Lecture Overview

- Week 2: Intro to Kotlin
- Week 3: Lifecycle and Layouts

- *Week 4: (Architectural) Design Patterns*
- *Week 5: Persisting Data*
- *Week 6: Architectural Components*
- ***Week 7: Sensing in Android***

- Week 8: Background and Foreground Services
- Week 10: Context
- Week 11: Releasing Apps
- Week 12: Guest lecture

COM4510/6510
Software Development for Mobile Devices

Lecture 7: Sensing and brief intro to location

Temitope (Temi) Adeosun
The University of Sheffield
t.adeosun@sheffield.ac.uk

Lecture Overview/Learning Objectives

- Sensors overview
  - An example
- Location awareness
  - Issues and warnings
- Lab tutorial :
  - Sensing

Learning Objectives: At the end of this week's lecture and labs, you should be able to:

- Identify, and describe the key components of the Android sensor framework and their uses
- Implement sensor monitoring using the Android sensor framework
- Recognize the key elements of using the Google play service for location awareness
- Implement a basic location aware Android app

Sensors Overview

- Most Android-powered devices have built-in sensors that measure **motion**, **position**, and various **environmental conditions**
- These sensors are
  - capable of providing raw data with **high precision and accuracy**
  - useful to monitor three-dimensional device movement or positioning
  - useful to monitor changes in the ambient environment near a device

https://developer.android.com/guide/topics/sensors/sensors_overview

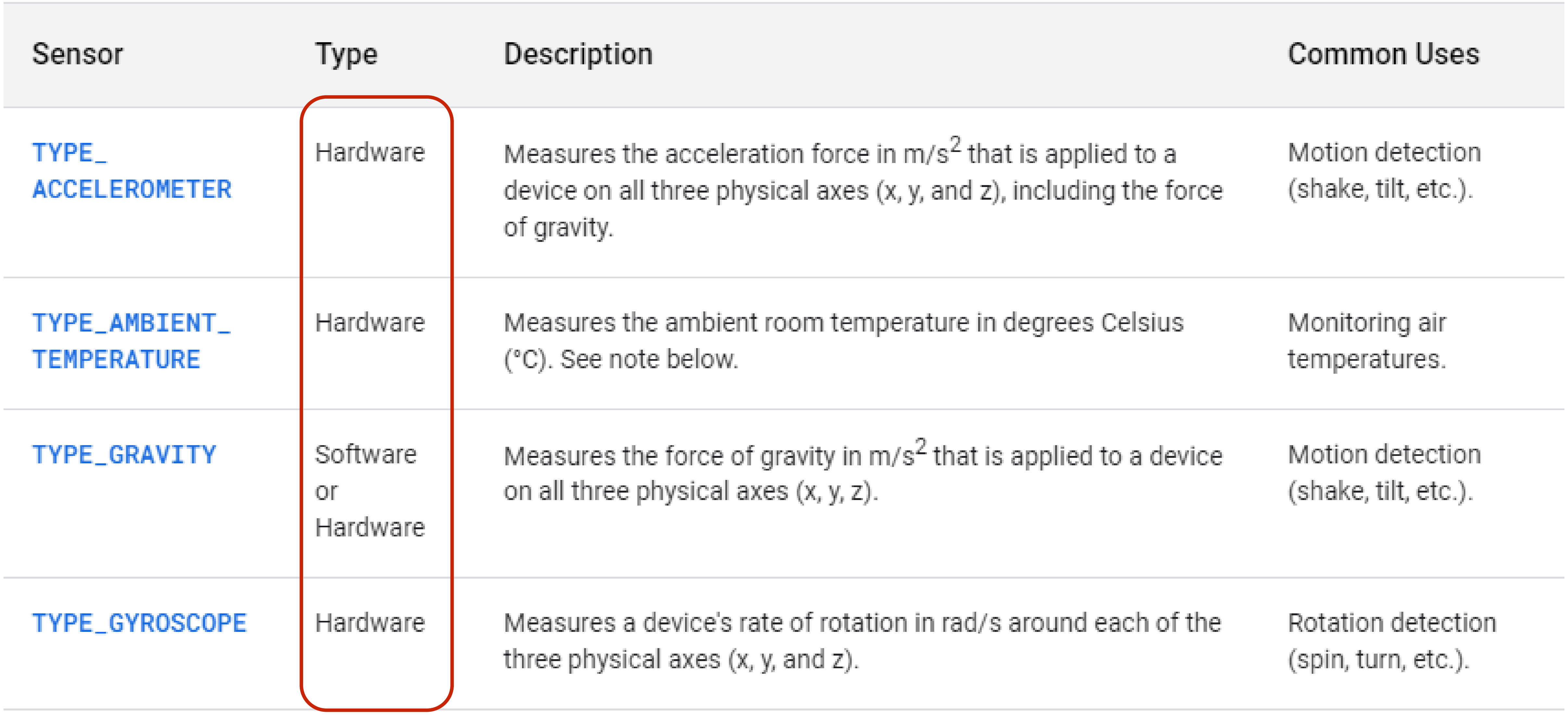
The University Of Sheffield.

Sensor Types

- The Android platform supports three broad categories of sensors:
  - **Motion sensors**: measure acceleration forces and rotational forces along three axes (accelerometers, gravity sensors, gyroscopes, and rotational vector sensors)
  - **Environmental sensors**: measure various environmental parameters
    - ambient air temperature and pressure, illumination, and humidity (barometers, photometers, and thermometers)
  - **Position sensors**: measure the physical position of a device (orientation sensors and magnetometers)

Hardware and Software Sensors

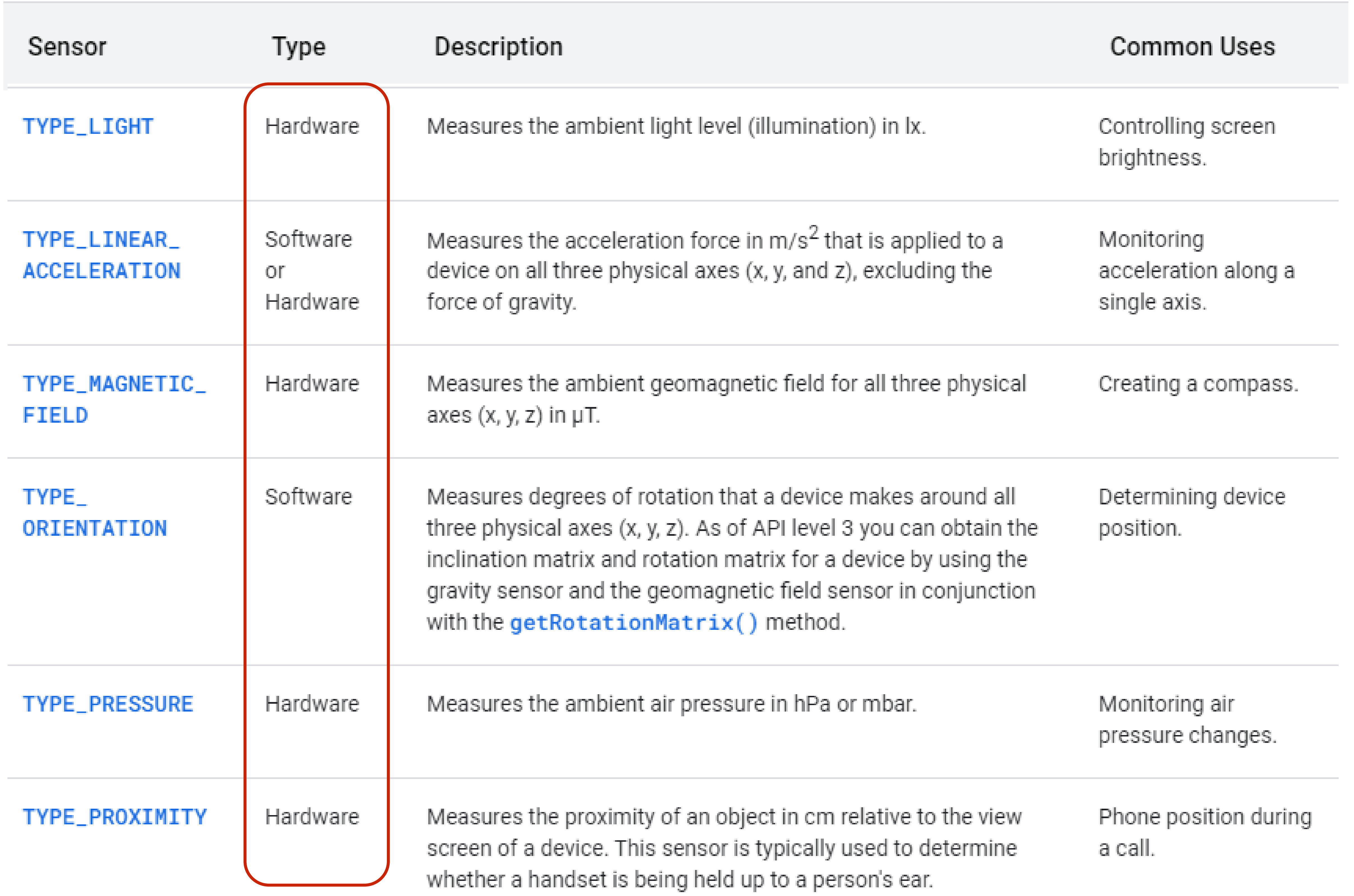
- Note:
  - Some of these sensors are **hardware-based** (e.g. accelerometer, photometer) and some are **software-based** (gravitational, orientation)
  - You cannot tell the difference (you use them in the same way)



