WYKŁAD 6

Obsługa sensorów urządzeń mobilnych

Programowanie Aplikacji Mobilnych
dr inż. Mateusz Pomianek

Android · iOS · Flutter · Sensor Fusion · Privacy

Plan wykładu

Sensory urządzeń mobilnych —> definicja i taksonomia

Sensor to przetwornik przekształcający **wielkość fizyczną** na **sygnał elektryczny**.
Smartfon zawiera 30–40+ sensorów sprzętowych i wirtualnych.

Ruch i orientacja	Środowiskowe	Pozycja	Biometryczne	Obrazowe
Akcelerometr	Barometr	GPS (GNSS)	Odcisk palca	Kamera RGB
Żyroskop	Termometr	Magnetometr	Face ID	Kamera ToF
Obrotomierz	Higrometr	Proximity	Tętno (PPG)	Kamera termiczna
Sensor kroków	Światłomierz	NFC	SpO_2	LiDAR (Pro)
Pochylenie	Sensor UV	Wi-Fi/BLE beacon	EEG	Iris scanner

40+
Sensorów iPhone 15 Pro

70%
Smartfonów z combined IMU

1 kHz
Max częstotliwość próbkowania IMU

< 1mA
CHRE zużycie mocy (always-on)

Sensory wirtualne (fusion) = obliczane z kilku źródeł sprzętowych, np. Rotation Vector (acc+gyro+mag) · **Linear Acceleration** · **Gravity** · **Orientation**

Architektura warstwowa Android Sensor Framework

Android Sensor Framework (android.hardware) standaryzuje dostęp do wszystkich sensorów przez jednolite API, niezależnie od producenta sprzętu.

Aplikacja (Java/Kotlin/Native)	SensorManager · SensorEventListener · SensorEvent · SensorAdditionalInfo
Android Framework	SensorService (system_server) · Sensor Fusion Engine · Calibration Service · CHRE
HAL 2.0 (AIDL/HIDL)	ISensors → ISensorsCallback · Batch/Flush protocol · Direct Channel (shared memory)
Kernel: Linux IIO Subsystem	Industrial I/O drivers · iio:device → sysfs attributes · DMA ring buffer
Sprzęt MEMS + SoC DSP/DSP Hub	MEMS die (silicon springs) · ADC (Σ-Δ) · Dedicated DSP core (CHRE/M-coprocessor)

CHRE (Context Hub Runtime Environment): dedykowany low-power DSP: przetwarza sensory gdy CPU uśpiony. Pobór < 1 mA. Kluczowy dla Always-On (pedometr, OK Google, crash detection).

Android SensorManager: rejestracja i odczyt sensorów

Każda **aplikacja** wchodzi w **interakcję z sensorami** przez **SensorManager**.
To centralny **punkt dostępu**, unikalny per **kontekst systemowy**.

Kluczowe klasy API

SensorManager

getSystemService(SENSOR_SERVICE). Metody: getDefaultSensor(type), registerListener(), unregisterListener(). Jeden na aplikację.

Sensor

Obiekt czujnika. Właściwości: getType(), getName(), getVendor(), getResolution(), getMaximumRange(), getPower() [mA].

SensorEvent

Zdarzenie: values[] (float[]), sensor, accuracy (0–3), timestamp [ns od boot]. Jedno zdarzenie per wywołanie callbacku.

Częstotliwości próbkowania

NORMAL (200ms) · GAME (20ms) · FASTEST (0ms, max HW). Android 12+: żądana szybkość to wskazówka: nie gwarancja HW.

Accuracy Callback

onAccuracyChanged() → ACCURACY_HIGH / MEDIUM / LOW / UNRELIABLE. Kluczowe dla magnetometru: kalibracja ósemką.

Implementacja SensorEventListener

Kotlin

```kotlin
class SensorActivity : AppCompatActivity(),
    SensorEventListener {

  private lateinit var sensorMgr: SensorManager
  private var accel: Sensor? = null

  override fun onCreate(b: Bundle?) {
    super.onCreate(b)
    sensorMgr = getSystemService(SENSOR_SERVICE)
        as SensorManager
    accel = sensorMgr
        .getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
  }

  override fun onResume() {
    super.onResume()
    accel?.also { sensor ->
      sensorMgr.registerListener(
        this, sensor, SENSOR_DELAY_GAME,
        5_000_000L // maxReportLatencyUs = 5 s batch
      )
    }
  }
  override fun onPause() {   // ZAWSZE odrejestruj!
    super.onPause()
    sensorMgr.unregisterListener(this)
  }
```

Architektura iOS Core Motion Framework

Apple Core Motion unifikuje dostęp do IMU. Akcelerometr, żyroskop, magnetometr, barometr i pedometr przez jeden obiekt **CMMotionManager**.

CMMotionManager

Centralny obiekt IMU. Jeden na aplikację (singleton). Zarządza startowaniem i zatrzymywaniem wszystkich czujników IMU.

CMDeviceMotion

Fuzja: attitude (roll/pitch/yaw), rotationRate, gravity, userAcceleration, magneticField. Najlepsze API do ogólnego użytku.

CMAccelerometerData

Surowe dane akcelerometru: acceleration.x/y/z [g]. Zawiera grawitację + przyspieszenie użytkownika łącznie.

CMGyroData

Surowy żyroskop: rotationRate.x/y/z [rad/s]. Dostępny od iPhone 4. Bez fuzji akumuluje drift kątowy.

CMAltimeter

Barometr: relativeAltitude [m od startu], pressure [kPa]. Zawsze aktywny (CHRE equivalent). Bardzo low-power.

CMPedometer

Kroki, dystans, piętro (floorsAscended/Descended), pace, cadence. Historia do 7 dni wstecz. Sensor zdrowotny.

CMMotionActivityManager

Rozpoznawanie aktywności: stationary, walking, running, automotive, cycling, unknown. Confidence: low/medium/high.

CMHeadphoneMotionManager

AirPods Pro/Max: orientacja głowy jako IMU. Head tracking dla Spatial Audio i nowych form interakcji.

Motion Coprocessor (Apple M-series): przetwarza sensory niezależnie od CPU/GPU: Always-On fitness tracking < 1 mA. Dane zdrowotne trafiają do **HealthKit**.

iOS Core Motion: implementacja Swift i układy odniesienia

CMMotionManager oferuje dwa modele odczytu:
polling (synchroniczny pull) i **bloki closure** (asynchroniczny push): push jest rekomendowany.

CMDeviceMotion: start i handler

```swift
let motion = CMMotionManager()
let queue  = OperationQueue()
func startSensors() {
  guard motion.isDeviceMotionAvailable else { return }
  motion.deviceMotionUpdateInterval = 1.0 / 60.0
  motion.startDeviceMotionUpdates(
    using: .xArbitraryZVertical, to: queue
  ) { [weak self] data, error in
    guard let d = data, error == nil else { return }
    DispatchQueue.main.async {
      self?.roll  = d.attitude.roll
      self?.pitch = d.attitude.pitch
      self?.yaw   = d.attitude.yaw
      self?.userAccel = d.userAcceleration
    }
  }
}
func stopSensors() {
  motion.stopDeviceMotionUpdates()
}
```

⚠ Zatrzymaj sensor w viewDidDisappear() / sceneDidEnterBackground()
aktywny sensor blokuje uśpienie CPU.

