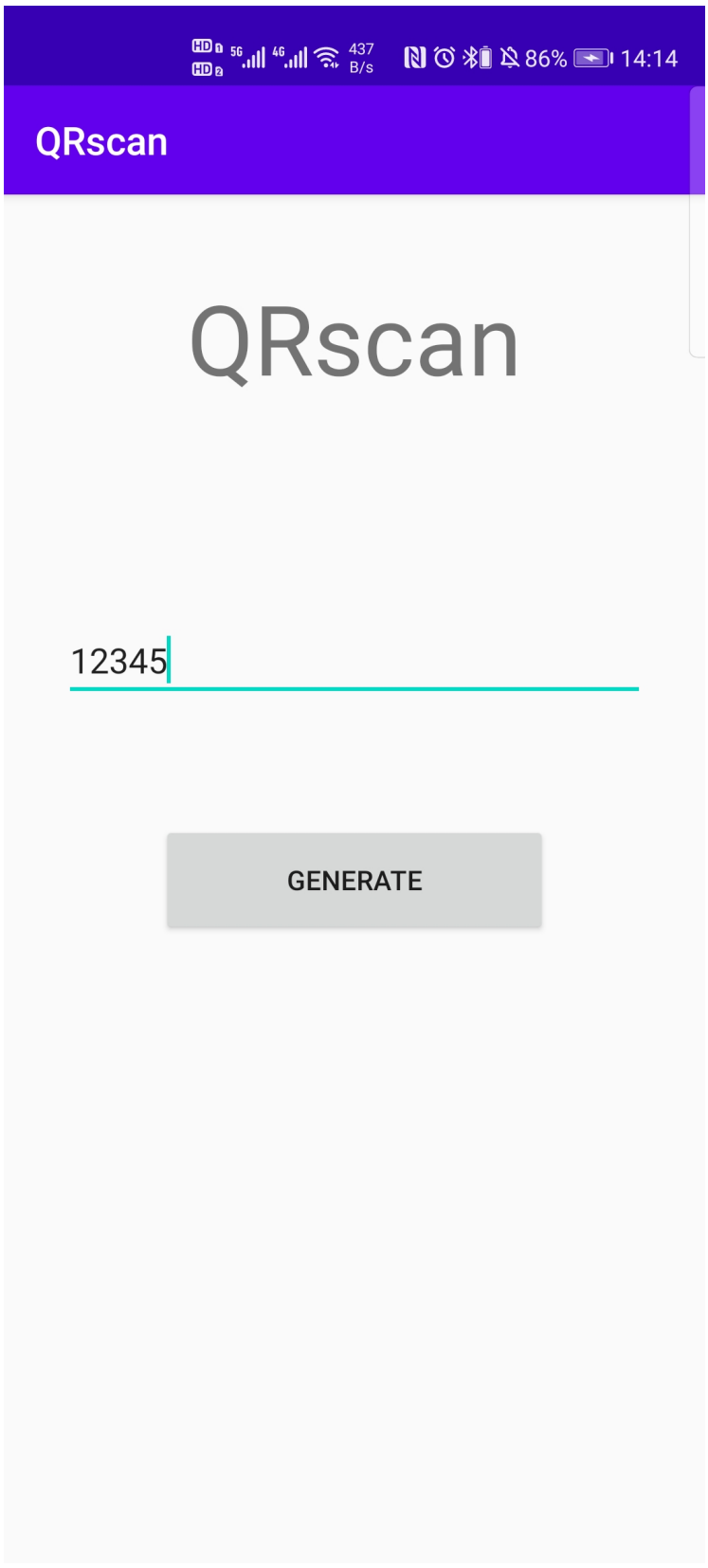

Figure 2: Demo for Encoder Front Panel.