Sensor	Type	Description	Common Uses
TYPE_ ACCELEROMETER	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_AMBIENT_ TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).

https://developer.android.com/guide/topics/sensors/sensors_overview#sensors-intro

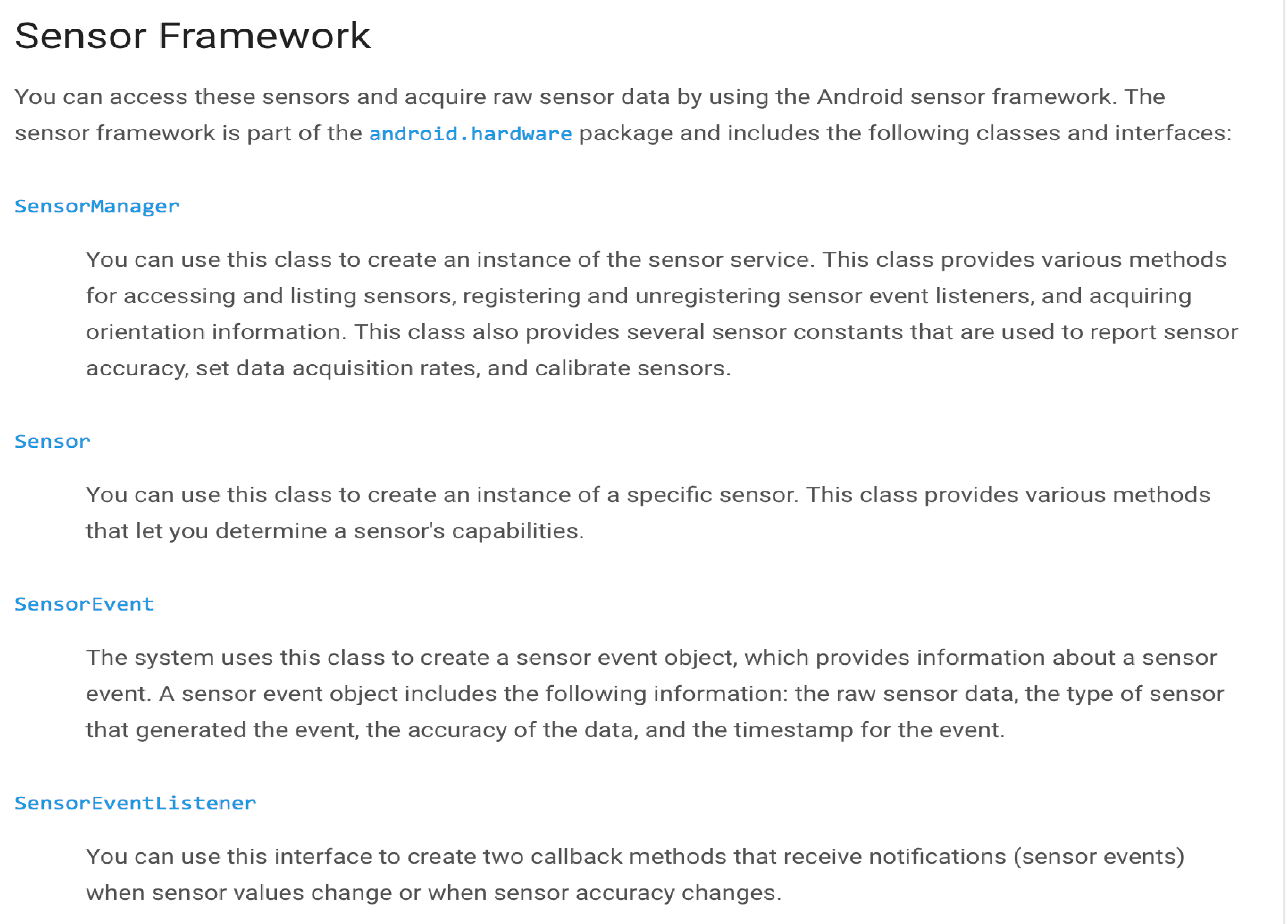
Hardware and Software Sensors



Sensor	Type	Description	Common Uses
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ ACCELERATION	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
TYPE_MAGNETIC_ FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.
TYPE_ ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the `getRotationMatrix()` method.	Determining device position.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

https://developer.android.com/guide/topics/sensors/sensors_overview#sensors-intro

Android Sensor Framework



Sensor Framework

You can access these sensors and acquire raw sensor data by using the Android sensor framework. The sensor framework is part of the `android.hardware` package and includes the following classes and interfaces:

`SensorManager`

> You can use this class to create an instance of the sensor service. This class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.

`Sensor`

> You can use this class to create an instance of a specific sensor. This class provides various methods that let you determine a sensor's capabilities.

`SensorEvent`

> The system uses this class to create a sensor event object, which provides information about a sensor event. A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.

`SensorEventListener`

> You can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

The University Of Sheffield.

Using sensors

Only needed outside activity/fragment

1

Instantiate SensorManager

```
val sensorManager = context.getSystemService(Context.SENSOR_SERVICE) as SensorManager
```

2

Get **Sensor** instance(s), if sensor manager is available

```
val sensor = sensorManager?.getDefaultSensor(Sensor.TYPE_LIGHT)
```

Handle different sensor config:
Detect sensors at runtime/
Filter for phone with specific sensors on Google play

- `if (sensor != null){}`
- `sensor?.let{}`

```
val sensorList = sensorManager?.getSensorList(Sensor.TYPE_ALL)
```

```
<uses-feature android:name="android.hardware.sensor.light"
  android:required="true" />
```

3

Monitor sensor events – by creating (see next slide) and registering (shown to the right) **SensorListener**; if sensor manager is available

Register targeted listener
a) containing class as sensor listener
b) sensor listener object

a

```
sensorManager?.let { it.registerListener(this, sensor,
SensorManager.SENSOR_DELAY_NORMAL) }
```

b

```
sensorManager?.let { it.registerListener(barometerEventListener, sensor,
        SensorManager.SENSOR_DELAY_NORMAL) }
```

4

Stop monitoring sensor event

```
sensorManager?.unregisterListener(this)

sensorManager?. unregisterListener(barometerEventListener)
```

Using sensors

b

sensor listener object

```
barometerEventListener  = object : SensorEventListener {

        override fun onSensorChanged(event: SensorEvent) {
                // A light sensor returns a single value.

                // Many sensors return 3 values for x, y, z axis.

        }
        override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
                // Do something here if sensor accuracy changes.
        }

sensorManager?.let {
    it.registerListener(barometerEventListener, sensor,
        SensorManager.SENSOR_DELAY_NORMAL) }
```

3.1

SensorEventListener interface (Asynchronous) callback implementation.
SensorEvent object contains the sensor data at the time event was raised.

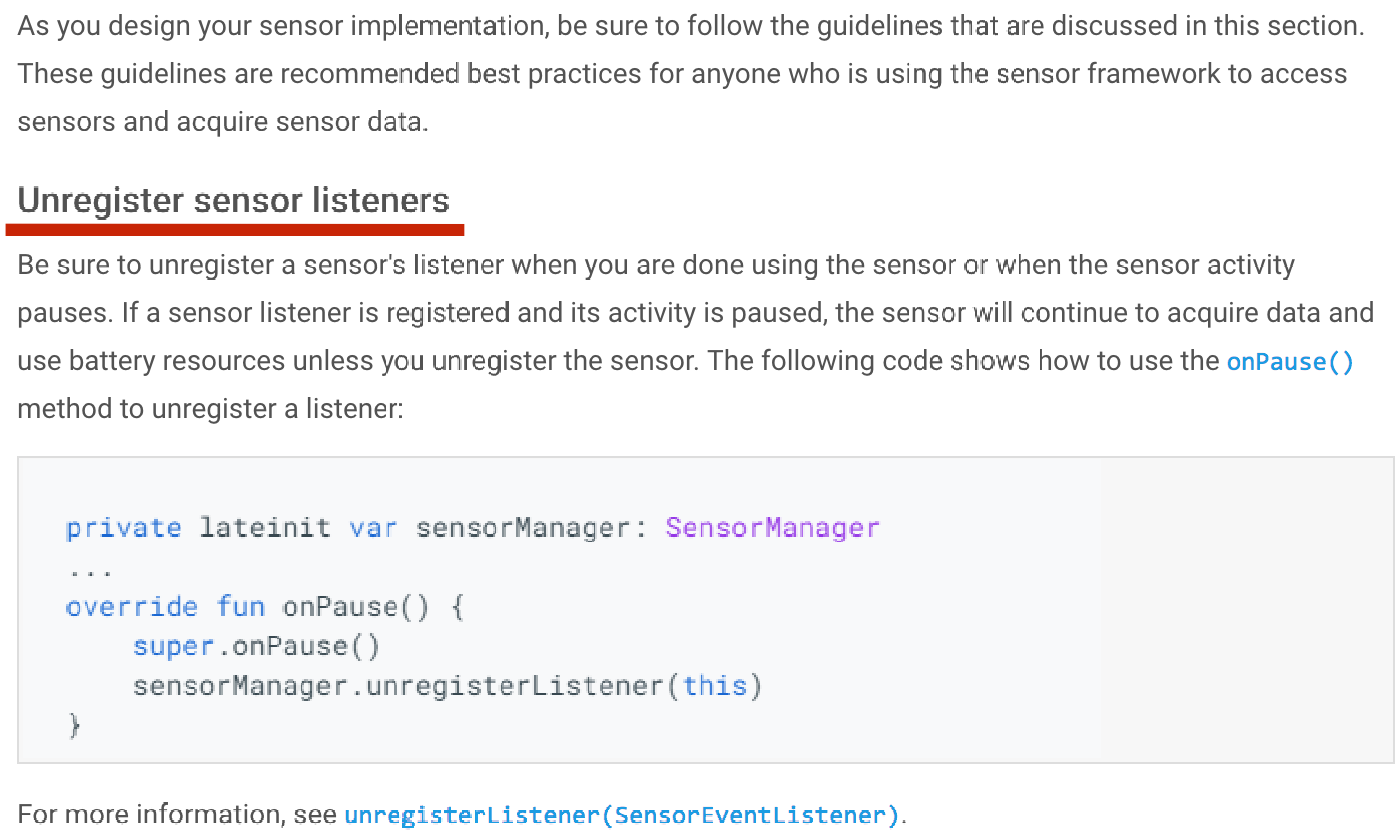
```
class MyActivity : SensorEventListener {
            ...
        sensorManager?.let {
            it.registerListener(this, sensor,
                SensorManager.SENSOR_DELAY_NORMAL) }

        override fun onSensorChanged(event: SensorEvent) {
                // A light sensor returns a single value.
                // Many sensors return 3 values for x, y, z axis.

        }
        override fun onAccuracyChanged(sensor Sensor?, accuracy: Int) {
                // Do something here if sensor accuracy changes.
        }
        }
```

