NPI P3: Android

Cristina Heredia, Alejandro Alcalde, *Universidad de Granada*

12 de febrero de 2016

Índice

1. Tutoriales de las aplicaciones Android

1.1. BrujulaCompass

Para realizar esta aplicación se ha tomado como base la brújula de la *ROM* MIUI. Se le ha añadido el reconocimiento de voz (*ASR*) y se modificó la la interfaz de la brújula para que mostrara hacia donde tiene que dirigirse el usuario en función del comando de voz. Veamos la primera pantalla:

1.1.1. Inicio de la aplicación

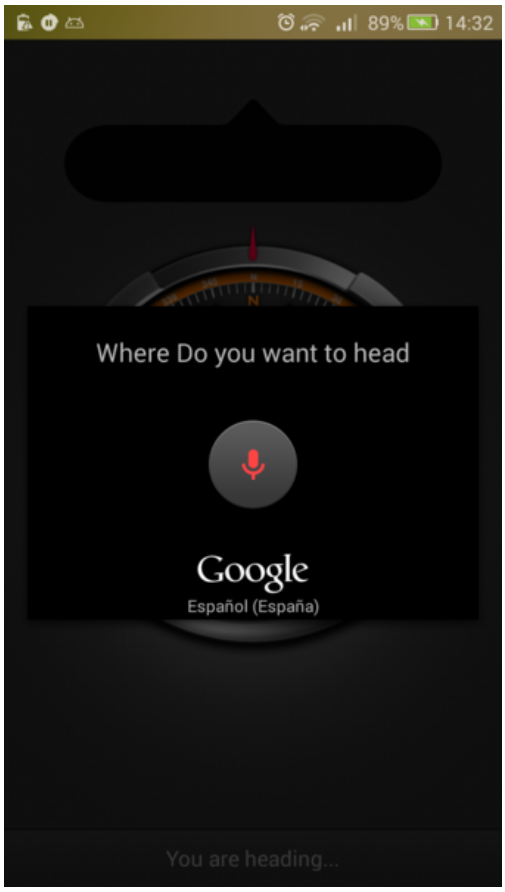

Figura 1: *Primera pantalla de la aplicación brújula*

Al mostrarse esta pantalla, el usuario debe proporcionar un comando de voz, por ejemplo *"Norte 10"*. Tras dar el comando, en la brujula se añadirá un marcador indicando dónde está el Norte + 10 grados. Además de esto, mediante una voz, se le irá indicando al usuario si debe girar a la derecha/izquierda o va en la dirección correcta:

Figura 2: *Indicaciones en la brujula*

Como vemos en la imagen, aparece un indicador rojo situado en el norte + 10 grados. Veamos otro ejemplo, Norte 45:

Figura 3: *Indicaciones en la brujula*

Para dar nuevas instrucciones de voz basta con tocar la brújula.
En la parte inferior de la pantalla, aparece el comando de voz reconocido.

1.1.2. Implementación

Se han necesitado de tres clases, La principal que implementa la actividad (`CompassActivity`), donde reside prácticamente toda la lógica de la aplicación. En ella se hace uso de los sensores magnético y el acelerómetro. La otra clase ha es una extensión de la clase `ImageVew` para crear nuestra propia vista, en este caso el compás y el indicador de la dirección indicada por el usuario.

Clase CompassActivity.java Esta clase es la principal y en la que se realiza toda la lógica, en ella se declarar y registran los sensores a usar (El magnético y

el acelerómetro). Ambos se obtienen en el método **onCreate** del siguiente modo:

```
private SensorManager mSensorManager;
private Sensor mMagneticSensor;
private Sensor mAccelerometer;

//...

mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mMagneticSensor =
 ↪   mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Para poder obtener actualizaciones frecuentes de los datos de los sensores es necesario declarar un **SensorEventListener** y registrarlo en el sistema, declararemos un único *listener* que será usado por los dos sensores:

```
private SensorEventListener mMagneticSensorEventListener = new
 ↪   SensorEventListener() {

