CPS- assignment 2

اسکنر سطح

اعضای گروه:

سید احسان حسن بیگی - ۸۱۰۱۹۷۶۱۹

علی بابابیگ - ۸۱۰۱۹۷۴۶۳

محمدمهدی سرابی زاده - ۸۱۰۱۹۷۵۲۴

فاطمه بهرامی - ۸۱۰۱۹۶۶۳۹

مراحل کار

1. طرز کار با سنسورهای ژیروسکوپ و شتاب‌سنج در اندروید

ابتدا طرز کار این سنسورها و طریقه‌ی استفاده از آن‌ها را بررسی کردیم. هردوی این سنسورها سنسورهای حرکتی هستند که برای تشخیص حرکت دستگاه از آن‌ها استفاده می‌شود. سنسورهای حرکتی آرایه‌های چندبعدی از sensorEvent ها در اختیار ما میگذارند. این برای سنسور شتاب‌سنج به صورت یک آرایه‌ی سه بعدی شامل اطلاعات درباره‌ی نیرو در سه جهت x و y و z است و برای سنسور ژیروسکوپ به صورت یک آرایه‌ی سه بعدی شامل اطلاعات نرخ rotation هر سنسور در سه محور x و y و z است.

برای گرفتن این دیتا از OS ابتدا تابع startSensor رو فراخوانی میکنیم.

```
private void startSensors() {
    Application app = getApplication();
    SensorManager sensorManager = (SensorManager) app.getSystemService(Context.SENSOR_SERVICE);

    initListeners();

    gyroscopeSensor = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
    sensorManager.registerListener(gyroscopeListener, gyroscopeSensor, samplingPeriodUs: 500000);

    accelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
    sensorManager.registerListener(accelerometerListener, accelerometerSensor, samplingPeriodUs: 500000);
}
```

در این قسمت یک callback رجیستر میکنیم که OS در بازه‌های زمانی مشخص شده آن را فراخوانی می‌کند. وقتی OS این callback را فراخوانی می‌کند توابع onSensorChanged برای سنسورهای ژیروسکوپ و شتاب‌سنج صدا زده می‌شوند.

در انتهای برنامه برای این که resource leak نداشته باشیم تابع stopSensors را کال می‌کنیم تا callbackهای رجیستر شده را آزاد کنیم.

2. کد برنامه

برای پیاده‌سازی این اپلیکیشن کد آن را در دو قسمت اصلی پیاده‌سازی کردیم. این دو قسمت شامل اکتیویتی و ویومدل هستند. قسمت اکتیویتی نمایش ویوها رو هندل میکنه و ویومدل لاجیک برنامه رو هندل میکنه. قسمت اصلی کد که محاسبات ناهمواری‌ها، خوندن دیتا از سنسورها و ... هستند در کلاس ویومدل قرار دارند.

3. استفاده از لایبرری MPChart برای نمایش دیتای ناهمواری‌های سطح در نمودار

برای استفاده از این کتابخانه ابتدا dependency آن را به فایل build.gradle اضافه می‌کنیم. برای نمایش دیتا کد نمایش یک BarChart را به اکتیویتی اضافه میکنیم. محاسبه‌ی مقادیر x و y این نمودار در تابع calculateShowingDate در ویومدل پیاده‌سازی شده است.

4. محاسبات مربوط به نمودار

برای محاسبه‌ی مقادیر ناهمواری نسبت به جابه‌جایی در راستای محور x از فرمول‌های زیر کمک گرفتیم:

$$v = \int a dt \qquad x = \int v dt \qquad a = g\sin(\theta) \qquad x = \frac{1}{2}at^2$$

