

SENG440

Week 8

Camera & Location

Ben Adams (benjamin.adams@canterbury.ac.nz)

Sensor questions

- What uses the most battery?
 - Depends on device but general rule:
 - GPS, accelerometer and gyroscope the most
 - Light sensor and compass less power
 - Can use `getPower()` fun in `Sensor` class to get the power in mA used by this sensor while in use
- We will go through more code examples next week

This Week

- Camera API
- CameraX
- MLKit Integration
- Location

Camera

- Request the Camera app to take the picture and return it
- Full access to the camera within your app

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

```
<uses-feature android:name="android.hardware.camera.any" />  
<uses-permission android:name="android.permission.CAMERA" />
```

Take a photo with photo app

```
val REQUEST_IMAGE_CAPTURE = 1

private fun dispatchTakePictureIntent() {
    Intent(MediaStore.ACTION_IMAGE_CAPTURE).also { takePictureIntent ->
        takePictureIntent.resolveActivity(packageManager)?.also {
            startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE)
        }
    }
}
```

Get thumbnail

- Thumbnail comes back from the Intent as an extra called “data”

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {  
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK) {  
        val imageBitmap = data.extras.get("data") as Bitmap  
        imageView.setImageBitmap(imageBitmap)  
    }  
}
```

Get full size image

- Photos saved in public external storage usually, but can be set to private data (deleted when the app is deleted).
- Requires `READ_EXTERNAL_STORAGE` and `WRITE_EXTERNAL_STORAGE` permissions
- Use `getExternalFilesDir(Environment.DIRECTORY_PICTURES)`

```
val REQUEST_TAKE_PHOTO = 1

private fun dispatchTakePictureIntent() {
    Intent(MediaStore.ACTION_IMAGE_CAPTURE).also { takePictureIntent ->
        // Ensure that there's a camera activity to handle the intent
        takePictureIntent.resolveActivity(packageManager)?.also {
            // Create the File where the photo should go
            val photoFile: File? = try {
                createImageFile()
            } catch (ex: IOException) {
                // Error occurred while creating the File
                ...
                null
            }
        }
        // Continue only if the File was successfully created
        photoFile?.also {
            val photoURI: Uri = FileProvider.getUriForFile(
                this,
                "com.example.android.fileprovider",
                it
            )
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI)
            startActivityForResult(takePictureIntent, REQUEST_TAKE_PHOTO)
        }
    }
}
```

Photo file

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

```
<application>
  ...
  <provider
    androidx.core.content.FileProvider
    android:name="android.support.v4.content.FileProvider"
    android:authorities="com.example.android.fileprovider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
      android:name="android.support.FILE_PROVIDER_PATHS"
      android:resource="@xml/file_paths"></meta-data>
    </provider>
  ...
</application>
```


Recording videos

```
const val REQUEST_VIDEO_CAPTURE = 1

private fun dispatchTakeVideoIntent() {
    Intent(MediaStore.ACTION_VIDEO_CAPTURE).also { takeVideoIntent ->
        takeVideoIntent.resolveActivity(packageManager)?.also {
            startActivityForResult(takeVideoIntent, REQUEST_VIDEO_CAPTURE)
        }
    }
}
```


```
override fun onActivityResult(requestCode: Int, resultCode: Int, intent: Intent) {
    if (requestCode == REQUEST_VIDEO_CAPTURE && resultCode == RESULT_OK) {
        val videoUri: Uri = intent.data
        videoView.setVideoURI(videoUri)
    }
}
```

Directly controlling the camera

- Sometimes want the **camera inside app**, e.g. a barcode scanner
- Create an instance of a **Camera object** on a different thread from Main/UI.
 - Will throw an exception if the camera is in use by another app.
- Create a **camera Preview** implementing `android.view.SurfaceHolder.Callback`
- Can programmatically **change the camera settings**, e.g. zoom, exposure compensation, landscape vs portrait.
- **Features:** focus areas, face detection, time lapse video, etc.
- 2 versions of API. The developer examples use first version (camera). The second version (camera2): [android.hardware.camera2](https://developer.android.com/reference/android/hardware/camera2)

CameraX

- **Camera jetpack support library** introduced in 2019 by Google.
- Easier to tap into capabilities of **different kinds of cameras** (e.g. effects) on different devices.
- **Backward compatible** with Camera API
- ImageAnalysis class that implements an analysis() method to perform computer vision and machine learning.