    @Override
    public void onSensorChanged(SensorEvent event) {

        if (event.sensor == mMagneticSensor) {
            System.arraycopy(event.values, 0, mLastMagnetometer, 0,
 ↪   event.values.length);
            mLastMagnetometerSet = true;
        } else if (event.sensor == mAccelerometer) {
            System.arraycopy(event.values, 0, mLastAccelerometer, 0,
 ↪   event.values.length);
            mLastAccelerometerSet = true;
        }

        if (mLastAccelerometerSet && mLastMagnetometerSet) {
            SensorManager.getRotationMatrix(mR, null, mLastAccelerometer,
 ↪   mLastMagnetometer);
            SensorManager.getOrientation(mR, mOrientation);
            float azimuthInRadians = mOrientation[0];
            float azimuthInDegress = (float)
 ↪   (Math.toDegrees(azimuthInRadians) + 360) % 360;

            mTargetDirection = -azimuthInDegress;
        }
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }
};
```

La función `onSensorChanged` será llamada cada vez que se actualicen los datos de los sensores.

Una vez tenemos una referencia a los sensores y el *listener* creado, hay que registrarlos en el método `onResume` y des-registrarlos en el `onPause`:

```
@Override
protected void onResume() {
    super.onResume();

    if (mMagneticSensor != null) {
        mSensorManager.registerListener(mMagneticSensorEventListener,
↪   mMagneticSensor,
                SensorManager.SENSOR_DELAY_GAME);
    }
    if (mAccelerometer != null) {
        mSensorManager.registerListener(mMagneticSensorEventListener,
↪   mAccelerometer,
                SensorManager.SENSOR_DELAY_GAME);
    }
    // ...
}

@Override
protected void onPause() {
    super.onPause();

    if (mMagneticSensor != null) {
        mSensorManager.unregisterListener(mMagneticSensorEventListener);
    }
    if (mAccelerometer != null) {
        mSensorManager.unregisterListener(mMagneticSensorEventListener);
    }

    // ...
}
```

El reconocimiento de voz se inicializa en el siguiente método:

```
/**
 * Starts listening for any user input.
 * When it recognizes something, the <code>processAsrResult</code> method is
↪   invoked.
 * If there is any error, the <code>processAsrError</code> method is invoked.
 */
private void startListening() {
    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
↪   RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
↪   getString(R.string.asr_prompt));
    intent.putExtra(RecognizerIntent.EXTRA_MAX_RESULTS, 2);
```

```
    try {
        startActivityForResult(intent, REQUEST_RECOGNIZE);
    } catch (ActivityNotFoundException e) {
        //If no recognizer exists, download from Google Play
        showDownloadDialog();
    }
}
```

Esto intentará lanzar el reconocedor de voz, si el dispositivo no lo tiene instalado lanzará un diálogo pidiendo al usuario que lo instale:

```
private void showDownloadDialog() {
    AlertDialog.Builder builder =
            new AlertDialog.Builder(this);
    builder.setTitle(R.string.asr_download_title);
    builder.setMessage(R.string.asr_download_msg);
    builder.setPositiveButton(android.R.string.yes,
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog,
                                    int which) {
                    //Download, for example, Google Voice Search
                    Intent marketIntent =
                            new Intent(Intent.ACTION_VIEW);
                    marketIntent.setData(
                            Uri.parse("market://details?"
                                    + "id=com.google.android.voicesearch"));
                }
            });
    builder.setNegativeButton(android.R.string.no, null);
    builder.create().show();
}
```

Una vez que el usuario habla, se recoge el resultado en el `onActivityResult` y decidimos cómo interpretarlo, en este caso se parsea de forma bastante primitiva si el usuario dijo *este, norte, sur u oeste* junto al número de grados:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
 ↪  {
    if (requestCode == REQUEST_RECOGNIZE &&
            resultCode == Activity.RESULT_OK) {
        ArrayList<String> matches =
                data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);

        String[] tokens = matches.get(0).split(" ");

        if (tokens.length == 2) {
            mHeadedDirection = Float.parseFloat(tokens[1]);
```

```
            mLocationTextView.setText(String.format(getString(R.string.heading_text),
↪   matches.get(0)));

            switch (tokens[0].toLowerCase()) {
                case "este":
                    mHeadedDirection += 90;
                    break;
                case "sur":
                case "surf":
                    mHeadedDirection += 180;
                    break;
                case "oeste":
                    mHeadedDirection += 270;
                    break;
            }

            Toast.makeText(this, R.string.asr_ask_again,
                    Toast.LENGTH_LONG).show();

        } else {
            Toast.makeText(this, R.string.asr_error,
                    Toast.LENGTH_LONG).show();
        }
    }
}
```

El punto cardinal junto con los grados servirán para situar el indicador que muestre al usuario hacia dónde debe dirigirse.