a

Containing class as sensor listener

Best Practices

- Always register



As you design your sensor implementation, be sure to follow the guidelines that are discussed in this section. These guidelines are recommended best practices for anyone who is using the sensor framework to access sensors and acquire sensor data.

Unregister sensor listeners

Be sure to unregister a sensor's listener when you are done using the sensor or when the sensor activity pauses. If a sensor listener is registered and its activity is paused, the sensor will continue to acquire data and use battery resources unless you unregister the sensor. The following code shows how to use the `onPause()` method to unregister a listener:

```
private lateinit var sensorManager: SensorManager
...
override fun onPause() {
    super.onPause()
    sensorManager.unregisterListener(this)
}
```

For more information, see `unregisterListener(SensorEventListener)`.

Best Practices

- On Android 9 (API 28) and higher, gather sensor data in the foreground (background sensor gathering may continue after activity is paused). See more here.
- Avoid using deprecated methods or sensor types
- Verify sensors before you use them

You could require the availability of a sensor through the Manifest file

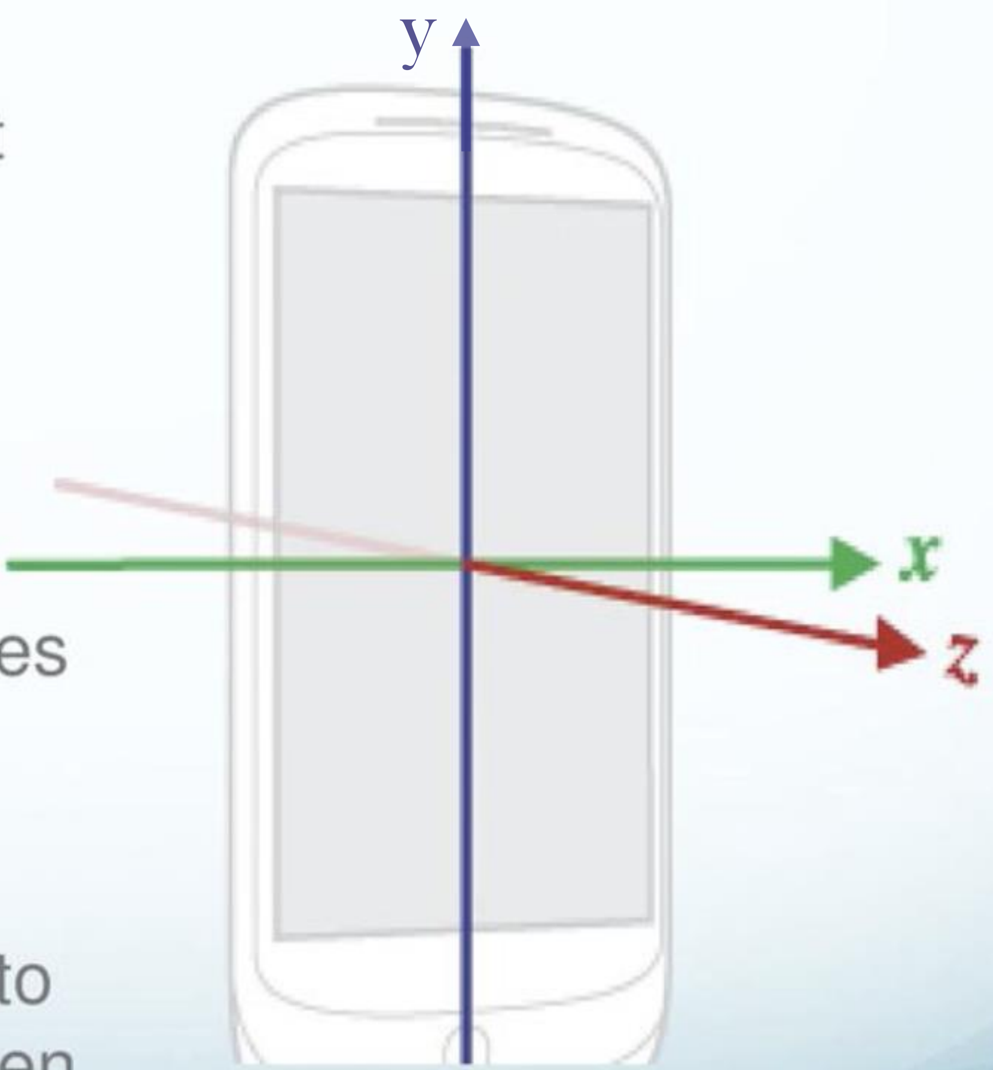
```
<uses-feature android:name="android.hardware.sensor.light"
    android:required="true" />
```

Only use if your app cannot run without the sensor.

- Choose sensor delays carefully
  - API 31 (Android 12 and higher), places rate limit on some motion and position sensors (to protect collecting potentially sensitive information about user). See more here.

Sensor Coordinate System

- Default orientation
  - X axis – horizontal pointing right
  - Y axis – vertical pointing up
  - Z axis – pointing toward the outside of the screen face.
- Coordinate system does not change when orientation changes
- You can use getOrientation() to determine screen rotation and use remapCoordinateSystem() to map sensor coordinates to screen coordinates.



The University Of Sheffield.

A simple MVC example (Lab uses MVVM)

```
class LightActivity: AppCompatActivity(), SensorEventListener {                    3

    private var sensorManager: SensorManager? = null                                1
    private lateinit var lightSensor: Sensor                                        2

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val binding = ActivityLightBinding.inflate(layoutInflater)
        setContentView(binding.root)

        sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        lightSensor = sensorManager?.getDefaultSensor(Sensor.TYPE_PRESSURE)!!       1.1, 2.1
    }

    override fun onSensorChanged(event: SensorEvent?) {                             3.1
        event?.let{
            Toast.makeText(
                this@LightActivity,
                "Sensor value: ${it.values[0].toString()}",
                Toast.LENGTH_LONG)
                .show()
        }
    }

    override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {               3.1
        // Do something when sensor accuracy changes
    }

    override fun onResume() {
        super.onResume()
        sensorManager?.let{                                                         3
            it.registerListener(this@LightActivity, lightSensor, SensorManager.SENSOR_DELAY_NORMAL)
        }
    }

    override fun onPause() {
        super.onPause()
        sensorManager?.unregisterListener(this)                                     4
    }

}
```

Time in Android

- Generally expressed with a java Date type
- However in sensors (using sensorEvent.timestamp) it is returned in
  - nanoseconds (1/1,000,000,000 of a second)
  - elapsed time since boot
  - can concert to milliseconds: nanoseconds_value/1,000,000
- Converting msecs to dates:
  - new Date(miliseconds)
- Converting date to msecs
  - new Date().getTime()
  - since 1.1.1970 (standard Unix time or epoch)
- Converting msecs to Strings

```
val date = Date(miliseconds)
val formatter: DateFormat = SimpleDateFormat("HH:mm:ss")
formatter.timeZone = TimeZone.getTimeZone("UTC")
val dateString = formatter.format(date)
```

Working with date/time can get complex. See documentation

The University Of Sheffield.

