دانشگاه تهران
دانشکده‌ی مهندسی برق و کامپیوتر

گزارش نهایی پروژه سیستم‌های سایبر-فیزیکی

پیاده سازی گام شمار با قابلیت تشخیص مسیر حرکت

گروه:
نسترن علیپور ۸۱۰۱۹۶۵۱۵
ایمان مرادی ۸۱۰۱۹۶۵۶۰
رامین فریاد هریس ۸۱۰۱۹۵۴۴۷
پارسا صدری سینکی ۸۱۰۱۹۵۵۲۶

استاد:
دکتر مهدی کارگهی
دکتر مهدی مدرسی

ترم بهار ۱۴۰۰

فهرست مطالب

1- مقدمه

○ محدوده پروژه

امروزه، دستگاه های تلفن همراه به سرعت در حال توسعه هستند و استفاده از آنها رو به افزایش است. در حال حاضر رایج ترین سیستم عامل برای تلفن های همراه، سیستم عامل اندروید است.
این دستگاه های اندرویدی به دلیل استفاده از سنسور های متعدد که دقت و تنوع آن ها به طور روزافزون، رو به افزایش است، امکانات سودمندی را در قالب برنامه های مختلف در اختیار کاربران قرار می دهند . به عنوان مثال می توان از برنامه های healthcare نام برد. برنامه هایی که در هنگام ورزش تعداد گام، مسافت طی شده و کالری مصرف شده کاربر را ثبت می کنند و به او گزارش می دهند. در همین راستا در این پروژه قصد داریم تا یک گام شمار طراحی کنیم.
تاکنون برنامه های متعددی در این حوزه توسعه یافته است اما ما قصد داریم علاوه بر امکان شمارش گام ، مسیر طی شده کاربر را نیز بدون استفاده از GPS و تنها با استفاده از سنسور های داخلی تلفن همراه رسم کنیم تا در صورت عدم دسترسی به GPS یا برای کاهش مصرف باتری نسبت به استفاده از GPS ، بتوان با استفاده از این برنامه، مسیر طی شده را مشاهده کرد و بر روی آن آنالیز های بیشتری انجام داد.

○ اهداف پروژه

- طراحی گام شمار برای کمک به بهبود و افزایش رکورد ورزش روزانه کاربر.
- پیاده سازی گام شمار بدون استفاده از سنسور گام شمار اندروید برای تلفن های همراهی که از این سنسور پشتیبانی نمی کنند.
- امکان مسیریابی در زمانی که سیگنال GPS موجود نیست یا خطا زیادی دارد. مثل فضای داخلی ساختمان ها یا بین ساختمان های بلند.
- تشخیص چرخش کامل و نیمه کامل
- تشخیص بازگشت به نقطه ی اولیه

2- معرفی پلتفرم و ابزارهای استفاده شده در پروژه

سخت افزار:

یک دستگاه تلفن همراه با سیستم عامل اندروید. این تلفن همراه باید حداقل سنسور های Accelerometer و Magnetometer و Rotation Vector را داشته باشد تا بتواند به طور حداقلی کار کند.

موجود بودن سنسور های Light، Proximity، Gravity، Step Counter،Step Detector می توانند به دقت محاسبات کمک کنند.

نرم افزار:

زبان اصلی استفاده شده برای برنامه نویسی این پروژه زبان جاوا بوده. پروژه بر روی محیط Android Studio زده و کامپایل می شود و بر روی تلفن های همراه تست می شود.

برای توسعه همزمان کد ها بین اعضای گروه از GitLab استفاده شد.

3- بیان چالش ها

- عدم پایداری خروجی قطب نما و تغییر های ناگهانی خروجی آن.
با پیاده سازی یک فیلتر پایین گذر توانستیم اثر نویز های کوچک که فرکانس بیشتری دارند را از روی میدان مغناطیسی برای قطب نما حذف کنیم.

- خطا بالا در تعیین جهت در فضای داخلی خانه، حدود ۹۵ درصد خطا ها خطای بالا ۲۳ درجه داشتند.
فیلتر پایین برای چالش هم کاربرد داشت و باعث بوجود آمدن خروجی ثابت تری شد. یک راه حل دیگر برای این چالش کالیبره کردن سنسور های تلفن همراه قبل شروع استفاده از برنامه بود. که این عمل توسط حرکت تلفن همراه در دست به شکل ۸ امکان پذیر می باشد.

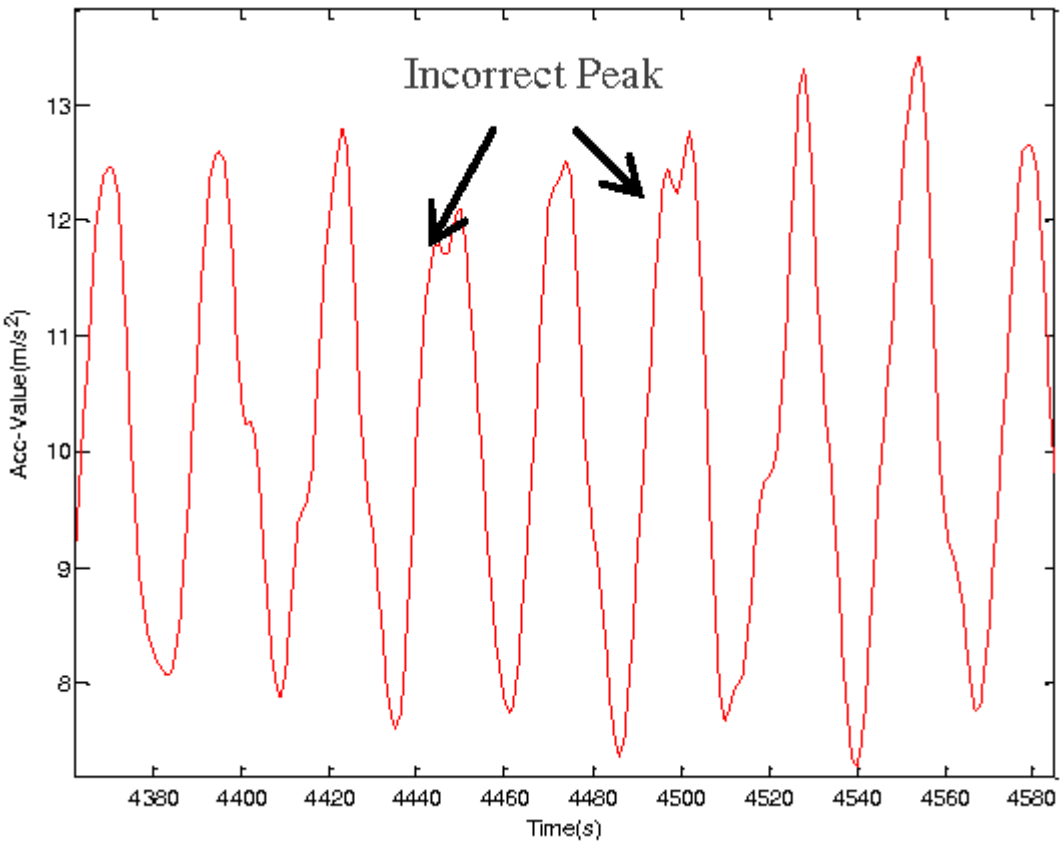
- تفاوت شمال جغرافیای با شمال مغناطیسی.
راه حل این مشکل محاسبه magnetic declination هست که از پیاده سازی آن توسط World Magnetic Model استفاده کردیم.

- خطا بالا قطب نما در هنگام برخورد پا با زمین هنگام قدم برداشتن.
حل این مشکل به این صورت بود که داده های مربوط به قطب نما بین هر دو قدم را در یک لیست نگهداری کردیم و از داده مربوط به قبل برخورد پا به زمین استفاده کردیم برای تشخیص جهت حرکت تا دقت بیشتری داشته باشد.

- تفاوت زیاد مقدار شتاب گام ها هنگامی که تلفن همراه در جیب است یا در دست کاربر.
برای تشخیص گام ها نیاز به تعیین مقدار threshold برای یافتن قله سیگنال های شتاب سنج هست. ولی مشکل این است که هنگامی که تلفن همراه کاربر در دست کاربر قرار دارد یا در جیبش قله سیگنال شتاب سنج از حدود $1\,m/s^2$ به حدود $10\,m/s^2$ تغییر پیدا می کند. برای همین یک ماژول تشخیص این که تلفن همراه در جیب قرار دارد یا در دستان کاربر پیاده سازی کردیم که با توجه به این تشخیص مقدار threshold را تعیین کنیم.

- تفاوت زیاد مقدار شتاب هنگام گام های قدم زدن عادی و دویدن.
برای تشخیص بین این دو و تعیین threshold مناسب برای آن ها از Activity Recognition API درون Google Play Services استفاده کردیم تا نوع فعالیت را تشخیص دهیم و مقادیر مناسب را برای threshold قرار دهیم.
.
- تشخیص گام هنگامی که تلفن همراه جابجا می شود ولی کاربر در حال راه رفتن نیست.
برای حل این مشکل هم از Activity Recognition API استفاده کردیم و گام های شمرده شده که در هنگام پیاده روی تشخیص داده نشده بودند را شمارش نکردیم. که با این روش توانستیم مقدار زیادی خطا های شمارش در طول روز را کاهش دهیم.

- احتساب چند گام با فاصله زمانی بسیار کم در اثر یک گام.
در هر گام برای هر پا دو قله بوجود می آید در سیگنال شتاب سنج. در صورتی که این دو قله هر دو مقداری بیشتر از threshold تعیین شده داشته باشند، دو گام به ازای این یک گام ثبت می شود. برای حل این مشکل ما زمان حداقلی برای فاصله بین دو ام تعین کردیم تا در صورتی که دو قله نزدیک تر از این فاصله زمانی نسبت به هم رخ بدهند شمارده نشوند.
در تصویر زیر نمونه یکی از این موارد را می توانید ببینید.



- تشخیص میزان شتاب هنگام قدم زدن در شرایط متفاوت.