Para lograr el efecto de giro de la brújula, se rota la imagen con cada actualización de los sensores:

```
protected Runnable mCompassViewUpdater = new Runnable() {
    @Override
    public void run() {
        if (mPointer != null && !mStopDrawing) {
            if (mDirection != mTargetDirection) {

                // calculate the short routine
                float to = mTargetDirection;
                if (to - mDirection > 180) {
                    to -= 360;
                } else if (to - mDirection < -180) {
                    to += 360;
                }

                // limit the max speed to MAX_ROTATE_DEGREE
                float distance = to - mDirection;
                float MAX_ROATE_DEGREE = 1.0f;
                if (Math.abs(distance) > MAX_ROATE_DEGREE) {
                    distance = distance > 0 ? MAX_ROATE_DEGREE : (-1.0f *
↪   MAX_ROATE_DEGREE);
                }
```

```
                // need to slow down if the distance is short
                mDirection = normalizeDegree(mDirection
                        + ((to - mDirection) *
↪   mInterpolator.getInterpolation(Math
                        .abs(distance) > MAX_ROATE_DEGREE ? 0.4f : 0.3f)));
                mPointer.updateDirection(mDirection);

                if (mHeadedDirection != -1) {
                    mUserHint.updateDirection(mDirection + mHeadedDirection);
                }
            }
            updateDirection();
            mHandlerCompass.postDelayed(mCompassViewUpdater, 20);
        }
    }
};
```

Como vemos, el `Runnable` se llama a sí mismo para mantenerse en ejecución `mHandlerCompass.postDelayed(mCompassViewUpdater, 20);`, de igual modo, habrá que escibir esta línea en el método `onResume`.

Los métodos `updateDirection` son métodos definidos en las clases que veremos ahora, que representan la brujula y el indicador.

Clase CompassView.java Los dos métodos más importantes de esta clase son:

```
@Override
protected void onDraw(Canvas canvas) {
    if (compass == null) {
        compass = getDrawable();
        compass.setBounds(0, 0, getWidth(), getHeight());
    }

    canvas.save();
    canvas.rotate(mDirection, getWidth() / 2, getHeight() / 2);
    compass.draw(canvas);
    canvas.restore();
}

public void updateDirection(float direction) {
    mDirection = direction;
    invalidate();
}
```

que se encargan de rotar la brújula cada vez que se llama al método `updateDirection` desde el `Runnable` visto anteriormente.

Permisos requeridos para el AndroidManifest

```
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

1.1.3. Referencias

- Comass de MIUI | github.com/MiCode
- Pro Android 5 | amazon.es
- Código de la aplicación | github.com/algui91/BrujulaCompass

1.2. GPSQR

En esta aplicación se lee un destino mediante códigos QR, tras esto, se puede iniciar la navegación con *Google Maps* (Usando la librería Android-GoogleDirectionLibrary). En la aplicación se muestran dos mapas. En el de abajo aparece el destino al que debemos llegar, además, se va dibujando un camino por el que el usuario va pasando. En el mapa de arriba se ve el mapa desde el punto de vista *StreetView*. Veamos la aplicación:

Figura 4: *GPSQR*

El *Floating Action Button* de abajo a la izquierda lanza el lector de QRs, que usa una simplificación de la librería *Zxing*. Cuando se escanea una localización, veremos lo siguiente:

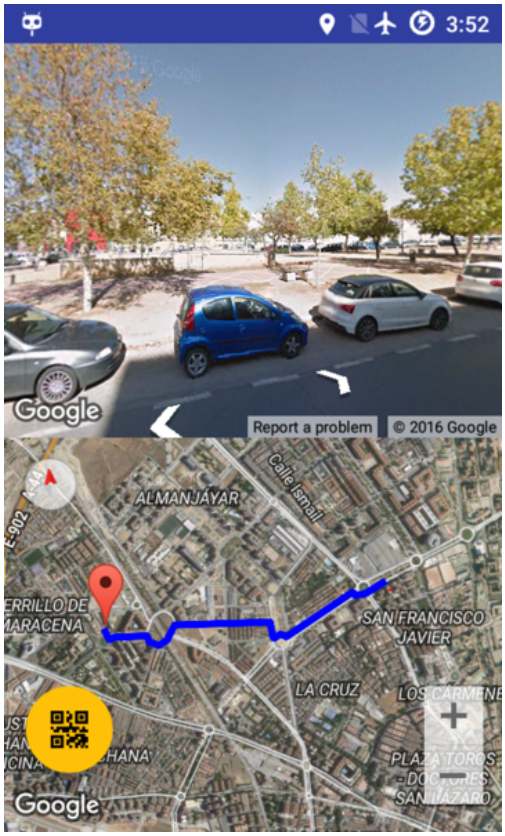

Figura 5: *Codigo QR leido con el destino*

Una vez leido el QR, solo resta pulsar el marcador rojo para iniciar la navegación con *Google Maps*. La ruta calculada por la *API* de *Google* es la azul, mientras que la ruta real tomada por el usuario aparecerá en rojo.

1.2.1. Implementación

Esta aplicación tiene dos clases, una para la primera y única pantalla y otra es un servicio que se ejecuta en segundo plano, encargado de obtener la localicación del usuario a un intervalo regular. Echemos primero un vistazo al Servicio.

Clase LocationUpdaterService.java Extiende de `Service` e implementa las siguientes interfaces:

```
public class LocationUpdaterService extends Service implements
        GoogleApiClient.ConnectionCallbacks,
```

```
        GoogleApiClient.OnConnectionFailedListener,
        LocationListener {
          // ....
        }
```

Con nombres bastantes descriptivos. Al iniciar el servicio, en su método `onCreate` se construye el cliente para la API de google del siguiente modo:

```
/**
 * Builds a GoogleApiClient. Uses the {@code #addApi} method to request the
 * LocationServices API.
 */
protected synchronized void buildGoogleApiClient() {
    if (BuildConfig.DEBUG) {
        Log.d(TAG, "Building GoogleApiClient");
    }
    mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .addApi(LocationServices.API)
            .build();
    createLocationRequest();
}
```

También se inicializa el `BroadcastManager` que usaremos para enviar las actualizaciones de la posición a la pantalla principal.

```
mBroadcaster = LocalBroadcastManager.getInstance(this);
```

El método `onCreate` se llama una única vez, al crear el servicio. Los comandos a ejecutar se colocan en el método `onStartCommand`, este metodo lo crearemos como *Sticky*, de modo que si el sistema finaliza el proceso del servicio, se volverá a ejecutar, volviendo así a obtener actualizaciones de localización:

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    super.onStartCommand(intent, flags, startId);
    if (BuildConfig.DEBUG) {
        Log.d(TAG, "OnStartCommand");
    }

    if (mGoogleApiClient.isConnected()) {
      LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
              mLocationRequest, this);
    }
```

```
    return START_STICKY;
}
```

Una vez hecho esto, se llamarán a las interfaces implementadas con los datos necesarios para obtener la localización actual, en este caso la interfaz que nos interesa es `onLocationChanged`:

```
@Override
public void onLocationChanged(Location location) {
    mCurrentLocation = location;
    mLastUpdateTime = DateFormat.getTimeInstance().format(new Date());
    sendResult(new LatLng(mCurrentLocation.getLatitude(),
 ↪   mCurrentLocation.getLongitude()));

    if (BuildConfig.DEBUG) {
        Log.d(TAG, mCurrentLocation.getLatitude() + ", " +
 ↪   mCurrentLocation.getLongitude());
    }
}
```

El método `sendResult` se usa para emitir un `Broadcast` al sistema y que la aplicación sea capaz de recibir el mensaje, incluso si no está en primer plano:

```
/**
 * This method send the current location to the activity who called the
 ↪   service, This way the
 * location can be used in the UI.
 *
 * @param message The location
 */
private void sendResult(LatLng message) {
    Intent intent = new Intent(COPA_RESULT);
    if (message != null)
        intent.putExtra(COPA_MESSAGE, message);
    mBroadcaster.sendBroadcast(intent);
}
```

Por último para que el Servicio funcione debemos registrarlo en el *Manifest* añadiendo la siguiente etiqueta:

```
<application>
  <!--//.....-->
  <service android:name=".LocationUpdaterService"/>
  <!--//.....-->
</application>
```

Clase MapsActivity.java Esta clase es la interfaz gráfica de la aplicación y donde se muestran los dos mapas, para poder trabajar con ellos se implementan las siguientes interfaces:

```
public class MapsActivity extends FragmentActivity implements
        OnMapReadyCallback,
        StreetViewPanorama.OnStreetViewPanoramaChangeListener,
        OnStreetViewPanoramaReadyCallback {

      private GoogleMap mMap;
      private StreetViewPanorama mStreetViewPanorama;

      // ...
}
```