Decoder Front-end

The decoder panel contains a *SurfaceView* object to preview the camera result, an *ImageView* object to show an edge of a square (indicating scanning area) at the center, and another *ImageView* object to show the captured image containing the QR code. When we use the decoder, the effect is shown below. The rectangle is generated by

```
<shape
    xmlns:android="http://schemas.android.com/apk/res/android">
    <corners android:radius="1dp" />
    <stroke android:color="@android:color/holo_blue_bright" android:
       width="3dp" />
</shape>
```

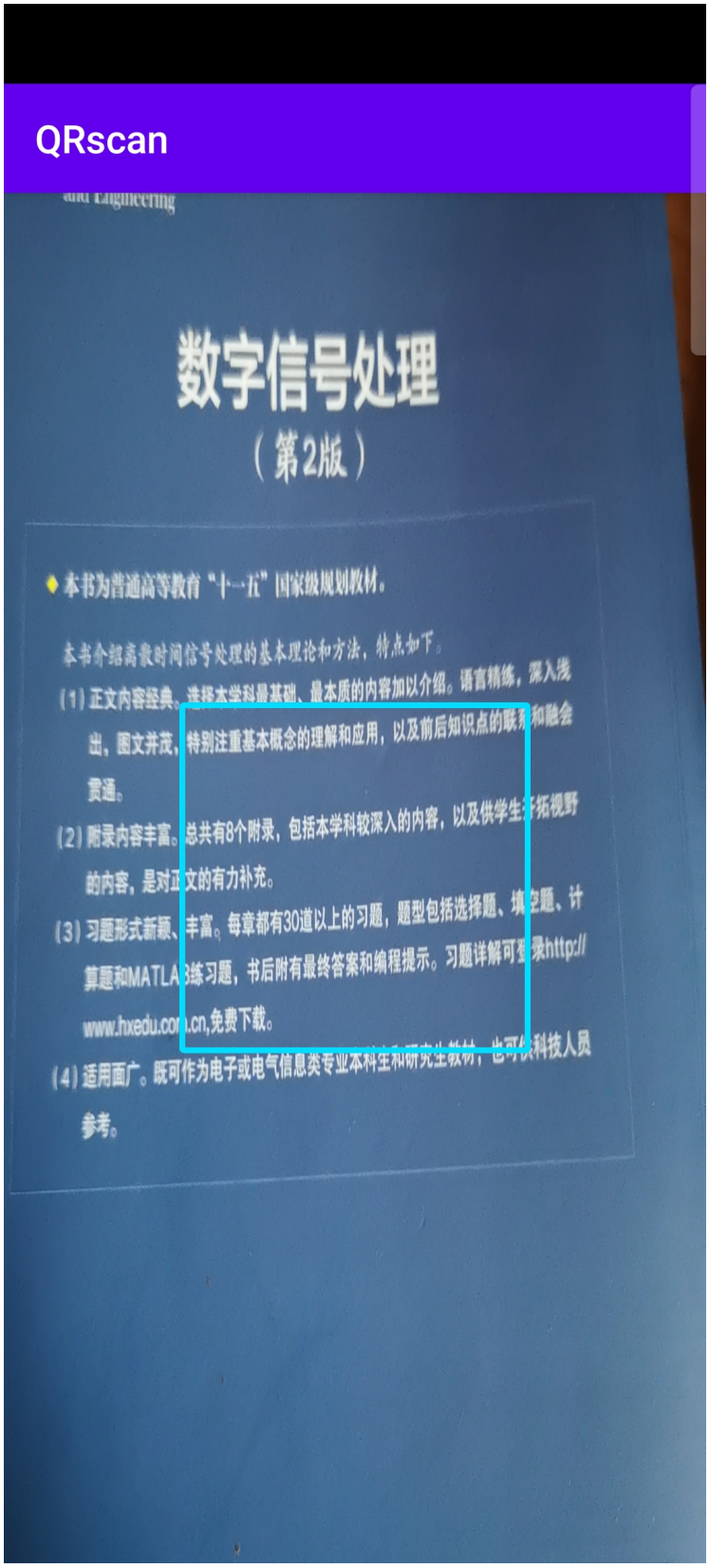

Figure 3: Demo for Decoder Front Panel.