در دیتای محاسبه شده از این فرمول‌ها مقادیری نویز و خطا داشتیم و برای درست‌کردن دیتای نهایی به صورت تجربی اصلاحاتی به فرمول آن اضافه کردیم و در نهایت دیتای به دست آمده با دقت خوبی دیتای ناهمواری‌ها رو درست نمایش می‌دهد. این اصلاحات در کد زیر قابل مشاهده هستند:

```
zAccel = currentAccelZ;
// converging velocity to 0 when acceleration is low
zVel = (Math.abs(zAccel) < Z_ACCEL_THRESH) ? (float) (zVel * Z_VEL_CONV_RATE)
        : zVel + (float) (zAccel * accelerometerDelta);
// added cap for vel
zVel = (zVel > Z_VEL_CAP) ? Z_VEL_CAP : (zVel < -Z_VEL_CAP) ? -Z_VEL_CAP : zVel;

float displacement = (float) (zVel * accelerometerDelta);
// using gyroscope to augment ascending/descending data
zDist += (currentRotationY < -Y_ROT_THRESH) ? GYRO_AUG_STEP
        : (currentRotationY > Y_ROT_THRESH) ? -GYRO_AUG_STEP : 0;
// scaling displacement
// zDist += (displacement > 0) ? displacement * UPWARD_SCALE : displacement *
// DOWNWARD_SCALE;
// added cap for dist
zDist = (zDist > Z_DIST_CAP) ? Z_DIST_CAP : (zDist < -Z_DIST_CAP) ? -Z_DIST_CAP : zDist;
```

در نهایت مقدار متغیر zDist به عنوان y نمودار نمایش داده می‌شود.

5. **استفاده از LiveData برای نمایش نمودار ناهمواری‌های سطح به صورت real time**

برای رساندن دیتای ناهمواری‌ها به صورت real time به اکتیویتی که وظیفه‌ی نمایش آن را بر عهده دارد از لایو دیتا استفاده کردیم. لایو دیتا یک استریم دیتا از ویومدل به اکتیویتی است که در کد ما showingRealTimeEntriesLiveData لایودیتایی است که دیتای real time را به اکتیویتی می‌رساند تا اکتیویتی آن را به نمودار جهت نمایش آن بدهد.

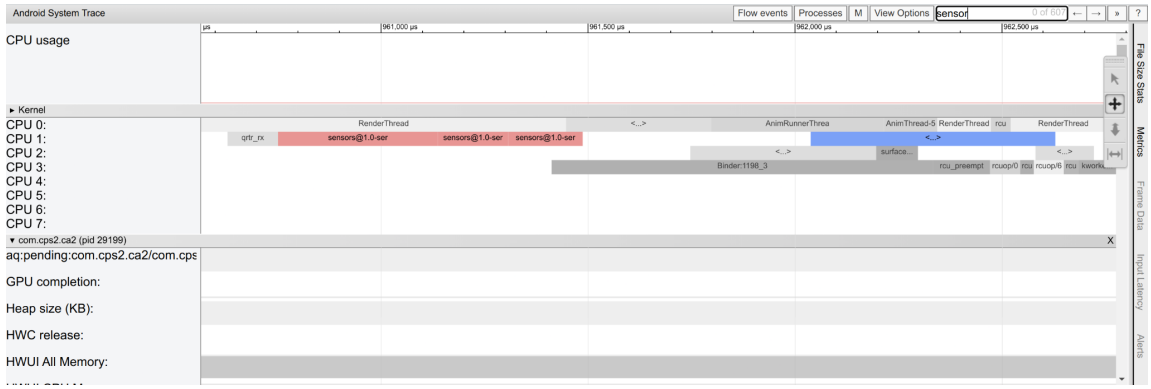
6. **ساختن keystore و ساخت apk نهایی**

برای گرفتن خروجی نهایی از ابزار android studio برای ساختن keystore و بیلد شدن فایل apk نهایی استفاده کردیم.

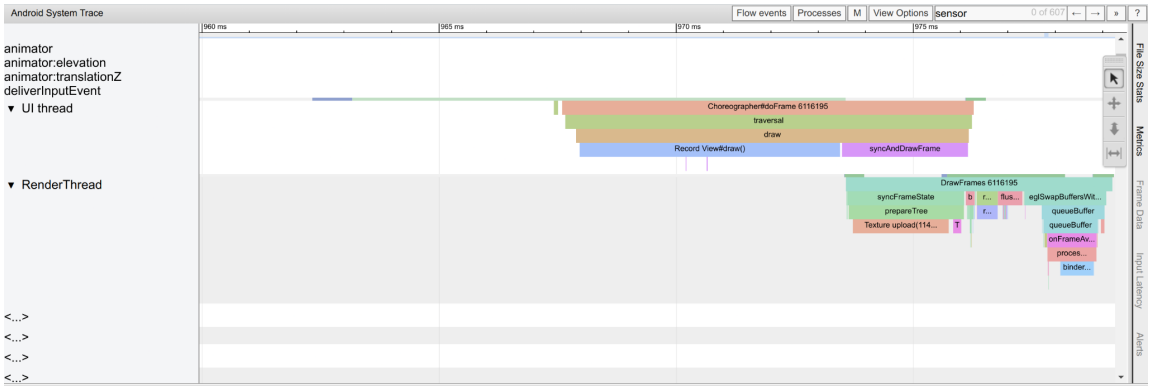
پرسش ها

7. تصویر زیر، دوره‌ی تناوب دریافت داده از سنسور - که تقریباً برابر با 50ms است - را نشان می‌دهد.