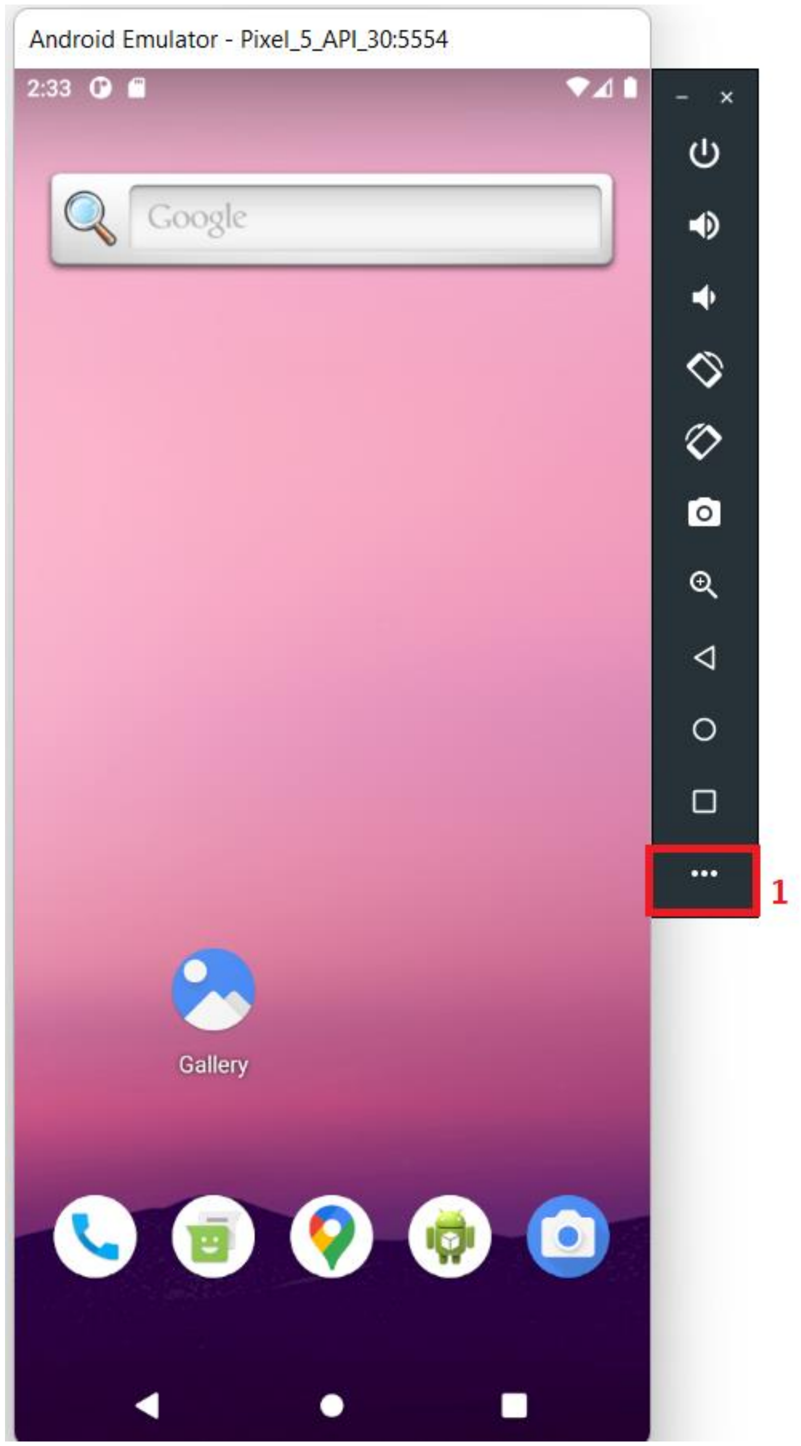
Time since last rebooted

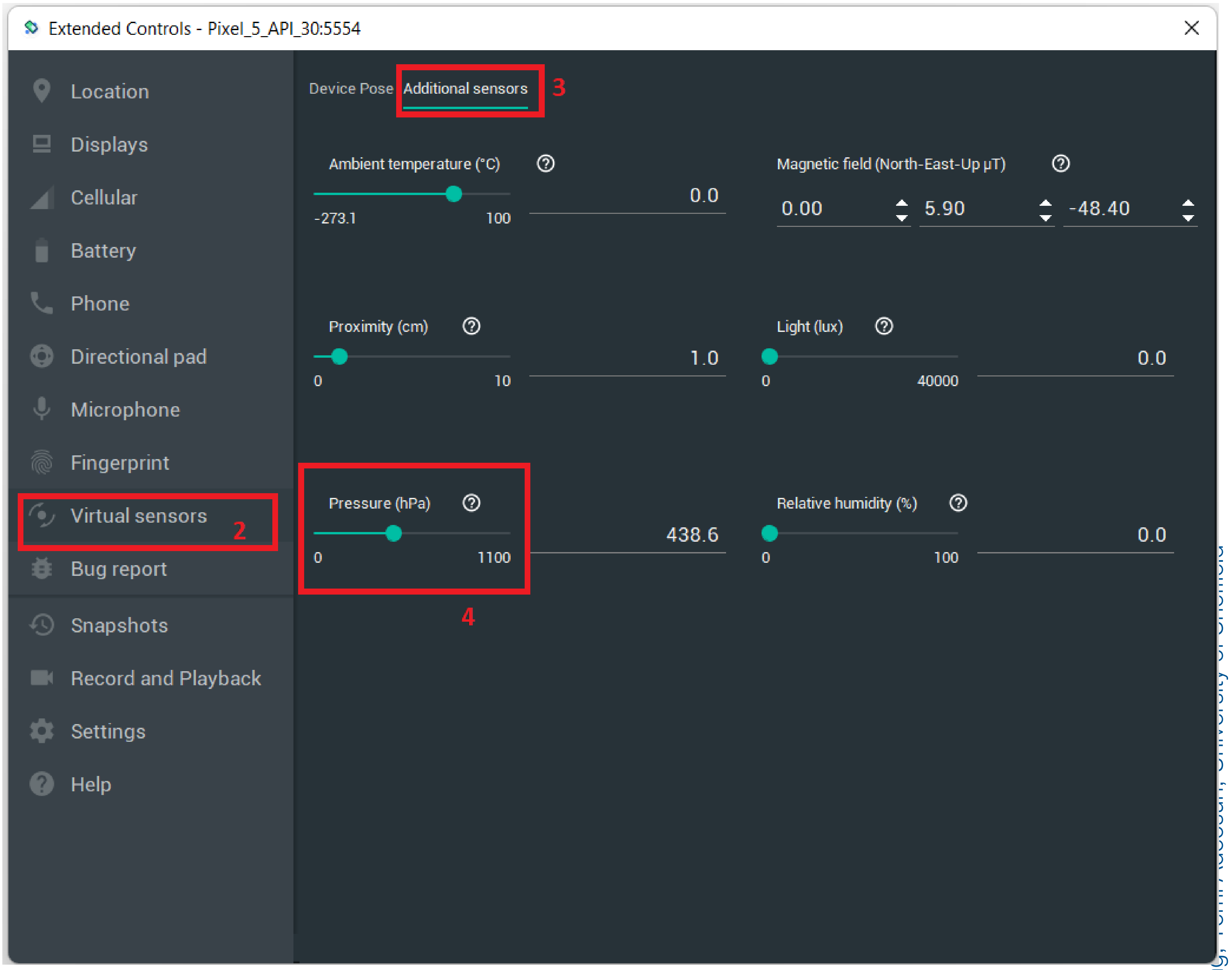
```
// http://androidforums.com/threads/how-to-get-time-of-last-system-boot.548661/
timePhoneWasLastRebooted = System.currentTimeMillis() - SystemClock.elapsedRealtime()
```

nano to milli seconds

```
actualTimeInMseconds = timePhoneWasLastRebooted + (event.timestamp / 1000000.0).toLong()
```

How to test on Emulator





Ambient temperature (°C): 0.0 (range -273.1 – 100)

Magnetic field (North-East-Up μT): 0.00, 5.90, -48.40

Proximity (cm): 1.0 (range 0 – 10)

Light (lux): 0.0 (range 0 – 40000)

Pressure (hPa): 438.6 (range 0 – 1100)

Relative humidity (%): 0.0 (range 0 – 100)

Location Awareness

Add Google play service dependencies:

```
implementation 'com.google.android.gms:play-services-location:${play_service_version}'

implementation 'com.google.android.gms:play-services-maps:${play_service_version}’
```

At the time of writing, play service version is 18.0.0

https://developer.android.com/training/location/index.html

Location Awareness

- One of the unique features of mobile applications is **location awareness**
- Mobile users take their devices with them everywhere, and adding location awareness to your app offers users a more **contextual experience**

The University Of Sheffield.

Use Google Play Services

- Always use the location APIs available in **Google Play Services** in your app
  - with automated location tracking, geofencing, and activity recognition The Google Play services location APIs
- **Google Play Services** are preferred over the Android framework location APIs (android.location)
  - as a way of adding **location awareness** to your app
- Note: Since this class is based on the Google Play services client library, make sure you install the latest version before using the sample apps or code snippets. To learn how to set up the client library with the latest version, see Setup in the Google Play services guide.