```
private fun startCamera() {
    val cameraProviderFuture = ProcessCameraProvider.getInstance(this)

    cameraProviderFuture.addListener(Runnable {
        // Used to bind the lifecycle of cameras to the lifecycle owner
        val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

        // Preview
        val preview = Preview.Builder()
            .build()
            .also {
                it.setSurfaceProvider(viewFinder.createSurfaceProvider())
            }

        // Select back camera as a default
        val cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA

        try {
            // Unbind use cases before rebinding
            cameraProvider.unbindAll()

            // Bind use cases to camera
            cameraProvider.bindToLifecycle(
                this, cameraSelector, preview)
        } catch (exc: Exception) {
            Log.e(TAG, "Use case binding failed", exc)
        }
    }, ContextCompat.getMainExecutor(this))
}
```

startCamera is called when the view is created.

```
private fun startCamera() {
    val cameraProviderFuture = ProcessCameraProvider.getInstance(this)

    cameraProviderFuture.addListener(Runnable {
        // Used to bind the lifecycle of cameras to the lifecycle owner
        val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

        // Preview
        val preview = Preview.Builder()
            .build()
            .also {
                it.setSurfaceProvider(viewFinder.createSurfaceProvider())
            }

        // Select back camera as a default
        val cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA

        try {
            // Unbind use cases before rebinding
            cameraProvider.unbindAll()

            // Bind use cases to camera
            cameraProvider.bindToLifecycle(
                this, cameraSelector, preview)
        } catch (exc: Exception) {
            Log.e(TAG, "Use case binding failed", exc)
        }
    }, ContextCompat.getMainExecutor(this))
}
```

Create instance of Camera provider.
Binds lifecycle of the camera to the
lifecycle of the owner (e.g., fragment)

```
private fun startCamera() {
    val cameraProviderFuture = ProcessCameraProvider.getInstance(this)

    cameraProviderFuture.addListener(Runnable {
        // Used to bind the lifecycle of cameras to the lifecycle owner
        val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

        // Preview
        val preview = Preview.Builder()
            .build()
            .also {
                it.setSurfaceProvider(viewFinder.createSurfaceProvider())
            }

        // Select back camera as a default
        val cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA

        try {
            // Unbind use cases before rebinding
            cameraProvider.unbindAll()

            // Bind use cases to camera
            cameraProvider.bindToLifecycle(
                this, cameraSelector, preview)
        } catch (exc: Exception) {
            Log.e(TAG, "Use case binding failed", exc)
        }
    }, ContextCompat.getMainExecutor(this))
}
```

Set to run camera to run on thread.
In this case the main thread.

```
private fun startCamera() {
    val cameraProviderFuture = ProcessCameraProvider.getInstance(this)

    cameraProviderFuture.addListener(Runnable {
        // Used to bind the lifecycle of cameras to the lifecycle owner
        val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

        // Preview
        val preview = Preview.Builder()
            .build()
            .also {
                it.setSurfaceProvider(viewFinder.createSurfaceProvider())
            }


        // Select back camera as a default
        val cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA

        try {
            // Unbind use cases before rebinding
            cameraProvider.unbindAll()

            // Bind use cases to camera
            cameraProvider.bindToLifecycle(
                this, cameraSelector, preview)
        } catch (exc: Exception) {
            Log.e(TAG, "Use case binding failed", exc)
        }
    }, ContextCompat.getMainExecutor(this))
}
```

Create Preview for the camera. This is the view that shows the updated camera image.

viewFinder is a `androidx.camera.view.PreviewView` defined in the view's layout template.



```
private fun startCamera() {
    val cameraProviderFuture = ProcessCameraProvider.getInstance(this)

    cameraProviderFuture.addListener(Runnable {
        // Used to bind the lifecycle of cameras to the lifecycle owner
        val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

        // Preview
        val preview = Preview.Builder()
            .build()
            .also {
                it.setSurfaceProvider(viewFinder.createSurfaceProvider())
            }


        // Select back camera as a default
        val cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA

        try {
            // Unbind use cases before rebinding
            cameraProvider.unbindAll()

            // Bind use cases to camera
            cameraProvider.bindToLifecycle(
                this, cameraSelector, preview)
        } catch (exc: Exception) {
            Log.e(TAG, "Use case binding failed", exc)
        }
    }, ContextCompat.getMainExecutor(this))
}
```

Select which camera: front or back.