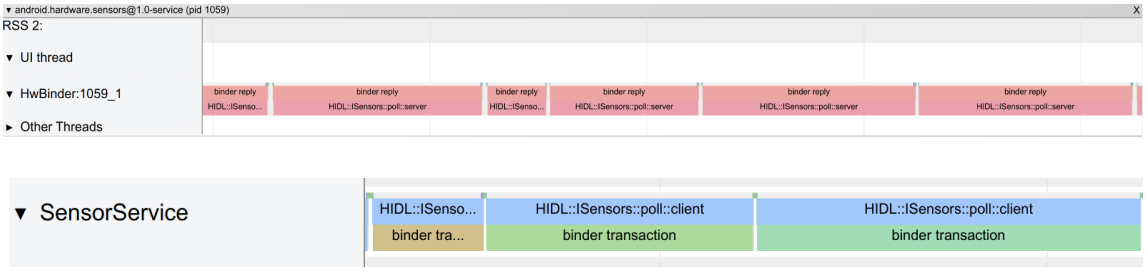
همان‌طور که در تصویر زیر مشخص است، در هر بار دریافت داده از سنسور - که فرکانس آن در کد تعیین شده است - ابتدا android.hardware.sensors@1.0-service در قالب ۳ ریسه‌ی مختلف فعال شده (بازه‌ی صورتی رنگ) و مدتی پس از آن، SensorService (بازه‌ی آبی رنگ) اقدام به فعالیت می‌کند.



8. تصویر زیر از نظر زمانی، متناظر با نوشته شدن داده‌های دریافت شده در تصویر بالا است. خط نازک آبی رنگ که در لحظه‌ی 962ms آغاز شده است، دقیقاً بعد از اتمام نوار آبی رنگ در تصویر قبلی است. اتمام کار Render Thread (نوار سبز رنگ با نام DrawFrames) که در تصویر زیر در لحظه‌ی 979ms رخ داده است، بیانگر نمایش داده‌های دریافتی از UI Thread می‌باشد. با توجه به این توضیحات، نمایش داده‌های دریافتی از سنسور، به اندازه‌ی 17ms به طول انجامیده است.



تصاویر زیر نیز مربوط به اجرای thread های سنسور می‌باشند:



9. به طور کلی مقدار تعیین‌شده و معینی برای نرخ نمونه‌برداری از این سنسورها وجود ندارد و بسته به نیازمندی می‌تواند بسیار متغیر باشد. با این حال، هر سنسور سقف مشخصی برای فرکانس نمونه‌برداری دارد؛ به عنوان مثال، برخی سنسورهای شتاب‌سنج محدودیت فرکانسی در حدود 100Hz الی 125Hz دارند.

   ما به صورت تجربی و با انتخاب مقادیر مختلف، به فرکانس 20Hz (دوره تناوب 50ms) رسیدیم.

10. در صورت استفاده از NDK، امکان استفاده مجدد از کد نوشته شده به زبان C/C++ در اپ اندروید از طریق java native interface را داریم. با این کار به دلیل اجرای مستقیم برنامه بر روی پردازنده (به جای interpret شدن توسط ماشین مجازی Dalvik)، سرعت اجرا بالاتر می رود و شاهد performance بهتری خواهیم بود. به همین جهت، NDK برای کارهای CPU intensive مانند بازی‌های ویدیویی و شبیه‌سازی فیزیکی مناسب‌تر است. همچنین، کد C/C++ نوشته شده برای اندروید (با استفاده از NDK) می‌تواند در پلتفرم های دیگر مثل ویندوز و iOS نیز اجرا شود که باعث می‌شود اپلیکیشن ما multiplatform باشد.