Encoder

The structure of encoder is relatively easy. After we get the input text by

```
String contentStr = textContent.getText().toString();
```

We can use the tools provided by zxing to create a *BitMatrix* object representing the QR code. Then by transforming the *BitMatrix* into *BitMap* then into *ImageView* we can demonstrate the QR code in a visible manner. Finally, we use an alert box to demonstrate the result QR code.

```
if (!contentStr.equals("")) {
    BitMatrix matrix = new MultiFormatWriter().encode(contentStr,
        BarcodeFormat.QR_CODE, 300, 300);
    int width = matrix.getWidth();
    int height = matrix.getHeight();

    int[] pixels = new int[height * width];

    for (int x = 0; x < height; x++)
        for (int y = 0; y < width; y++)
            if (matrix.get(x, y))
                pixels[x*height+y] = Color.BLACK;

    Bitmap bitmap = Bitmap.createBitmap(width, height, Bitmap.Config.
        ARGB_8888);
    bitmap.setPixels(pixels, 0, width, 0, 0, width, height);

    ImageView image1 = new ImageView(QREncoder.this);
    image1.setImageBitmap(bitmap);

    new AlertDialog.Builder(QREncoder.this)
            .setTitle(R.string.app_name)
            .setIcon(android.R.drawable.ic_dialog_info)
            .setView(image1)
            .setPositiveButton("Confirm", new DialogInterface.
                OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which)
                    {
                    dialog.dismiss();
                }
            })
            .show();
}
else {
    Toast.makeText(QREncoder.this, "Text cannot be empty", Toast.
        LENGTH_SHORT).show();
}
```

Of course empty-text detection is added to ensure robustness. The result for input “12345” is shown in the following figure.