Always check that GooglePlayServices are available in the country where you plan to release

Location permission

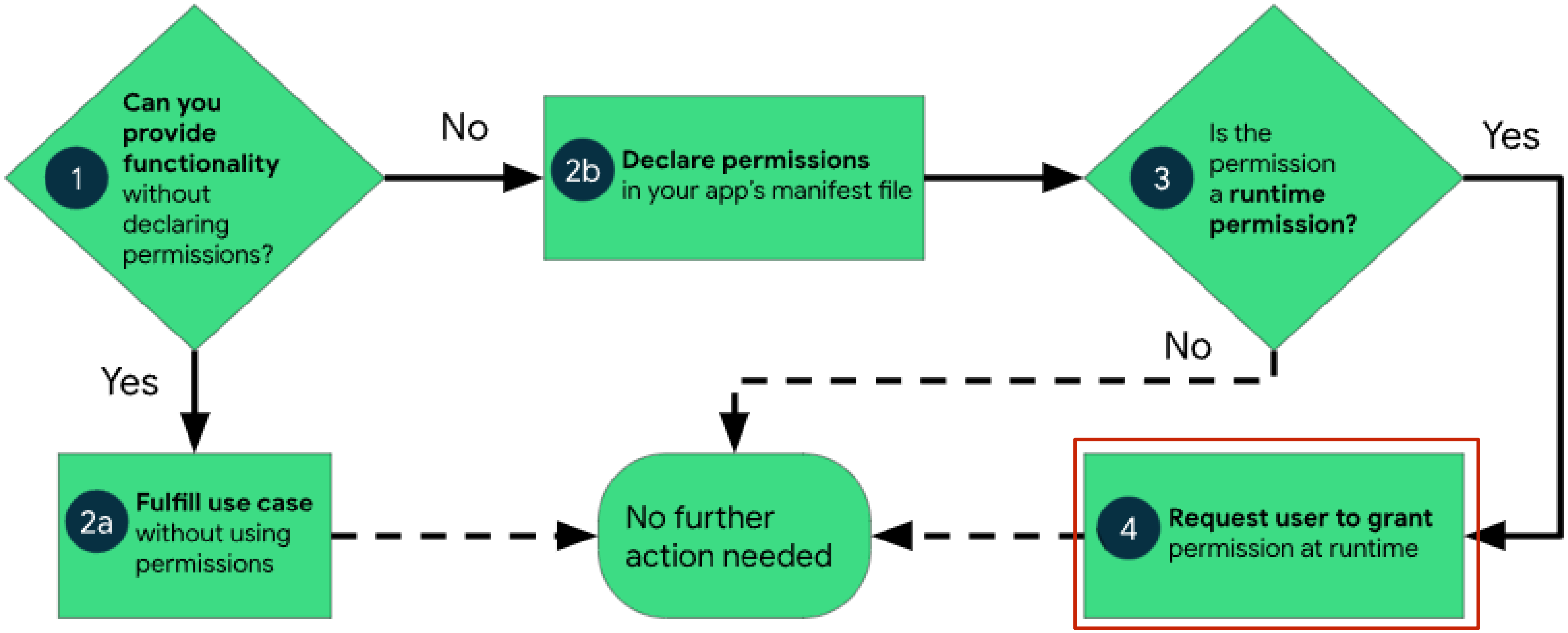
- Requesting location access requires making runtime permission request (vs. install time permission request)
- Location permissions consist two elements:
  - Location access category – foreground or background (only foreground discussed here)
  - Accuracy: two supported by Android - precise and approximate
- Foreground location access. Suited for
  - One-time, defined/limited time location request
  - app is visible
  - app running a foreground service
  - Declare foreground location intention by requesting ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION in Android Manifest:

```
<!-- Always include this permission -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

<!-- Include only if your app benefits from precise location access. -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

https://developer.android.com/training/location/permissions

Location permission

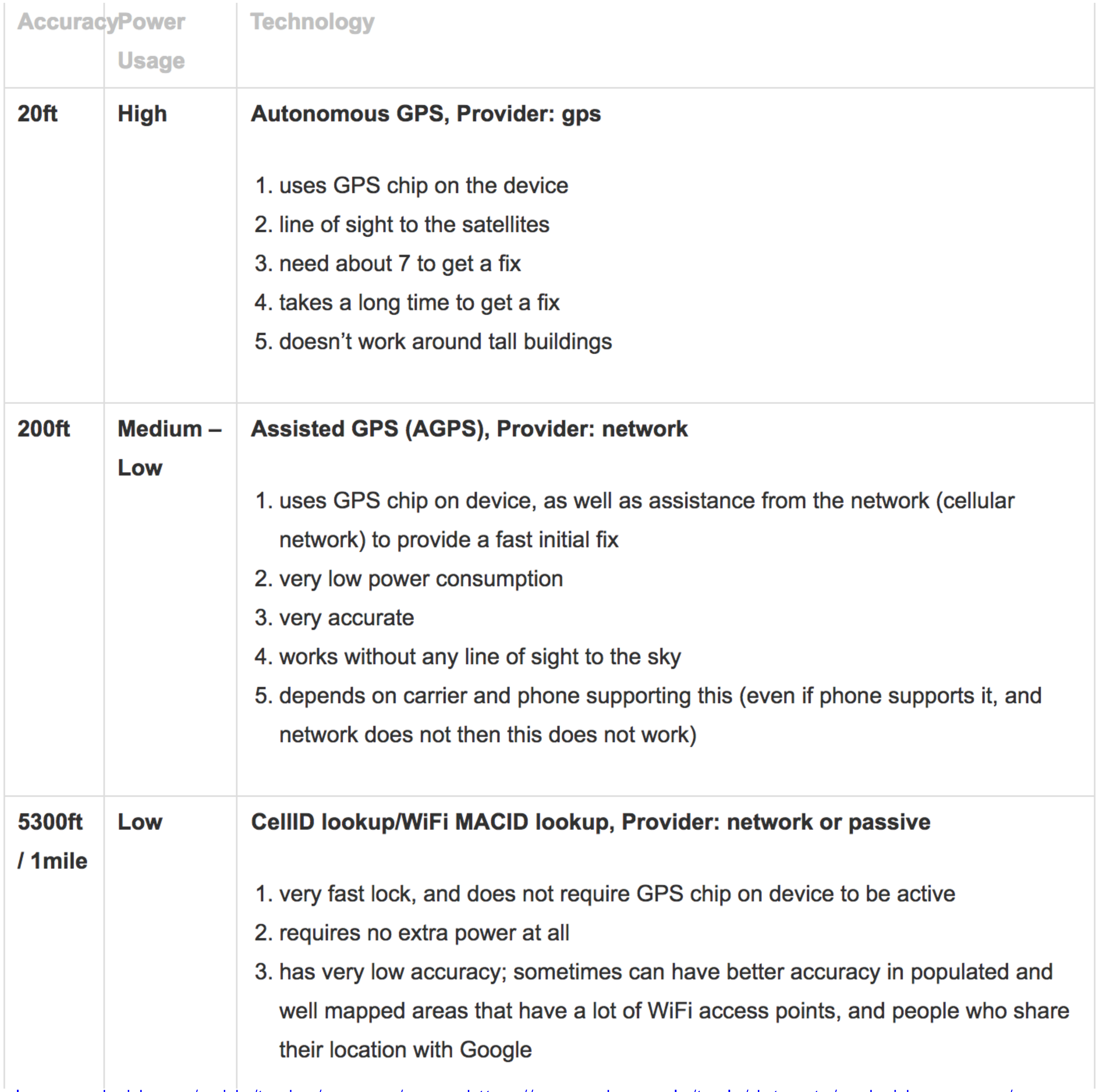
- Approximate(ACCESS_COARSE_LOCATION): network - city block level - ~100m-1000m
- Precise (ACCESS_FINE_LOCATION): estimates device location to about 160ft(50m), even better (~1-50m)
- Follow the Android permission workflow and permissions best practice when requesting permissions:


1 Can you provide functionality without declaring permissions?
No
2b Declare permissions in your app's manifest file
3 Is the permission a runtime permission?
Yes
Yes
No
2a Fulfill use case without using permissions
No further action needed
4 Request user to grant permission at runtime

https://developer.android.com/training/location/permissions

Location Providers

https://developerlife.com/2010/10/20/gps/



Accuracy	Power Usage	Technology
20ft	High	**Autonomous GPS, Provider: gps** 1. uses GPS chip on the device 2. line of sight to the satellites 3. need about 7 to get a fix 4. takes a long time to get a fix 5. doesn't work around tall buildings
200ft	Medium – Low	**Assisted GPS (AGPS), Provider: network** 1. uses GPS chip on device, as well as assistance from the network (cellular network) to provide a fast initial fix 2. very low power consumption 3. very accurate 4. works without any line of sight to the sky 5. depends on carrier and phone supporting this (even if phone supports it, and network does not then this does not work)
5300ft / 1mile	Low	**CellID lookup/WiFi MACID lookup, Provider: network or passive** 1. very fast lock, and does not require GPS chip on device to be active 2. requires no extra power at all 3. has very low accuracy; sometimes can have better accuracy in populated and well mapped areas that have a lot of WiFi access points, and people who share their location with Google

GNNS (Interesting mention): https://developer.android.com/guide/topics/sensors/gnss, https://research.google/tools/datasets/android-accuracy/

The University Of Sheffield.

Fused Location

- The **fused location provider** is one of the location APIs in Google Play Services
- It manages the underlying location technology and provides **a simple API** so that you can specify requirements at a high level, like high accuracy or low power
- It also **optimizes the device's use** of battery power.
  - the old location service had the concept of provider:
    - GPS (high precision), Network (mid precision), WIFI, etc.
    - This has now been changed to optimize battery usage
      - Wireless is often the most accurate in cities thanks to the mapping of wireless networks.

Compare with: https://developerlife.com/2010/10/20/gps/

Using FusedLocationProvider

1 Request foreground location permission in manifest as required

2 Create an instance of a FusedLocationProvideClient

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

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


lateinit var locationClient: FusedLocationProviderClient


override fun onCreate(savedInstanceState: Bundle?) {
  // ...
  locationClient = LocationServices.getFusedLocationProviderClient(this)
  // ...
}
```

Note: here is it in the onCreate(). You may want to do this in a Service/WorkManager if you are tracking location in the background

- FusedLocationProviderClient instance exposes methods to get location information:
  - getLastLocation(): gets location estimate quickly with minimal battery usage. Location might be outdated if no client used location service recently
  - getCurrentLocation(): get fresh, accurate location consistently; can cause location computation
    - Recommended method for getting location – safter than alternatives like requestLocationUpdates()
  - requestLocationUpdates(): requests location update with a callback to continuously receive location.
    - Increased battery consumption – especially if location is unavailable or request isn't stopped correctly

The University Of Sheffield.

