Mobile Programming

Mobile Sensors

Lecturer: Ali Najimi
Faculty of Computer Engineering
Sharif University of Technology
Fall 2027

Accelerometer | Gyroscope | GPS | Biometrics | Camera | Proximity | Light | Compass



Table of Contents

1. Introduction to Mobile Sensors
2. Sensor Categories Overview
3. Motion Sensors — Accelerometer and Gyroscope
4. Motion Sensors — Magnetometer and Gravity
5. Position Sensors — GPS and Location
6. Position Sensors — Proximity and Step Counter
7. Environmental Sensors — Light, Pressure, Humidity
8. Biometric Sensors — Fingerprint and Face ID
9. Camera and Microphone Sensors
10. Connectivity Sensors — NFC, Bluetooth, Wi-Fi

Table of Contents

11. Flutter Sensor Packages Overview
12. `sensors_plus` — Motion and Position
13. `geolocator` — GPS and Location
14. `local_auth` — Biometric Authentication
15. Battery and Performance Considerations
16. Exercise: Sensor Dashboard and Live Wallpaper
17. Exercise: Bank Card App
18. Summary and Key Takeaways

Introduction to Mobile Sensors

Modern smartphones contain **10-20+ sensors** that enable context-aware experiences.

Why sensors matter:

- **Context awareness** — Apps adapt to user environment (dark mode from light sensor).
- **Health and fitness** — Step counting, heart rate, sleep tracking.
- **Navigation** — GPS + compass for turn-by-turn directions.
- **Security** — Fingerprint and face recognition for authentication.
- **Gaming** — Tilt controls via accelerometer and gyroscope.
- **AR/VR** — Camera + motion sensors for immersive experiences.

> A modern flagship phone has more sensors than a 1990s space probe.

Sensor Categories Overview

Category	Sensors	Primary Use
Motion	Accelerometer, Gyroscope, Gravity, Linear Acceleration	Device movement, tilt, rotation
Position	GPS, Magnetometer (Compass), Proximity	Location, heading, near/far
Environmental	Light, Pressure (Barometer), Temperature, Humidity	Ambient conditions
Biometric	Fingerprint, Face ID/Iris	Authentication, security
Media	Camera (RGB/Depth/IR), Microphone	Photos, video, audio, AR
Connectivity	NFC, Bluetooth, Wi-Fi, Cellular	Proximity, communication
Health	Heart Rate, SpO2, Step Counter, Body Temperature	Fitness, medical

Motion Sensors — Accelerometer

The **accelerometer** measures acceleration forces (including gravity) on 3 axes.

How it works:

- MEMS (Micro-Electro-Mechanical Systems) technology.
- Measures acceleration in m/s2 along X, Y, Z axes.
- At rest, reports **~9.81 m/s2** upward (gravity).

Use cases:

- Screen rotation (portrait / landscape).
- Shake detection (undo action).
- Tilt-based game controls.
- Fall detection (Apple Watch).
- Pedometer (step counting).

> **Note:** Accelerometer data includes gravity. `linear_acceleration`: gravity-free data.

Motion Sensors — Accelerometer in Flutter

Package: `sensors_plus`

```
# pubspec.yaml
dependencies:
  sensors_plus: ^6.0.1
```

```
import 'package:sensors_plus/sensors_plus.dart';

// Listen to accelerometer events
accelerometerEvents.listen((AccelerometerEvent event) {
  print('X: ${event.x}, Y: ${event.y}, Z: ${event.z}');
  // At rest on table: X~0, Y~0, Z~9.81
});
```

> **Tip:** Sensor events fire at **high frequency** (~100-200 Hz). Throttle updates for UI.

Motion Sensors — Gyroscope

The **gyroscope** measures angular velocity (rotation speed) around 3 axes.

How it works:

- Measures rotation rate in **rad/s** around X, Y, Z axes.
- Coriolis effect on vibrating MEMS structure.
- Does NOT measure absolute angle — only rotation speed.

Use cases:

- Camera image stabilization.
- Gesture recognition (flick, twist).
- 3D game controls (steering, aiming).
- Drone stabilization.
- Augmented Reality (AR) head tracking.