Pull vs Push

Pull: startAccelerometerUpdates() → Timer → accelerometerData? Prostszy, może miss zdarzeń. Push: startAccelerometerUpdates(to:queue:handler:): rekomendowane.

Układy odniesienia (reference frames)

xArbitraryZVertical

Oś Z pionowo (grawitacja). X dowolnie: nie wymaga magnetometru. Zalecane dla gier i ogólnych zastosowań.

xMagneticNorthZVertical

X → magnetyczna Północ. Wymaga sprawnego magnetometru. Dla kompasów i map.

xTrueNorthZVertical

X → geograficzna Północ (korekcja deklinacji). Wymaga dostępu do lokalizacji GPS.

xArbitraryCorrectedZVertical

Jak xArbitrary ale z korekcją dryfu: dokładniejszy długoterminowo, wyższy koszt.

CMAttitude.multiply(byInverse:)

Oblicz delta-orientację względem punktu referencyjnego: attitude.multiply(byInverse: referenceAttitude) → relative rotation.

Akcelerometr: fizyka, filtrowanie i fastosowania

Akcelerometr mierzy przyspieszenie właściwe: siłę na jednostkę masy, włącznie z grawitacją. Wynik w m/s² lub g (1g ≈ 9.81 m/s²).

Budowa MEMS

Beleczki (proof mass) zawieszone na sprężynach krzemowych. Przyspieszenie → ugięcie → zmiana pojemności kondensatora → ADC → wartość cyfrowa.

Układ osi (right-handed)

X: poziomo (right), Y: pionowo (up), Z: prostopadle do ekranu (toward user). Spoczynek Android: [0, 0, 9.81] m/s². iOS: [0, 0, 1] g: inna konwencja!

Separacja grawitacji

Linear Acceleration = Total − Gravity. Android TYPE_LINEAR_ACCELERATION (wirtualny). iOS: d.userAcceleration. Konieczne filtrowanie low-pass.

Bias i kalibracja

Bias (zero-g offset): stały błąd systematyczny. Temperature drift: zmiana biasu z temperaturą. Fabrycznie kalibrowane, drift rośnie z wiekiem.

Zastosowania

Rotacja ekranu · Detekcja upadku · Pedometr · Tilt steering gry · Crash detection (iOS 16) · Shake gesture · Fitness intensity (METs).

Filtr dolno- i górnoprzepustowy

Kotlin

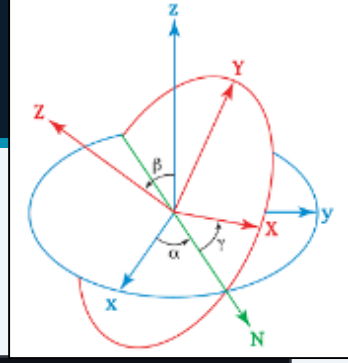
```kotlin
// Low-pass: izolacja grawitacji
const val ALPHA = 0.8f  // wyższe = więcej smoothing
var gravity = FloatArray(3)

fun separateGravity(e: SensorEvent) {
  gravity[0] = ALPHA*gravity[0] + (1-ALPHA)*e.values[0]
  gravity[1] = ALPHA*gravity[1] + (1-ALPHA)*e.values[1]
  gravity[2] = ALPHA*gravity[2] + (1-ALPHA)*e.values[2]
  // High-pass: izolacja ruchu użytkownika
  val linX = e.values[0] - gravity[0]
  val linY = e.values[1] - gravity[1]
  val linZ = e.values[2] - gravity[2]
  // Tilt: kąt pochylenia urządzenia
  val pitch = atan2(-linX,
      sqrt(linY*linY + linZ*linZ))
  val roll  = atan2(linY, linZ)
}
```

Detekcja potrząśnięcia (Shake)

Kotlin

```kotlin
val SHAKE_THRESHOLD = 2.5f // g
val magnitude = sqrt(x*x + y*y + z*z) / GRAVITY
if (magnitude > SHAKE_THRESHOLD) onShakeDetected()
```

Żyroskop: prędkość kątowa, kwaterniony i AHRS



Żyroskop mierzy **prędkość kątową** (angular velocity) wokół **3 osi** w rad/s.
Całkowanie daje kąt: ale z driftem, który rośnie w czasie.

Zasada Efektu Coriolisa

Wibrująca masa rezonuje z freq. naturalną. Obrót → siła Coriolisa → prostopadłe ugięcie → pomiar pojemnościowy. Rozdzielczość: 0.01°/s.

Drift żyroskopu: problem

Całkowanie rad/s → kąt. Błąd biasu kumuluje się: 0.1°/s drift → po 10 min błąd 60°! Konieczna fuzja z akcelerometrem lub magnetometrem.

Kwaterniony vs Euler Angles

Kąty Eulera (roll, pitch, yaw) → gimbal lock przy pitch=±90°. Kwaterniony q=[w,x,y,z] |q|=1: pozbawione degeneracji. Standard VR/AR/lotnictwo.

TYPE_ROTATION_VECTOR

Sensor wirtualny Android: kwaternion z fuzji acc+gyro+mag. Najdokładniejsza orientacja. TYPE_GAME_ROTATION_VECTOR: bez mag, stabilniejszy przy magnesach.

Zastosowania

VR/AR head tracking · OIS stabilizacja obrazu · Sterowanie kamerą w grach · Handwriting angle · Dead reckoning nawigacja bez GPS.

Odczyt Rotation Vector i kwaternionów

Kotlin

```kotlin
// TYPE_ROTATION_VECTOR → macierz → kąty Eulera
override fun onSensorChanged(e: SensorEvent) {
  if (e.sensor.type == Sensor.TYPE_ROTATION_VECTOR) {
    val rotMat = FloatArray(9)
    SensorManager.getRotationMatrixFromVector(
        rotMat, e.values)
    val orientation = FloatArray(3)
    SensorManager.getOrientation(rotMat, orientation)
    val azimuth = Math.toDegrees(orientation[0].toDouble())
    val pitch   = Math.toDegrees(orientation[1].toDouble())
    val roll    = Math.toDegrees(orientation[2].toDouble())
    // Lub bezpośrednio kwaternion:
    val quat = FloatArray(4)
    SensorManager.getQuaternionFromVector(quat, e.values)
    // quat = [w, x, y, z]
  }
}
```

Systemy orientacji (IMU → AHRS → INS):

IMU	AHRS	INS
acc+gyro: drift bez referencji zewnętrznej	IMU+mag: pełna orientacja z kursem	AHRS+GPS: fuzja kursów i pozycji

Magnetometr: kompas cyfrowy i pole magnetyczne

Magnetometr (AMR lub Hall-effect) mierzy natężenie pola magnetycznego Ziemi w µT: umożliwia określenie kierunku geograficznego.

