به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

سیستم‌های نهفته و بی‌درنگ

گزارش طراحی و پیاده‌سازی اسکنر سطح اندروید

سجاد گندم مالمیری

علیرضا توکلی

عرفان رزاقی

کاوه معصومی

1401/02/25

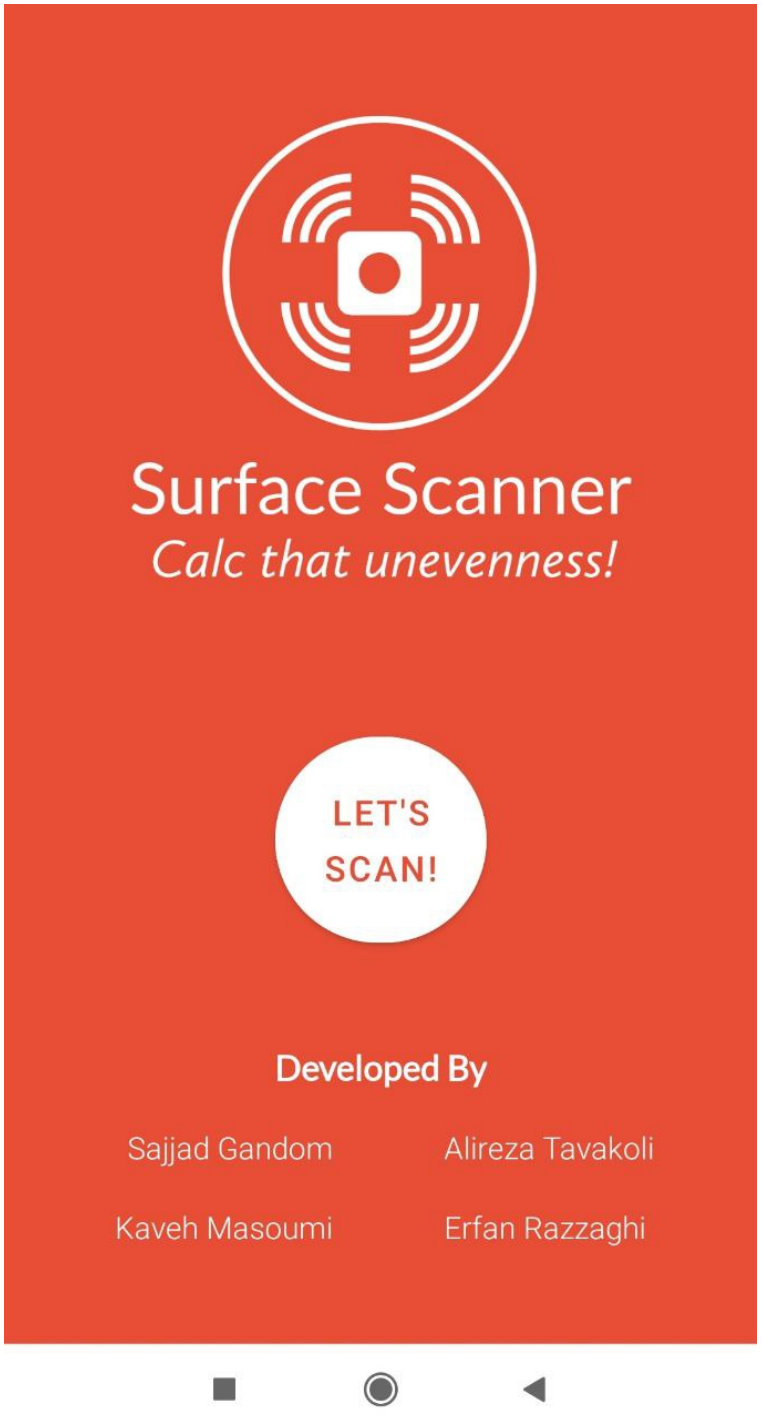
فهرست

بخش 1 – توضیح اجزای سیستم

این برنامه ی اندروید از دو Activity اصلی تشکیل شده است:

- MainActivity:

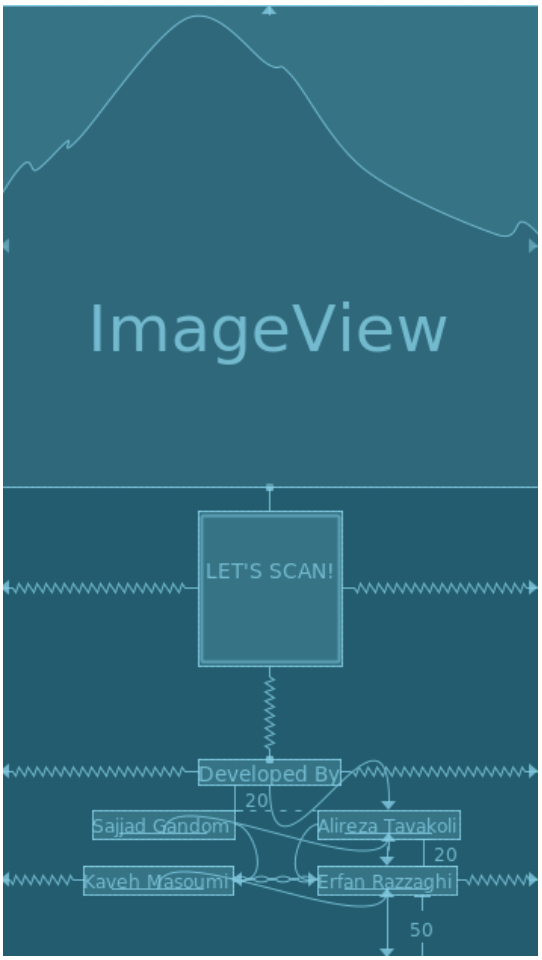
این صفحه صرفا شامل یک دکمه‌ی "Let's scan" بوده که شخص را به صفحه‌ی اصلی اسکن کردن هدایت می‌کند. علاوه بر آن نام توسعه دهندگان این پروژه در آن آمده است.