```
private fun startCamera() {
    val cameraProviderFuture = ProcessCameraProvider.getInstance(this)

    cameraProviderFuture.addListener(Runnable {
        // Used to bind the lifecycle of cameras to the lifecycle owner
        val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

        // Preview
        val preview = Preview.Builder()
            .build()
            .also {
                it.setSurfaceProvider(viewFinder.createSurfaceProvider())
            }

        // Select back camera as a default
        val cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA

        try {
            // Unbind use cases before rebinding
            cameraProvider.unbindAll()

            // Bind use cases to camera
            cameraProvider.bindToLifecycle(
                this, cameraSelector, preview)

        } catch (exc: Exception) {
            Log.e(TAG, "Use case binding failed", exc)
        }

    }, ContextCompat.getMainExecutor(this))
}
```

Bind use cases. This is where you can put an analyzer in the pipeline, e.g., face detector.

Image analysis

```
val imageAnalysis = ImageAnalysis.Builder()
    .setTargetResolution(Size(1280, 720))
    .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
    .build()

imageAnalysis.setAnalyzer(executor, ImageAnalysis.Analyzer { image ->
    val rotationDegrees = image.imageInfo.rotationDegrees
    // insert your code here.
})

cameraProvider.bindToLifecycle(this as LifecycleOwner, cameraSelector, imageAnalysis, preview)
```

- Can create your own subclass of `ImageAnalysis.Analyzer`
- This is how you include machine learning based analyzers

MLKit

- Machine learning libraries optimized for mobile
 - Android
 - iOS
- Used to be part of Google Firebase
- Vision
 - Text recognition, face detection, post detection, selfie segmentation, barcode scanning, image labeling, object detection and tracking, digital ink recognition
- Natural language processing
 - Identify languages, translate text, smart replies, entity extraction

Resources

- **CameraX**

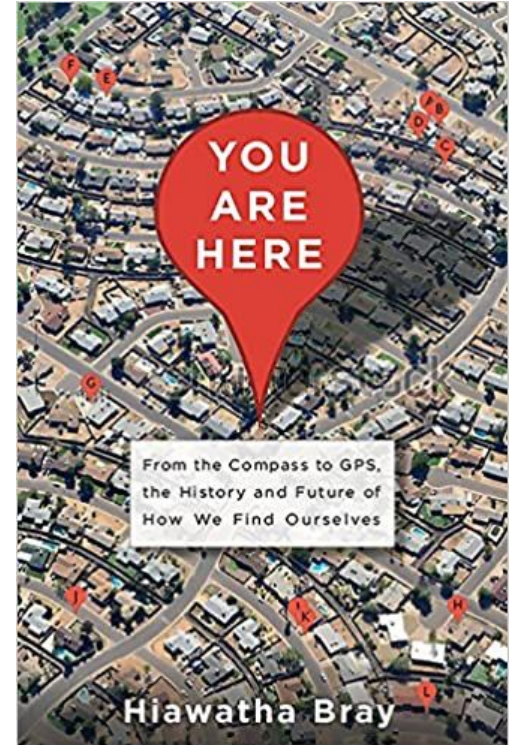