Zasada i wyzwania pomiarowe:

Pole magnetyczne Ziemi

Wartość: 25–65 µT (zależy od lokalizacji). 3-osiowy pomiar (Bx, By, Bz). Inklinacja: kierunek pola ≠ poziom (ważne dla tilt-compensated compass).

Hard Iron Distortion

Stałe magnesy w urządzeniu (głośnik, aparat, silniki wibracji): addytywny bias. Kalibracja: baza stała: eliminowana przez zebranie danych i wyznaczenie offset.

Soft Iron Distortion

Materiały ferromagnetyczne: zniekształcenie kształtu pola (elipsa zamiast koła). Korekcja macierzą 3×3. Kalibracja figure-8 zbiera próbki z wszystkich orientacji.

Deklinacja magnetyczna

Kąt między magnetyczną a geograficzną Północą (0–30°). Android: GeomagneticField(lat,lon,alt,time).getDeclination(). Konieczne dla true north heading.

```kotlin
val azimuth = atan2(By, Bx) * 180f / PI  // -180..180
val bearing = (azimuth + 360f) % 360f     // 0..360°
```

Android i iOS: API magnetometru:

Kotlin/Swift

```kotlin
// Android: sensor types
// TYPE_MAGNETIC_FIELD          → surowe µT
// TYPE_MAGNETIC_FIELD_UNCALIBRATED → + bias estymacja

// Tilt-compensated kompas (acc + mag)
val R = FloatArray(9); val I = FloatArray(9)
SensorManager.getRotationMatrix(R, I,
    gravityValues, geomagneticValues)
val orientation = FloatArray(3)
SensorManager.getOrientation(R, orientation)
val azimuth = Math.toDegrees(
    orientation[0].toDouble())  // 0..360°

// iOS: CLLocationManager
locationManager.startUpdatingHeading()
func locationManager(_ m: CLLocationManager,
    didUpdateHeading h: CLHeading) {
  let magnetic = h.magneticHeading   // 0..360
  let trueNorth = h.trueHeading      // wymaga GPS
  let accuracy  = h.headingAccuracy  // stopnie
}
```

Zastosowania magnetometru

Kompas · Indoor navigation (geomagnetic fingerprinting) · Detekcja metalowych obiektów · NFC trigger · Czy telefon na metalowej powierzchni.

Ograniczenia środowiskowe

Elektronika, stale budynków, magnesy: odchylenie do 50° w biurowcach. Zawsze sprawdzaj headingAccuracy przed użyciem!

Barometr i sensory środowiskowe: ciśnienie, światło, proximity

Sensory środowiskowe dostarczają kontekstu fizycznego otoczenia: kluczowe dla fitness, smart home i adaptacyjnego UX.

Barometr (ciśnienie atmosferyczne):

Zasada pomiaru

MEMS piezooporowy: membrana ugina się pod ciśnieniem → zmiana rezystancji. Zakres: 300–1100 hPa. Rozdzielczość: 0.1 Pa → ΔZ ≈ 8 mm.

Altitude estimation

H = 44330×(1−(P/P_0)^0.1903) [m]. P_0=1013.25 hPa. Dokładność ±1 m przy stabilnych warunkach. Kalibruj GPS: drift z pogodą.

iOS Floors Climbed

CMPedometer.floorsAscended/floorsDescended: Apple oblicza piętro z barometru. Dane w HealthKit. Wymagany sensor ciśnienia.

Kotlin/Swift

```kotlin
// Android
Sensor.TYPE_PRESSURE  // event.values[0] [hPa]
val alt = SensorManager.getAltitude(
    SensorManager.PRESSURE_STANDARD_ATMOSPHERE, hPa)
// iOS
altimeter.startRelativeAltitudeUpdates(to: .main) {
  d, _ in
  let relAlt  = d!.relativeAltitude.floatValue // [m]
  let pressure = d!.pressure.floatValue        // [kPa]
}
```

Sensor światła (ALS)

TYPE_LIGHT [lux]. Auto-brightness ekranu. Zakres: 0–100 000 lux. Odpowiedź < 50 ms.

Sensor proximy

TYPE_PROXIMITY: 0 (blisko) / 5 cm (daleko): binarny. IR LED + IR detektor. Wyciszenie ekranu podczas połączenia.

Sensor temperatury

TYPE_AMBIENT_TEMPERATURE: wycofany z większości urządzeń. ThermalManager (Android 10+) monitoruje SoC.

UV Index Sensor

Rzadki: Samsung Galaxy S. Mierzy UV 0–11+. Typowo w smartwatch i wearables dla sport apps.

Sensor wilgotności **TYPE_RELATIVE_HUMIDITY**: rzadki na telefonach, częstszy w Samsung Galaxy S i smartwatch oraz czujnikach IoT.

Geomagnetic Rotation Vector

TYPE_GEOMAGNETIC_ROTATION_VECTOR: orientacja z acc+mag bez żyroskopu. Mniejsze zużycie baterii kosztem dokładności dynamicznej.

GPS i GNSS: lokalizacja satelitarna

GPS (USA) jest częścią GNSS: obejmuje GPS, GLONASS (Rosja), Galileo (EU) i BeiDou (Chiny). Multi-constellation → lepsza dokładność i availability.

Zasada trilateracji

Min. 3 satelity → pozycja 3D. 4. satelita → eliminacja błędu zegara odbiornika. Więcej → wyższa dokładność (PDOP).

TTFF (Time To First Fix)

Cold start: 45–90 s (pobieranie almanach). Warm: 5–30 s. Hot: < 2 s. A-GPS (Assisted) skraca cold TTFF < 5 s przez sieć.

Dokładność smartfona

CEP 3–5 m przy otwartym niebie. SBAS (WAAS/EGNOS) < 1 m. Multipath w miastach: 10–50 m. Dual-Frequency (L1+L5) < 1 m.

Dual-Frequency L1+L5

iPhone 15 Pro, Pixel 6+: korekcja jonosferyczna: precyzja < 1 m w terenie miejskim. Przyszłość precyzyjnej nawigacji mobilnej.

Błędy i zakłócenia

Multipath (odbicia), spoofing (fałszywe sygnały), efekty troposferyczne/ jonosferyczne, canyon miejski. Wnioski z jakości fix: HDOP, numSatellites.

Fused Location Provider (Android)

Kotlin

```kotlin
val fusedClient = LocationServices
    .getFusedLocationProviderClient(this)

val request = LocationRequest.Builder(
    Priority.PRIORITY_HIGH_ACCURACY, 1000L)
  .setMinUpdateDistanceMeters(1f)
  .setWaitForAccurateLocation(true)
  .build()

val callback = object : LocationCallback() {
  override fun onLocationResult(r: LocationResult) {
    val loc = r.lastLocation ?: return
    val lat  = loc.latitude      // stopnie
    val lon  = loc.longitude     // stopnie
    val alt  = loc.altitude      // metry MSL
    val acc  = loc.accuracy      // CEP 68% [m]
    val spd  = loc.speed         // [m/s]
    val bear = loc.bearing       // kurs [°]
  }
}
fusedClient.requestLocationUpdates(
    request, callback, mainLooper)
// ZAWSZE zatrzymaj:
fusedClient.removeLocationUpdates(callback)
```

iOS CLLocationManager

requestWhenInUseAuthorization() / requestAlwaysAuthorization(). desiredAccuracy: kCLLocationAccuracyBest. distanceFilter [m] minimalizuje wywołania.