Combine with accelerometer for full 6-axis orientation (sensor fusion).

Motion Sensors — Gyroscope in Flutter

```dart
import 'package:sensors_plus/sensors_plus.dart';

// Rotation rate in rad/s
gyroscopeEvents.listen((GyroscopeEvent event) {
  print('RotX: ${event.x}, RotY: ${event.y}, RotZ: ${event.z}');
});

// Gravity sensor (gravity direction only)
gravityEvents.listen((GravityEvent event) {
  print('Gravity X: ${event.x}, Y: ${event.y}, Z: ${event.z}');
});

// Linear acceleration (no gravity)
linearAccelerationEvents.listen((LinearAccelerationEvent event) {
  print('Accel X: ${event.x}, Y: ${event.y}, Z: ${event.z}');
});
```

Motion Sensors — Magnetometer

The **magnetometer** measures the Earth's magnetic field — acts as a **digital compass**.

How it works:

- Measures magnetic field strength in **uT** (microtesla) on 3 axes.
- Hall effect or magnetoresistive sensor.
- Earth's magnetic field is approximately **25-65 uT**.

Use cases:

- Compass apps (direction/heading).
- Map orientation (rotate map to face direction).
- Metal detection (field disturbance).
- Indoor navigation.

> **Warning:** Keep away from magnets and speakers — they distort readings. Calibrate by moving phone in a figure-8 pattern.

Motion Sensors — Magnetometer in Flutter

```dart
import 'package:sensors_plus/sensors_plus.dart';
import 'dart:math';

magnetometerEvents.listen((MagnetometerEvent event) {
  print('Mag X: ${event.x}, Y: ${event.y}, Z: ${event.z}');

  // Calculate compass heading (azimuth)
  final heading = atan2(event.y, event.x) * (180 / pi);
  final normalized = heading < 0 ? heading + 360 : heading;
  print('Compass heading: ${normalized.toStringAsFixed(1)} deg');
});
```

Full orientation (rotation vector):

```dart
rotationVectorEvents.listen((RotationVectorEvent event) {
  print('Rot: X=${event.x}, Y=${event.y}, Z=${event.z}');
});
```

Position Sensors — GPS and Location

GPS (Global Positioning System) provides outdoor location using satellite signals.

How it works:

- Triangulates position from **4+ satellites**.
- Accuracy: **3-5 meters** (open sky), worse indoors.
- Cold start: ~30s to get first fix.
- Assisted GPS (A-GPS) uses cell towers + Wi-Fi for faster fix.

Position Sensors — GPS and Location

Flutter package: `geolocator`

```
dependencies:
  geolocator: ^13.0.2
```

Other location sources:

- **Wi-Fi positioning** — Indoor, ~10-50m accuracy.
- **Cell tower triangulation** — ~100-500m accuracy.
- **BLE beacons** — Indoor, ~1-5m accuracy.

Position Sensors — GPS in Flutter

```dart
import 'package:geolocator/geolocator.dart';
// Check and request permission
LocationPermission perm = await Geolocator.checkPermission();
if (perm == LocationPermission.denied) {
  perm = await Geolocator.requestPermission();
}
// Get current position
Position pos = await Geolocator.getCurrentPosition(
  desiredAccuracy: LocationAccuracy.high,
);
print('Lat: ${pos.latitude}, Lon: ${pos.longitude}');
print('Altitude: ${pos.altitude}m, Speed: ${pos.speed}m/s');
// Stream location updates
Geolocator.getPositionStream(
  locationSettings: LocationSettings(
    accuracy: LocationAccuracy.high,
    distanceFilter: 10, // Update every 10 meters
  ),
).listen((Position p) {
  print('New pos: ${p.latitude}, ${p.longitude}');
});
```

Position Sensors — Geocoding

Convert between coordinates and addresses using `geocoding`.

```yaml
dependencies:
  geocoding: ^3.0.0
```

```dart
import 'package:geocoding/geocoding.dart';

// Coordinates -> Address (Reverse Geocoding)
List<Placemark> placemarks = await placemarkFromCoordinates(
  35.6892, 51.3890, // Tehran
);
print(placemarks.first.street);    // Street name
print(placemarks.first.locality);  // City
print(placemarks.first.country);   // Country

// Address -> Coordinates (Forward Geocoding)
List<Location> locations = await locationFromAddress(
  "Sharif University of Technology, Tehran",
);
print('Lat: ${locations.first.latitude}');
```

Position Sensors — Proximity Sensor

The **proximity sensor** detects if an object is **near** the phone (binary: near/far).