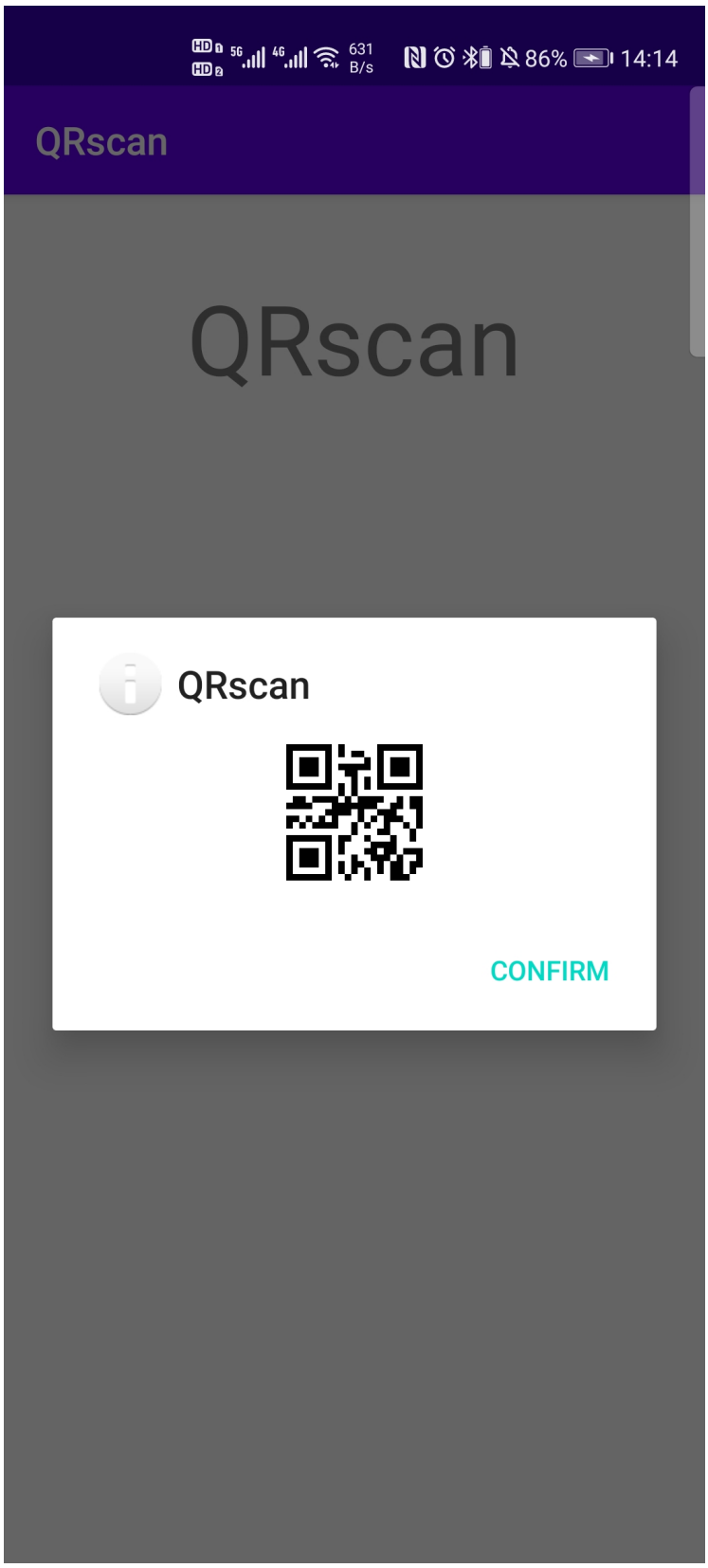

Figure 4: Demonstration for Encoder Result.

Decoder

The provided code uses *android.hardware.Camera* package, which is abandoned in our development environment (Android >= 5.0). Therefore, we completely re-write the decoder module using *android.hardware.camera2* package instead to realize similar functions.

For the class *QRDecoder* we need it to implement *SurfaceHolder.Callback* method and *View.onClickListener* method, shown as below.

```
public class QRDecoder extends AppCompatActivity implements
    SurfaceHolder.Callback, View.OnClickListener
```

The *Callback* method is used to control the *SurfaceView* object to show the camera previews, and the *onClickListener* method provides the function that when we click the screen, the screen captures the figure and recognizes the QR code in the available area (restricted by the square area). The member variables are shown below.

```
private SurfaceView surfaceView = null;
private SurfaceHolder surfaceHolder = null;
private CameraDevice cameraDevice = null;
private CameraCaptureSession cameraCaptureSession = null;
ImageReader imageReader = null;
ImageView imageView;

private Handler childHandler;

// const to control the orientation of the taken figure
private static final SparseIntArray ORIENTATIONS = new SparseIntArray()
    ;
static {
    ORIENTATIONS.append(Surface.ROTATION_0, 90);
    ORIENTATIONS.append(Surface.ROTATION_90, 0);
    ORIENTATIONS.append(Surface.ROTATION_180, 270);
    ORIENTATIONS.append(Surface.ROTATION_270, 180);
}
```

Some surface view components are shown below.

```
private void initSurfaceView() {
    surfaceView = this.findViewById(R.id.preview_view);
    imageView = this.findViewById(R.id.preview_img);
    surfaceView.setOnClickListener(this);
    surfaceHolder = surfaceView.getHolder();
    surfaceHolder.addCallback(QRDecoder.this);
    surfaceHolder.setFormat(PixelFormat.TRANSPARENT);
}

@Override
```

```
11 public void onCreate(Bundle savedInstanceState) {
12     super.onCreate(savedInstanceState);
13     requestWindowFeature(Window.FEATURE_NO_TITLE);
14     getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
           WindowManager.LayoutParams.FLAG_FULLSCREEN);
15     setContentView(R.layout.decoder);
16
17     initSurfaceView();
18 }
19
20 public void surfaceCreated(SurfaceHolder holder) {
21     Log.i(getString(R.string.app_name), "SurfaceHolder.Callback:
           Surface Created.");
22     initCameraPreview();
23 }
24
25 @Override
26 public void surfaceChanged(SurfaceHolder holder, int format, int width,
       int height) {
27     Log.v(getString(R.string.app_name), "SurfaceHolder.Callback:
           Surface Changed.");
28 }
29
30 @Override
31 public void surfaceDestroyed(SurfaceHolder holder) {
32     Log.v(getString(R.string.app_name), "SurfaceHolder.Callback:
           Surface Destroyed.");
33     if (cameraDevice != null) {
34         cameraDevice.close();
35         QRDecoder.this.cameraDevice = null;
36     }
37 }
```

The *onCreate()* function initializes the decode panel. The *surfaceCreated()* function will call the *initCameraPreview()* function to show the camera preview on the *SurfaceView* object. The *surfaceDestroyed()* function shuts down the camera.