در پیاده‌سازی این صفحه صرفا یک listener برای دکمه تعریف شده که کاربر را پس از ویبره‌ی گوشی به صفحه‌ی اصلی هدایت کند:

```java
public class MainActivity extends AppCompatActivity {

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

        Button letsScanBtn = findViewById(R.id.letsScanBtn);
        letsScanBtn.setOnClickListener(v -> {
            vibrate();
            Intent intent = new Intent(getBaseContext(), ScannerActivity.class);
            startActivity(intent);
        });

    }

    private void vibrate() {
        Vibrator v = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            v.vibrate(VibrationEffect.createOneShot( milliseconds: 100, VibrationEffect.DEFAULT_AMPLITUDE));
        } else {
            v.vibrate( milliseconds: 50);
        }
    }
}
```

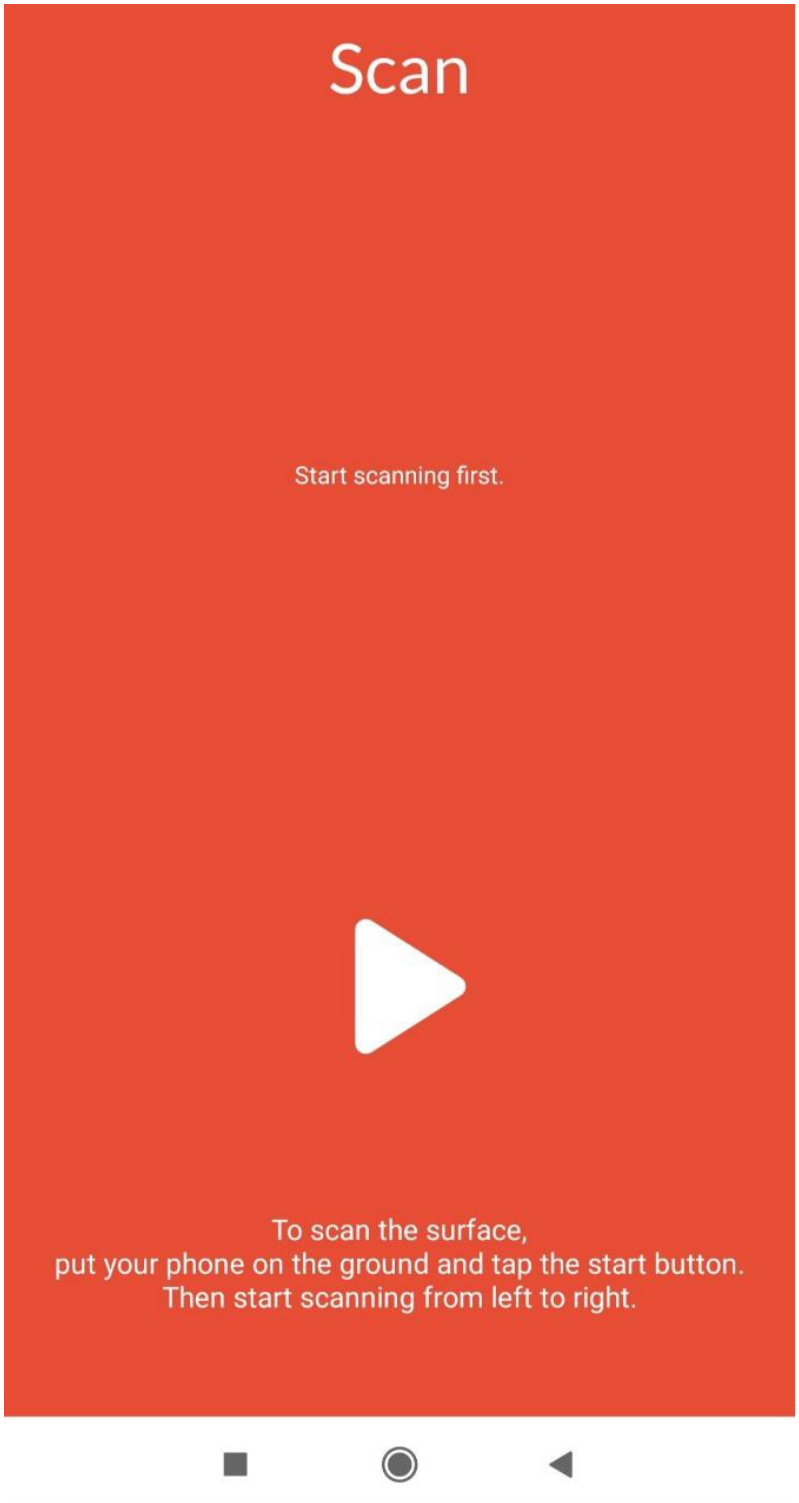
همچنین رابط کاربری آن از این اجزا تشکیل شده است:



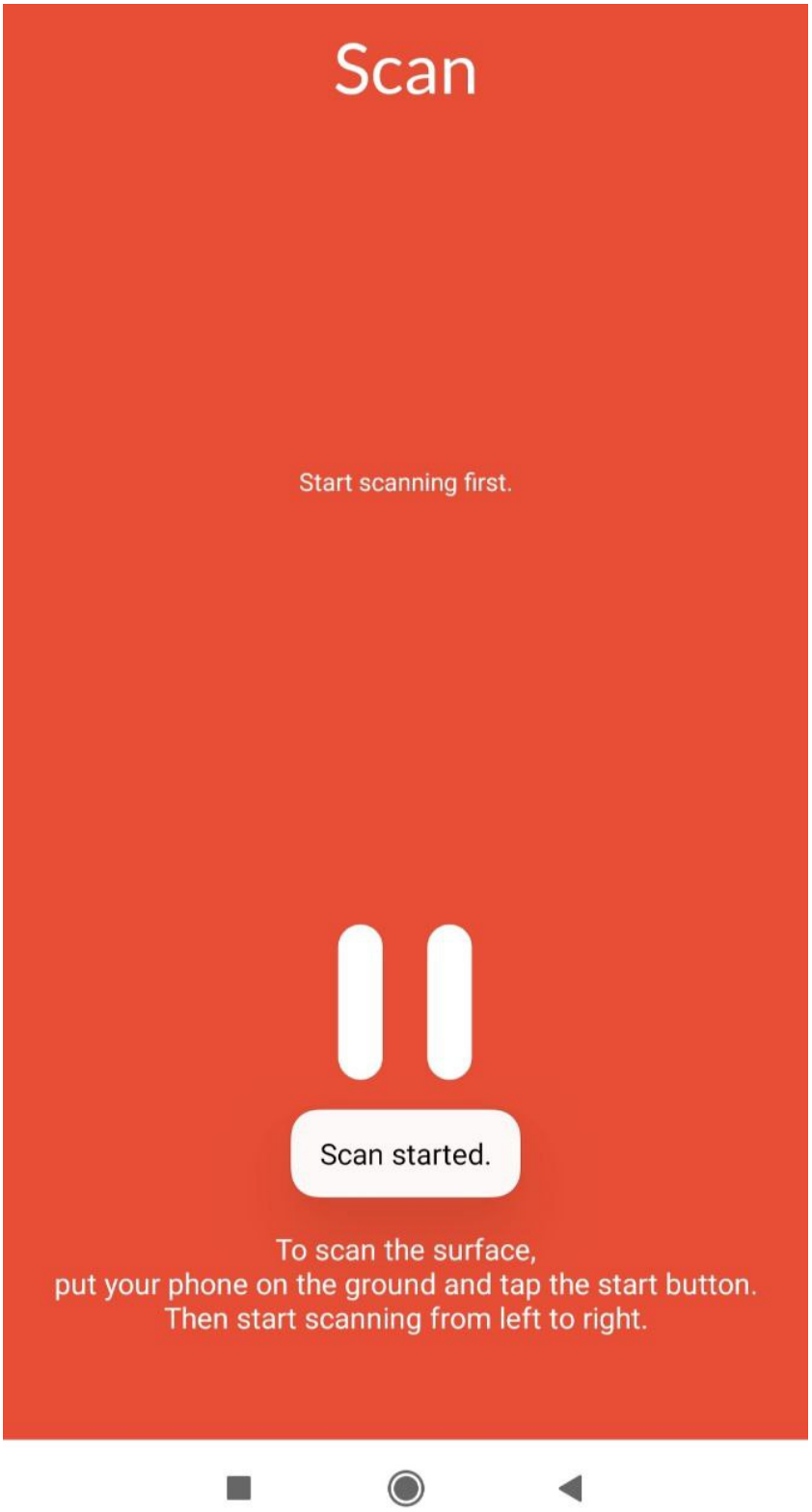
- ScannerActivity:

این صفحه برای اسکن کردن که هدف اصلی پروژه است استفاده می‌شود. روال استفاده از این صفحه بدین صورت است:

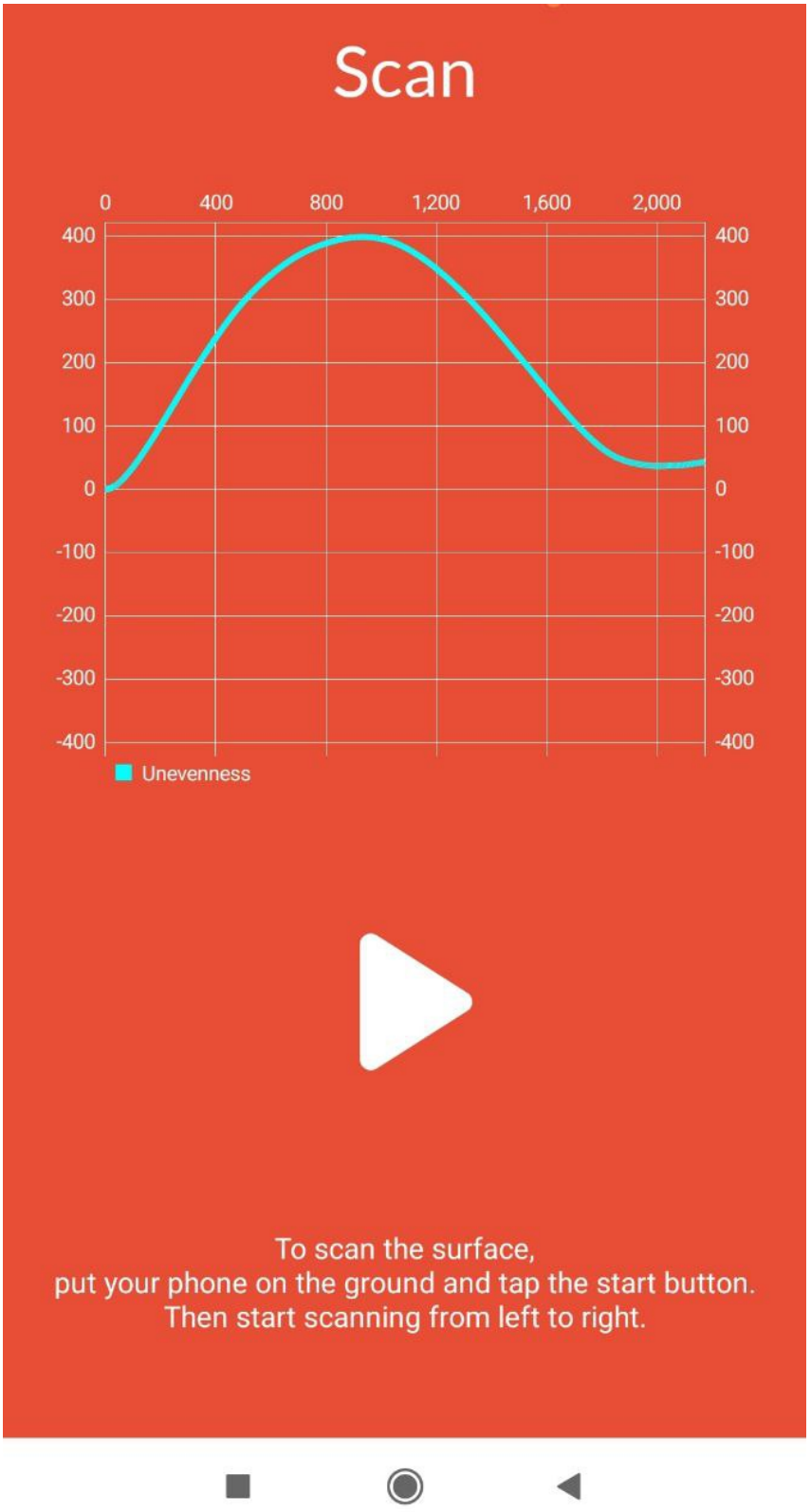
ابتدا کاربر با این صفحه مواجه می‌شود:



سپس برای شروع اسکن کردن، همانطور که در راهنمای پایین گفته شده است، باید گوشی را بر روی زمین قرار داده و دکمه‌ی شروع را ضربه بزند که در این صورت کاربر این تصویر را خواهد دید:



پس از اسکن درست سطح و زدن دکمه‌ی خاتمه نیز کاربر نتیجه‌ی نهایی را در بالای همین صفحه به صورت یک نمودار می‌بیند:



طرز کار این صفحه بدین صورت است که ابتدا در onCreate صفحه یک سری موارد انجام می‌شوند مانند نسخه‌گیری از شی اصلی SurfaceScanner (که منطق محاسبه و تشخیص سطح در آن قرار دارد)، مقداردهی اولیه و آماده‌سازی فرمت گراف نتیجه و Listener برای دکمه‌ی شروع و پایان اسکن در آن قرار می‌گیرد:

```
private boolean isScanning = false;
private SurfaceScanner surfaceScanner;
float chartYaxisRange = 30f;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_scanner);
    this.getSupportActionBar().hide();

    surfaceScanner = new SurfaceScanner( activity: this);

    setupChart();

    PlayPauseView playPauseView = findViewById(R.id.triggerScanBtn);
    playPauseView.fadeIn();
    playPauseView.setOnClickListener(v -> {
        vibrate();
        if (!isScanning) {
            playPauseView.setState(PlayPauseView.STATE_PLAY);
            startScanning();
        }
        else {
            playPauseView.setState(PlayPauseView.STATE_PAUSE);
            stopScanning();
        }
        isScanning = !isScanning;
    });
}
```