How it works:

- Infrared LED + IR detector.
- Detects objects within **~5 cm**.
- Binary output: **0 (near)** or **1 (far)**.

Primary use case:

- Turn off screen during phone calls (ear proximity).
- Pocket detection (lock screen).

Position Sensors — Proximity Sensor

Flutter package: `proximity_sensor`

```dart
import 'package:proximity_sensor/proximity_sensor.dart';

ProximitySensor.events.listen((bool isNear) {
  if (isNear) {
    print('Object detected near screen');
  } else {
    print('No object near screen');
  }
});
```

Position Sensors — Step Counter

The **step counter** uses accelerometer data to detect walking steps.

How it works:

- Hardware step detector (Android `TYPE_STEP_COUNTER`).
- Persists across sleep — counts total steps since boot.
- Uses pattern recognition on acceleration peaks.

Position Sensors — Step Counter

Flutter packages:

```
dependencies:
  pedometer: ^0.0.6          # Step count + activity recognition
  # OR
  health: ^11.0.0             # Apple Health / Google Fit integration
```

```
import 'package:pedometer/pedometer.dart';

Pedometer.stepCountStream.listen((StepCount count) {
  print('Steps: ${count.steps}');
});

Pedometer.activityStream.listen((ActivityEvent activity) {
  print('Activity: ${activity.type}'); // WALKING, RUNNING, STILL
});
```

Environmental Sensors — Light Sensor

The **ambient light sensor** measures light intensity in **lux** (lx).

How it works:

- Photodiode that measures visible light intensity.
- Range: **0-65,000+ lux**.
- Dark room: ~50 lux, Office: ~300-500 lux, Direct sun: ~100,000 lux.

Use cases:

- **Auto-brightness** — Adjust screen brightness.
- **Camera exposure** — Auto ISO/shutter speed.
- **Smart home** — Turn on lights when dark.

Environmental Sensors — Light Sensor

Flutter package: `light_sensor`

```dart
import 'package:light_sensor/light_sensor.dart';

LightSensor.luxStream.listen((int lux) {
  print('Ambient light: $lux lux');
  if (lux < 50) {
    print('Dark environment');
  } else if (lux > 10000) {
    print('Bright sunlight');
  }
});
```

Environmental Sensors — Barometer

The **barometer** measures atmospheric pressure for altitude estimation.

How it works:

- Measures air pressure in **hPa** (hectopascals).
- Normal sea-level pressure: **1013.25 hPa**.
- Can detect altitude changes of **+/-1 meter**.

Use cases:

- **Floor detection** — Which floor of a building you're on.
- **Weather prediction** — Falling pressure = incoming storm.
- **Altitude for GPS** — Improves GPS vertical accuracy.
- **Hiking/fitness apps** — Elevation tracking.

Environmental Sensors — Barometer

```dart
import 'package:sensors_plus/sensors_plus.dart';
import 'dart:math';

barometerEvents.listen((BarometerEvent event) {
  print('Pressure: ${event.pressure} hPa');
  final alt = 44330 * (1 - pow(event.pressure / 1013.25, 0.1903));
  print('Estimated altitude: ${alt.toStringAsFixed(1)}m');
});
```

Environmental Sensors — Temperature and Humidity

Some devices include **temperature** and **humidity** sensors.

Temperature sensor:

- Measures ambient temperature in **degrees Celsius**.
- Found in some Samsung, Motorola devices.
- Use case: Weather apps, smart home.

Humidity sensor:

- Measures relative humidity in **%RH**.
- Combined with temperature for **dew point** calculation.
- Use case: Comfort index, weather apps.