- CameraX Overview: [CameraX overview | Android Developers](#)
- Getting Started with CameraX Codelab: [Getting Started with CameraX](#)
- CameraX Demo App: [android/camera-samples github.com](#)

- **MLKit**

- [ML Kit | Google for Developers](#)

Android and Location

- Android device knows your location via multiple methods
 - GPS
 - Cell-ID (cell tower)
 - Wi-Fi networks
 - Network Location Provider combines Cell-ID and Wi-Fi data



Global Positioning System (GPS)

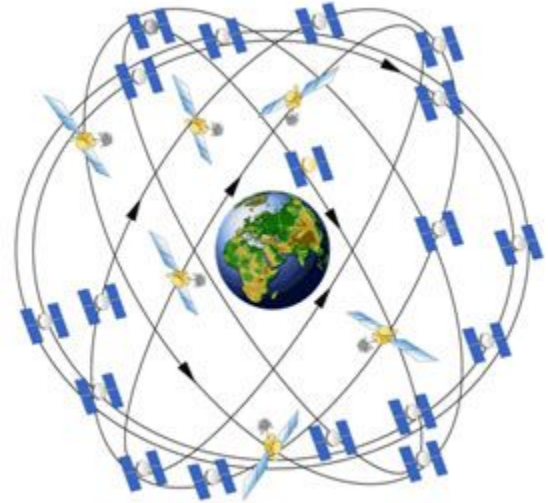
- US System that provides position, navigation, and timing
- Space Segment, Control Segment, User Segment
- US Air Force and Space Force(!) develops, maintains, and operates the space segment and control segment
- Other countries have developed their own system. E.g. BeiDou



GPS Space Segment

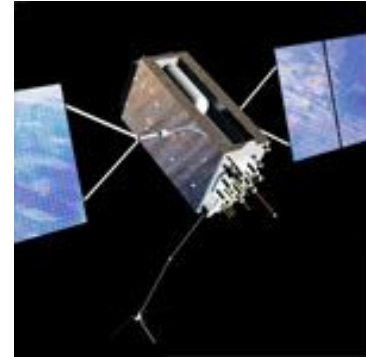
- 31 core satellites
- Medium earth orbit, 20k km above the earth
- 6 orbital planes with 4 satellites each
- Generally 4 satellites in line of sight at any spot on the Earth

[GPS.gov: Space Segment](https://www.gps.gov/space-segment)



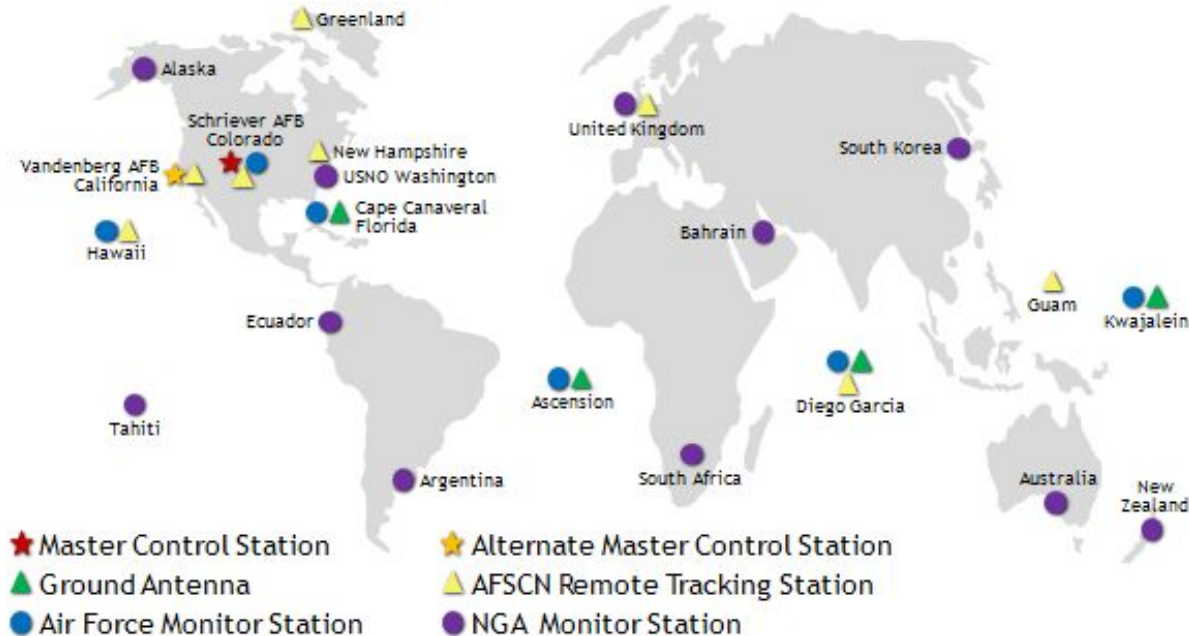
GPS Space Segment

- Satellites circle the earth twice a day
- Upgraded over time with different generations of satellites
- Current generation of satellites being developed by Lockheed - Martin (FOCS)



GPS Control Segment

- Ground facilities that monitor transmissions, perform analysis, and send commands and data to satellites



GPS User Segment

- Onboard clocks with accuracy of 1 nanosecond (1 billionth of a second).
- Satellites transmit one-way.
- Receiver calculates position and course by comparing time signals from multiple satellites with the known position of those satellites.

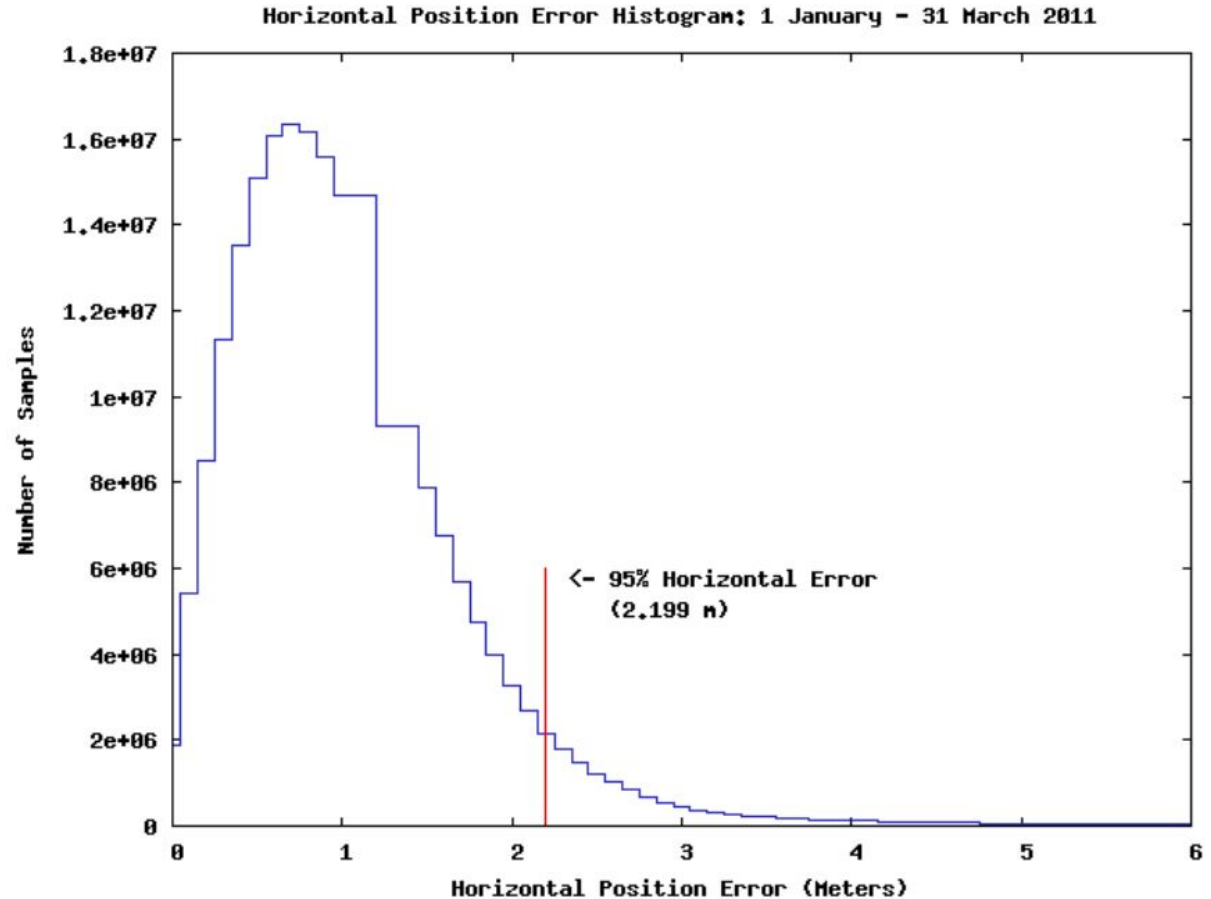
CRAZY PHENOMENON		IF IT WORKED, COMPANIES WOULD BE USING IT TO MAKE A KILLING IN...	ARE THEY?
REMOTE VIEWING	OIL PROSPECTING		
DOWSING			
AURAS	HEALTH CARE COST REDUCTION		
HOMEOPATHY			
REMOTE PRAYER			
ASTROLOGY	FINANCIAL/BUSINESS PLANNING		
TAROT			
CRYSTAL ENERGY	REGULAR ENERGY		