Lokalizacja sieciowa i Indoor Positioning

GPS zawodzi w budynkach i gęstej zabudowie: hybrydowe techniki lokalizacji kompensują ograniczenia każdego z systemów.

Wi-Fi Positioning (WPS)

Dok: 5–40 m

RSSI triangulacja z baz AP w Google/Apple Location DB. Dokładność: 5–40 m. Fingerprinting: survey bazą danych AP.

Cell Tower Triangulation

Dok: 100–2000 m

MCC+MNC+LAC+CellID → baza OpenCellID/Google. Fallback gdy GPS/Wi-Fi niedostępne. Działa zawsze (GSM).

BLE Beacon Proximity

Dok: 1–5 m

iBeacon / Eddystone. RSSI → dystans path loss model. Indoor muzeum, lotnisko, centrum handlowe. < 5 m.

Geomagnetic Fingerprinting

Dok: 1–3 m

Mapa anomalii pola magnetycznego budynku. Bez infrastruktury: tylko baza fingerprints i magnetometr telefonu.

Ultra-Wideband (UWB)

Dok: < 30 cm

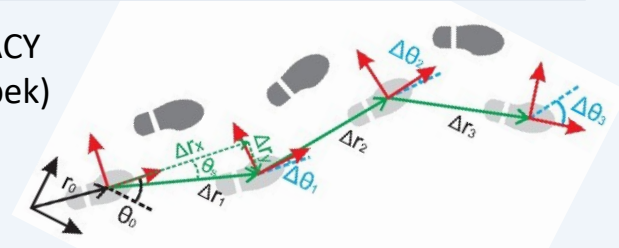
iPhone 11+ (chip U1). TOF (Time of Flight) < 30 cm precyzji. AirDrop, Find My, AirTag. Wymaga UWB peer.

Dead Reckoning (PDR)

Dok: 2–10 m drift

Pedestrian Dead Reckoning: krok + kierunek z IMU. Zliczanie kroków + azymut z kompasu → track. Błąd 1–5%.

Fused Location Priority: HIGH_ACCURACY (GPS+all) · BALANCED_POWER_ACCURACY (Wi-Fi+cell) · LOW_POWER (cell only) · NO_POWER (passive: korzysta z fix innych apek)



Fuzja sensorów: filtr komplementarny i teoria kalmana

Sensor fusion to algorytmiczne **łączenie** danych z **wielu czujników**.
Każdy sensor ma komplementarne wady i zalety, razem **dają dokładniejszy wynik**.

Filtr Komplementarny: prosta i skuteczna fuzja

Idea dwóch komplementarnyh źródeł

Akcelerometr: stabilny długoterminowo, szumny w dynamicznym ruchu. Żyroskop: szybki i precyzyjny krótkoterminowo, drift długoterminowy. Waga α łączy oba.

Kotlin

```kotlin
// α dobierane empirycznie: 0.96–0.99
val α = 0.98f; val dt = 0.02f  // 50 Hz
// Akcelerometr → kąt (długoterminowo stabilny)
val accelAngle = atan2(ay, az) * 180f / PI.toFloat()
// Żyroskop → całkowanie (krótkoterminowo precyzyjny)
gyroAngle += gyroPitch * dt
// Fuzja: α * gyro + (1-α) * accelerometer
angle = α * (angle + gyroPitch * dt) + (1-α) * accelAngle
```

Madgwick / Mahony Filter

Zaawansowane filtry komplementarne w kwaternionach, czyli poprawnie obsługuje gimbal lock. AHRS standard w dronach, VR i robotyce. β parametr reguluje gain.

Ograniczenia filtra komplementarnego

Stałe α: nie adaptuje się do dynamiki. Nie modeluje szumu pomiarowego. Wystarczający dla 90% aplikacji mobilnych *(gry, UI, fitness)*.

Filtr Kalmana: optymalna fuzja bayesowska

Idea filtru Kalmana

Bayesowski estymator stanu: predykcja modelem ruchu → korekta pomiarem. Minimalizuje błąd MMSE przy gaussowskim szumie procesowym i pomiarowym.

1. Predykcja stanu

$$\hat{x}_k^- = F_k \cdot \hat{x}_{k-1} + B_k \cdot u_k$$
$$P_k^- = F_k \cdot P_{k-1} \cdot F_k T + Q_k$$

2. Kalman Gain

$$K_k = P_k^- \cdot H_k T \cdot (H_k \cdot P_k^- \cdot H_k T + R_k)^{-1}$$

3. Korekcja pomiarem

$$\hat{x}_k = \hat{x}_k^- + K_k \cdot (z_k - H_k \cdot \hat{x}_k^-)$$
$$P_k = (I - K_k \cdot H_k) \cdot P_k^-$$