Antes de poder trabajar con cualquiera de los mapas hemos de esperar a que estén inicializados, el momento de hacer esta inicialización es cuando el sistema llama a `OnMapReadyCallback, OnStreetViewPanoramaReadyCallback`:

```
/**
 * Manipulates the map once available.
 * This callback is triggered when the map is ready to be used.
 * This is where we can add markers or lines, add listeners or move the
 ↪  camera.
 * If Google Play services is not installed on the device, the user will be
 ↪  prompted to install
 * it inside the SupportMapFragment. This method will only be triggered once
 ↪  the user has
 * installed Google Play services and returned to the app.
 */
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
    UiSettings uiSettings = mMap.getUiSettings();
    uiSettings.setMapToolbarEnabled(true);
    uiSettings.setZoomControlsEnabled(true);
}

@Override
public void onStreetViewPanoramaReady(StreetViewPanorama panorama) {
    mStreetViewPanorama = panorama;
    mStreetViewPanorama.setOnStreetViewPanoramaChangeListener(this);
    mStreetViewPanorama.setStreetNamesEnabled(true);
}
```

Entre otras cosas, en el método `onCreate` inicializamos el `BroadcastReceiver` que nos permitirá recibir actualizaciones desde el servicio, e iniciamos el servicio:

```
@Override
protected void onCreate(final Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_maps);
//  ...
    mLocationReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            mPreviousLocation = mCurrentLocation;
            mCurrentLocation =
 ↪   intent.getParcelableExtra(LocationUpdaterService.COPA_MESSAGE);
            updateMap();
            mLocationsList.add(mCurrentLocation);

            if (BuildConfig.DEBUG) {
                Log.d(TAG, "LocationList size: " + mLocationsList.size());
            }
        }
    };

    mRequestLocationIntent = new Intent(this, LocationUpdaterService.class);
    startService(mRequestLocationIntent);
// ...
}
```

La función `updateMap` es la encargada de dibujar las líneas del camino seguido por el usuario.

Iniciar la navegación hasta destino Cuando se lanza el lector QR y se obtiene el destino, se crea una ruta a seguir, en esta ruta es posible hacerla siguiendo el camino mostrado en el mapa, o lanzando la navegación con *Google Maps*. Para lo último, es neceario pulsar sobre el marcador de destino y posteriormente pulsar el icono de *Google Maps*:

```
LatLng firstLocation = new LatLng(mCoord[0], mCoord[1]);
mMap.addMarker(new MarkerOptions().position(firstLocation).title("Dest"));
mMap.moveCamera(CameraUpdateFactory.newLatLng(firstLocation));
mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(firstLocation, 21.0f));

GoogleDirection.withServerKey(getString(R.string.google_maps_server_key))
        .from(mCurrentLocation)
        .to(new LatLng(mCoord[0], mCoord[1]))
        .transportMode(TransportMode.WALKING)
        .execute(new DirectionCallback() {
            @Override
            public void onDirectionSuccess(Direction direction) {
                if (direction.isOK()) {
                    Toast.makeText(getApplicationContext(), "DIRECTION KOK",
 ↪  Toast.LENGTH_LONG).show();
```

```
                    ArrayList<LatLng> directionPositionList =
↪   direction.getRouteList().get(0).getLegList().get(0).getDirectionPoint();
                    PolylineOptions polylineOptions =
↪   DirectionConverter.createPolyline(getApplicationContext(),
↪   directionPositionList, 5, Color.BLUE);
                    mMap.addPolyline(polylineOptions);
                } else {
                    Toast.makeText(getApplicationContext(), "NOT OK" +
↪   direction.getStatus(), Toast.LENGTH_LONG).show();
                }
            }

            @Override
            public void onDirectionFailure(Throwable t) {
                Toast.makeText(getApplicationContext(), "Failure",
↪   Toast.LENGTH_LONG).show();
            }
        });
```

Permisos requeridos para el AndroidManifest

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

1.2.2. Referencias y agradecimientos

- stackoverflow.com
- github.com/googlesamples/android-play-location
- github.com/akexorcist/Android-GoogleDirectionLibrary
- gist.github.com/blackcj
- github.com/googlemaps/android-samples
- Icono QR

1.3. Photo Gesture

Para realizar esta aplicación se ha usado una librería llamada PatterLock.

En esta aplicación se le pide al usuario que establezca un patrón de bloqueo, puede ser tan complejo como el patrón de bloqueo usado en Android. Una vez establecido, cuando se introduzca correctamente la aplicación tomará una foto a los 3 segundos. A continuación mostramos la pantalla principal de la aplicación.