The Location Object

- Methods provide access to location object
  - If your app can continuously track location, it can deliver **more relevant information** to the user
    - e.g. to find their way while walking or driving
  - Available information:
    - **latitude and longitude**
    - **bearing (horizontal direction of travel)**
    - **altitude**
    - **speed of the device**
    - **time (in millisec in 01/01/1970), elapsedRealTimeNano (nanosec since boot)**

Last known location

- The precision of the location returned by this call is determined by the permission setting you put in your app manifest
- To request the last known location use the lastlocation property in Kotlin (property value is provided by getLastLocation()
- Implement addOnSuccessListener – asynchronous callback invoked when location is returned.

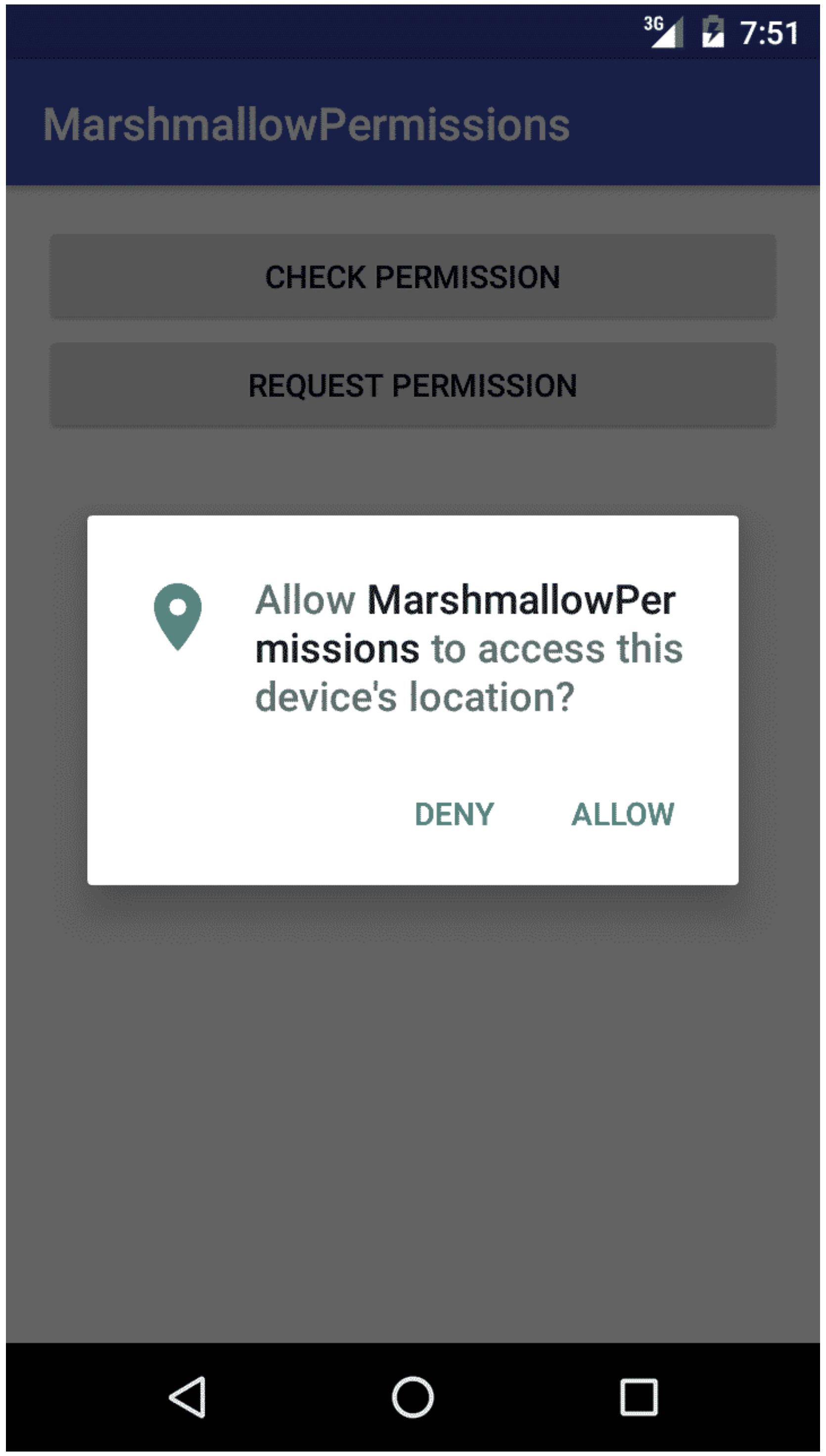
```
locationClient.lastLocation.addOnSuccessListener {Location: location ->
    when(it){
        null -> TODO("Handle rare situation where location is null ")
        else -> {TODO("Do something with the location object")}
    }
}
```

- location object may be null: location setting turned off; device hasn’t record location yet (new/factory reset device), Google play service restarted

Asking for permission

https://developer.android.com/training/permissions/requesting

- Since Android 6 Location is considered a **dangerous permission**
  - So it is considered not given by the user until it is explicitly requested



Allow **MarshmallowPermissions** to access this device's location?

lastLocation permission check

- lastLocation will raise compile time error because of attempt to access location object without checking location permission is granted
- Always check permission is granted before accessing any data that needs runtime permission
  - Note the code snippet on the next slide can be generated by Android studio by accepting the suggestions for the compile time error
- This check needs to run before the call to lastLocation. See more about requesting location permissions

lastLocation permission check

```
if (ActivityCompat.checkSelfPermission(
        this,
        Manifest.permission.ACCESS_FINE_LOCATION
    ) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(
        this,
        Manifest.permission.ACCESS_COARSE_LOCATION
    ) != PackageManager.PERMISSION_GRANTED
) {
    // TODO: Consider calling
    // requestPermissionLauncher.launch(
    //     Manifest.permission.ACCESS_FINE_LOCATION)
    // here to request the missing permissions, and then overriding
    //   public void onRequestPermissionsResult(int requestCode, String[] permissions,
    //                                          int[] grantResults)
    // to handle the case where the user grants the permission. See the documentation
    // for ActivityCompat#requestPermissions for more details.

}
```

- This is single permission check example. Permission checks are needed from Android 6 – API 23. Older devices just grant the permissions. Please check more details about handle permissions.

Best practices

https://developer.android.com/training/permissions/usage-notes

Tenets of working with Android permissions

We recommend following these tenets when working with Android permissions:

#1: Only use the permissions necessary for your app to work. Depending on how you are using the permissions, there may be another way to do what you need (system intents, identifiers, backgrounding for phone calls) without relying on access to sensitive information.

#2: Pay attention to permissions required by libraries. When you include a library, you also inherit its permission requirements. You should be aware of what you're including, the permissions they require, and what those permissions are used for.

#3: Be transparent. When you make a permissions request, be clear about what you’re accessing, and why, so users can make informed decisions. Make this information available alongside the permission request including install, runtime, or update permission dialogues.

#4: Make system accesses explicit. Providing continuous indications when you access

Best practices

Important read: https://developer.android.com/training/permissions/usage-notes

Tenets of working with Android permissions (In addition to the usage notes linked above)

We recommend following these tenets when working with Android permissions:

#1: Only use the permissions necessary for your app to work. Depending on how you are using the permissions, there may be another way to do what you need (system intents, identifiers, backgrounding for phone calls) without relying on access to sensitive information.

#2: Pay attention to permissions required by libraries. When you include a library, you also inherit its permission requirements. You should be aware of what you're including, the permissions they require, and what those permissions are used for.

#3: Be transparent. When you make a permissions request, be clear about what you're accessing, and why, so users can make informed decisions. Make this information available alongside the permission request including install, runtime, or update permission dialogues.

Best practices

Important read: https://developer.android.com/training/permissions/usage-notes

#4: Make system accesses explicit. Providing continuous indications when you access sensitive capabilities (for example, the camera or microphone) makes it clear to users when you’re collecting data and avoids the perception that you're collecting data surreptitiously.