برای محاسبه و بدست آوردن مقادیر مناسب برای threshold در شرایط مختلف ما نمودار سیگنال شتاب سنج را هنگام راه رفتن در شرایط مختلف ذخیره کردیم و مقدار تقریبی قله های سیگنال شتاب سنج را در شرایط مختلف بدست

۴- نزدیکترین نمونه های مشابه

برای تشخیص مسیر حرکت بدون استفاده از GPS راه حل پیش فرض گرفتن انتگرال دوم از شتاب تلفن همراه است. به علت اینکه فقط با انتگرال دوم داده های سنسور شتاب سنج به تنهایی نمی شود جابجایی تلفن همراه را محاسبه کرد چون خطا به شدت زیادی پیدا می کند بعد از چند ثانیه، تعدادی زیادی مقاله و پروژه با محدود کردن نوع حرکت و جابجایی، یا داشتن دانش پیشینی نسبت به محیط و یا استفاده از سیگنال های کمکی و خارجی در محیط سعی در حل این مسئله داشته اند.
برای مثال برای محدود کردن نوع حرکت و جابجایی تعدادی مقاله فرض کرده بودند کاربر فقط در خودرو در حال حرکت است و در راستای z هیچ نوع شتابی ندارد و به این صورت سعی در کاهش خطا محاسبه انتگرال دوم داده های سنسور شتاب سنج داشه باشد. یا اینکه به جای محاسبه انتگرال دوم شتاب کاربر با استفاده از فرض اینکه کاربر در حال قدم زدن است و هر قدم به طور متوسط طول ثابتی دارد سعی کردند مسافت طی شده کاربر را با استفاده از یک گام شمار بدست بیاورند.
برای مثال مربوط به استفاده از سیگنال های موجود در محیط هم تعدادی تلاش کرده بودند با استفاده از یک سیگنال ثابت در محیط مثل بلوتوث یا WiFi خطا در موقعیت یابی تلفن همراه را کاهش بدهند.
یه غیر از این پروژه هایی که مستقیما هدف مشابه ای با پروژه ما داشتند پروژه زیاد دیگری هم بودند که تشابه فقط با بخشی از پروژه ما داشتند. مثلا تعدادی پروژه مربوط به بخش گام شمار بودند یا تعدادی مربوط به تشخیص جهت حرکت یا جهت های جغرافیایی بودند.
در منابع لیستی از پروژه ها و مقالات مشابه با دسته بندی مربوطه آورده شده است.

5- مبانی فنی پروژه

- ارائه راه‌حل پیشنهادی بصورت کلی

راه حل کلی:

```
while IsWalking() == true do
    if DetectStep(accelerometerData) == true then
        direction = GetDirection(magnetometerData, accelerometerData);
        location = GetLastLocation();
        newLocation = CalculateMovement(location, direction);
        DrawMovementGraph(movementGraph, newLocation);
    else
end
```

تشخیص مسیر حرکت:
راه حل پیشنهادی ما برای بدست آوردن مسیر حرکت این است که هر گام کاربر را تشخیص دهیم و در لحظه گام جهت حرکت کاربر را نیز محاسبه کنیم. با این روش می توانیم مسیر حرکت کاربر را با هر گام بروزرسانی کنیم.

- ارائه راه‌حل با جزییات

تشخیص گام:
برای تشخیص گام ابتدا مقدار شتاب گرانشی را از شتاب در هر راستا کم می کنیم. سپس اندازه شتاب کلی تلفن همراه محاسبه می شود. در مرحله بعد قله های این سیگنال تشخیص داده می شوند. قله های تشخیص داده شده باید چند شرط لازم را داشته باشند تا آن ها را به عنوان گام شمارش کنیم. اولین شرط این است که فرکانس قله ها باید کمتر از مقدار حداکثر فرکانس گام های انسان باشد. دو شرط دیگر این است که اندازه شتاب در قله باید از یک مقدار آستانه گام بیشتر باشد و از مقدار یک مقدار آستانه نویز کمتر باشد.
برای کاهش خطا هنگامی که کاربر در حال حرکت نیست از ActivityRecognitionAPI موجود در Google Play Service استفاده کردیم.
برای محاسبه مقدار متغیر های آسانه گام و آستانه نویز ابتدا تشخیص می دهیم که تلفن همراه در جیب کاربر می باشد یا در دستش قرار دارد. سپس از ActivityRecognitionAPI استفاده کردیم تا تشخیص دهیم کاربر در حال دویدن هست یا راه رفتن. متناسب با این که تلفن همراه در چه وضعیتی قرار دارد و کاربر با چه سرعتی حرکت می کند این دو متغیر را مقدار دهی کردیم.

تشخیص جهت حرکت:
برای تشخیص جهت حرکت ما از یک روش اصلی و یک روش کمکی استفاده کردیم. برای تشخیص جهت روش اصلی استفاده از سنسور مغناطیس سنج بود تا توسط آن جهت گیری تلفن همراه را در جهت های جغرافیایی پیدا کنیم.
روش دوم استفاده شده این بود که ابتدا یک جهت تلفن همراه را در زمان ابتدایی توسط روش قبلی بدست بیاوریم و سپس از آن زمان به بعد با استفاده از سنسور ژیروسکوپ مقدار چرخش را نسبت به جهت ابتدایی محاسبه کنیم.

تشخیص موقعیت تلفن همراه:

برای تشخیص موقعیت تلفن همراه، یعنی این که تلفن در جیب کاربر قرار دارد یا در دست فرد، ما با استفاده از سنسور های نور، نزدیکی و شتاب سنج سه پارامتر نور محیط، نزدیکی تلفن همراه به بدن کاربر و زاویه گوشی را تشخیص می دهیم که با استفاده از این پارامتر ها می توان تشخیص داد که تلفن همراه در جیب کاربر قرار دارد یا خیر.

فیلتر داده های سنسور ها:

داده های ورودی از سنسور ها ممکن است دارای مقداری نویز باشند یا این که بیش از حد به عوامل محیطی مثل لرزش حساسیت داشته باشند و به همین دلیل خطا در الگوریتم های استفاده شده ایجاد کنند. به همین منظور از تعدادی فیلتر برای سنسور هایی که به مشکل بر خوردیم استفاده کردیم.

اولین فیلتر مورد استفاده فیلتر Low Pass بود که برای حذف سیگنال های با فرکانس بالا از داده های سنسور استفاده شد. با استفاده از این فیلتر ما توانستیم مقدار خروجی قطب نما را بسیار با ثبات تر کنیم و شتاب گرانشی را از داده های سنسور شتاب سنج حذف کنیم.

فیلتر بعدی که استفاده کردیم فیلتر Complementary بود که با استفاده از آن خروجی دو روش مکمل تشخیص جهت مان را به یک خروجی واحد با خطا کمتر تبدیل کردیم.

یکی دیگر از فیلتر های مورد استفاده فیلتر Median بود که با استفاده از آن خطا تشخیص جهت به هنگام ضربه وارده زمان گام را توانستیم حذف کنیم و از جهت حرکت تشخیص داده شده بین دو گام استفاده کردیم.

علاوه بر فیلتر ها یک مشکل دیگری که وجود داشت Drift انتگرال داده های سنسور ژیروسکوپ یا داده های سنسور Game Rotation Vector بود که با ریست کردن محاسبات آن ها بعد حدود هر ۲۰ ثانیه توانستیم این مشکل را حل کنیم.

6- پیاده‌سازی‌های انجام شده

- شکست کار بین اعضای تیم

نام اعضا	کار های انجام شده توسط فرد
پارسا صدری سینکی	بهبود شمارش گام ها ، افزودن فیلترهای low pass و Complementary و Median روی قطب نما ، افزودن پلات نمایش مسیر ، پیاده سازی کد اولیه تشخیص چرخش ، افزودن امکان استفاده از Activity Recognition گوگل برای یافتن نوع اکتیویتی ، افزودن تشخیص مکان قرار گیری گوشی ، پیاده سازی تشخیص بازگشت به نقطه ی اول
ایمان مرادی	پیاده سازی تشخیص چرخش ۳۶۰ درجه ای و ۱۸۰ درجه ای کاربر ، افزودن امکان متوقف کردن کامل سرویس ها ، افزودن رابط سنسور Step Detector
نسترن علی پور	افزودن کلاس های رابط سنسور های Accelerometer و Magnetic Field برای قطب نما . ایجاد قطب نما در صفحه ی مسیریاب ، افزودن سرویس محاسبه ی مسیر ، استایل layout های برنامه
رامین فریاد	پیاده سازی کد پایه ی شمارش گام ها ، پیاده سازی نوتیفیکیشن

- مشخصات محیط توسعه

برای توسعه ی این برنامه از ادیتور Android Studio استفاده کردیم که دارای قابلیت وصل گوشی و دیباگ برنامه در هنگام اجرا است. برای استفاده از این ادیتور و نوشتن برنامه ی اندروید کافی است که Android SDK روی لپ تاپ نصب شده باشد.

- تشریح پیاده‌سازی

در این قسمت پیاده سازی برنامه با جزئیات آبجکت ها توضیح داده میشود. برنامه از یه بخش اصلی تشکیل شده است:

- بخش شمارش گام ها که در سرویس StepCounterService انجام و در StepCounterAcitivity نمایش داده میشود.
- بخش نمایش مسیر که در سرویس RoutingService انجام و در RountingActivity نمایش داده میشود.
- بخش تشخیص چرخش کامل و نیمه کامل که در کلاس LocalDirection انجام و با استفاده از Turning180DegreeAlert و Turniing360DegreeAlert به صورت الرت نمایش داده میشود.

در ادامه جزئیات هر بخش را شرح می دهیم.

نحوه ی ذخیره ی داده های برنامه در دیتابیس:

برای ذخیره ی داده های اصلی برنامه ، مانند تعداد گام های طی شده و مسیر طی شده ، از یک دیتابیس استفاده میکنیم. به این منظور از کلاس sharePreferences در اندروید استفاده می کنیم.