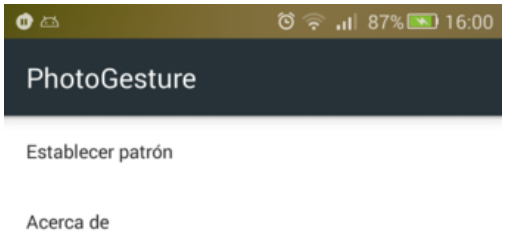

Figura 6: *Pantalla principal de photoGesture*

Al pulsar *"Establecer patrón"* veremos lo siguiente:

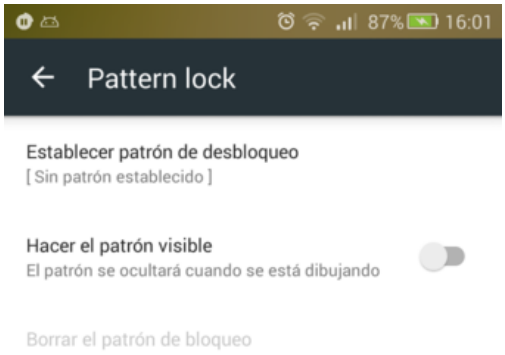

Figura 7: *Establecer patrón*

Es posible hacer que el patrón no sea visible cuando lo introducimos, para añadir una capa extra de seguridad.

Cuando pulsemos *Establecer patrón* se nos pedirá que lo dibujemos dos veces, para confirmarlo:

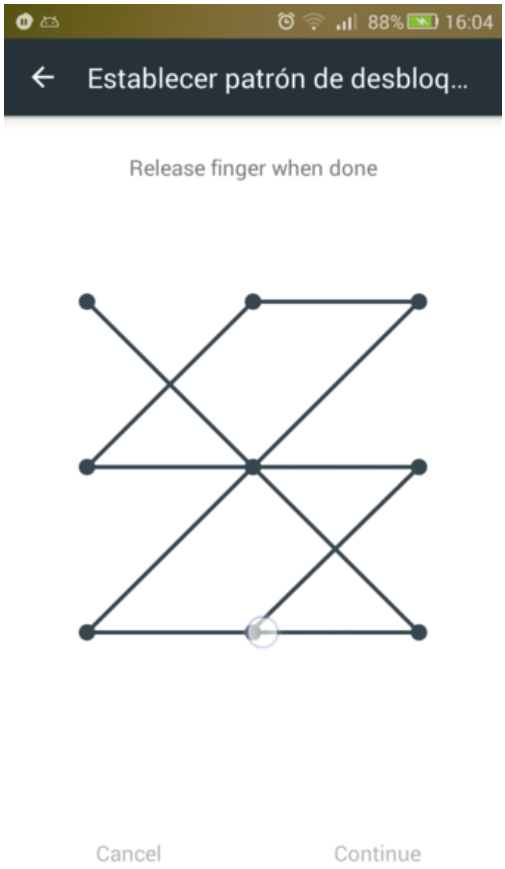

Figura 8: *Dibujando el patrón*

Hecho esto, cuando volvamos a la pantalla principal, en lugar de "Establecer patrón" aparecerá "Echar foto". Si pulsamos sobre ese botón, se nos pide el patrón establecido. Si se introduce bien, aparecerá la cámara con una cuenta atrás, al llegar a 0 se echará una foto:

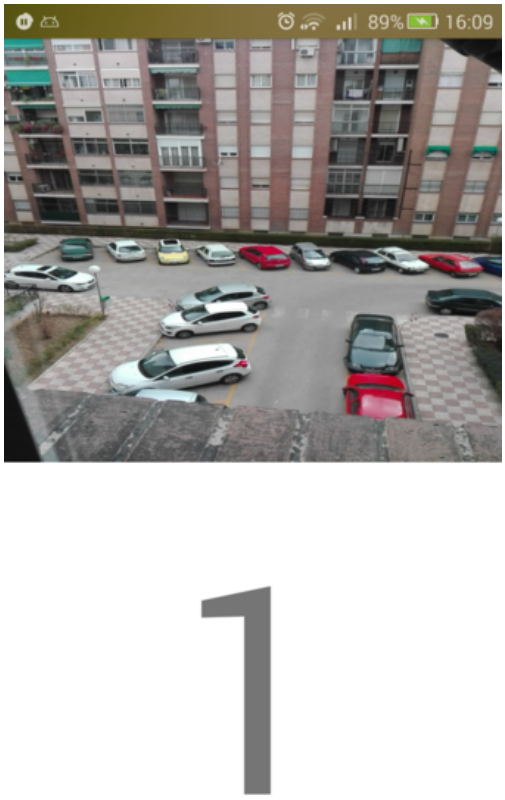

Figura 9: *Cuenta atrás para echar la foto*

La foto se guardará en la galería.