CURSES, HEXES	THE MILITARY		
RELATIVITY	GPS DEVICES		✓
QUANTUM ELECTRODYNAMICS	SEMICONDUCTOR CIRCUIT DESIGN		✓

EVENUALLY, ARGUING THAT THESE THINGS WORK MEANS ARGUING THAT MODERN CAPITALISM ISN'T *THAT* RUTHLESSLY PROFIT-FOCUSED.

GPS User Segment

- Accuracy normally within 5 - 10 meters.
- Precision requires accuracy of clocks and timing signal on the order of 20 nanoseconds.
- The Special and General theories of Relativity must be taken into account to achieve the desired accuracy.
- Special relativity predicts clocks on satellites go slower, on the order of 10 microseconds per day.
- General relativity predicts the mass of the earth will have an effect.

GPS Accuracy

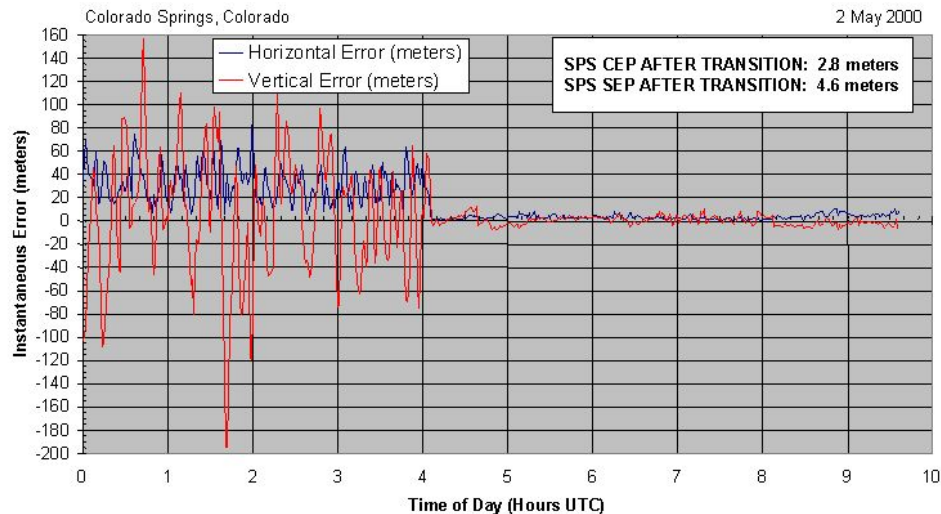


GPS Accuracy

- **Selective Availability:** intentional degradation of signals for civilian use ended in 2000



SA Transition -- 2 May 2000



GPS Accuracy

- Civilian GPS: aka SPS
- Military GPS: aka PPS
- Military broadcasts on two frequencies, civilian only one
- "This means military users can perform *ionospheric correction*, a technique that reduces radio degradation caused by the Earth's atmosphere. With less degradation, PPS provides better accuracy than the basic SPS. "



Android and Location

Android and Location

Currently **3 methods** of obtaining location:

- GPS - Global Positioning System
- Network
 - combines cell tower triangulation and wireless networks
- Passive
 - not a real provider, just piggy back off other applications
 - similar to software sensors

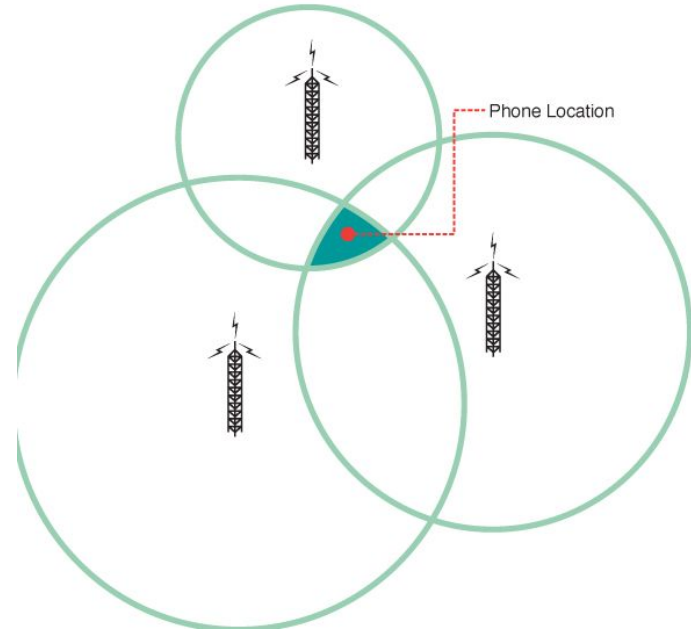
GPS

- Most **accurate**, but only works **outdoors**
- Quickly **consumes battery power**
- **Delay** in acquiring satellites or re-acquiring **if signal lost**



Network

- Combines **cell tower triangulation** and in range **wireless networks**
- If no cellular capability or plan on device (tablets?) then just wireless networks



Location permissions

- Foreground location
 - One-off access to location, e.g. sending a message with location
 - Visible indicator to user
- Request coarse or fine-grained location