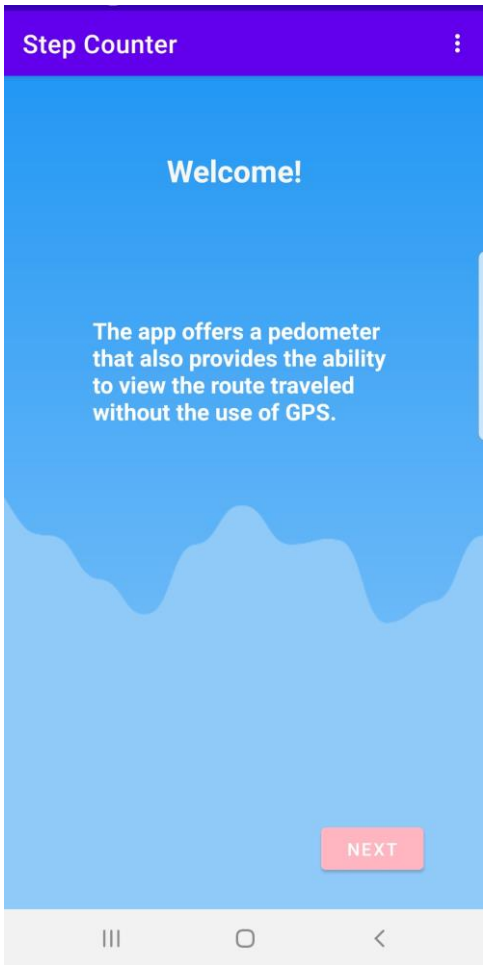
MainActivity

اکتیویتی اصلی برنامه ، MainActivity است. این کلاس از AppCompatActivity ارث بری میکند. AppCompatActivity کلاسی پایه برای اکتیویتی هایی است که می خواهند از برخی از ویژگی های جدیدتر پلت فرم در دستگاه های Android قدیمی استفاده کنند.

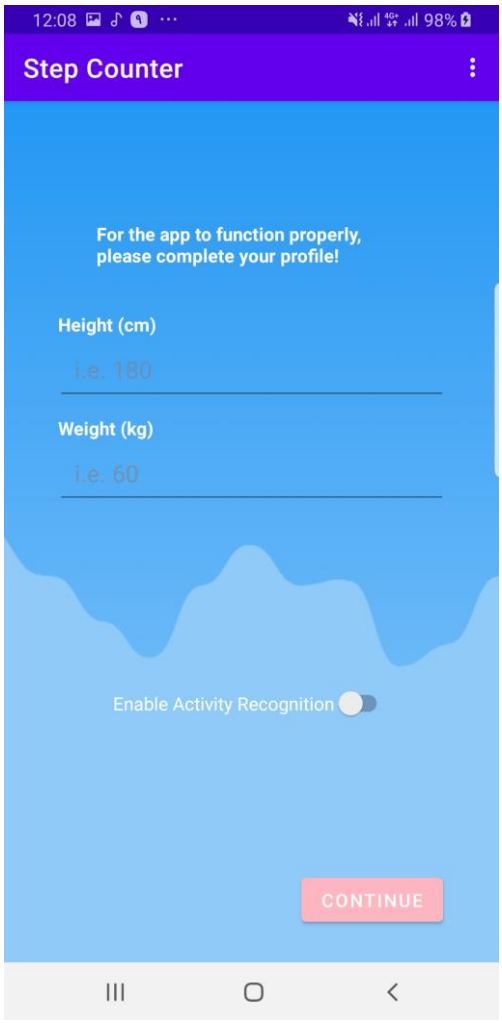
در این برنامه از سرویس های گوگل پلی برای تشخیص نوع حرکت استفاده میشود و در هنگام باز شدن برنامه نیاز است تا پرمیشن ACTIVITY_RECOGNITION درخواست کنیم. بنابر این با استفاده از ActivityCompat.requestPermissions این درخواست را به صورت یک پنجره ی pop-up ارسال میکنیم.

این اکتیویتی دارای سه layout است ، قصد داریم تا هنگام ورود به برنامه یک توتوریال به کاربر نمایش دهیم و برنامه را معرفی کنیم ، در صفحه ی دیگری اطلاعات قد و وزن فرد را دریافت میکنیم و در نهایت وارد قسمت اصلی برنامه شویم. به این منظور از سه layout زیر استفاده کرده ایم:

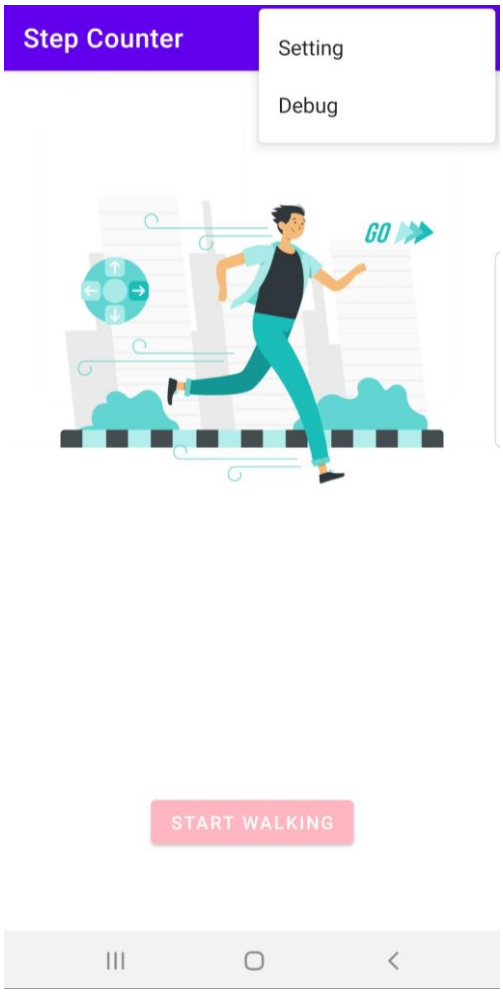
1. tutorial.xml: در این layout به معرفی برنامه میپردازیم دارای آبجکت های زیر است:
   a. دو آبجکت از نوع textView که متن معرفی برنامه در آن ها نوشته شده است.
   b. یک آبجکت از نوع Button که با کلیک آن میتوان به layout بعدی رفت.



2. tutorial_get_info.xml: در این layout اطلاعات فرد را دریافت میکنیم:
   a. دو textView که در آن ها تایتل height و weight نوشته شده است.
   b. دو EditText با شناسه ها Height_tutorial و Weight_tutorial که در آنها کاربر اطلاعات قد و وزن خود را وارد میکند.
   c. یک سوییچ با شناسه ی recognitionEnSwitch که کاربر انتخاب میکند از activity recognition گوگل برای شناسایی نوع فعالیت استفاده شود.



3. activity_main.xml: این layout،layout اصلی این امتیویتی است که با هر بار وارد شدن به برنامه آن را مشاهده میکنیم. در صورتی که دو layout قبلی تنها برای اولین استفاده از برنامه نمایش داده میشوند. در این layout ، آبجکت های زیر را قرار داده ایم:
   a. یک button با شناسه ی R.id.startWalking که با کلیک کردن آن میتوان به اکتیویتی StepCounterActivity رفت.
   b. یک ActionBar که به صورت سه نقطه در Toolbar این صفحه قابل مشاهده است. آیتم های زیر در این ActionBar وجود دارد:
      i. تنظیمات : که در آن میتوان اطلاعات قد و وزن و اجازه ی استفاده از Activity recognition را که کاربر قبلا وارد کرده بود ، اصلاح کرد. با کلیک این آیتم به اکتیویتی SettingActivity وارد میشویم که این اکتیویتی را در ادامه توضیح خواهیم داد.
      ii. ابزار دیباگ : که برای نمایش اطلاعات دریافت شده از سنسور های شمارش گام ، استفاده میشود. با کلیک این آیتم به اکتیویتی StepCounterDebugActivity وارد میشویم که رفتار این اکتیویتی را نیز در ادامه شرح خواهیم داد.



SettingsActivity

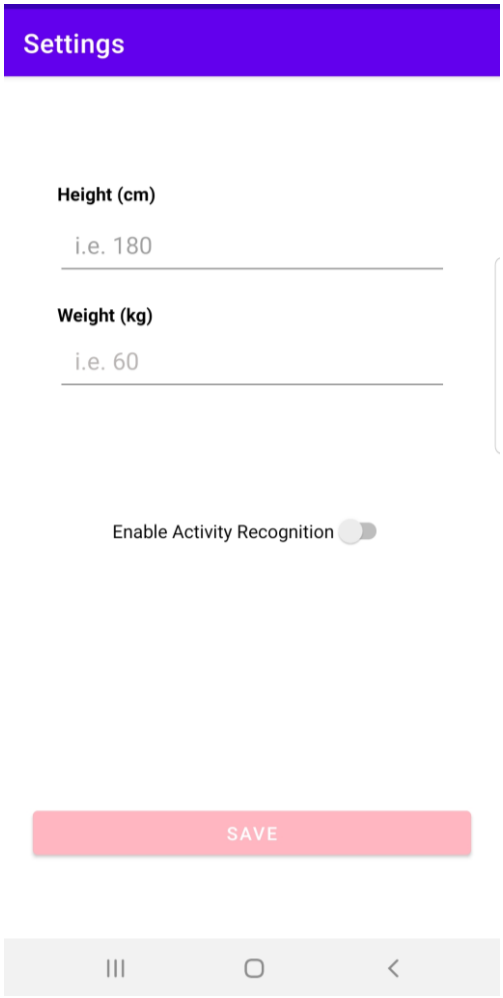
اکتیویتی SettingsActivity که پیش تر از آن نام برده شد ، برای تعیین کردن تنظیمات اولیه ی برنامه شامل موارد زیر است:

قد: میزان قد فرد که با یک hint نشان داده شده است که مقداری بر اساس سانتی متر است.

وزن: میزان وزن فرد که با یک hint نشان داده شده است که مقداری بر اساس کیلوگرم است.

یک سوئیچ با شناسه ی R.id.recognitionEnSwitch که برای مشخص کردن این است که آیا سرویس گوگل پلی (google play services) در سرویس StepCounterService مورد استفاده قرار گیرد یا خیر. در این مورد در ادامه توضیحات بیشتری می دهیم.