Environmental Sensors — Temperature and Humidity

Flutter access: Platform channels (not all devices have these).

```
// Android: SensorManager.getDefaultSensor(TYPE_AMBIENT_TEMPERATURE)
// iOS: CMHumiditySensor (limited availability)
```

Biometric Sensors — Fingerprint

Fingerprint sensors provide biometric authentication using fingerprint patterns.

How it works:

- **Capacitive** — Most common. Uses tiny capacitor arrays to map ridges/valleys.
- **Ultrasonic** (Samsung) — Uses sound waves for 3D fingerprint map.
- **Optical** (under-display) — Uses light to capture fingerprint image.

Biometric Sensors — Fingerprint

Flutter package: `local_auth`

```
dependencies:
  local_auth: ^2.3.0
```

Android manifest:

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

iOS Info.plist:

```
<key>NSFaceIDUsageDescription</key>
<string>This app needs biometric access for authentication</string>
```

Biometric Sensors — Fingerprint in Flutter

```dart
import 'package:local_auth/local_auth.dart';
final localAuth = LocalAuthentication();
// Check if device supports biometrics
bool canAuth = await localAuth.canCheckBiometrics;
// OR
bool isSupported = await localAuth.isDeviceSupported();
// Get available biometric types
List<BiometricType> availableBiometrics =
    await localAuth.getAvailableBiometrics();
// [BiometricType.fingerprint, BiometricType.face, BiometricType.weak]
// Authenticate with fingerprint
bool authenticated = await localAuth.authenticate(
  localizedReason: 'Please authenticate to access your data',
  options: AuthenticationOptions(
    stickyAuth: true,
    biometricOnly: true, // No fallback to PIN
  ),
);
if (authenticated) {
  print('User authenticated successfully!');
}
```

Biometric Sensors — Face ID

Face ID (Apple) and **Face Unlock** (Android) use facial recognition.

How it works (Apple Face ID):

- **TrueDepth camera** projects 30,000 IR dots onto face.
- IR camera captures the dot pattern.
- Neural engine matches against stored facial data.
- Works in dark, with glasses, hats, and masks.

How it works (Android Face Unlock):

- Front camera + ML-based face detection.
- Some devices use IR (like Apple), others use RGB only.
- Less secure than fingerprint for payments.

Biometric Sensors — Face ID

Flutter usage: Same `local_auth` package — the OS chooses the method.

```dart
// Same code works for both fingerprint and face
bool auth = await localAuth.authenticate(
  localizedReason: 'Scan your face or fingerprint',
  options: AuthenticationOptions(biometricOnly: true),
);
```

Camera Sensors

Modern phones have **multiple cameras** with various sensor types.

Camera types:

- **Wide** — Standard camera (24-35mm equivalent).
- **Ultrawide** — 120+ degree field of view.
- **Telephoto** — 2-10x optical zoom.
- **Depth (ToF)** — Time-of-flight for depth mapping.
- **IR** — Infrared for Face ID, night vision.

Camera Sensors

Flutter package: `camera`

```
dependencies:
  camera: ^0.11.0
```

```dart
import 'package:camera/camera.dart';

// Get available cameras
final cameras = await availableCameras();
final firstCamera = cameras.first;

// Create controller
final controller = CameraController(
  firstCamera,
  ResolutionPreset.high,
  enableAudio: false,
);
await controller.initialize();

// Take photo
final image = await controller.takePicture();
```

Camera Sensors — AR and Depth

Depth sensors enable Augmented Reality and 3D scanning.

Technologies:

- **LiDAR** (Apple iPad Pro, iPhone Pro) — Light detection and ranging.
- **ToF** (Samsung, others) — Time-of-flight infrared.
- **Stereo vision** — Dual camera depth estimation.

Camera Sensors — AR and Depth

Flutter packages for AR:

```
dependencies:
  ar_flutter_plugin: ^0.7.3    # ARCore/ARKit wrapper
  # OR
  model_viewer_plus: ^1.0.0    # 3D model viewer
```

Use cases:

- **AR gaming** — Pokemon Go, furniture placement (IKEA).
- **Measurement** — Measure distances with camera.
- **3D scanning** — Create 3D models of objects.
- **Navigation** — AR walking directions.
- **Virtual try-on** — Glasses, clothes, makeup.