همچنین آماده‌سازی اولیه‌ی گراف به این صورت است و شامل مواردی مانند رنگ خطوط، بازه‌ی محورها و موارد مربوط به Place Holder Text آن میشود که در زیر قابل مشاهده هستند:

```
private void setupChart() {
    LineChart chart = findViewById(R.id.resultGraph);
    chart.getDescription().setEnabled(false);

    chart.setNoDataText("Start scanning first.");
    chart.setGridBackgroundColor(Color.WHITE);
    chart.setNoDataTextColor(Color.WHITE);
    chart.setBorderColor(Color.WHITE);
    chart.getXAxis().setTextColor(Color.WHITE);
    chart.getXAxis().setGridColor(Color.WHITE);
    chart.getXAxis().setAxisLineColor(Color.WHITE);

    chart.getAxisLeft().setTextColor(Color.WHITE);
    chart.getAxisLeft().setGridColor(Color.WHITE);
    chart.getAxisLeft().setAxisLineColor(Color.WHITE);
    chart.getAxisLeft().setZeroLineColor(Color.WHITE);
    chart.getAxisRight().setTextColor(Color.WHITE);
    chart.getAxisRight().setGridColor(Color.WHITE);
    chart.getAxisRight().setAxisLineColor(Color.WHITE);
    chart.getAxisRight().setZeroLineColor(Color.WHITE);

    chart.getLegend().setTextColor(Color.WHITE);

    chart.setDrawingCacheBackgroundColor(Color.WHITE);

    chart.getAxisLeft().setAxisMinimum(-chartYaxisRange);
    chart.getAxisLeft().setAxisMaximum(chartYaxisRange);
    chart.getAxisRight().setAxisMinimum(-chartYaxisRange);
    chart.getAxisRight().setAxisMaximum(chartYaxisRange);
}
```

لاجیک و منطق اصلی برنامه در شیء Surface Scanner قرار دارد و ما در هنگام شروع اسکن، متد startScan آن را صدا می‌زنیم و در هنگام پایان آن، با فراخوانی متد stopScan نتیجه را دریافت می‌کنیم و سپس آن را رسم می‌کنیم:

```
private void startScanning() {
    Toast.makeText(getBaseContext(), "Scan started.", Toast.LENGTH_SHORT).show();
    surfaceScanner.startScan();
}

private void stopScanning() {
    Toast.makeText(getBaseContext(), "Scan stopped.", Toast.LENGTH_SHORT).show();
    ArrayList<Pair<Double, Double>> positions = surfaceScanner.stopScan();
    displayGraph(positions);
}
```

در Constructor شیء Surface Scanner مواردی مانند ایجاد Listener برای دو سنسور Gyroscope و Accelerometer صورت می‌گیرد:

```
public SurfaceScanner(ScannerActivity activity) {

    SensorManager sensorManager = (SensorManager) activity.getSystemService(Context.SENSOR_SERVICE);
    Sensor gyroscopeSensor = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);

    SensorEventListener gyroscopeSensorListener = new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent event) {
            gyrLastValues.update(event.values[0], event.values[1], event.values[2]);
        }

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

        }
    };
    sensorManager.registerListener(gyroscopeSensorListener,
            gyroscopeSensor, SensorManager.SENSOR_DELAY_NORMAL);


    Sensor accelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    SensorEventListener accelerometerSensorListener = new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent event) {
            accLastValues.update(event.values[0], event.values[1], event.values[2]);
        }

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

        }
    };
    sensorManager.registerListener(accelerometerSensorListener,
            accelerometerSensor, SensorManager.SENSOR_DELAY_NORMAL);
}
```

در متد startScan آن مقدار دهی اولیه‌ی دوباره به پارامترهای شیء صورت می‌گیرد تا امکان چندین بار اسکن کردن وجود داشته باشد. همچنین یک TimerTask جدید تعریف می‌شود که هر sampleRateMs میلی‌ثانیه (که برابر 20 میلی‌ثانیه است) عمل نمونه‌گیری از سنسورها و محاسبه‌ی position گوشی را با فراخوانی calculatePosition انجام می‌دهد:

```
public void startScan() {
    this.angle = 0.0;
    this.velocity = 0.0;
    this.accLastValues.clear();
    this.gyrLastValues.clear();
    this.positions.clear();
    this.positions.add(new Pair<>(0.0, 0.0));

    Timer timer = new Timer();

    this.timerTask = new TimerTask() {
        @Override
        public void run() {
            calculatePosition();
        }
    };
    timer.schedule(this.timerTask,  delay: 0, this.sampleRateMs);
}
```

با توجه به این که ما در هر لحظه سرعت زاویه‌ای را داریم، می‌توانیم زاویه‌ی دستگاه را با افق به راحتی به دست آوریم که همان‌طور که می‌بینید، این سرعت زاویه‌ای در زمان نمونه‌برداری ضرب شده و با زاویه‌ی قبلی جمع می‌شود. هم‌چنین با داشتن شتاب در راستای x دستگاه و داشتن سرعت قبلی، می‌توان به راحتی سرعت جدید را به دست آورد؛ که این سرعت با توجه به زمان نمونه برداری و مکان قبلی، مکان جدید دستگاه را به ما می‌دهد. در انتها نیز مکان جدید را ذخیره کرده تا پس از اتمام نمودار مربوطه را بکشیم.

```
public void calculatePosition() {
    double sampleRateS = (double) this.sampleRateMs / 1000;

    double theta = (-this.gyrLastValues.getY() * sampleRateS) + this.angle;
    double newVelocity = this.accLastValues.getX() * sampleRateS + velocity;

    Pair<Double, Double> lastPosition = this.positions.get(this.positions.size() - 1);
    Pair<Double, Double> newPositions = new Pair<>(
        lastPosition.first + newVelocity * Math.cos(theta),
        lastPosition.second + newVelocity * Math.sin(theta)
    );
    this.positions.add(newPositions);
    this.velocity = newVelocity;
    this.angle = theta;
}
```