Active Location Seeking

- If lastLocation returns a rather old location, you may want to **seek locations actively**
  - this costs a considerable amount of battery
  - do it only if needed
  - do not start/stop unnecessarily location seeking
    - once started let it run until needed
  - stop it as soon as possible/no longer needed

Important read: https://developer.android.com/training/location/request-updates

A word on location setting

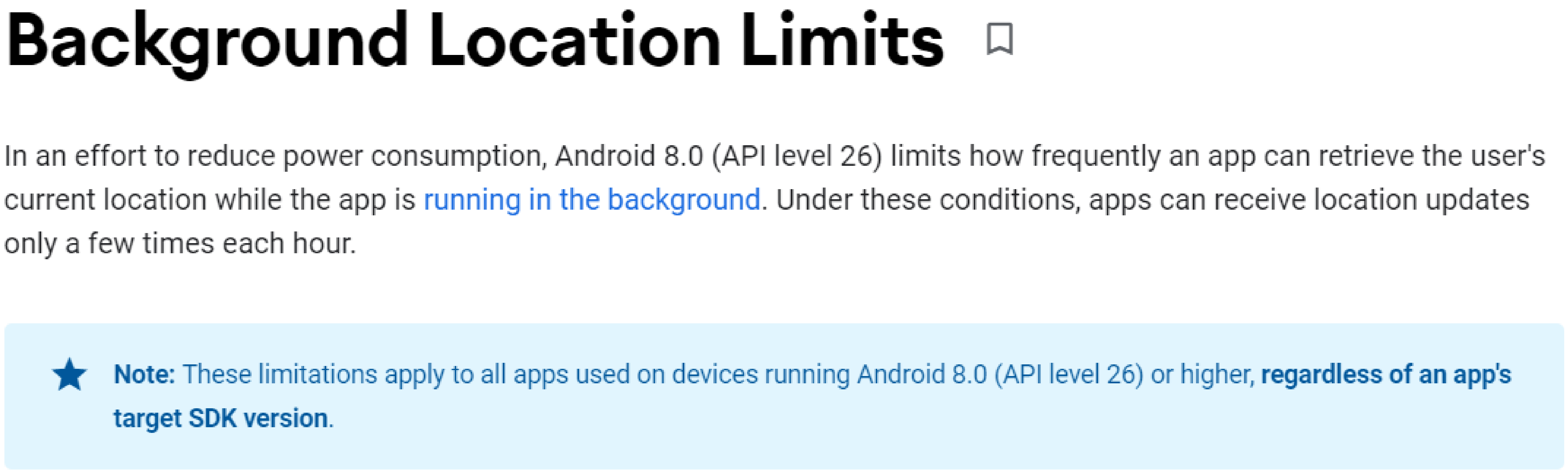
- The need of your app might vary at run time:
  - Request update more frequently as speed changes
  - Demand higher accuracy
- Use the Settings client
  - Remember to prompt the user if you need permission for new setting

Further reading: https://developer.android.com/training/location/change-location-settings

The University Of Sheffield.

Issues and Warnings ⚠

- Locations are **never** precise
- Lat/long/speed. etc. are approximate and sometimes way off
  - e.g. accuracy > 3000m on motorways
    - even if accuracy is high-ish (e.g. 30m) the location may be way off (800m)
  - GNNS will improve this
- Location has typically max frequency of 1 second
- Paths need cleaning

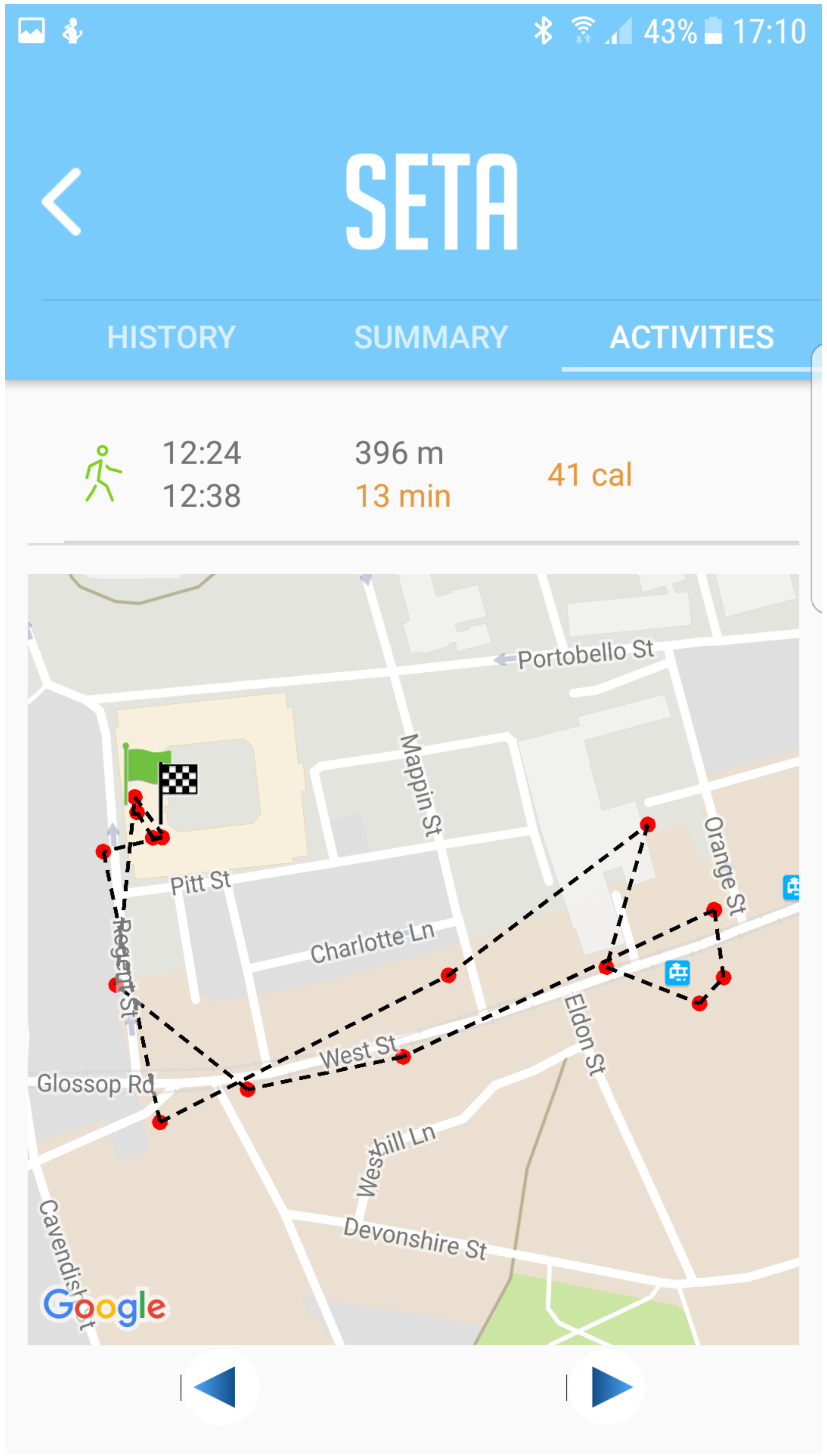


Background Location Limits

In an effort to reduce power consumption, Android 8.0 (API level 26) limits how frequently an app can retrieve the user's current location while the app is running in the background. Under these conditions, apps can receive location updates only a few times each hour.

★ **Note:** These limitations apply to all apps used on devices running Android 8.0 (API level 26) or higher, **regardless of an app's target SDK version**.

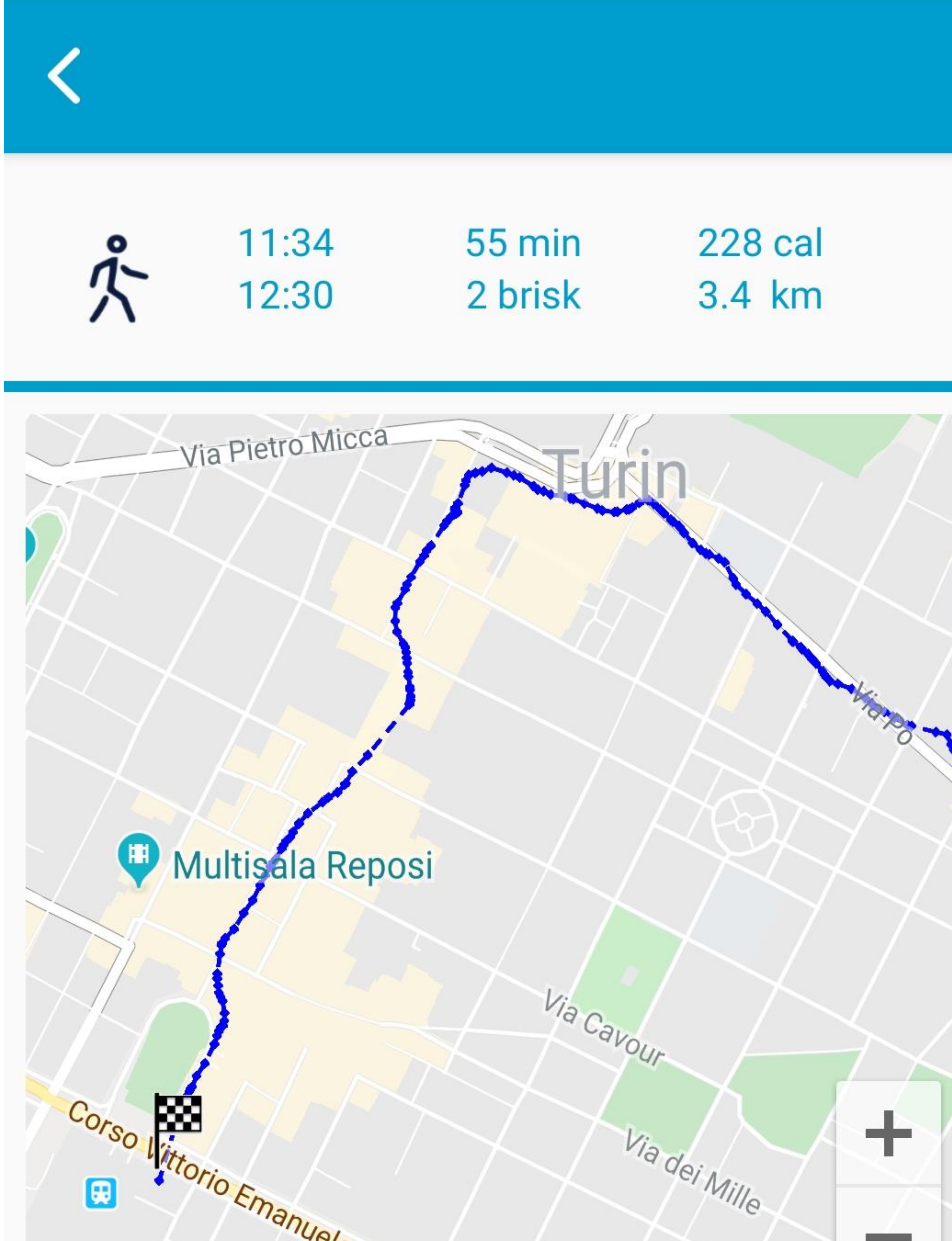
Imprecision



- Location services have predictive capabilities
  - so more frequent requests means better precision
  - See 3 seconds frequency in the image