Microphone Sensor

The **microphone** captures audio for various applications.

How it works:

- MEMS microphone converts sound waves to electrical signals.
- Sample rate: up to **48 kHz** (CD quality).
- Multiple microphones for noise cancellation and spatial audio.

Flutter package: `record` or `flutter_sound`

```
dependencies:
  record: ^5.1.0          # Audio recording
  # OR
  flutter_sound: ^9.0.0 # Recording + playback
```

Microphone Sensor

Use cases:

- Voice recording and notes.
- Speech recognition (STT).
- Music detection (Shazam-like).
- Environmental sound classification.
- Voice commands.

> **Important:** Always request microphone permission before recording.

Connectivity Sensors — NFC

NFC (Near Field Communication) enables short-range wireless communication.

How it works:

- Range: **~4 cm** (touch-to-share).
- Frequency: **13.56 MHz**.
- Data rate: up to **424 kbps**.
- Two modes: **read/write** and **peer-to-peer**.

Use cases:

- **Contactless payments** — Apple Pay, Google Pay.
- **Transit cards** — Tap to ride.
- **Smart posters** — Tap for info.
- **Access cards** — Hotel rooms, offices.
- **Android Beam** (deprecated) / **AirDrop** alternative.

Connectivity Sensors — NFC

Flutter package: `nfc_manager`

```
dependencies:
  nfc_manager: ^3.5.0
```

Connectivity Sensors — NFC in Flutter

```dart
import 'package:nfc_manager/nfc_manager.dart';
// Check NFC availability
bool isAvailable = await NfcManager.instance.isAvailable();
// Start NFC session
NfcManager.instance.startSession(
  onDiscovered: (NfcTag tag) async {
    print('Tag discovered: ${tag.data}');
    // Read NDEF records
    final ndef = Ndef.from(tag);
    if (ndef != null) {
      for (var record in ndef.cachedMessage!.records) {
        print('Payload: ${String.fromCharCodes(record.payload)}');
      }
    }
    // Write NDEF record
    final newMessage = NdefMessage([
      NdefRecord.createText('Hello NFC!'),
    ]);
    await ndef.write(newMessage);
  },
);
```

Connectivity Sensors — Bluetooth

Bluetooth enables short-to-medium range wireless communication.

Bluetooth versions:

- **Classic (BR/EDR)** — Headphones, speakers, file transfer. Range: ~10m.
- **BLE (Bluetooth Low Energy)** — IoT, wearables, beacons. Range: ~100m.
- **Bluetooth 5.0+** — 2x speed, 4x range, 8x broadcast capacity.

Connectivity Sensors — Bluetooth

Flutter packages:

```
dependencies:
  flutter_blue_plus: ^1.32.0   # BLE communication
  # OR
  bluetooth_enable: ^0.0.1     # Enable Bluetooth
```

```dart
import 'package:flutter_blue_plus/flutter_blue_plus.dart';

// Scan for BLE devices
FlutterBluePlus.startScan(timeout: Duration(seconds: 10));
FlutterBluePlus.scanResults.listen((results) {
  for (ScanResult r in results) {
    print('${r.device.platformName}: ${r.rssi} dBm');
  }
});
```

Connectivity Sensors — Wi-Fi and Cellular

Wi-Fi sensing and **cellular** provide additional context.

Wi-Fi:

- RSSI (signal strength) for proximity estimation.
- Wi-Fi RTT (Round-Trip Time) for indoor positioning (~1m accuracy).
- Wi-Fi Direct for peer-to-peer communication.

Cellular:

- Cell tower ID for coarse location.
- Signal strength for coverage mapping.
- 5G enables new AR/VR use cases.