سپس وقتی کاربر دکمه‌ی توقف را می‌زند، تابع stopScan آن صدا زده می‌شود که در آن صرفا TimerTask مذکور متوقف شده و نتیجه‌ی position های محاسبه‌شده‌ی گوشی برگردانده می‌شود:

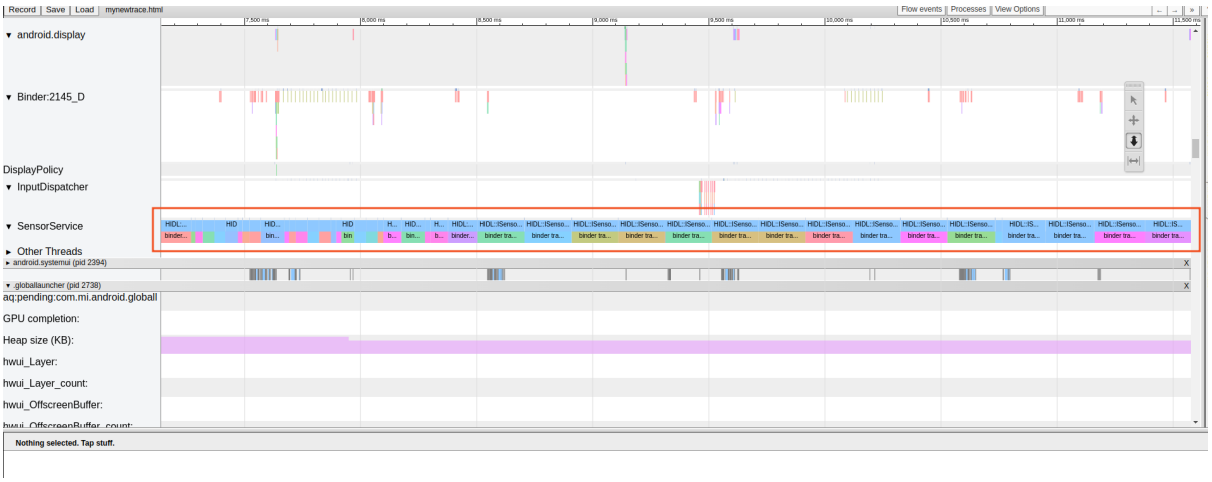
```
public ArrayList<Pair<Double, Double>> stopScan() {
    this.timerTask.cancel();
    return this.positions;
}
```

سپس این نتیجه‌ی برگردانده‌شده در ScannerActivity با فراخوانی تابع displayGraph به این طریق بر روی گراف نمایش داده می‌شود:

```
private void displayGraph(ArrayList<Pair<Double, Double>> positions) {
    LineChart chart = findViewById(R.id.resultGraph);
    float extendedChartYaxisRange = chartYaxisRange;

    List<Entry> entries = new ArrayList<~>();
    for (Pair<Double, Double> position : positions) {
        entries.add(new Entry(position.first.floatValue(), position.second.floatValue()));

        if (Math.abs(position.second) >= extendedChartYaxisRange)
            extendedChartYaxisRange = (float) (1.2 * Math.abs(position.second));
    }
    chart.getAxisLeft().setAxisMinimum(-extendedChartYaxisRange);
    chart.getAxisLeft().setAxisMaximum(extendedChartYaxisRange);
    chart.getAxisRight().setAxisMinimum(-extendedChartYaxisRange);
    chart.getAxisRight().setAxisMaximum(extendedChartYaxisRange);

    Collections.sort(entries, new EntryXComparator());

    LineDataSet dataSet = new LineDataSet(entries, label: "Unevenness");
    dataSet.setColor(Color.CYAN);
    dataSet.setDrawValues(false);
    dataSet.setDrawCircles(false);
    dataSet.setLineWidth(3.0f);

    LineData lineData = new LineData(dataSet);
    chart.setData(lineData);
    chart.animateX( durationMillis: 500);
}
```