1.3.1. Implementación

Se ha reutilizado el ejemplo que el autor de la librería creo para demostar su uso, y se ha modificado para ajustarlo a los requisitos de la práctica. Para ello, cuando se introduce correctamente el parón, se incia la actividad de hacer una foto:

```
@Override
protected boolean isPatternCorrect(List<PatternView.Cell> pattern) {

    boolean isCorrect = PatternLockUtils.isPatternCorrect(pattern, this);
    if (isCorrect) {
```

```
            startActivity(new Intent(this, MakePhotoActivity.class));
        }

        return isCorrect;
    }
```

En la actividad `MakePhotoActivity` se muestra la pantalla dividida en dos, arriba aparece la cámara y abajo una cuenta atrás de 3 segundos, cuando esta cuenta atrás llegue a cero, se hará la foto.

En el método `onCreate` se inicializa la cámara se crean dos *callbacks*, uno que se ejecuta al hacer la foto y otro para reproducir un sonido al echar la foto:

```
mPicture = new Camera.PictureCallback() {

    @Override
    public void onPictureTaken(byte[] data, Camera camera) {

        File pictureFile = getOutputMediaFile(MEDIA_TYPE_IMAGE);
        if (pictureFile == null) {
            Log.e(TAG, "Error creating media file, check storage permissions:
 ↪  ");
            return;
        }

        try {
            FileOutputStream fos = new FileOutputStream(pictureFile);
            fos.write(data);
            fos.close();

            new MediaScannerWrapper(getApplicationContext(),
 ↪  pictureFile.getPath(), "image/jpeg").scan();
        } catch (FileNotFoundException e) {
            Log.d(TAG, "File not found: " + e.getMessage());
        } catch (IOException e) {
            Log.d(TAG, "Error accessing file: " + e.getMessage());
        }

    }
};

mShutterSound = MediaPlayer.create(this, R.raw.shut);

mShutter = new Camera.ShutterCallback() {
    @Override
    public void onShutter() {
        mShutterSound.start();
        finish();
    }
};
```

Al llegar la cuenta atrás a cero, se llama al método `takePicture` de la cámara

y se le pasan los *callbacks anteriores*:

```
mCamera.takePicture(mShutter, null, mPicture);
```

Mostrando una vista previa de la cámara Para crear la parte superior de la pantalla, en la que se muestra una vista previa de la cámara, hay que crear una clase extendiendo de `SurfaceView` que hemos llamado `CameraPreview`, esta clase implementa la interfaz `SurfaceHolder.Callback`:

```
/** A basic Camera preview class */
public class CameraPreview extends SurfaceView implements
↪  SurfaceHolder.Callback {

    private SurfaceHolder mHolder;
    private Camera mCamera;

    public CameraPreview(Context context, Camera camera) {
        super(context);
        mCamera = camera;

        mCamera.setDisplayOrientation(90);
        // Install a SurfaceHolder.Callback so we get notified when the
        // underlying surface is created and destroyed.
        mHolder = getHolder();
        mHolder.addCallback(this);
        // deprecated setting, but required on Android versions prior to 3.0
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    public void surfaceCreated(SurfaceHolder holder) {
        // The Surface has been created, now tell the camera where to draw
↪   the preview.
        try {
            mCamera.setPreviewDisplay(holder);
            mCamera.startPreview();
        } catch (IOException e) {
            Log.d("TAG", "Error setting camera preview: " + e.getMessage());
        }
    }

    public void surfaceDestroyed(SurfaceHolder holder) {
        // empty. Take care of releasing the Camera preview in your activity.
        mCamera.release();
    }

    public void surfaceChanged(SurfaceHolder holder, int format, int w, int
↪   h) {
        // If your preview can change or rotate, take care of those events
↪   here.
        // Make sure to stop the preview before resizing or reformatting it.
```

```
        if (mHolder.getSurface() == null) {
            // preview surface does not exist
            return;
        }

        // stop preview before making changes
        try {
            mCamera.stopPreview();
        } catch (Exception e) {
            // ignore: tried to stop a non-existent preview
        }

        // set preview size and make any resize, rotate or
        // reformatting changes here

        // start preview with new settings
        try {
            mCamera.setPreviewDisplay(mHolder);
            mCamera.startPreview();

        } catch (Exception e) {
            Log.d("TAG", "Error starting camera preview: " + e.getMessage());
        }
    }
}
```