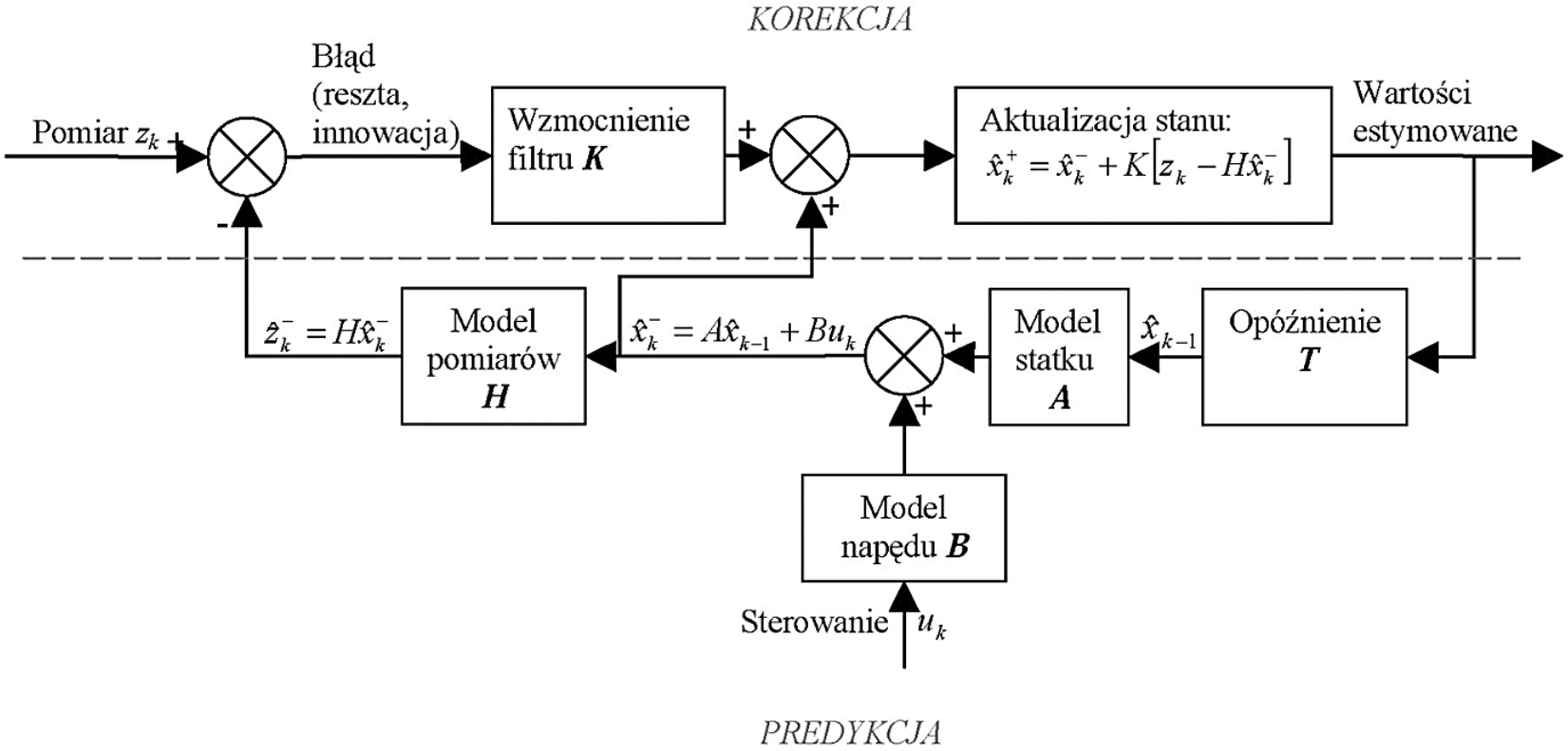
EKF: Extended Kalman Filter

Dla nieliniowych systemów (GPS+IMU): linearyzacja przez Jakobiany. Używany w: Google Maps dead reckoning, autonomiczne drony, AR HMD.

UKF i Particle Filter

Unscented KF *(sigma points)* i Monte Carlo: silnie nieliniowe systemy. Wyższy koszt obliczeniowy. SLAM, robotyka mobilna.

Fuzja sensorów: filtr komplementarny i teoria kalmana



Źródło: Paweł Zalewski, „Modele z filtrem kalmana i rozmyte w systemach dynamicznego pozycjonowania”, 411, Górnictwo i Geoinżynieria • Rok 35 • Zeszyt 4/1 • 2011

Orientacja 3D: kwaterniony, SLERP i reprezentacje obrotu

Precyzyjna orientacja 3D wymaga kwaternionów. To odpowiedź na gimbal lock kątów Eulera i niestabilność macierzy rotacji.

Definicja kwaternionu jednostkowego

q = w + xi + yj + zk, gdzie $w^2+x^2+y^2+z^2=1$. Obrót o θ wokół osi n: q=[cos(θ/2), sin(θ/2)·n]. Pozbawiony gimbal lock.

Konwersja Kwaternion → Euler

roll = atan2(2(wy+xz), 1−2(y^2+z^2)) pitch = arcsin(2(wy−zx))
yaw = atan2(2(wz+xy), 1−2(x^2+y^2))

SLERP: Spherical Linear Interpolation

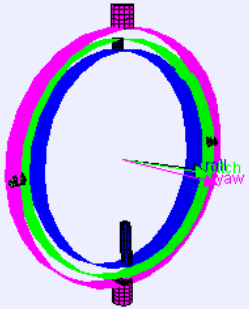
Interpolacja między orientacjami po sferze jednostkowej:
$q(t) = q_0 \cdot (q_0^{-1} \cdot q_1)^t$. Smooth animacja kamery, korekcja VR.

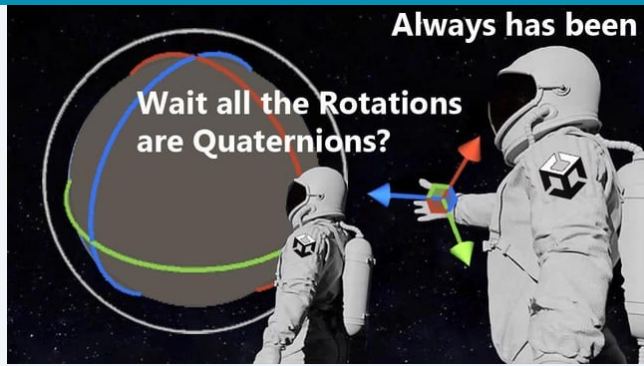
Gimbal Lock: dlaczego jest problemem

Przy pitch=±90° dwie osie obrotowe pokrywają się → utrata stopnia swobody. Widoczne w modelowaniu 3D i stabilizatorach gimbal. Kwaterniony: brak tego problemu.

Kotlin

```kotlin
// Odczyt kwaternionu orientacji (Android)
if (e.sensor.type == Sensor.TYPE_ROTATION_VECTOR) {
  val q = FloatArray(4)
  SensorManager.getQuaternionFromVector(q, e.values)
  // q[0]=w, q[1]=x, q[2]=y, q[3]=z
}
```





Porównanie reprezentacji orientacji

Kąty Eulera

✓ Intuicyjne, czytelne dla człowieka
✗ Gimbal lock, zależność od kolejności obrotu

Macierz rotacji

✓ Szybkie transformacje wektorów (SIMD)
✗ 9 wartości, 6 redund.: normalizacja trudna

Kwaternion

✓ Brak gimbal lock, płynna interpolacja SLERP
✗ Nieintuicyjny, podwójna okładka (q ≡ -q)

Axis-Angle

✓ Geometrycznie jasny: oś + kąt
✗ Nieciągły przy 0° i 360°, trudny do składania

Kamera jako sensor: Camera2 API i Computer Vision

CameraX (Jetpack): ImageAnalysis

Kotlin

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

imageAnalysis.setAnalyzer(executor) { proxy ->
  val bitmap = proxy.toBitmap()
  val rotation = proxy.imageInfo.rotationDegrees
  val input = InputImage.fromBitmap(bitmap, rotation)