```
<manifest ... >
  <!-- To request foreground location access, declare one of these permissions. -->
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
</manifest>
```

```
<manifest ... >
  <!-- Required only when requesting background location access on
       Android 10 (API level 29) and higher. -->
  <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
</manifest>
```

Getting Location

- Older, **detailed approach**
 - Flexible, detailed, but developers typically make mistakes
 - Consumes heaps of battery power: GPS receiver requires large amounts of power to amplify signal from satellites
- Newer, higher level, more abstract approach using **Google Play Services**

Location Manager



Location API

- **LocationManager** – provides access to system location services
 - Get instance of `LocationManager` using `getSystemService` method using `LOCATION_SERVICE` (e.g. GPS or Network)
- **LocationListener** – implement class by overriding methods:
 - `onLocationChanged(location: Location)`
 - **Location** contains the data about the location
 - `onStatusChanged`
 - `onProviderEnabled`
 - `onProviderDisabled`

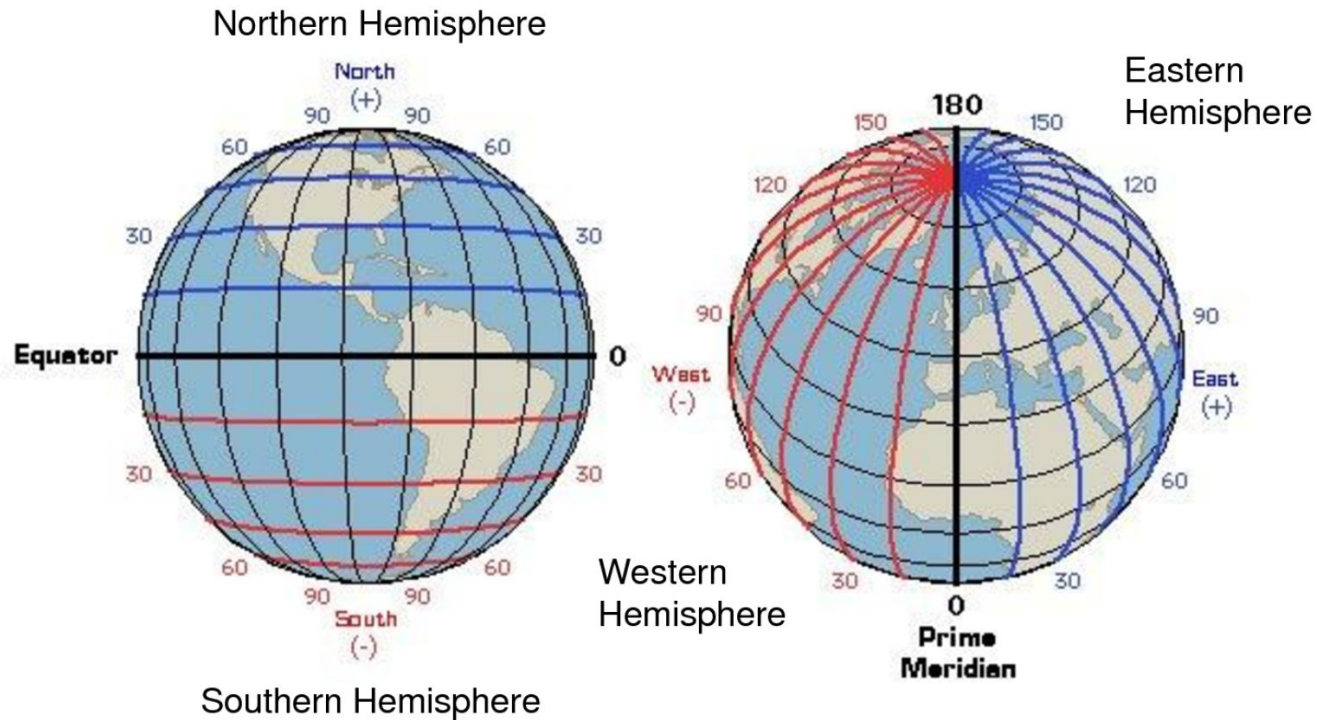
Location Data

- onLocationChanged method in the LocationListener receives Location objects
- toString shown
- latitude, longitude, estimated accuracy in meters, bearing