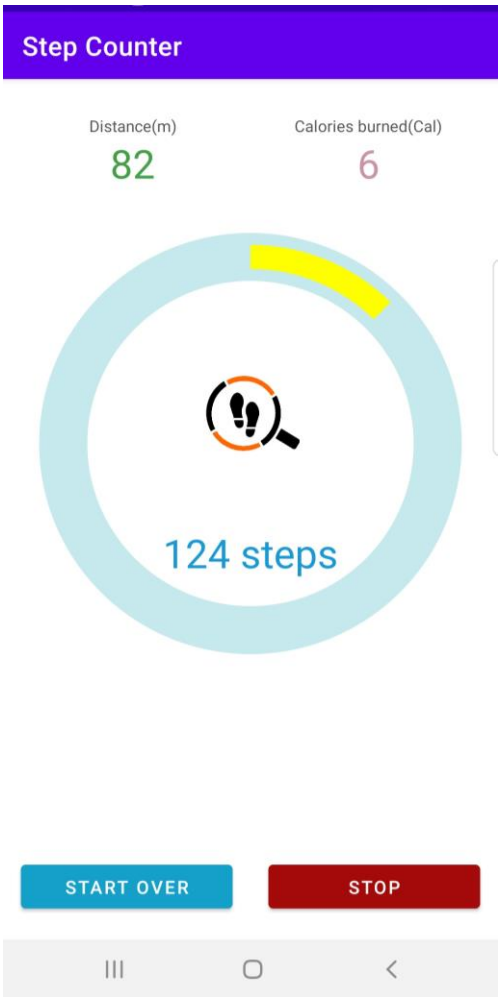
در انتها یک button سیو داریم. اطلاعات وارد شده در صورتی ذخیره میشوند که این کلیک را فشار دهیم. بنابراین در هنگام کلیک این دکمه بررسی میکنیم که آیا متنی در فرم های قد و وزن وارد شده است. در صورتی که عدد باشند ، آن ها را در دیتابیس در ستون های height , weight ذخیره میکنیم. برای ذخیره ی آن ها در دیتابیس از Shared Preferences استفاده میکنیم.



StepCounterActivity

StepCounterActivity یکی از مهمترین اکتیویتی های این پروژه است که در آن تعداد گام های طی شده ، محاسبه و نمایش داده میشود. این اکتیویتی دارای یک سرویس با نام StepCounterService است که محاسبات و تشخیص گام در این سرویس انجام میگیرد.

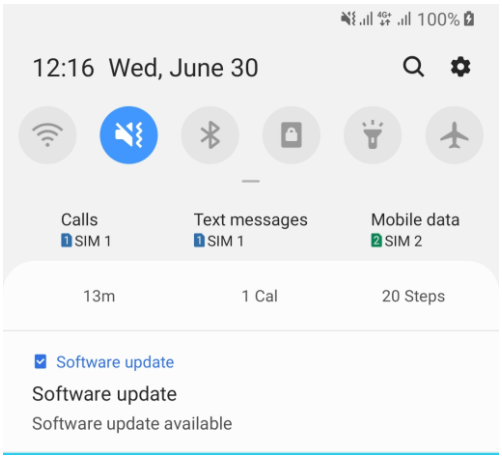
در این اکتیویتی یک تایمر ایجاد کرده ایم. به این صورت که در فاصله زمانی های ۱ ثانیه یک تابع با نام updateStepCounterLayout صدا زده میشود و محتوای نمایشی این اکتیویتی آپدیت میشود. layout این اکتیویتی دارای یک باکس است که تعداد گام های طی شده ، میزان کالری سوزانده شده و مسافت طی شده را نشان میدهد. همچنین دارای یک progressBar به شکل حلقه است که تعداد گام های طی شده را به صورت گرافیکی تر نمایش میدهد. (شکل ۲). در layout این صفحه دو کلید قرار داده ایم. کلید یک دکمه ی 'Start Over' در صفحه وجود دارد که در onClick این دکمه ، تعداد گام های طی شده ی ذخیره شده در دیتابیس ، به صفر آپدیت میشود. همچنین یک دکمه ی 'Show Movement' نیز در این صفحه وجود دارد که با کلیک آن می توان به اکتیویتی RoutingActivity رفت.



در این اکتیویتی در هنگام ایجاد شدن (onCreate) دو تابع startStepCounter() و startRouting() را فراخوانی می کنیم. در این توابع intent هایی برای کلاس های سرویس StepCounterService و RoutingService ایجاد میکنیم و با استفاده از آنها ، سرویس ها را start میکنیم تا شمارش گام ها و تشخیص مسیر حرکت ، از لحظه ی کلیک دکمه ی startWalking و ایجاد StepCounterActivity شروع شود.

StepCounterService

همانطور که پیشتر گفته شد ، StepCounterService محاسبات اصلی برای تشخیص گام را انجام میدهد. علاوه بر انجام محاسبات این سرویس یک نوتیفیکیشن را در پنل گوشی کاربر ایجاد میکند که گام ها و کالری سوزانده شده و مسافت طی شده را به صورت real-time نمایش میدهد.



ابتدا حرکت فرد و نوع حرکت فرد را بررسی میکنیم. به این منظور با استفاده از BroadcastReceiver مقدار های تولید شده توسط کلاس DetectedActivitiesIntentService را دریافت می کنیم. این مقادیر احتمال اینکه فرد در حال حاضر در کدام وضعیت

- در حال حرکت (on foot)
- راه رفتن (walking)
- دویدن (running)

قرار دارد را مشخص میکنند. (اگر یک یک فرد on foot باشد ، در یکی از دو وضعیت walking و یا running قرار دارد). تابع handleUserActivity() با دریافت این احتمال ها ، نوع حرکت را تعیین میکند.

یک sensorManager در این سرویس ایجاد کرده ایم تا برای کنترل و ایجاد سنسور ها مورد استفاده قرار گیرد. برای تشخیص گام از دو سنسور استفاده کردیم. اولین سنسور تشخیص گام گوشی است که در صورتی که این سنسور در گوشی وجود داشته باشد از آن استفاده میکنیم. اما از آنجایی که بسیاری از گوشی ها این سنسور را ندارند ، از یک سنسور شتاب سنج به عنوان جایگزین استفاده کرده ایم. به این صورت که اگر تغییرات شتاب در یک محدوده ی مشخصی بود ، یک گام تلقی شود.

در تابع onSensorChanged که زمانی که مقدار گزارش شده توسط سنسور تغییر کند ، صدا زده میشود ، مقدار های جدید شتاب در راستای x , y, z دریافت میکنیم. سپس مقدار اندازه ی شتاب را حساب میکنیم. سپس میزان شتاب گرانشی را حذف میکنیم تا شتاب خالص گوشی به دست آید.

این سرویس دارای یک تایمر است که هر ۰,۲ ثانیه ، یک آبجکت از نوع TimerTask را ایجاد میکند و تابع run() از این آبجکت اجرا میشود. در این تابع اگر در حال استفاده از سنسور تشخیص گام هستیم ، تعداد گام تشخیص شمارش شده را به تعداد گام های طی شده اضافه میکنیم. در حالتی که از سنسور شتاب سنج استفاده میکنیم نیز اگر تعداد تغییرات شتاب در محدوده ی مشخصی بود (بین ۰ تا ۶ پیک) در این حالت یک گام به گام های طی شده اضافه میکنیم. در فاصله ی زمانی ۰,۲ حداکثر میتوان یک گام برداشت بنابراین به صورت real-time و دقیق تمام گام ها را شمارش می کنیم. در انتهای این تابع notification را آپدیت میکنیم و تعداد جدید گام های طی شده را در دیتابیس ذخیره میکنیم.

برای محاسبه ی کالری سوخت شده و مسافت طی شده ، با استفاده از اطلاعات قد و وزنی که فرد در صفحه ی setting وارد کرده است و گام های شمارش شده ، این مقادیر را محاسبه میکنیم. فرمول های محاسبه ی این مقادیر به صورت زیر است:

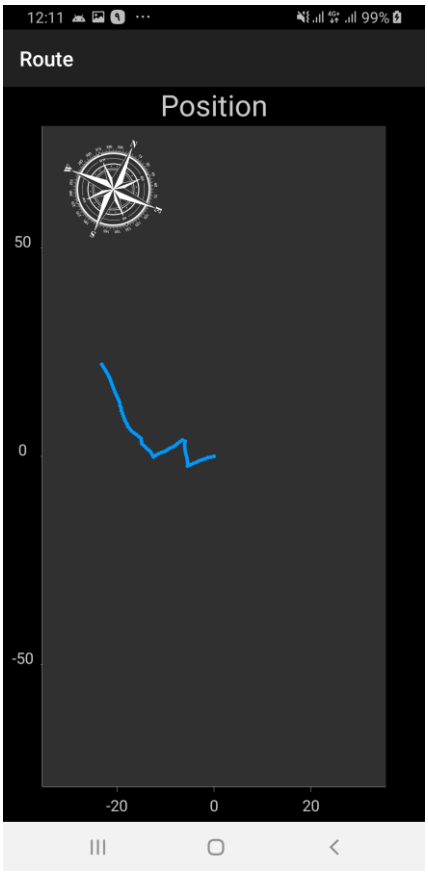
$$\square\square\square\square\square\square\square\square = \square\square\square\square\square * \square\square\square\square\square\square * 0.3937 * 0.414 * 0.0254$$

$$\square\square\square\square\square\square\square\square\square = \square\square\square\square\square * (0.035 * \square) + ((\square/\square) * (0.029 * \square))/150$$

که a متوسط شتاب حرکت و m وزن فرد برحسب کیلوگرم و h قد فرد بر حسب متر و height قد فرد بر حسب سانتی متر است.