  // ML Kit barcode scanning
  barcodeScanner.process(input)
    .addOnSuccessListener { barcodes ->
        barcodes.forEach { processBarcode(it) }
    }
    .addOnCompleteListener { proxy.close() }
}
```

Camera2 vs CameraX

Camera2 (API 21+): pełna kontrola ISO, shutter, AF/AE/AWB, RAW.
CameraX: wysoki poziom abstrakcji, lifecycle-aware, rekomendowane dla ML. AVFoundation (iOS): analogicznie.

Formaty pikseli

YUV_420_888: wydajny, 3 płaszczyzny, konieczna konwersja do Bitmap.
RGBA_8888: bezpośrednio Bitmap, duże bufory pamięci.
JPEG: gotowy, wolniejszy.

Kamera mobilna ewoluowała w universalny sensor. Dokumenty, AR, ML, depth sensing i health monitoring. Wszystko z jednego układu optycznego.

Computer Vision: zastosowania on-device

OCR (Tekst)
ML Kit Text Recognition v2: 119 języków. Off-line, < 50 ms. Latin, cyrylica, CJK.

Barcode / QR
ML Kit Barcode: 1D/2D, < 30 ms. AVMetadataObject (iOS). Płatności, linki, AR triggery.

Object Detection
TFLite SSD MobileNet v2: 80 klas. 30+ FPS na mid-range. Tracking mode dla ciągłości.

Pose Estimation
ML Kit Pose: 33 keypoints ciała. Fitness, rehabilitacja, dance games, sports analysis.

Face Detection/Mesh
ML Kit Face: landmarks, contours, emocje. MediaPipe Face Mesh: 468 punktów twarzy.

Depth / LiDAR / ToF
iPhone Pro LiDAR: ARKit MeshAnchors. Google ToF: depth maps. ARCore depth fusion.

ARCore, ARKit i Visual-Inertial Odometry

AR łączy kamerę z IMU i LiDAR (world tracking) aby umieszczać obiekty wirtualne w przestrzeni fizycznej z milimetrową dokładnością.

ARKit (iOS 11+)

- World Tracking: Visual-Inertial Odometry
- Plane Detection (horizontal, vertical, angled)
- LiDAR Scene Reconstruction (iPhone Pro)
- Image/Object Tracking: ReferenceImage
- Face Tracking: TrueDepth 52-pt mesh
- Depth API: ARDepthData + smoothing

ARCore (Android)

- Motion Tracking: Feature Points + IMU
- Plane Detection + anchors na powierzchniach
- Light Estimation: HDR environment probes
- Depth API: Depth Image (phone cameras)
- Cloud Anchors: wspólne AR experiences
- Geospatial API: Street View 3D alignment

WebXR (browser)

- Hit Testing: ray casting do sceny AR
- DOM Overlay: HTML nad warstwą AR
- Immersive-ar + Immersive-vr sessions
- Light Estimation API (eksperymentalne)
- Anchors API dla persystencji sceny
- Depth Sensing (Chrome flag)

Visual-Inertial Odometry (VIO) to serce każdego systemu AR

Feature Tracking

FAST/ORB punkty śledzone między klatkami kamery: relative motion pose estimation.

IMU Pre-integration

Gyro+acc @ 1000 Hz między klatkami 30 Hz kamery: eliminacja motion blur.

Bundle Adjustment

Optymalizacja mapy keyframes i trajektorii (g2o/Ceres). Pełna rekonstrukcja sceny.

LiDAR Mesh Fusion

iPhone Pro LiDAR @ 15 fps → ARMeshAnchor: rzeczywiste siatki 3D dla okludowania.

Programowanie reaktywne: strumienie danych sensorów

Dane sensorów to strumienie zdarzeń. Reaktywne podejście (Kotlin Flow, Swift AsyncStream, RxJava) redukuje callback hell i upraszcza transformacje.

Kotlin Flow: sensor jako callbackFlow

Kotlin

```kotlin
fun SensorManager.sensorFlow(
  type: Int,
  period: Int = SENSOR_DELAY_GAME
): Flow<SensorEvent> = callbackFlow {
  val listener = object : SensorEventListener {
    override fun onSensorChanged(e: SensorEvent) {
      trySend(e)  // non-blocking
    }
    override fun onAccuracyChanged(s: Sensor, a: Int) {}
  }
  val sensor = getDefaultSensor(type)
    ?: throw SensorUnavailableException(type)
  registerListener(listener, sensor, period)
  awaitClose { unregisterListener(listener) }
}

// Użycie w ViewModel
val accelState = sensorMgr
  .sensorFlow(Sensor.TYPE_ACCELEROMETER)
  .map { Triple(it.values[0], it.values[1], it.values[2]) }
  .debounce(100)
  .stateIn(viewModelScope,
      SharingStarted.WhileSubscribed(), null)
```

iOS Swift: AsyncStream / Combine

Swift

```swift
// AsyncStream wrapper
func accelStream()
    -> AsyncStream<CMAccelerometerData> {
  AsyncStream { cont in
    motion.accelerometerUpdateInterval = 1.0/50
    motion.startAccelerometerUpdates(to:.main) {
      d, _ in guard let d else { return }
      cont.yield(d)
    }
    cont.onTermination = { _ in
      self.motion.stopAccelerometerUpdates()
    }
  }
}
// Użycie async/await
@MainActor func listen() async {
  for await data in accelStream() {
    let mag = sqrt(
      pow(data.acceleration.x, 2) +
      pow(data.acceleration.y, 2) +
      pow(data.acceleration.z, 2))
    if mag > threshold { handleShake() }
  }
}
```

Kluczowe operatory reaktywne dla sensorów

`map`	surowe ADC → jednostki fizyczne	`filter`	odrzuć zdarzenia < threshold
`debounce`	max 1 UI update na 50 ms	`distinctUntilChanged`	emituj gdy wartość się zmieniła
`scan`	rolling average, kąt z całkowania	`combineLatest`	fuzja acc+mag → kompas

RxJava dla Android

Observable.create() + SensorEventListener. Operatory: observeOn(AndroidSchedulers.mainThread()), throttleLast(50, MILLISECONDS), map, filter.

Thread Safety: ważne

Dane sensora przychodą na dowolnym wątku. Zawsze dispatch na main thread przed aktualizacją UI. Kotlin: flowOn(Dispatchers.Main). Swift: DispatchQueue.main.async.

DSP na urządzeniu mobilnym: filtrowanie sygnałów sensorów

Surowe dane sensorów zawierają szum, drift i artefakty.
cyfrowe przetwarzanie sygnałów (DSP) jest konieczne dla użytecznych pomiarów aplikacji.

Moving Average SMA/EMA

SMA: $\bar{x}_n=(1/N)\cdot\Sigma x_n$. EMA: $x_n=\alpha\cdot x_n+(1-\alpha)\cdot x_{n-1}$. EMA adaptuje szybciej. O(1) memory. Wygładzanie kroków, altitude. Opóźnienie N/2 próbek (SMA).

Butterworth IIR Filter

Maksymalnie płaskie pasmo przepustowe. Biquad sekcje (SOS). Obliczanie współczynników offline w Python: scipy.signal.butter(). Real-time on-device.

FIR Filter (windowed)

Liniowa faza: brak zniekształceń fazowych. Kaiser/Hamming okna. Wyższe opóźnienie niż IIR ale stabilniejszy. Dla orientacji preferowany.

Median Filter

Odporność na spike outliers (artefakty). Zastępuje próbkę medianą N sąsiadów. Skuteczne dla GPS position glitches. O(N log N).

FFT: analiza częstotliwościowa

Krok dominuje 1.5–3 Hz. Jazda < 1 Hz. Drżenie > 5 Hz. FFT do klasyfikacji aktywności, analizy tętna PPG i wykrywania anomalii.

Model szumów sensorów (Allan Deviation)

Białe (Gaussian white noise)

ARW (Angle Random Walk) [°/√h] dla żyroskopu. VRW dla akcelerometru. Dominuje na wysokich częstotliwościach.

Bias Instability (flicker)

Powolna losowa zmiana biasu. Widoczna w Allan Deviation w środkowych częstotliwościach (0.001–1 Hz). Trudna do korekcji.

Rate Random Walk (drift)

Silnie skorelowany, kumulatywny drift. Akumulacja błędu pozycji: $\sigma^2\_pos \sim t^3$. Główne wyzwanie nawigacji inercyjnej.

Quantization Noise (ADC)

16-bit acc, zakres ±4g: rozdzielczość ~0.12 mg. ±0.5 LSB losowy błąd kwantyzacji. Redukcja dithering/oversampling.

Allan Deviation plot: narzędzie analizy szumów IMU: identyfikuje ARW, Bias Instability, RRW i Rate Ramp. Kluczowe przy doborze filtrów.

Sensory biometryczne: odcisk palca, Face ID i PPG

Sensory biometryczne realizują uwierzytelnianie użytkownika.
Dane biometryczne nigdy nie opuszczają Secure Enclave / StrongBox urządzenia.

Odcisk palca: technologie sensora

Pojemnościowy (capacitive)
Siatka mikroelektrod: różnica pojemności grzebienie/bruzdy. Pod-szybkowe iPhone (Touch ID gen 2). Odporny na zabrudzenia.

Optyczny (optical in-display)
Ekran OLED jako źródło światła, CMOS pod ekranem. Popularny Android. Podatny na zaawansowane reprodukcje odcisków.

Ultradźwiękowy (Qualcomm)
3D Sonic piezoelektryczny. Działa przez wodę, brud, cienkie folie. 3D obraz linii papilarnych. Samsung Galaxy S flagship.

Android BiometricPrompt API

Kotlin