```
onLocationChanged CALLED:  
Location[gps  
30.286450,-97.736539 acc=50  
et=+1h8m52s912ms  
alt=175.70001220703125  
vel=2.4394498 bear=110.0  
{Bundle[mParcelledData.dataSize  
=44]]]
```

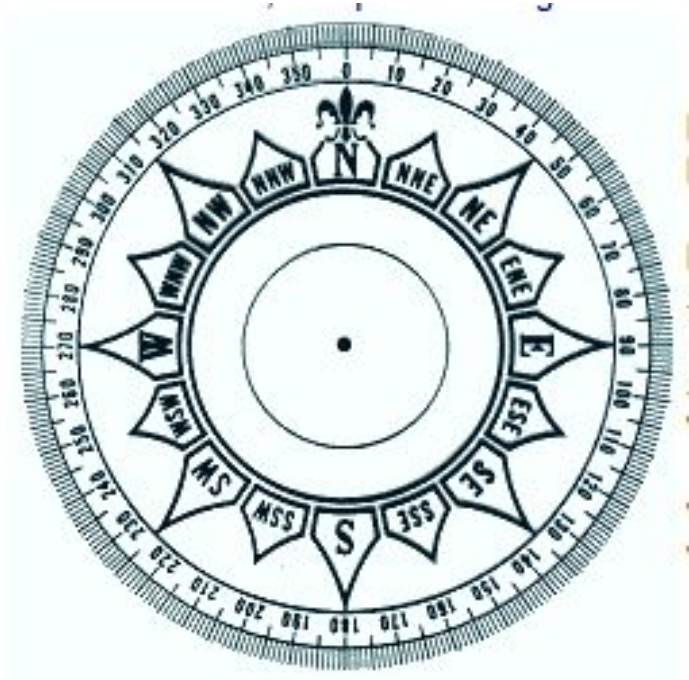
The diagram illustrates the mapping of location data fields from a `toString` output to a list of attributes. A red box highlights the coordinates `30.286450,-97.736539` in the output, which corresponds to the `latitude, longitude` attribute in the list. A green box highlights the `acc=50` in the output, which corresponds to the `estimated accuracy` attribute. A blue box highlights the `bear=110.0` in the output, which corresponds to the `bearing` attribute. Arrows indicate the mapping from the list attributes to the corresponding parts of the output string.

Latitude and Longitude



Bearing

- Direction
- 360 degrees
- degrees east of north
- 0 = north
- 90 = east
- 180 = south
- 270 = west



Sample GPS Locations

- ET = Elapsed time passed since device start up
- Altitude in meters
- Units for velocity: meters / sec

onLocationChanged CALLED:

Location[gps
30.286450,-97.736539 acc=50

et=+1h8m52s912ms

alt=175 70001220703125

vel=2.4394498 bear=110.0

{Bundle[mParcelledData.dataSize
=44]}}

Google Play Location services

- Google Play services added as gradle dependencies:
 - See: <https://developers.google.com/android/guides/setup>