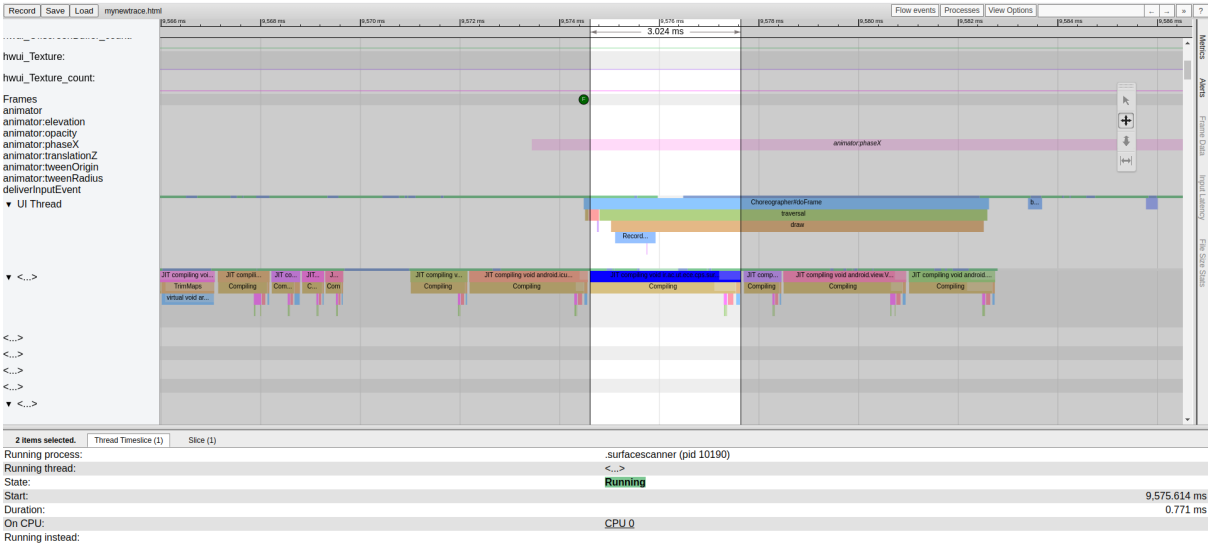
بخش 2 – پاسخ سوالات

- از وقتی که درخواست خواندن داده به سنسور داده شده تا گرفتن داده چه اتفاقاتی در سطح سیستم‌عامل افتاده است؟ توضیح خود را با systrace توضیح داده و توجیه کنید.



- چه مدت زمانی طول می‌کشد تا تغییرات اسکن شده از سطح بر اساس مقداری که از سنسور خوانده شده است، روی صفحه‌ی نمایش ظاهر شود؟ (تصویر واضح از systrace قرار داده شود.)

همانطور که در تصویر زیر دیده میشود، زمان اجرای تابع displayGraph که تغییرات را بر روی نمودار رسم میکند برابر 3.042ms است:



- بهترین دوره تناوب برای خواندن مقادیر سنسور شتاب‌سنج و ژیروسکوپ چه مقدار است؟ با استدلال توجیه شود.

این دوره ی تناوب باید بزرگتر از sample rate خود سنسور باشد تا بتواند محاسبات را انجام دهد. هم‌چنین بعد از این شرط با توجه به برنامه‌ی نوشته شده، فرض می‌گیریم که در این دوره‌ی تناوب، شتاب و سرعت زاویه‌ای دریافت شده ثابت است؛ پس هر چه دوره تناوب کم‌تر باشد، محاسبات ما محاسبات درست‌تری خواهد بود. با توجه به این که در دستگاه‌های مختلف این sample rate متفاوت است، نمی‌توان عدد کلی‌ای برای همه‌ی دستگاه‌ها استفاده کرد.

- اگر از Android NDK به جای Android SDK استفاده می‌شد، اسکنر شما چه مزایا و معایبی خواهد داشت؟

اندروید NDK مجموعه ابزاریست که به توسعه‌دهندگان این توانایی را میدهد که از کدهای از پیش‌نوشته‌شده ی خود به زبان C++/C دوباره‌استفاده (reuse) کرده و آن را در برنامه های اندروید خود به کمک Java Native Interface ترکیب نمایند. از آنجایی که این برنامه‌ها به طور مستقیم بر روی پردازنده اجرا می‌شوند (و نه روی مفسر Dalvik Virtual Machine در حالت عادی استفاده از SDK) باعث میشود که کارایی و سرعت برنامه‌ها بیشتر باشد. اما از طرفی باعث میشود پیچیدگی برنامه‌ها بیشتر شده که در نتیجه باید در حالت های خاص از آن استفاده نمود و همیشه چک کرد که آیا API معادلی در Android Framework برای کار مورد نظر ما فراهم شده است یا خیر.

استفاده از Android NDK باعث می‌شود که برنامه‌ی اسکنر ما Multi-Platform باشد و از کدهای C++/C آن بتوان در ios و Windows نیز استفاده نمود. اما باید دقت کنیم که کارهایی مانند Memory Management را در این صورت باید خودمان به عهده بگیریم.

- در مورد سنسورهای hardware-based و software-based تحقیق نمایید و هر یک را تشریح نمایید. هر کدام از سنسورهای مورد استفاده در این تمرین در کدام دسته قرار می‌گیرند؟

سنسورهای hardware-based قطعات فیزیکی‌ای هستند که بر روی یک دستگاه سوار شده اند. آن‌ها دیتاهای خود را از طریق اندازه‌گیری مستقیم مشخصات خاص فیزیکی محیط بدست می‌آورند مانند شتاب، قدرت میدان ژئومغناطیسی، تغییرات زاویه‌ای و .. .