Connectivity Sensors — Wi-Fi and Cellular

Flutter packages:

```
dependencies:
  connectivity_plus: ^6.0.3    # Network status
  network_info_plus: ^6.0.0    # Wi-Fi info
```

```dart
import 'package:connectivity_plus/connectivity_plus.dart';

// Check connectivity
final result = await Connectivity().checkConnectivity();
if (result.contains(ConnectivityResult.wifi)) {
  print('Connected to Wi-Fi');
} else if (result.contains(ConnectivityResult.mobile)) {
  print('Connected to cellular');
}
```

Flutter Sensor Packages Overview

Motion and Position

- `sensors_plus` — Accelerometer, gyroscope, magnetometer, barometer
- `geolocator` — GPS location
- `geocoding` — Address to coordinates
- `proximity_sensor` — Proximity detection
- `pedometer` — Step counting
- `flutter_compass` — Compass heading

Environmental

- `light_sensor` — Ambient light
- `battery_plus` — Battery level/status

Biometric and Security

- `local_auth` — Fingerprint and Face ID
- `nfc_manager` — NFC read/write
- `secure_application` — Screen capture prevention

Media

- `camera` — Camera access
- `record` — Audio recording

Connectivity

- `flutter_blue_plus` — BLE
- `connectivity_plus` — Network status
- `network_info_plus` — Wi-Fi info
- `nearby_connections` — P2P

sensors_plus — Complete API

The `sensors_plus` package provides access to all motion and environmental sensors.

```dart
import 'package:sensors_plus/sensors_plus.dart';

// All available sensor streams:
accelerometerEvents         // Acceleration + gravity (m/s2)
gyroscopeEvents             // Rotation rate (rad/s)
magnetometerEvents          // Magnetic field (uT)
gravityEvents               // Gravity vector only (m/s2)
linearAccelerationEvents    // Acceleration without gravity (m/s2)
rotationVectorEvents        // Device orientation (quaternion)
barometerEvents             // Atmospheric pressure (hPa)

// Each event provides x, y, z values:
accelerometerEvents.listen((event) {
  // event.x, event.y, event.z
});
barometerEvents.listen((event) {
  // event.pressure
});
```

Note: Not all sensors are available on all devices. Always check availability.

geolocator — Complete API

```dart
import 'package:geolocator/geolocator.dart';

// Permission management
LocationPermission perm = await Geolocator.checkPermission();
perm = await Geolocator.requestPermission();
// PermissionStatus: denied, deniedForever, whileInUse, always

// Current position (one-shot)
Position pos = await Geolocator.getCurrentPosition(
  locationSettings: LocationSettings(
    accuracy: LocationAccuracy.high,
    timeLimit: Duration(seconds: 10),
  ),
);

// Position stream (continuous)
Stream<Position> stream = Geolocator.getPositionStream(
  locationSettings: LocationSettings(
    accuracy: LocationAccuracy.high,
    distanceFilter: 5, // min distance between updates
  ),
);

// Distance between two points
double distance = Geolocator.distanceBetween(
  35.6892, 51.3890, // Tehran
  32.6539, 51.6660, // Isfahan
);
print('Distance: ${(distance / 1000).toStringAsFixed(0)} km');
```

local_auth — Biometric Authentication

```dart
import 'package:local_auth/local_auth.dart';
import 'package:local_auth/error_codes.dart' as auth_error;

final localAuth = LocalAuthentication();

// Check device capability
bool canCheck = await localAuth.canCheckBiometrics;
bool deviceSupported = await localAuth.isDeviceSupported();

// List available biometrics
List<BiometricType> biometrics = await localAuth.getAvailableBiometrics();
// BiometricType.fingerprint
// BiometricType.face
// BiometricType.iris (rare)
// BiometricType.weak (Android class 2 biometry)

// Authenticate
try {
  bool auth = await localAuth.authenticate(
    localizedReason: 'Please verify your identity',
    options: AuthenticationOptions(
      stickyAuth: true,       // Resume after app switch
      biometricOnly: false,   // Allow PIN fallback
      useErrorDialogs: true,  // Show error dialogs
    ),
  );
} on PlatformException catch (e) {
  if (e.code == auth_error.notAvailable) {
    // No biometrics enrolled
  }
}
```

Battery and Performance Considerations

Sensor usage impacts battery life significantly.

Best Practices:

- **Unsubscribe** from sensor streams when not needed.
- Use `distanceFilter` for GPS (avoid continuous updates).
- Lower sensor `accuracy` when high precision is not needed.
- Batch sensor readings for processing.
- Use `WorkManager` for background sensor tasks.