RoutingActivity

در اکتیویتی RoutingActivity مسیر طی شده توسط کاربر را نمایش می دهیم. در این صفحه علاوه بر نمایش مسیر یک قطب نما نیز قرار داده ایم که جهت حرکت را نیز داشته باشیم. این کلاس دارای یک تایمر است که هر ۱ ثانیه ، یک آبجکت از کلاس UpdateGraph را new میکند . این کلاس از نوع TimerTask است بنابراین دارای یک تابع run به صورت پیش فرض است و زمانی که یک آبجکت از این کلاس ایجاد شود ، این تابع اجرا میگردد. در این تابع ابتدا زاویه ی گوشی با شمال جغرافیایی را محاسبه میکنیم و مسیر را آپدیت میکنیم. برای محاسبه ی این زاویه از یک سرویس به نام RoutingService استفاده کرده ایم که جلوتر این سرویس را شرح خواهیم داد. در تصویر زیر صفحه ی مسیریاب را مشاهده میکنیم که علاوه بر نمایش مسیر ، یک قطب نما نیز در این صفحه وجود دارد:



RoutingService

این سرویس در زمان ایجاد StepCounterActivity شروع به کار کرده است. در این کلاس برای محاسبه ی مداوم و real time اینکه زاویه ی حرکت فرد نسبت به شمال چه میزان است ، از یک تایمر با interval برابر با ۰,۰۷ ثانیه استفاده کرده ایم. دلیل استفاده از این فاصله ی زمانی این است که به طور معمول در هر ثانیه یک فرد میتواند حداکثر سه گام بردارد و ما نیاز داریم بین هر دو گام حداقل دو بار نمونه برداری کنیم. با این تایمر میتوان به ازای هر گام طی شده ، به صورت real time مسیر را آپدیت کرد. بنابراین در فاصله زمانی های ۰,۰۷ ثانیه یک آبجکت از کلاس UpdateGradient ایجاد میشود. این کلاس از TimerTask ارث بری میکند. بنابراین دارای یک تابع run است که در هنگام ایجاد آبجکت اجرا میشود و در آن محاسبات اصلی صورت میگیرد. به این صورت که زاویه ی گوشی دریافت میشود و اگر در این بازه ی زمانی ۰,۰۷ ثانیه ، گامی توسط سرویس StepCounterService شناسایی شده بود ، مسیر طی شده را آپدیت میکنیم و یک نقطه ی جدید با توجه به بردار زاویه ی گوشی به آن اضافه میکنیم. همچنین در تابع

checkReturnToStartingPoint() بررسی میشود که آیا نقطه ی جدید ، نزدیک به نقطه ی شروع است ، در صورتی که فاصله ی آن ها از یکدیگر کمتر از $\sqrt{5}$ باشد ، یک الرت با استفاده از ویجت Toast در گوشی نمایش می دهیم.

برای محاسبه ی زاویه گوشی ، کلاس RoutingService دارای یک آبجکت از کلاس Orientation است. این کلاس اطلاعات دو سنسور Accelerometer و MagneticField را دریافت میکند و با استفاده از آن ها زاویه نسبت به شمال جغرافیایی را برمیگرداند.

در ادامه نحوه ی به دست آوردن مقادیر سنسور های Accelerometer و MagneticField و فیلتر هایی که بر سر خروجی آن ها قرار داده ایم ، توضیح میدهیم.

سنسور Accelerometer در مکان یاب

برای استفاده از استفاده از سنسور های سخت افزاری گوشی نیاز است تا از API های مناسب برای آن ها استفاده کنیم. این API ها در کلاس SensorEventListener تعریف شده اند و با ارث بری از این کلاس میتوان به توابع مفید آن دست پیدا کرد. از آنجایی که در این پروژه از تعدادی سنسور مختلف استفاده شده است ، از یک رابط بین کلاس SensorEventListener و آن ها استفاده شده است. این رابط همان کلاس GameSensorListener است. در این کلاس sensorManager به عنوان ورودی دریافت میشود.
سنسور magnetic field تغییرات قطب های زمین را نشان میدهد. اما برای اینکه بالا و پایین گوشی تشخیص داده شود ، نیاز است تا از سنسور acceleration نیز استفاده گردد.

کلاس Accelerometer

کلاس Accelerometer که از این کلاس ارث بری میکند ، در تابع createSensor() خود یک سنسور با تایپ TYPE_ACCELEROMETER که همان سنسور شتاب سنج است ، ایجاد میکند و در تابع onSensorChanged هر بار که شتاب دستگاه تغییر میکند ، اطلاعات شتاب جدید در سه محور مختصاتی را دریافت میکنیم ، سپس آن را از یک low Pass Filter عبور میدهیم. دلیل آن هم این است که سنسور accelerometer دارای noise زیادی است و در صورتی که گوشی تکان یک دفعه ای داشته باشد ، ممکن است برای مدتی مقدار نشان داده شده ، noise زیادی داشته باشد. بنابراین با استفاده از Low Pass Filter میتوان تغییرات را smooth تر کرد. پیاده سازی این فیلتر به این گونه است که مقدار قبلی را با ضریبی از اختلاف مقدار جدید و مقدار قبلی ، جمع میکند و پاسخ را باز میگرداند. این روش باعث میشود که خروجی نهایی ، به مقدار خروجی قبلی نزدیک تر باشد و نویز ها تا حدی حذف شوند.

کلاس Magnetometer

کلاس Magnetometer نیز مشابه Accelerometer از کلاس GameSensorListener ارث بری میکند. در این کلاس یک سنسور از تایپ TYPE_MAGNETIC_FIELD که همان سنسور قطب نما است ، ایجاد میشود. تابع پیش فرض onSensorChanged زمانی که قطب های جغرافیایی تغییر کند و در حقیقت زاویه با شمال عوض شود ، در این صورت مقادیر جدید را در هر سه محور دریافت میکنیم و آن را مانند شتاب سنج ، از یک فیلتر low Pass Filter عبور میدهیم. دلیل استفاده از این فیلتر نیز مانند قبل ، وجود نویز زیاد در خروجی این سنسور است.

کلاس InPocketDetector

در هنگام راه رفتن ، اگر گوشی در دستان فرد باشد ، الگوی تغییرات شتاب و محدوده ی تغییرات آن متفاوت از زمانی است که گوشی در جیب فرد قرار دارد. بنابراین باید با توجه به مکان قرار گیری گوشی ، شروط تغییرات شتاب برای تشخیص گام ، لحاظ گردد.
به این منظور کلاس InPocketDetector را پیاده سازی کرده ایم. این کلاس برای تحقق این هدف از ۳ سنسور استفاده میکند.

اولین سنسور light است که میزان روشنایی را مشخص میکند. در زمانی که گوشی در جیب است عددی کمتر از ۱ نشان میدهد اما زمانی که بیرون است عددی در حدود ۱۵۰ را تشخیص میدهد. به منظور استفاده از این سنسور گوشی ، یک سنسور با تایپ TYPE_LIGHT ایجاد میکنیم.
دومین سنسور Proximity است. این سنسور نزدیکی گوشی به بدن را نشان میدهد. زمانی که گوشی در داخل جیب است معمولا مقدار صفر و زمانی که بیرون است عددی در حدود پنج را نشان میدهد. به این منظور سنسور با تایپ TYPE_PROXIMITY ایجاد میکنیم.
دلیل اینکه علاوه بر سنسور روشنایی از سنسور مجاورت استفاده میکنیم این است که ممکن است فرد در شب در حال پیاده روی باشد ، بنابراین در این حالت سنسور روشنایی اطلاعات زیادی به ما نمیدهد.
سومین سنسوری که مورد استفاده قرار میگیرد سنسور شتاب گرانشی است. دلیل آن این است که زاویه قرار گیری گوشی را متوجه شویم ، زمانی که گوشی در دست فرد قرار دارد و با آن کار میکند ، احتمالا گوشی زاویه ی ۰ و یا ۱۸۰ درجه دارد و زمانی که در جیب باشد ، زاویه ای حدود ۹۰ درجه دارد.
در شرایطی که مقدار سنسور proximity کمتر از ۱ باشد و نور کمتر از ۲ باشد باشد ، تشخیص داده میشود که گوشی در جیب قرار دارد.

BackgroundDetectedActivitiesService

پیش از این گفتیم که محدوده ی تغییرات شتاب در هنگام دویدن و راه رفتن متفاوت است و برای هر کدام از این وضعیت ها ، شروط تشخیص یک گام ، متفاوت است. همچنین زمانی که فرد حرکت نمیکند ، نیز ممکن است گوشی با تکان های کوچک دچار تغییرات شتاب شود ، اما نباید شمارش گام صورت گیرد. بنابراین باید با یک مکانیزمی این وضعیت ها را تشخیص دهیم.
این کلاس برای تشخیص اینکه فرد در حرکت است و یا گوشی در حالت سکون قرار دارد مورد استفاده قرار میگیرد. به این منظور از API های google play service استفاده شده است.

DetectedActivitiesIntentService

در این کلاس هر بار با تغییر activity نوع آن شناسایی می شود و آن نوع آن را برودکست میکند. دلیل آن هم این است که ارتباطات آسنکرون باشد و زمانی که StepCounterService قصد داشت تا نوع اکتیویتی را دریافت کند با استفاده از BroadcastReceiver مقدار آن را دریافت کند.
سه مقدار از DetectedActivitiesIntentService برودکست می شود که در یک intent و در قابل Extra در آن ذخیره میشوند. اولین مقدار (on_foot_confidence) احتمال در حال حرکت بودن فرد به صورت پیاده را مشخص میکند ، دومین مقدار (walking_confidence) احتمال راه رفتن را مشخص میکند و آخرین مقدار (running_confidence) احتمال دویدن را مشخص میکند.
در ادامه دو کلاس Publisher و Subscriber توضیح داده میشود که از الگوی برنامه نویسی انتشار و دریافت استفاده میکنند.

Publisher

این کلاس مشخص کننده ی یک آبجکت با قابلیت نشر دارد. دارای یک تابع publish است :هر زمانی مورد استفاده قرار میگیرد که یک اتفاق به خصوصی برای یک آبجکت با این نوع شکل میگیرد و میخواهد که تعدادی آبجکت دیگر را باخبر کند.

Subscriber

این کلاس مشخص کننده ی یک آبجکت با قابلیت دریافت اخبار از آبجکتی دیگر است. در حقیقت هر آبجکت از نوع Publisher دارای تعدادی آبجکت از نوع Subscriber است. دارای یک تابع update است. زمانی که یک آبجکت از نوع Publisher قصد دارد تا Subscriber هایش را از تغییر با خبر کند ، یک تابع با نام update را برای آن آبجکت ها صدا میزند.

RotationVector

این کلاس برای دریافت اطلاعات از سنسور Game Rotation Vector مورد استفاده قرار میگیرد. به این منظور از کلاس SensorListener ارث بری میکند و یک نوع آبجکت Publisher است. بنابراین دارای توابع createSensor() و onSensorChanged() و publish() است. در تابع createSensor() یک آبجکت sensor با تایپ TYPE_GAME_ROTATION_VECTOR ایجاد میکنیم. هر بار که این سنسور داده ی متفاوتی را دریافت کند ، تابع onSensorChanged() فراخوانی میشود. در این تابع مقدار زاویه ی چرخش نسبت به حالت اولیه ی گوشی در جهت x را محاسبه میکند و سپس subscriber خود که آبجکت Turning Detector است را باخبر میکند.