از طرف دیگر سنسورهای software-based با اینکه سنسورهای hardware-based را شبیه‌سازی می‌کنند اما در واقع قطعات فیزیکی نیستند. آن‌ها دیتای خود را از طریق محاسبات بر روی یک یا چند سنسور hardware-based بدست می‌آورند به همین خاطر به آن‌ها سنسورهای مجازی یا ترکیبی نیز می‌گویند. از جمله‌ی این سنسورها میتوان سنسور شتاب خطی و سنسور جاذبه را نام برد.

سنسورهای Gyroscope و Accelerometer ای که در این پروژه استفاده کردیم هر دو از نوع سنسورهای hardware-based هستند.

- چه تفاوتی بین تعریف سنسور به صورت wake-up و non-wake-up وجود دارد؟ ضمن تشریح مزایا و معایب هر کدام، مشخص کنید که انجام این کار تاثیری بر نحوه دریافت بروزرسانی سنسورها و نتیجه ی اسکن دارد یا خیر؟

وضعیت‌های مختلف روشن‌بودن یک SoC عبارتند از: On, Idle, Suspend.

- وضعیت On وقتی است که SoC در حال کار است.

- وضعیت Idle وقتی است که SoC روشن شده اما در حال انجام هیچ task ای نیست.
- وضعیت Suspend وقتی است که سیستم روشن نشده و روی حالت مصرف کم باتری قرار دارد. در این حالت مصرف باتری دستگاه ۱۰۰ بار از وضعیتی که دستگاه در وضعیت On بوده کمتر است.

سنسورهای Non-wake-up سنسورهایی هستند که مانع رفتن SoC به وضعیت Suspend نمی‌شوند و همچنین آن را برای گزارش داده بیدار نمی‌کنند. به طور دقیق‌تر درایورها اجازه ندارند که wake-lock ها را نگه دارند و وظیفه‌ی برنامه‌هاست که که با گرفتن partial wake lock ها اطلاعات را وقتی صفحه خاموش است از سنسورهای non-wake-up دریافت کنند. وقتی SoC در وضعیت Suspend است، سنسورها باید به کار خود ادامه داده و event های جدید تولید کنند، که در یک صف FIFO ی سخت افزاری نگهداری می‌شوند. این event های درون FIFO وقتی SoC بیدار میشود به اپلیکیشن‌ها تحویل داده می‌شوند. اگر FIFO زیادی کوچک باشد نمی‌تواند همه‌ی event ها را نگهداری کند، که در این صورت event های قدیمی‌تر پاک شده و از دست می‌روند. به محض اینکه SoC از حالت Suspend خارج می‌شود همه‌ی event های FIFO گزارش شده و عملیات‌ها به طور عادی ادامه پیدا می‌کنند. برنامه‌هایی که از سنسورهای Non-wake-up استفاده می‌کنند یا باید از یک wake lock استفاده کرده و جلوی Suspend شدن سیستم را بگیرند و وقتی به سنسورها نیاز ندارند از آن‌ها unregister شوند، و یا انتظار داشته باشند که event هایی که در حالت Suspend رخ می‌دهند را از دست بدهند.

برخلاف سنسورهای non-wake-up، سنسورهای wake-up به ما اطمینان می‌دهند که داده‌ها به صورت مستقل از وضعیت SoC ارسال شده‌اند. در زمانی که SoC بیدار باشد، سنسورهای wake-up مانند سنسورهای non-wake-up عمل می‌کنند. وقتی SoC در حالت خواب است، سنسورهای wake-up باید SoC را بیدار کنند تا event ها را تحویل بدهند. البته آن‌ها هنوز باید اجازه دهند SoC به وضعیت suspend برود، اما وقتی نیاز بود event ای گزارش شود باید آن را بیدار کنند. در زمان گزارش یک رخداد باید SoC را بیدار کنند و event ها را تحویل دهند پیش از آن که حداکثر زمان گزارش به سر برسد یا FIFO ی سخت‌افزار پر شود. برای این که مطمئن شویم که application ها زمان کافی برای دریافت event را قبل از این که SoC به خواب برود را دارند، درایور می‌بایست یک timeout wake lock را هر زمان که event ای گزارش شد به مدت ۲۰۰ میلی‌ثانیه فعال کند. پس SoC نباید اجازه داشته باشد بعد از یک wake up interrupt به مدت ۲۰۰ میلی‌ثانیه به وضعیت sleep برود. این نیازمندی در ورژن‌های بعدی اندروید از بین می‌رود و ما تا آن زمان به این timeout wake lock نیازمندیم.

تا اندروید KitKat این که یک سنسور wake-up باشد یا non-wake-up بستگی به نوع سنسور داشت، که اکثرا non-wake-up بودند به جز سنسورهای مجاورت و تشخیص‌دهنده‌ی حرکت‌های قابل توجه.

از اندروید Lollipop به بعد این قضیه با استفاده از یک flag در هنگام تعریف سنسور قابل انتخاب است. می‌توان از خروجی SensorManager.getDefaultSensor(sensorType) مشاهده کرد که اکثر اپلیکشن‌ها از کدام نوع سنسور بسته به تایپ سنسورهای تعریف شده استفاده می‌کنند.