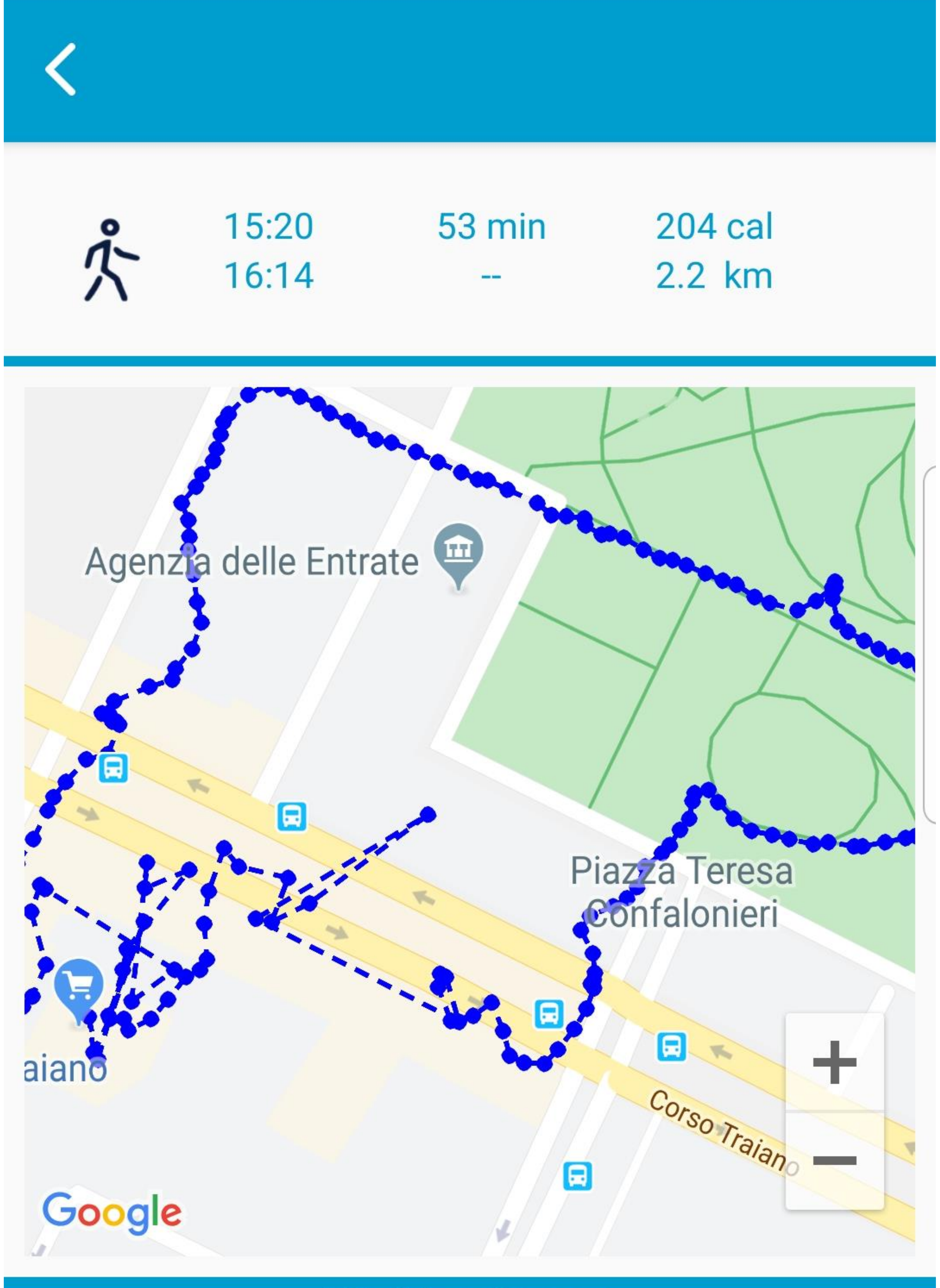
Also see:

https://developer.android.com/training/location/change-location-settings



The University Of Sheffield.

but still…

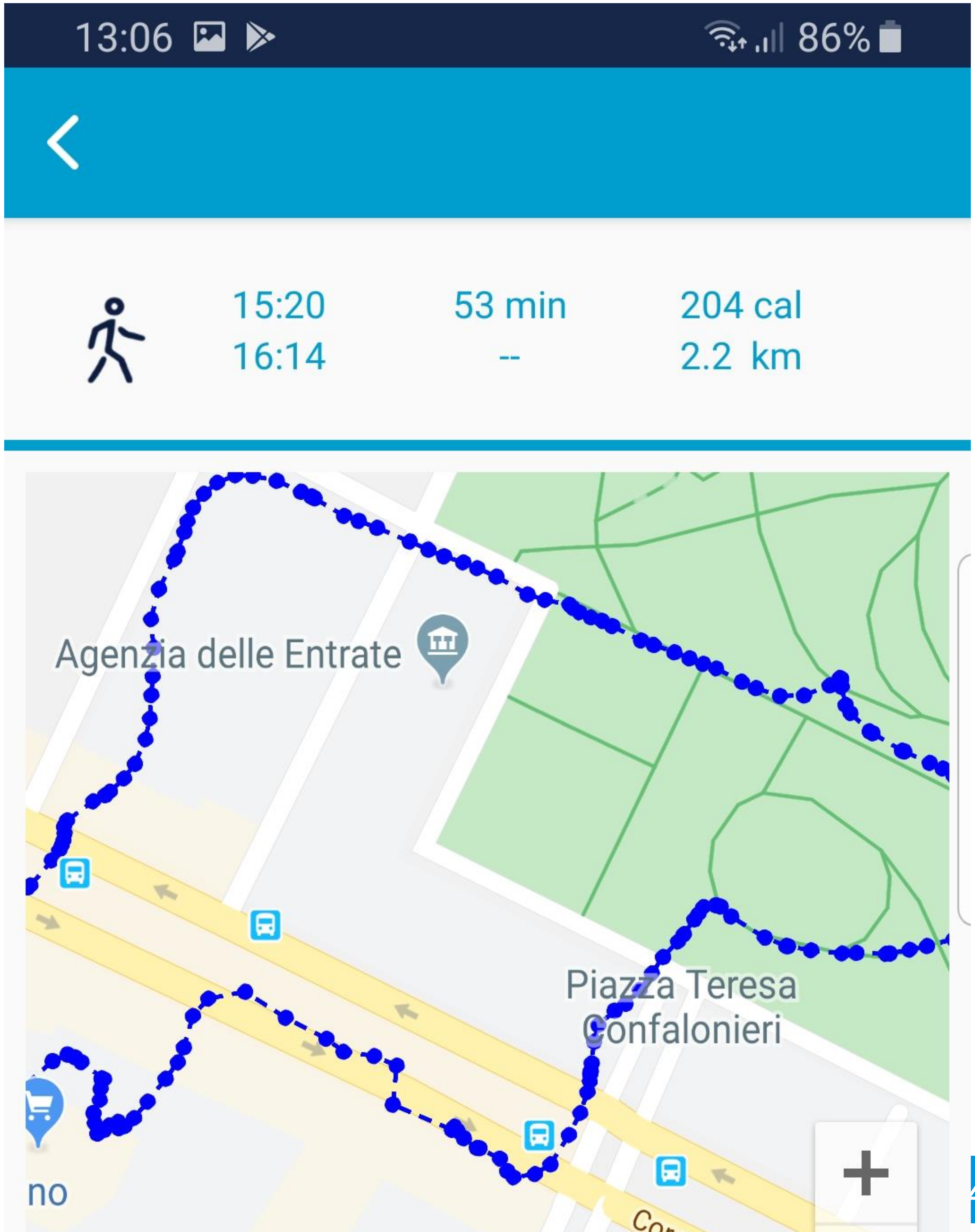

15:20 16:14

53 min --

204 cal 2.2 km

Using smoothing

- Different kind of smoothing functions can be used
  - e.g. Kalman Filters
- They are not easy to use and can be computationally expensive
- Augmented with additional data for improvement
  - e.g. using a step counter



Summary

- Sensors overview
  - An example
- Location awareness
  - Issues and warnings

Lab tutorial :

- Sensing