LocalDirection

یک سرویس دیگر از برنامه ، تشخیص چرخش ۳۶۰ و یا ۱۸۰ درجه ای است. به این منظور کلاس LocalDirection را ایجاد کردیم. در این کلاس ابتدا یک sensorManager ایجاد میکنیم و سپس دو آبجکت از نوع TurningDetector ایجاد میکنیم. این آبجکت ها برای تشخیص چرخش به اندازه ی زاویه ی مورد نظر است. زمانی که یکی از این آبجکت ها چرخش را detect کردند ، باید یک الرت در صفحه چاپ شود که به این اندازه چرخش اتفاق افتاده است. به این منظور از کلاس Turning180DegreeAlarm و Turning360DegreeAlarm استفاده کرده ایم که جلو تر در مورد این دو کلاس صحبت میکنیم.

TurningDetector

این کلاس برای تشخیص چرخش مورد استفاده قرار میگیرد. به عنوان ورودی یک sensorManager و زاویه ی چرخش مورد نظر بر اساس رادیان (turningDegree) و میزان زمانی که چرخش در آن باید اتفاق بیفتد (timeThreshold) را دریافت میکند. به در این کلاس نیاز داریم تا زاویه ی فعلی گوشی نسبت به حالت ابتدایی را پیدا کنیم بنابراین یک آبجکت از نوع RotationVector ایجاد میکنیم و آن را start() میکنیم.

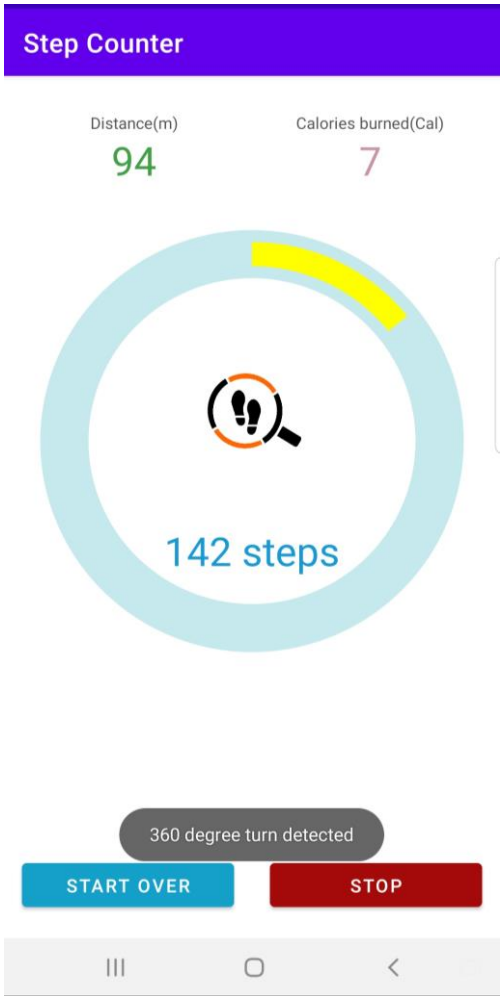
برای مکانیزم کارایی این کلاس با بقیه ی سیستم باید گفت که این کلاس subscriber سنسورهای مورد استفاده ی خود و publisher ایونت تشخیص چرخش است. که در پروژه این کلاس یکی از دو سنسور RotationVector و یا Gyroscope را subscribe میکند تا هرگاه زاویه ی گوشی تغییر کرد از آن مطلع شود. به علاوه کلاس LocalDirection بر روی یک آبجکت از این کلاس که چرخش های زاویه ی مورد نظر را کشف میکند، subscribe کرده و میتواند به نحو دلخواه خود چرخش را به کاربر اطلاع دهد. بنابراین، این کلاس دو اینترفیس Publisher و Subscriber را implement میکند.

تشخیص چرخش به این صورت انجام میپذیرد که ۳۶۰ درجه را به باکت هایی با سایز دلخواه (با قابلیت تغییر در پیاده سازی برای تست بهتر) تقسیم میکنیم، مثلا باکت هایی ۵ درجه ای. بنابراین باکت اول بازه ی [۰ ، ۵) را شامل میشود و .... حال برای هر باکت آخرین زمانی که جهت گوشی در این زاویه بوده است را نگه میداریم. در نتیجه در هر لحظه برای هر باکت آخرین زمانی که جهت گوشی در آن زاویه بوده است را خواهیم داشت. حال هنگامی که آپدیت چرخش گوشی trigger شود، با توجه به جهت جدید گوشی باکت ها را به دو ترتیب ترتیب ساعتگرد و پادساعتگرد چک میکنیم و در صورتی که همه ی باکت های بین زاویه ی کنونی گوشی و باکتی که چرخش از آن شروع شده،زمان ثبت شده برای حضور در آن باکت ها بعد از مدت timeThreshold باشد در این صورت میتوان گفت که همه ی این باکت ها را در چند ثانیه ی قبل دیده ایم و چرخشی صورت گرفته است. برای فهم بهتر مثالی را در نظر بگیرید که به دنبال تشخیص چرخش ۱۸۰ درجه در حداکثر زمان ۳ ثانیه هستیم. هر آپدیت چرخش گوشی توسط سنسور به اطلاع آبجکت ساخته شده ی کلاس TurningDetector میرسد. این کلاس با استفاده از تابع degreeToBucketNumber جهت جدید گوشی را گرفته و شماره ی باکت را به عنوان خروجی میدهد. فرض کنید که این باکت، باکتِ زاویه های [۱۸۵ ، ۱۸۰) باشد. حال برای تشخیص چرخش، که میتواند چرخشی ساعتگرد و یا پادساعتگرد باشد، الگوریتمی را اجرا میکنیم. فرض کنید میخواهیم چرخش ساعتگرد را چک کنیم. در این صورت همه ی باکت های بین [۰ ، ۵) تا [۱۸۵ ، ۱۸۰) باید در ۳ ثانیه ی اخیر دیده شده باشند. پس کافی است که بین این باکت ها iterate کرده و چنانچه همه ی باکت ها آخرین زمان دیده شدنشان در طول ۳ ثانیه ی گذشته بود، متوجه میشویم که چرخش اتفاق افتاده است. برای تشخیص چرخش پادساعتگرد هم همین الگوریتم با این تفاوت که باکت هایی که باید انتظار داشته باشیم در آن ها بوده ایم، متفاوت اند. بعد از تشخیص چرخش این آبجکت به تمام subscriber هایش اطلاع میدهد

که چرخش اتفاق افتاده است و بعد از آن لیست زمان چرخش در باکت ها را پاک میکند. در پایان هر آپدیت چرخش اطلاعات آخرین چرخش را فارغ از اینکه چرخش در آن اتفاق افتاده است یا نه ذخیره میکنیم. حال پارامتر ها و حالت های مختلف را شرح میدهیم.
برای استفاده ی بهینه از حافظه ۳۶۰ درجه را به باکت های کوچکتری شکسته ایم که اولین نکته ی نگران کننده دقت چرخش است. که با در نظر گرفتن باکت های ۵ درجه ای دقت لازم به دست می آید(در هنگام آزمایش تفاوت حسی ای بین باکت های پنج درجه ای و یک درجه ای و حتی نیم درجه ای حس نشد، پارامتر دیگری که در این دقت میتواند موثر باشد سرعت سنسور است که با کم کردن باکت ها برای جلوگیری از خطای رد شدن باکت یک زاویه باید فرکانس سنسور را هم افزایش داد، بنابراین در اینجا با رعایت نسبت خوبی از دقت به هزینه توانسته ایم تشخیص چرخش را به خوبی پیاده سازی کنیم) استفاده از باکت ها در پیاده سازی کمک کرده است که حجم دیتایی که در صورت غیر باکت کردن زاویه ها نگه داریم و از آن مهمتر عملکرد سی پی یو برای تشخیص چرخش در دیتای زیاد است. زیرا در صورت غیر باکت کردن دیتا ها باید زمان و درجه ی همه ی دیتا های سنسور از زمان حاضر تا timeTreshold ثانیه ی قبل را ذخیره کنیم و در هر بار آپدیت سنسور باید همه ی این لیست را برای تشخیص چرخش پیمایش کنیم. مثلا چنانچه فرکانس آپدیت سنسور ۱۰۰۰ باشد باید در هر آپدیت سنسور این دیتا ذخیره میشود و در نتیجه حداقل ۳۰۰۰ دیتا خواهیم داشت که در هر آپدیت سنسور باید آنها را پیمایش کنیم. بنابراین در یک ثانیه باید در order حدود ۱۰۰۰ * ۳۰۰۰ دستور اجرا کنیم که از لحاظ پرفورمنسی الگوریتم مناسبی نخواهد بود.
در نهایت دو instance از این کلاس برای تشخیص چرخش های ۳۶۰ درجه در زمان ۷ ثانیه و چرخش های ۱۸۰ درجه در زمان ۳ ثانیه در کلاس LocalDirection ساخته و استفاده شده اند.
برای اعلام این چرخش باید یک الرت در صفحه ی گوشی ایجاد شود ، این الرت را کلاس Turning180DegreeAlarm و Turning360DegreeAlarm انجام میدهند که برای انتشار این خبر باید آبجکت از نوع TurningDetector را از نوع Publisher و آبجکت از نوع Turning360DegreeAlarm و Turning180DegreeAlarmرا از نوع Subscriber تعریف کنیم و آن را در لیست Subscriber های TurningDetector قرار دهیم. زمانی که این آبجکت تابع publish() را فراخوانی میکند ، در تابع update این آبجکت ها یک الرت از نوع Toast ایجاد شده و پیغام چرخش به اندازه ی TurningDegree نمایش داده میشود.