Battery Impact by Sensor:

Sensor	Battery Drain
GPS (high accuracy)	**High**
GPS (low accuracy)	Medium
Accelerometer	Low
Gyroscope	Low-Medium
Bluetooth scan	Medium
NFC	Very Low
Camera	**High**

> **Rule:** Only subscribe to sensors when your screen is visible. Unsubscribe in `dispose()`.

Exercise: Sensor Dashboard, Live Wallpaper

Goal: Build a Flutter app that demonstrates ALL sensors with a parallax live wallpaper.

1. **Parallax wallpaper** — Background moves based on accelerometer/gyroscope.
2. **Biometric lock** — Fingerprint/Face ID to unlock sensor dashboard.
3. **Compass** — Real-time compass using magnetometer.
4. **GPS map** — Show current location on a map.
5. **Light meter** — Display ambient light level with auto-theme.
6. **Step counter** — Count steps using pedometer.
7. **Proximity detector** — Detect when phone is near object.
8. **Barometer/altitude** — Show pressure and estimated altitude.
9. **NFC reader** — Read NFC tags.
10. **Shake detection** — Detect phone shake gesture.

Exercise: Encrypted Bank Card App

Objective: Build a notes app with encrypted storage.

Requirements:

1. SQLite database with `sqflite`.
2. Notes model with card number, cvv2, exp.
3. CRUD operations via `CardRepository`.
4. Encryption per card using `flutter_secure_storage`.
5. Search functionality.
6. NFC.
7. Biometric Lock.

Summary: Key Takeaways

Sensor Types

- **Motion:** Accelerometer, Gyroscope, Magnetometer, Gravity, Linear Acceleration.
- **Position:** GPS, Proximity, Step Counter.
- **Environmental:** Light, Barometer, Temperature, Humidity.
- **Biometric:** Fingerprint, Face ID, Iris.
- **Media:** Camera (RGB/Depth/IR), Microphone.
- **Connectivity:** NFC, BLE, Wi-Fi, Cellular.

Best Practices

- Always check sensor availability before subscribing.
- Unsubscribe from sensors in `dispose()`.
- Request runtime permissions properly.
- Use sensor fusion for better accuracy.
- Throttle high-frequency sensor updates for UI.
- Consider battery impact of continuous sensing.
- Handle missing sensors gracefully (fallback UI).

Summary: Key Takeaways

Key Packages

- `sensors_plus` — All motion sensors.
- `geolocator` + `geocoding` — GPS.
- `local_auth` — Biometrics.
- `nfc_manager` — NFC.
- `flutter_blue_plus` — BLE.
- `light_sensor` — Ambient light.
- `camera` — Camera access.

Exercise Highlights

- Parallax wallpaper using accelerometer.
- Biometric lock with fingerprint/Face ID.
- Real-time compass with magnetometer.
- GPS location with geocoding.
- Auto-theme with light sensor.
- NFC tag reader.
- Shake detection gesture.
- Using secure storage in a real life app.

References

- Flutter `sensors_plus`: https://pub.dev/packages/sensors_plus
- Flutter `geolocator`: https://pub.dev/packages/geolocator
- Flutter `geocoding`: https://pub.dev/packages/geocoding
- Flutter `local_auth`: https://pub.dev/packages/local_auth
- Flutter `nfc_manager`: https://pub.dev/packages/nfc_manager
- Flutter `flutter_blue_plus`: https://pub.dev/packages/flutter_blue_plus
- Flutter `camera`: https://pub.dev/packages/camera
- Flutter `light_sensor`: https://pub.dev/packages/light_sensor
- Flutter `connectivity_plus`: https://pub.dev/packages/connectivity_plus
- Flutter `battery_plus`: https://pub.dev/packages/battery_plus

References

- Android Sensor Overview: https://developer.android.com/guide/topics/sensors/sensors_overview
- Apple Core Motion: https://developer.apple.com/documentation/coremotion
- Kalman Filter: https://en.wikipedia.org/wiki/Kalman_filter

Thank You! 🙌

Questions?

"I shook my phone to undo a bug,
then the compass pointed to my ex's house,
and the proximity sensor said 'too close.'
Sensors know too much."
— A developer after building a sensor app