```kotlin
val prompt = BiometricPrompt(this, mainExecutor,
  object : BiometricPrompt.AuthenticationCallback() {
    override fun onAuthenticationSucceeded(r: Result) {
      unlockContent()  // r.cryptoObject?.cipher
    }
  })
val info = BiometricPrompt.PromptInfo.Builder()
  .setTitle("Potwierdź tożsamość")
  .setAllowedAuthenticators(BIOMETRIC_STRONG
      or DEVICE_CREDENTIAL)
  .build()
prompt.authenticate(info)
```

Apple Face ID: TrueDepth Camera

Structured Light 3D
IR projektor + IR kamera: 30 000 punktów IR → chmura punktów 3D twarzy. Neural Engine → Secure Enclave. FAR < 1:1 000 000.

vs Android Face Unlock (2D)
Szybszy, tańszy: ale możliwy do oszukania zdjęciem. Przeznaczony dla wygody (convenience), NIE dla transakcji finansowych.

Secure Enclave / StrongBox
Apple SEP / Android StrongBox (HSM): izolowany koprocesor. Szablon biometrii nigdy nie wychodzi poza hardware secure storage. Klucze w TEE.

iOS LocalAuthentication

Swift

```swift
let ctx = LAContext()
ctx.evaluatePolicy(
  .deviceOwnerAuthentication,
  localizedReason: "Potwierdź tożsamość"
) { success, error in
  DispatchQueue.main.async {
    if success { self.unlockContent() }
  }
}
```

Pedometr i rozpoznawanie aktywności fizycznej

Step detection i activity recognition to kluczowe zastosowania IMU w fitness.
Miliardy urządzeń zbierają dane aktywności non-stop przez CHRE.

Algorytm detekcji kroków (Step Detection)

1. Filtrowanie sygnału
Bandpass 1–3 Hz: zakres częstotliwości kroków. Eliminacja DC bias (grawitacja) i szumu HF > 5 Hz.

2. Peak Detection
Lokalne maksima powyżej dynamicznego threshold. Schmidt trigger: hystereza dla odporności na szum.

3. False Step Rejection
Odrzuć kroki poza zakresem 0.4–2.5 kr/s. Min czas między krokami: 300 ms. Shake ≠ krok.

4. Dystans i kalibracja
Dystans = kroki × długość kroku. Model Weinberga: L = 0.41×√(acc_max) [m]. Kalibracja GPS.

```
Sensor.TYPE_STEP_COUNTER  // całk. od boot (CHRE)
Sensor.TYPE_STEP_DETECTOR // event per krok
// iOS: CMPedometer z HealthKit
CMPedometer.queryPedometerData(
    from: start, to: end) { data, _ in
  print(data?.numberOfSteps ?? 0)
}
```

Rozpoznawanie aktywności: cechy sygnału

Walking
Amplituda 0.5–2g · Freq 1.7–2.3 Hz · Regularne szczyty

Running
Amplituda 2–5g · Freq 2.5–4 Hz · Impulsowy charakter

Cycling
Niska amplituda pionowa · Regularne pedałowanie

Driving
Niskie przyspieszenie pionowe · Brak kroków · GPS > 10 km/h

Stairs Up/Down
Barometr zmiana ciśnienia + kroki z komponentem pionowym

ML dla Activity Recognition
Klasyczne: SVM, Random Forest (cechy w dziedzinie czasu i częstotliwości). Deep Learning: 1D-CNN na surowym sygnale IMU. Android Awareness API / CoreML (iOS).

Optymalizacja zużycia baterii: sensory i lokalizacja

Sensory to główni konsumenci energii: GPS wyczerpuje baterię w 4–6h, ciągły IMU przy SENSOR_DELAY_FASTEST: ~50 mW. **Oszczędzanie jest obowiązkiem**.

Orientacyjne zużycie mocy sensorów

Sensor	Tryb ciągły	Batched	Odpyt. co 5s	Uwagi
GPS (High Accuracy)	150–200 mW	N/D	5–15 mW	Największy konsument: używaj Balanced
Wi-Fi Scan	30–60 mW	5–10 mW	1 mW	OS throttle: min 30s / scan (Android 8+)
Akcelerometr 100 Hz	1.5–3 mW	0.1 mW	0.05 mW	CHRE < 0.5 mW: hardware batching
Żyroskop 100 Hz	5–8 mW	0.5 mW	0.1 mW	Wyłącz gdy brak aktywnego ruchu
Barometr	0.8 mW	0.15 mW	0.02 mW	Bardzo energooszczędny: zawsze aktywny
Kamera Preview 30fps	200–400 mW	N/D	N/D	Najdroższy: minimize resolution dla ML

Strategie oszczędzania energii

Sensor Batching — maxReportLatencyUs: bufor, obudź CPU raz/5s zamiast 500×/s. Redukcja wake-ups o 95%.

Adaptive Sampling — Zwiększ interval gdy statyczny (wariancja < threshold). STEP_DETECTOR: brak kroków → wolniej.

GPS OFF Indoor — Geofencing + WiFi scan → indoor → wyłącz GPS. Oszczędność 80% w trybie indoor.

WorkManager + Constraints — Zbieraj GPS gdy ładowanie: Constraints.requiresCharging(). Opóźnienie akceptowalne dla track logging.

Android Doze Mode i iOS Background: ograniczenia systemowe

OS aktywnie ogranicza dostęp do sensorów w tle. Dlatego **projektowanie** aplikacji musi uwzględniać **politykę energetyczną** i **dopuszczalne tryby pracy**.

Android Doze Mode (Android 6+)

Active: pełny dostęp

Bateria podłączona lub screen on. GPS, sieci, sensory: wszystko bez ograniczeń.

Doze Light (idle)

Screen off + nieruchomy. GPS wstrzymany. Sieci ograniczone. Alarmy defer. CHRE sensory działają (step, tilt).

Doze Deep

Screen off + statyczny + odłączony. Brak sieci, GPS, sync. Tylko high-priority FCM. Maintenance window rzadko.

Kotlin