The *initCameraPreview()* function mainly provides the function for decoding the QR code in the captured image (which is delivered by the *onClick()* function shown later), and the camera preview that is supported by *takePreview()* function later. For decoding, a bitmap is first generated by the captured image, and then the bitmap is transformed into a binary bitmap supported by zxing. Finally zxing decodes the binary bitmap and save the result in a *Result* object, which will be further returned to main activity control.

```
1 @RequiresApi(api = Build.VERSION_CODES.LOLLIPOP)
2 private void initCameraPreview() {
3     Log.v(getString(R.string.app_name), "Initialize Camera Preview.");
4     HandlerThread handlerThread = new HandlerThread("Camera2");
5     handlerThread.start();
```

```
6       childHandler = new Handler(handlerThread.getLooper());
7       Handler mainHandler = new Handler(getMainLooper());
8       String cameraID = "" + CameraCharacteristics.LENS_FACING_FRONT;
9       imageReader = ImageReader.newInstance(surfaceView.getWidth(),
            surfaceView.getHeight(), ImageFormat.JPEG, 1);
10      imageReader.setOnImageAvailableListener(new ImageReader.
            OnImageAvailableListener() {
11          @Override
12          public void onImageAvailable(ImageReader imgReader) {
13              if (cameraDevice != null)
14                  cameraDevice.close();
15              surfaceView.setVisibility(View.GONE);
16              imageView.setVisibility(View.VISIBLE);
17
18              // load image into imageView and display
19              Image image = imgReader.acquireNextImage();
20
21              ByteBuffer byteBuffer = image.getPlanes()[0].getBuffer();
22              byte[] bytes = new byte[byteBuffer.remaining()];
23              byteBuffer.get(bytes);
24              final Bitmap bitmap = BitmapFactory.decodeByteArray(bytes,
                    0, bytes.length);
25              if (bitmap != null) {
26                  Log.v(getString(R.string.app_name), "Bitmap: "+bitmap.
                        toString());
27                  imageView.setImageBitmap(bitmap);
28
29                  // scan QR code in the bitmap
30                  // transfer the bitmap into binary one
31                  int[] array = new int[bitmap.getWidth()*bitmap.
                        getHeight()];
32                  bitmap.getPixels(array, 0, bitmap.getWidth(), 0, 0,
                        bitmap.getWidth(), bitmap.getHeight());
33                  LuminanceSource source = new RGBLuminanceSource(bitmap.
                        getWidth(), bitmap.getHeight(), array);
34                  BinaryBitmap binaryBitmap = new BinaryBitmap(new
                        HybridBinarizer(source));
35
36                  Log.v(getString(R.string.app_name), binaryBitmap.
                        toString());
37
38                  Reader reader = new QRCodeReader();
39                  // read QR code
40                  try {
41                      Result result = reader.decode(binaryBitmap);
42                      String text = result.getText(); // equivalent to
                            result.toString()
43
44                      Intent intent = new Intent();
45                      Bundle bundle = new Bundle();
46                      bundle.putString("result", text);
```

```
47                      intent.putExtras(bundle);
48                      setResult(RESULT_OK, intent);
49                      finish();
50                  } catch (Exception e) {
51                      e.printStackTrace();
52                      setResult(RESULT_CANCELED);
53                      finish();
54                  }
55              }
56
57          }
58      }, mainHandler);
59      CameraManager cameraManager = (CameraManager) this.getSystemService
            (Context.CAMERA_SERVICE);
60      try {
61          if (ActivityCompat.checkSelfPermission(this, Manifest.
                permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
62              return;
63          }
64          cameraManager.openCamera(cameraID, DeviceStateCallBack,
                mainHandler);
65      } catch (Exception e) {
66          e.printStackTrace();
67      }
68  }
```