Turning360DegreeAlarm / Turning180DegreeAlarm

این کلاس ها از Subscriber های آبجکت های از نوع TurningDetector هستند. زمانی که یک چرخش به اندازه ی ۳۶۰ یا ۱۸۰ درجه صورت گیرد ، تابع publish() این آبجکت فراخوانی میشود. در این تابع ، تابع آپدیت subscriber ها فراخوانی میشود و در آن یک الرت از نوع ویجت Toast ایجاد میشود. در تصویر زیر یک نمونه الرت چرخش ۳۶۰ درجه ای نمایش داده شده است:

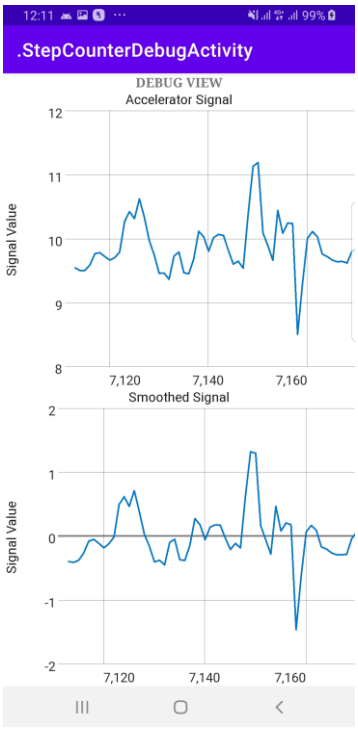


GyroOrientation

این کلاس در صورت موجود بودن سنسور Game Rotation Vector از این سنسور و در غیر این صورت از سنسور Gyroscope استفاده می کند. در صورت موجود نبودن هیچ یک از این سنسور تاثیری روی جهت یابی برنامه نخواهد داشت.
در صورت استفاده از سنسور Gyroscope، ابتدا جهت گیری اولیه را از قطب نما پیدا می کنیم و سپس در هر بازه زمانی میزان چرخش تلفن همراه را حول محور z محاسبه می کنیم و به جهت اولیه اضافه می کنیم. نحوه محاسبه چرخش هم به این صورت است که سنسور داده ای که بر می گرداند سرعت چرخش می باشد که ما این مقدار را در مدت بازه زمانی اش ضرب می کنیم و مقدار زاویه چرخش بدست می آید.
در صورت استفاده از سنسور Game Rotation Vector سنسور به ما میزان زاویه چرخش را بر می گرداند که ما با آن را به علاوه زاویه جهت گیری اولیه ای که از قطب نما بدست آورده ایم می کنیم. این سنسور ممکن است در هنگام شروع مقدار اولیه صفر را بر نگرداند که در این صورت باید این مقدار اولیه را نیز هر بار از زاویه خروجی از سنسور کم کنیم.
برای این که زاویه جهت گیری اولیه که از سنسور قطب نما می گیریم به اندازه کافی استیبل شده باشد هنگام شروع، ما مقداری زمان متناسب با آلفا تعریف شده برای فیلتر Low Pass صبر می کنیم.
نکته مهم آخر در پیاده سازی این کلاس این هست که زاویه خروجی این کلاس را مشابه به زاویه قطب نما پیاده سازی کنیم تا با هم مشکل تطابق نداشته باشند.

StepCounterDebugActivity

این اکتیویتی برای مشاهده مقادیر سنسور شتاب سنج در لحظه برای دیباگ و تنظیم پارامتر های گام شمار استفاده می شود. برای پیاده سازی آن نیز لیست داده های مقدار شتاب کلی دستگاه قبل و بعد از حذف شتاب گرانشی را به کتابخانه GraphView میدهیم تا همانطور که در شکل زیر می بینید دو نمودار خطی از آن داده ها را تولید کنیم.



7- تست عملکرد

- تست دقت گام شمار در دست

  - طرح تست

  پیاده روی یک مسیر حدود ۱۰ دقیقه ای در حالی که تلفن همراه در دست کاربر قرار دارد.

  - نحوه اجرای تست

  مقایسه نتایج بدست آمده از تعداد گام ها بین الگوریتم پیاده سازی شده، سنسور اندروید و برنامه Samsung Health.

  - نتایج تست‌های انجام شده

  سنسور اندروید: ۷۰۶

  برنامه سامسونگ: ۸۲۸

  گام شمار پیاده سازی شده: ۸۲۵

  - تحلیل نتایج

  سنسور اندروید برای شمارش گام به هنگامی که تلفن همراه در دست کاربر قرار دارد، تعدادی از گام های کاربر را نمی شمارد و به همین علت گام شمار دقیق تر شرکت سامسونگ استفاده کردیم.

  خطا گام شمار پیاده سازی شده نسبت به گامش شمار سامسونگ ۰,۳۶۲۳% بود که بسیار خطا نا چیزی می باشد.

- تست دقت گام شمار در جیب

  - طرح تست

  پیاده روی یک مسیر حدود ۱۰ دقیقه ای در حالی که تلفن همراه در جیب کاربر قرار دارد.

  - نحوه اجرای تست

  مقایسه نتایج بدست آمده از تعداد گام ها بین الگوریتم پیاده سازی شده و برنامه Samsung Health.

  - نتایج تست‌های انجام شده

  برنامه سامسونگ: ۱۰۰۹

  گام شمار پیاده سازی شده: ۸۶۹

  - تحلیل نتایج

  خطا گام شمار پیاده سازی شده نسبت به گامش شمار سامسونگ ۱۳,۸۷% بود که خطا قابل قبولی می باشد.

- تست تشخیص چرخش ۱۸۰ و ۳۶۰ درجه ای

  - طرح تست

  چرخش با گوشی در دست و چرخش های ۱۸۰ و ۳۶۰ درجه ای و چرخش های نزدیک به ۱۸۰ درجه - غیرفعال سازی و تست هر یک به صورت جداگانه و تست در حالت فعال سازی هر دو - تاثیر منفی bucketize کردن در دقت - تاثیر سایز باکت ها در دقت سنسور

  - نحوه اجرای تست

  مشاهده و بررسی عملکرد در حالاتی که چرخش به صورت کامل صورت گرفته و حرکتی شبیه چرخش با تغییر پارامتر ها

- نتایج تست‌های انجام شده

در هر دو زاویه ی چرخش دقت عالی بود. در الگوریتم باکت کردن در با کت سایز های کوچک(۱ درجه) مطلقا تفاوتی دیده نشد و در باکت سایز بزرگ (۵ درجه) جز حالاتی که با تلاش فراوان که نقطه کور های الگوریتم شبیه سازی شود باز هم تفاوتی دیده نشد (البته در باکت با سایز ۱۰ درجه ای افت دقت قابل درک بود). نتیجه ی تست جداگانه و تست هر دود detector با هم مثبت بود و هیچ کدام خللی بر دیگری وارد نمیکرد و عملکرد هر یک نیز در حالت با هم یکسان بود (وجود هر دو باعث ایجاد دیلی هایی در دیگری میشود) با توجه به سایز پنج درجه ای باکت ها با کاهش فرکانس سنسور خطای قابل درکی در تشخیص چرخش دیده نشد اما با توجه به عملکرد مناسب فرکانس TYPE_GAME_ROTATION_VECTOR برای سنسور در نظر گرفته شد.

- تست تشخیص مسیر طی شده

- طرح تست

پیاده روی یک مسیر حدود ۱۰ دقیقه ای در حالی که تلفن همراه در دست کاربر قرار دارد.

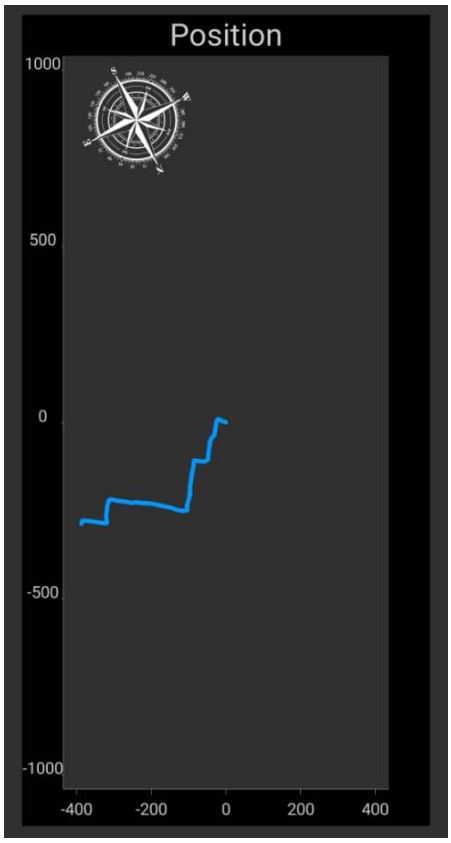
- نحوه اجرای تست

مشاهده و مقایسه مسیر تولید شده توسط برنامه بر روی نقشه واقعی محل پیاده روی از نقطه شروع تا نقطه پایان .

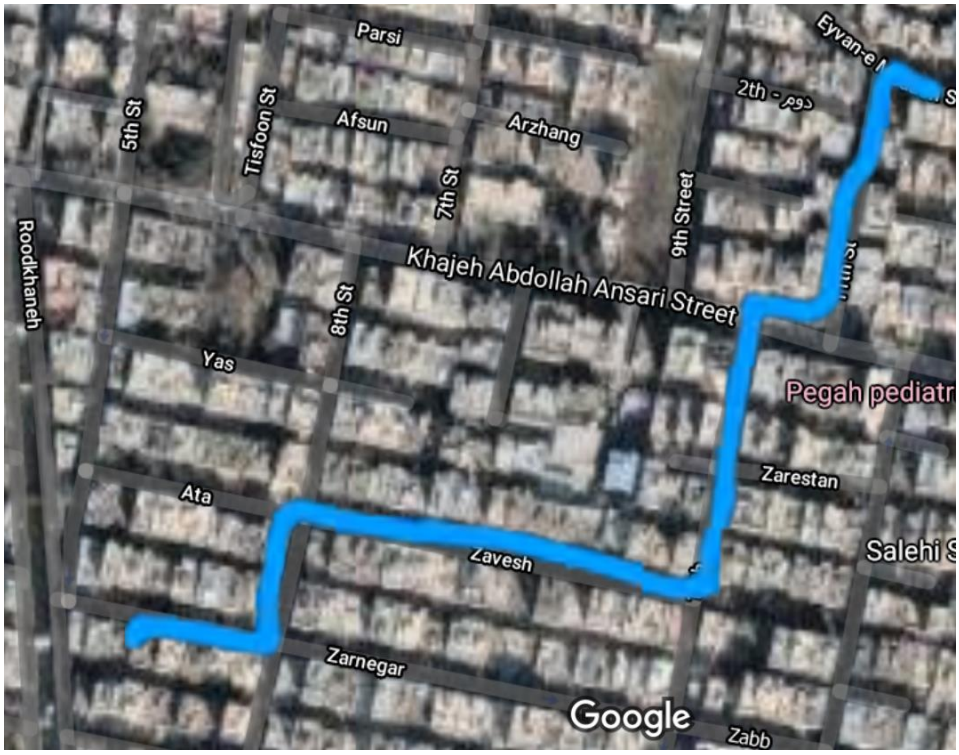
- نتایج تست‌های انجام شده

مسیر اصلی: ۷۰۰ متر

تصویر اصلی مسیر محاسبه شده:



مسیر محاسبه شده بر روی نقشه: ۶۸۹ متر



- تحلیل نتایج

مسیر کلی محاسبه شده تقریبا به خوبی منطبق بر کوچه های نقشه شده، به جز یک مورد حرکت به سمت جنوب در ابتدا مسیر که بنظر می رسد مقداری زود تر از مسافت واقعی بر روی نقشه پیچیدن کاربر کشیده شده است.
مسافت طی شده ٦٨٩ متر در مقابل ٧٠٠ متر تخمین زده شده توسط نقشه گوگل قرار دارد که حدود ١,٥٧% درصد خطا دارد که بسیار ناچیز می باشد.