```
dependencies {  
    implementation 'com.google.android.gms:play-services-location:18.0.0'  
}
```

- **Fused Location Provider**

- Provides best location information from multiple sources: GPS, network, ...
 - While optimizing for battery usage

Getting location with play services

- Use FusedLocationProviderClient

```
private lateinit var fusedLocationClient: FusedLocationProviderClient

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

    fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)
}
```

Getting last location

```
fusedLocationClient.lastLocation
    .addOnSuccessListener { location : Location? ->
        // Got last known location. In some rare situations this can be null.
    }
```

- Should check attributes of Location object for:
 - If location is significantly newer than previous estimate
 - If accuracy is better than previous estimate
 - What the provider is (e.g. GPS, network)

Location requests

- Invoke requestLocationUpdates method in the fused location client
- Pass it a location callback that implements onLocationResult, which can update the UI

```
locationCallback = object : LocationCallback() {  
    override fun onLocationResult(locationResult: LocationResult?) {  
        locationResult ?: return  
        for (location in locationResult.locations){  
            // Update UI with location data  
            // ...  
        }  
    }  
}
```

- <https://developer.android.com/training/location/request-updates>

Geofencing

- Set entrance and exit events.
- Requires fine location and background location permissions.
- Maximum 100 geofences per app.
- Use geofence builder to define a circle with center longitude and latitude, and radius in meters.
- <https://developer.android.com/training/location/geofencing>



Maps

- Various **Maps SDKs** available to incorporate mapping in your app.
- **Google Maps** play services
 - <https://developers.google.com/maps/documentation/android-sdk/overview>
- You do not need to rely on Google. Other **alternatives**, e.g. **Mapbox SDK**:
 - Built using OpenStreetMap data
 - Fully customizable map styling
 - <https://docs.mapbox.com/android/maps/guides/>
 - Google still better for some features, such as offline maps

More info

- Build **location based apps**:

<https://developer.android.com/training/location>

- Read the **Sensors Overview**:

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