The *DeviceStateCallBack* controls the call back state of the camera and their accordingly reactions. It is defined as follows.

```
1   private CameraDevice.StateCallback DeviceStateCallBack = new
        CameraDevice.StateCallback() {
2       @Override
3       public void onOpened(@NonNull CameraDevice camera) {
4           Log.v(getString(R.string.app_name), "DeviceStateCallBack:
                Camera Opened.");
5           cameraDevice = camera;
6           takePreview();
7       }
8
9       @Override
10      public void onDisconnected(@NonNull CameraDevice camera) {
11          Log.v(getString(R.string.app_name), "DeviceStateCallBack:
                Camera Disconnected.");
12          if (cameraDevice != null) {
13              cameraDevice.close();
14              QRDecoder.this.cameraDevice = null;
15          }
16      }
17
18      @Override
19      public void onError(@NonNull CameraDevice camera, int error) {
```

```
20          Log.v(getString(R.string.app_name), "DeviceStateCallBack:
                Return Camera Error.");
21          Toast.makeText(QRDecoder.this, "Camera open failed", Toast.
                LENGTH_SHORT).show();
22      }
23  };
```

We can see that when the camera opens we will take a preview, and the preview will later be sent on the screen. The *takePreview()* function is defined as follows. In this function, a *CaptureRequestBuilder* and a *CaptureRequest* object are created to fulfill the task. The builder does basic settings for capturing and builds the physical signal from the camera into recognizable *CaptureRequest* objects, and the *CaptureRequest* object will be shown on the screen in a visible figure manner.

```
1   private void takePreview() {
2       try {
3           final CaptureRequest.Builder previewRequestBuilder =
                cameraDevice.createCaptureRequest(CameraDevice.
                TEMPLATE_PREVIEW);
4           previewRequestBuilder.addTarget(surfaceHolder.getSurface());
5
6           cameraDevice.createCaptureSession(Arrays.asList(surfaceHolder.
                getSurface(), imageReader.getSurface()), new
                CameraCaptureSession.StateCallback() {
7               @Override
8               public void onConfigured(@NonNull CameraCaptureSession
                    session) {
9                   if (cameraDevice == null)
10                      return;
11                  cameraCaptureSession = session;
12                  try {
13                      // AF mode
14                      previewRequestBuilder.set(CaptureRequest.
                            CONTROL_AF_MODE, CaptureRequest.
                            CONTROL_AF_MODE_CONTINUOUS_PICTURE);
15                      // auto flash light
16                      previewRequestBuilder.set(CaptureRequest.
                            CONTROL_AE_MODE, CaptureRequest.
                            CONTROL_AE_MODE_ON_AUTO_FLASH);
17                      // display preview
18                      CaptureRequest previewRequest =
                            previewRequestBuilder.build();
19                      cameraCaptureSession.setRepeatingRequest(
                            previewRequest, null, childHandler);
20                  } catch (CameraAccessException e) {
21                      Log.e(getString(R.string.app_name), "Camera Access
                            Error.");
22                      e.printStackTrace();
23                  }
24              }
25
```

```
26                @Override
27                public void onConfigureFailed(@NonNull CameraCaptureSession
                      session) {
28                    Toast.makeText(QRDecoder.this, "Camera Preview Failed."
                          , Toast.LENGTH_SHORT).show();
29                }
30            }, childHandler);
31        } catch (Exception e) {
32            e.printStackTrace();
33        }
34    }
```

Finally, when we click on the screen, the app will take an image and the image will be sent to the *initCameraPreview()* function for further decoding. The main structure of the *takePicture()* function is the same as the *takePreview()* function, except that it does rotation if the picture is not taken vertically.

```
1   @Override
2   public void onClick(View v) {
3       takePicture();
4   }
5
6   private void takePicture() {
7       if (cameraDevice == null)
8           return;
9       final CaptureRequest.Builder captureRequestBuilder;
10      try {
11          captureRequestBuilder = cameraDevice.createCaptureRequest(
                CameraDevice.TEMPLATE_STILL_CAPTURE);
12          captureRequestBuilder.addTarget(imageReader.getSurface());
13          // AF mode
14          captureRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
                CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);
15          // auto flash light
16          captureRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE,
                CaptureRequest.CONTROL_AE_MODE_ON_AUTO_FLASH);
17
18          // get the orientation of the phone
19          int rotation = getWindowManager().getDefaultDisplay().
                getRotation();
20
21
22          captureRequestBuilder.set(CaptureRequest.JPEG_ORIENTATION,
                ORIENTATIONS.get(rotation));
23          CaptureRequest captureRequest = captureRequestBuilder.build();
24          cameraCaptureSession.capture(captureRequest, null, childHandler
                );
25      } catch (Exception e) {
26          e.printStackTrace();
27      }
28  }
```