Permisos requeridos para el AndroidManifest

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

1.3.2. Referencias

- Página oficial de Android | developer.android.com/guide/topics/media/camera
- Librería PatternLock | github.com/DreaminginCodeZH/PatternLock

1.4. Movement Sound

En esta aplicación se usa el acelerómetro y el giroscopio, para mostrar sus valores por pantalla. El giroscopio es capaz de detectar una rotación del dispositivo, al hacerlo, reproduce un sonido. Por contra, el acelerómetro detecta una sacudida del dispositivo y reproduce un sonido distinto. Debido a que no en todos los dispositivos se obtienen valores precisos para los sensores, solo se mostrarán aquellos que tengan sentido.

Figura 10: *Aplicación Movement Sound*

1.4.1. Implementación

Para instanciar ambos sensores en el `onCreate`:

```
// Get the sensors to use
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensorAcc = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
mSensorGyr = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

Cuando el tengamos datos nuevos de los sensores, los actualizaremos en el `onSensorChanged`:

```
@Override
public void onSensorChanged(SensorEvent event) {
```

```
    if (event.accuracy == SensorManager.SENSOR_STATUS_UNRELIABLE) {

        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            mAccx.setText(R.string.act_main_no_acuracy);
            mAccy.setText(R.string.act_main_no_acuracy);
            mAccz.setText(R.string.act_main_no_acuracy);
        } else if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {
            mGyrox.setText(R.string.act_main_no_acuracy);
            mGyroy.setText(R.string.act_main_no_acuracy);
            mGyroz.setText(R.string.act_main_no_acuracy);
        }
        return;
    }

    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        mAccx.setText("x = " + Float.toString(event.values[0]));
        mAccy.setText("y = " + Float.toString(event.values[1]));
        mAccz.setText("z = " + Float.toString(event.values[2]));
        detectShake(event);
    } else if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {
        mGyrox.setText("x = " + Float.toString(event.values[0]));
        mGyroy.setText("y = " + Float.toString(event.values[1]));
        mGyroz.setText("z = " + Float.toString(event.values[2]));
        detectRotation(event);
    }

}
```

Si el acelerómetro está activo, se intentará detectar una sacudida del dispositivo y se reproducirá un sonido, en el caso del giroscopio, se pretende detectar una rotación del dispositivo. Estos patrones se reconocen con las siguientes funciones:

```
// References:
//  - http://jasonmcreynolds.com/?p=388
//  - http://code.tutsplus.com/tutorials/using-the-accelerometer-on-android--
 ↪  mobile-22125

/**
 * Detect a shake based on the ACCELEROMETER sensor
 *
 * @param event
 */
private void detectShake(SensorEvent event) {
    long now = System.currentTimeMillis();

    if ((now - mShakeTime) > SHAKE_WAIT_TIME_MS) {
        mShakeTime = now;

        float gX = event.values[0] / SensorManager.GRAVITY_EARTH;
        float gY = event.values[1] / SensorManager.GRAVITY_EARTH;
        float gZ = event.values[2] / SensorManager.GRAVITY_EARTH;
```

```
        // gForce will be close to 1 when there is no movement
        double gForce = Math.sqrt(gX * gX + gY * gY + gZ * gZ);

        // Change background color if gForce exceeds threshold;
        // otherwise, reset the color
        if (gForce > SHAKE_THRESHOLD) {
            soundAcc.start();
        }
    }
}

/**
 * Detect a rotation in on the GYROSCOPE sensor
 *
 * @param event
 */
private void detectRotation(SensorEvent event) {
    long now = System.currentTimeMillis();

    if ((now - mRotationTime) > ROTATION_WAIT_TIME_MS) {
        mRotationTime = now;

        // Change background color if rate of rotation around any
        // axis and in any direction exceeds threshold;
        // otherwise, reset the color
        if (Math.abs(event.values[0]) > ROTATION_THRESHOLD ||
                Math.abs(event.values[1]) > ROTATION_THRESHOLD ||
                Math.abs(event.values[2]) > ROTATION_THRESHOLD) {
            soundGyro.start();
        }
    }
}
```

Permisos requeridos para el AndroidManifest

```
<uses-permission android:name="android.hardware.sensor.gyroscope"/>
<uses-permission android:name="android.hardware.sensor.accelerometer"/>
```

1.4.2. Referencias

- AndroidWearMotionSensors | github.com/drejkim