در مقابل، SDK از زبان جاوا استفاده می‌کند و پروژه‌های نمونه، ابزارهای توسعه، Android Studio IDE و API ها و کتابخانه‌های رایج اندروید را در اختیار برنامه‌نویس می‌گذارد. به این دلیل، توسعه‌ی اپلیکیشن را ساده‌تر و احتمالاً سریع‌تر می‌کند. SDK همچنین portability اپلیکیشن را با وجود متفاوت بودن معماری پردازنده‌ها تضمین می‌کند و مدیریت حافظه را به صورت خودکار انجام می‌دهد.

11. سنسورهای hardware-based کامپونت‌های فیزیکی هستند که در تلفن‌های هوشمند ساخته می‌شوند و داده‌هایشان را مستقیماً با اندازه‌گیری مشخصه‌های محیطی به دست می‌آورند. سنسور ژیروسکوپ و شتاب‌سنج مثال‌هایی از این دسته‌اند. در مقابل، سنسورهای software-based دستگاه‌های فیزیکی نیستند؛ ولی رفتار سنسورهای hardware-based را تقلید می‌کنند. سنسورهای software-based داده‌هایشان را از طریق یک یا چند سنسور hardware-based به دست می‌آورند؛ به همین خاطر به این سنسورها، سنسورهای مجازی یا کامپوزیت هم می‌گویند. سنسورهای تشخیص جهت[1]، گرانش، شتاب‌سنج خطی و ... مثال‌هایی از این دسته هستند.

ژیروسکوپ و شتاب‌سنج خطی دو سنسور اصلی مورد استفاده در این تمرین هستند که با استناد به این لینک به ترتیب، hardware-based و software-based هستند؛ البته لازم به ذکر است که خود سنسور شتاب‌سنج، hardware-based است.

سنسورهای ژیروسکوپ و شتاب‌سنج خطی، به ترتیب مشخصه‌های محیطی سرعت زاویه‌ای و شتاب (با حذف گرانش) را اندازه‌گیری می‌کنند. به طور کلی، شتاب‌سنج فقط می‌تواند حرکت خطی را تشخیص دهد؛ ولی ژیروسکوپ tilt و orientation را هم تشخیص می‌دهد.

12. تفاوت سنسورهای wake-up و non-wake-up را در نحوه‌ی تعاملشان با SoC می‌توان یافت. به‌طور کلی SoC از نظر توان مصرفی، سه state مختلف می‌تواند داشته باشد که عبارتند از: on (روشن و در حال اجرا) و idle (روشن ولی بی‌کار) و suspend (خواب).

[1] Orientation

سنسورهای wake-up همواره اطمینان حاصل پیدا می‌کنند که داده‌های ارسالی آن‌ها - صرف‌نظر از state فعلی SoC - به دست SoC می‌رسد. با توجه به این امر، در صورتی که SoC در وضعیت suspend باشد، سنسور اقدام به بیدار کردن SoC خواهد کرد.

سنسورهای non-wake-up همان‌طور که از نامشان نیز بر می‌آید، اقدام به بیدار کردن SoC - به منظور ارسال داده - نمی‌کنند؛ در عوض، این‌گونه سنسورها رویدادها و اطلاعاتی که جمع‌آوری کرده‌اند را در یک لیست FIFO (در لایه‌ی سخت‌افزار) ذخیره می‌کنند تا هر زمانی که SoC از وضعیت suspend خارج شد، آن‌ها را دریافت کند. در نتیجه، در صورتی که مدت زمان suspend بودن طولانی شود و یا رویدادهای زیادی در آن مدت رخ دهد، امکان از دست رفتن برخی داده‌های سنسور به علت تکمیل ظرفیت لیست FIFO وجود دارد.

با توجه به توضیحات فوق، سنسورهای wake-up در زمان suspend بودن SoC نیز داده‌های خود را ارسال کرده و همچنین هیچ loss ای ندارند؛ اما از سوی دیگر، سنسورهای non-wake-up اقدام به بیدار کردن SoC از وضعیت suspended نمی‌کنند و در توان مصرفی دستگاه صرفه‌جویی می‌کنند؛ چرا که SoC ها در حالت suspended، تا ۱۰۰ برابر کم‌تر از حالت on انرژی مصرف می‌کنند.

خروجی نهایی برنامه

در این بخش، خروجی نهایی برنامه با موانعی که در تصویر زیر مشخص هستند، آورده شده است.