The decode result of a QR code directing to a Wechat page for Tsinghua University Press is shown below.

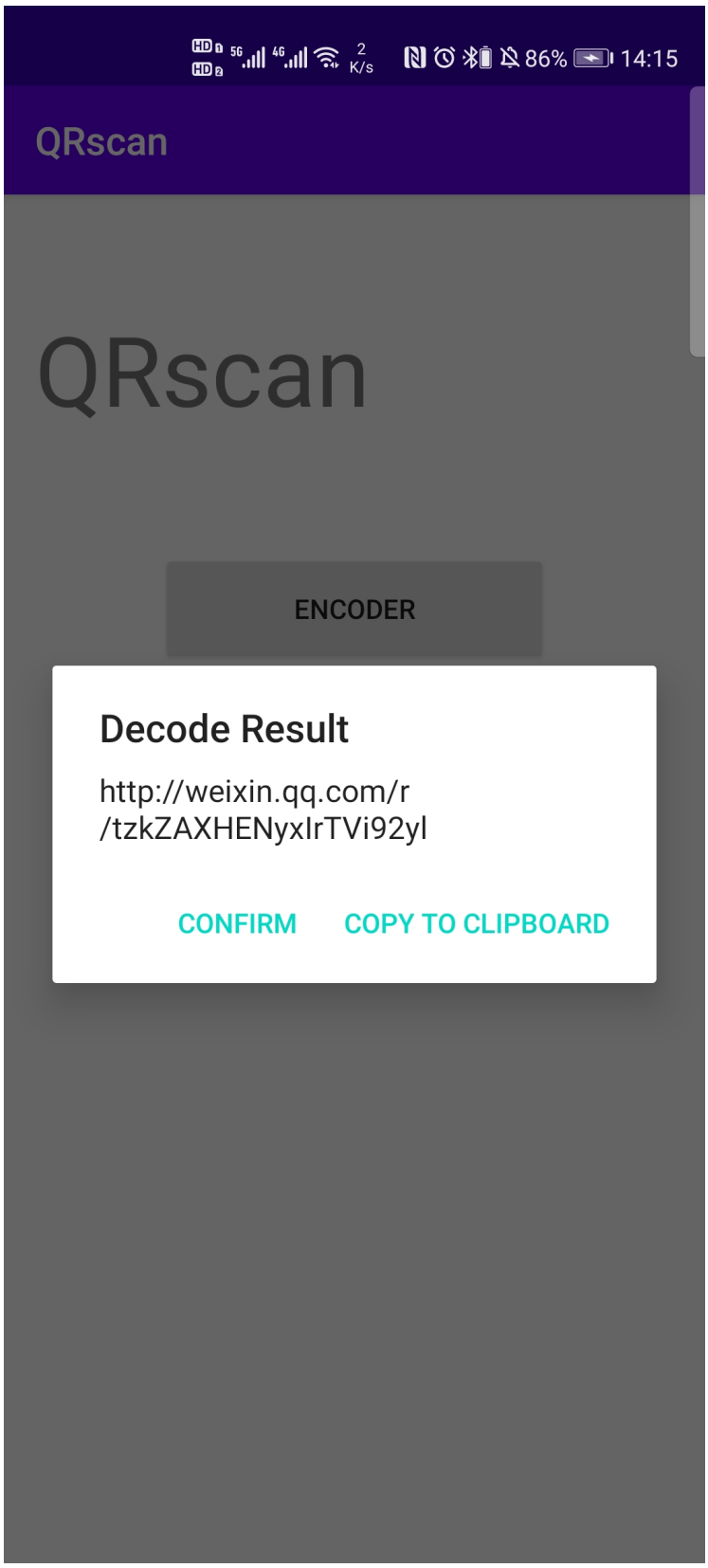

Figure 5: Decode Success Demonstration.