- تست تشخیص مقدار جابجایی

- طرح تست

پیاده روی یک مسیر حدود ١٠ دقیقه ای در حالی که تلفن همراه در جیب کاربر قرار دارد.

- نحوه اجرای تست

مقایسه مقدار جابجایی بین نقطه شروع و پایان و مقایسه آن با جابجایی واقعی.

- نتایج تست‌های انجام شده

مختصات شروع و پایان:
35.745303, 51.457606
35.742870, 51.453285
جابجایی واقعی: (٣٩٠,٨٤,٢٦٩,٩٥) متر
الگوریتم پیاده سازی شده: (٣٨٩,٧,٢٨٣,٨) متر
فاصله واقعی: ٤٧٥,٠١ متر
فاصله محاسبه شده: ٤٨٢,٠٨ متر

- تحلیل نتایج

زاویه Bearing محاسبه شده از مبدا به مقصد در واقعیت حدود ٣٤,٦٣ درجه می باشد و زاویه محاسبه شده ٣٦,٠٦ درجه می باشد.
الگوریتم پیاده سازی شده برای زاویه Bearing محاسبه شده ٣,٩٦% درجه خطا داشت و فاصله محاسبه شده ١,٤٦% خطا داشت.
درکل بعد از حدود ٧٠٠ متر پیاده روی خطایی در حدود ١٣,٨٩ متر الگوریتم پیاده سازی شده داشت که با توجه به این که نقاط شروع و پایان بر روی نقشه انتخاب شده بودند و خودشان دارای خطایی در حد چند متر می باشند نتیجه مطلوبی می باشد.

8- پاسخ به سوالات طراحی و تایید شده در پروپوزال

۱- با چه فرکانسی خواندن سنسور انجام شود؟

از آن جایی که سرعت حرکت سریع تر از متوسط برای پیاده روی حدود ٦ کیلومتر بر ساعت است که حدودا می شود ۰,٦ ثانیه برای هر گام و با توجه به قضیه نمونه برداری نایکوئیست-شانون، حداقل فرکانس نمونه برداری باید بزرگتر از دو برابر فرکانس بیشینه سیگنال اصلی باشد که حدودا می شود ۰,۳ ثانیه. برای ایجاد مقداری بازه اطمینان فرکانس نمونه برداری از سنسور شتاب سنج برای گام شمار را رو ۰,۲ ثانیه یا ۲۰۰ میلی ثانیه در نظر میگیریم.

سنسور Step Detector هم مشابه توجیه بالا فرکانس ۲۰۰ میلی ثانیه برایش مناسب می باشد.

از آنجایی که داده های سنسور های مغناطیس سنج و شتاب سنج را برای قطب نما از فیلتر Low Pass با مقدار آلفا ۱٦/۱ عبور می دهیم و با توجه به این که باید حداقل یک نمونه محاسبه جهت گیری تلفن همراه بین دو گام داشته باشیم تا خطا ضربه هنگام گام را حذف کنیم، ایده آل است که فرکانس نمونه برداری ما ۱٦*۲ یعنی ۳۲ برابر فرکانس گام باشد. این فرکانس حدودا معادل ۱۸,۷۵ میلی ثانیه می باشد. ما مقدار ۲۰ میلی ثانیه را که نزدیک مقدار ایده آل می باشد برای این دو سنسور برای قطب نما انتخاب کردیم.

برای سنسور های شتاب سنج، Proximity و نور برای تشخیص موقعیت تلفن همراه اصلا نیازی به فرکانس بالا ندارند و با حداقل فرکانس نمونه برداری خروجی مناسب را می دهند که ما مقدار ۲۰۰ میلی ثانیه را برایشان انتخاب کردیم.

برای سنسور های ژیروسکوپ و Game Rotation Vector فرکانس نمونه برداری ۲۰۰ هرتز برای دقت ایده آل پیشنهاد شده بود در مقالات، ولی به بالا بودن این فرکانس ما به فرکانس ۵۰ هرتز یا همان تاخیر ۲۰ میلی ثانیه بسنده کردیم، زیرا توانستیم نتیجه مطلوب را در این مقدار تاخیر بدست بیاوریم و نیازی به فرکانس نمونه برداری بیشتر برای این سنسور ها نبود.

۲- آیا سنسور های انتخاب شده مناسب اند یا گزینه های جایگزین دیگری وجود دارند؟

به طور کلی برنامه ما فقط نیاز به سنسور های شتاب سنج و مغناطیس سنج برای کار کردن دارد. هر چند در صورت موجود بودن سنسور های بیشتر برای افزایش دقت محاسبات ما از سنسور های بیشتری استفاده کردیم.

این سنسور های اضافه عبارتند از ژیروسکوپ و Game Rotation Vector برای کمک به جهت یابی. سنسور Proximity و نور برای تشخیص موقعیت تلفن همراه. سنسور Step Detector به عنوان جایگزین گام شمار.

به غیر از این سنسور ها در صورت موجود بودن سنسور Gravity می توانستیم محاسبه مقدار شتاب گرانشی را از برنامه مان حذف کنیم و به جای آن از این سنسور استفاده کنیم. دو مورد از کاربرد های این سنسور را می توان به محاسبه حذف شتاب گرانشی از شتاب بدست آمده در الگوریتم گام شمار و به بدست آوردن زاویه تلفن همراه در هنگام تشخیص موقعیت دستگاه اشاره کرد.

۳- در چه تایم فریمی یا با چه فرکانسی مسیر طی شده را پردازش کنیم؟

در صورت ذخیره مقادیر سنسور ها می توان تمام محاسبات مسیر را به طور غیر بی درنگ انجام داد، ولی با توجه به اینکه هدف این پروژه محاسبه بی درنگ مسیر طی شده می باشد، به غیر از محاسباتی که با هر بار تغییر در مقادیر سنسور ها انجام می شود بررسی وقوع هر گام با همان فرکانس محاسبه شده یعنی هر ۲۰۰ میلی ثانیه یک بار انجام می شود. محاسبه جهت حرکت نیز برای اینکه باید بیشتر از دو برابر فرکانس تشخیص گام باشد با فرکانس ۷۰ میلی ثانیه یکبار انجام می شود. محاسبات مربوط به رابط کاربری نیز هر ۱ ثانیه یکبار انجام می شود تا تاخیر بیش از اندازه برای کاربر نداشته باشند.

٤- دقت محاسبه مسیر در مقایسه با GPS به چه میزان است؟

مقایسه دقت مسیر محاسبه شده با GPS را می توانید در تست های عملکرد مشاهده کرد. ولی دقت این دو روش از نظر ذاتی با هم تفاوت دارند زیرا این روش محاسبه مسیر باعث می شود خطا ها در طول زمان جمع شوند و مانند GPS صرفا دارای یک خطای محاسباتی ساده نیست که بتوان این دو را با هم مقایسه کرد. در نتیجه با گذر زمان دقت این روش دائما کاهش پیدا می کند در حالی که دقت GPS تغییری در طول زمان نخواهد داشت.

9- مراجع

تشخیص مسیر طی شده با استفاده از سیگنال های خارجی:

https://github.com/I-Hope-Peace/In-outdoorSeamlessPositioningNavigationSystem

https://github.com/andreyukD/BLE-Beacon-Indoor-Positioning

تشخیص مسیر طی شده با استفاده از گام شمار:

https://www.semanticscholar.org/paper/AnDReck-%3A-Positioning-Estimation-using-Pedestrian-Sim%C3%B5es/85577e4ff29997eef0935169c55d934d5257e80e

https://github.com/nisargnp/DeadReckoning

https://github.com/nesl/dr_client

تشخیص مسیر طی شده با دانش قبلی نسبت محیط:

https://github.com/HyHend/SPSAM

تشخیص گام:

https://www.semanticscholar.org/paper/A-Novel-Walking-Detection-and-Step-Counting-Using-Kang-Huang/7e70085d2f44a0739e4f04b7353455c22ce5a8f7

https://github.com/lisymart/Pedometer

https://github.com/google/simple-pedometer

https://github.com/isibord/StepTrackerAndroid

https://github.com/jeeshnair/ubicomp

https://github.com/freddiejbawden/stepz

https://hub.packtpub.com/step-detector-and-step-counters-sensors/

تشخیص جهت حرکت:

https://www.researchgate.net/publication/266657734_I_am_a_smartphone_and_I_can_tell_my_user's_walking_direction

https://github.com/yogur/android-compass

سایر منابع:

1. https://developer.android.com/guide/topics/sensors/sensors_motion
2. https://developer.android.com/reference/android/hardware/SensorManager
3. https://www.youtube.com/watch?v=C7JQ7Rpwn2k
4. https://github.com/KalebKE/FSensor
5. https://www.researchgate.net/publication/309502392_SensingKit_Evaluating_the_Sensor_Power_Consumption_in_iOS_Devices
6. https://github.com/Sainathhiwale/ActivityRecognition