And if the decoder can’t recognize any QR code in the figure, the result is also shown below.

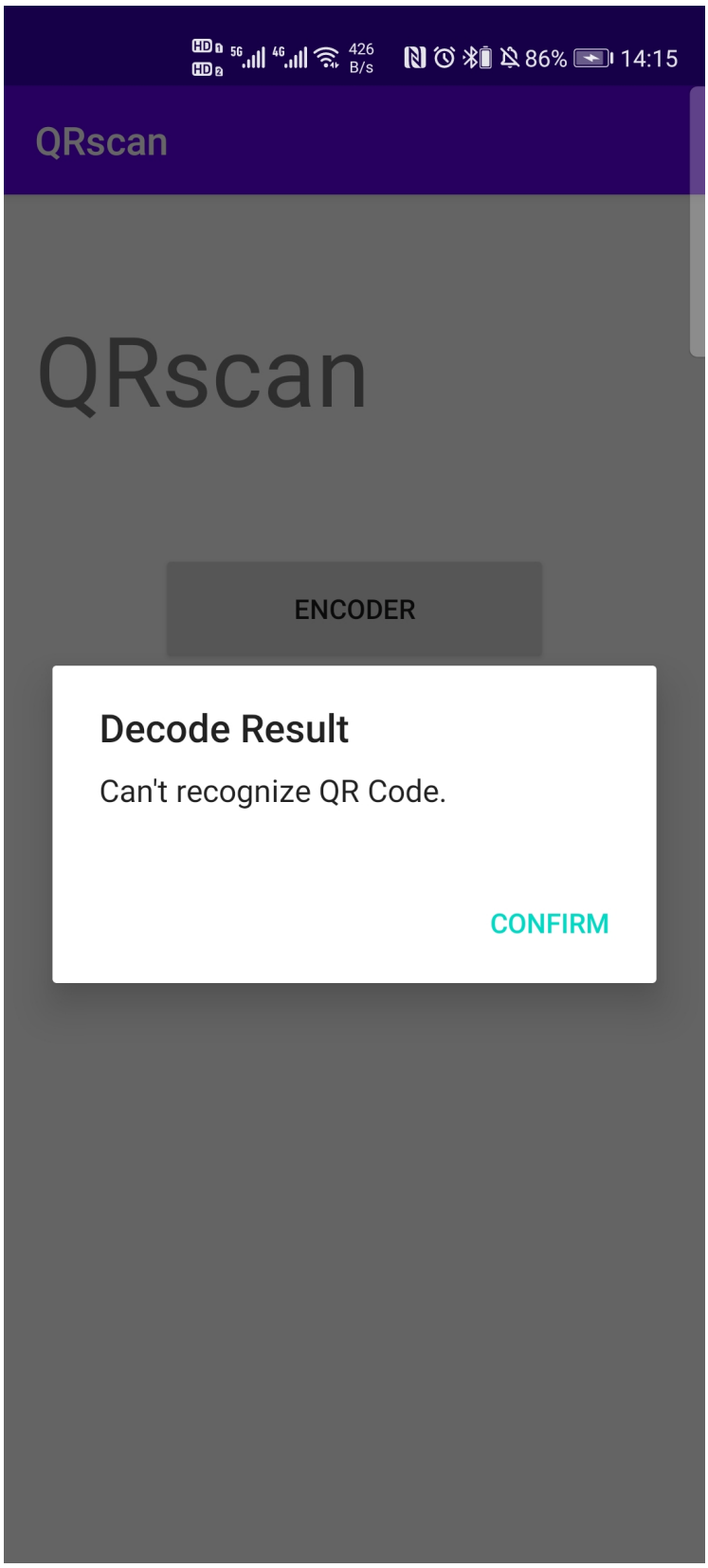

Figure 6: Decode Error Demonstration.