```kotlin
val pm = getSystemService(POWER_SERVICE) as PowerManager
// Sprawdź wykluczenie z optymalizacji baterii
pm.isIgnoringBatteryOptimizations(packageName)
// Foreground Service = dostęp do GPS w tle
startForegroundService(
    Intent(this, LocationService::class.java))
// WorkManager z NetworkType/Charging constraints
val work = OneTimeWorkRequestBuilder<GpsWork>()
  .setConstraints(Constraints.Builder()
      .setRequiresCharging(true).build())
  .build()
```

iOS Background Location Modes

Always Authorization

Info.plist: NSLocationAlwaysAndWhenInUseUsageDescription.
App Store: wymagane uzasadnienie. iOS 13+: przycisk Allow Once ogranicza.

Significant Location Changes

startMonitoringSignificantLocationChanges(): budzenie apki co ~500m (cell changes). Bardzo energooszczędne, minimalne uprawnienia.

Region Monitoring (Geofencing)

startMonitoring(for:CLCircularRegion): enter/exit callback. Max 20 regionów per app. OS monitoruje automatycznie.

Visit Monitoring

startMonitoringVisits(): auto-wykrywanie odwiedzin (przybycie/wyjście ze stałych miejsc). Minimalne zużycie baterii.

Android Privacy Dashboard (Android 12+): użytkownicy widzą które aplikacje używały GPS/kamery/mikrofonu w ostatnich 7 dniach → nadużycie = odinstalowanie.

Prywatność i uprawnienia: **Privacy by Design** dla sensorów

*"**Jeśli możesz zidentyfikować osobę na podstawie danych: to dane osobowe**" (RODO Art. 4).*
Trajektoria GPS ujawnia dom, pracę i lekarza.

Model uprawnień Android 12+

ACCESS_FINE_LOCATION
Runtime: dialog z opcjami: Precise / Approximate / Deny. Android 12: opcja przybliżonej lokalizacji (AGP ~3 km). Proś o Approximate gdy wystarczy.

ACCESS_BACKGROUND_LOCATION
Oddzielna zgoda: przekierowuje do Settings. Play Store: wymagane uzasadnienie pisemne. Bardzo trudna do uzyskania od użytkownika.

BODY_SENSORS (API 33+)
Wymagane dla PPG, tętna, SpO_2. Android 13: BODY_SENSORS_BACKGROUND oddzielnie. Dane zdrowotne: najwyższa klasa ochrony.

IMU: brak uprawnień!
Akcelerometr, żyroskop, barometr dostępne BEZ uprawnień runtime. Android 12+ ogranicza do 200 Hz dla 3rd-party. Wciąż wektor ataku!

```kotlin
// Pytaj o uprawnienia w kontekście użycia
val launcher = registerForActivityResult(
  RequestMultiplePermissions()) { handleResult(it) }
// NIE pytaj przy starcie apki: pytaj gdy użytkownik
// chce skorzystać z funkcji wymagającej uprawnienia
```

Privacy by Design: zasady dla sensorów

Data Minimization
Zbieraj tylko dane niezbędne do funkcji. GPS: kraj zamiast ulicy. IMU: 10 Hz zamiast 100 Hz jeśli wystarczy.

Purpose Limitation
Dane do nawigacji → nie wysyłaj do advertiserów. RODO Art. 5: cel zbierania musi być konkretny i legalny.

Storage Limitation
Nie przechowuj dłużej niż potrzeba. Auto-usuwanie po sesji lub N dniach. Minimalizacja logów lokalizacji.

Pseudonymization
Trajectory data: generalizacja + szum Laplace (Differential Privacy). Usuń identyfikatory przed analizą ML.

User Control (RODO DSAR)
UI do wyświetlania zbieranych danych, eksportu i usunięcia konta + danych. Android HealthConnect API.

Bezpieczeństwo danych sensorów: ataki i obrona

Dane sensorów to wektor ataku, side-channel przez IMU ujawnia PIN,
GPS śledzi codzienne trasy, kamera rozpoznaje twarze bez zgody.

⚠ Wektory ataków

IMU Keylogging (bez uprawnień)
Akcelerometr wykrywa wibracje klawiatury → rekonstrukcja PIN. Demo akademickie: 80% skuteczność.

GPS Trajectory Deanonymization
Codzienne trasy identyfikują dom i pracę z 95% dokładnością nawet po pseudonimizacji.

Ultrasonic Beacon Tracking
Reklamy TV emitują ultradźwięki → mikrofon smartfona → cross-device tracking bez zgody.

Sensor Fingerprinting (gait)
Unikalny bias i szum IMU każdego urządzenia → identyfikacja (fingerprint) bez cookies i IDFA.

Kamera w tle (malware)
Malware aktywuje kamerę i nagrywa bez wskaźnika. Identyfikacja twarzy, monitorowanie.

✓ Zabezpieczenia i mitigacje

Android 12: 200 Hz limit IMU
Ograniczenie częstotliwości dla 3rd-party. Fuzzing wartości. CHRE wymaga uprawnień.

Differential Privacy + Generalization
Szum Laplace'a do współrzędnych. Uogólnienie do poziomu dzielnicy. Usuwanie skrajnych punktów.

Mic Permission Indicator (Hardware)
iOS: pomarańczowy indicator hardwarowy. Android 12+: zielony pasek kamery/mikrofonu.

Sensor Randomization per session
Losowy offset do wartości sensorów per sesja. Android Privacy Sandbox ogranicza IDFA.

Hardware Privacy Indicator (non-bypassable)
Nie można wyłączyć software'm. iOS 14+: zielona kropka. Android 12+: status bar indicator.

Cross-Platform: sensory w Flutter

Flutter umożliwia obsługę sensorów z jednej bazy kodu Dart.
Bogaty ekosystem pakietów pub.dev z zestandaryzowanym, stream-based API.

Kluczowe pakiety Flutter dla sensorów

sensors_plus
IMU: AccelerometerEvent, GyroscopeEvent, MagnetometerEvent. Stream-based. Android+iOS+Web.

geolocator
GPS: getCurrentPosition(), getPositionStream(). LocationPermission. Platform channels.

camera
Camera2/AVFoundation. ImageStream per-frame. Exposure, zoom, flash control.

flutter_activity_recognition
CMMotionActivityManager (iOS) / ActivityRecognitionClient (Android). Aktywność: still/walking/running.

barometer
Ciśnienie atmosferyczne → Stream<BarometerEvent> w hPa. Low-power.

Platform Channels: własne API
Brak pakietu pub.dev → MethodChannel (jednorazowe wywołanie) lub EventChannel (ciągły stream) do natywnego Kotlin/Swift.

Expo vs bare
Expo SDK: sensors_plus wbudowany w Expo Sensors: zero konfiguracji natywnej.

Implementacja sensors_plus + geolocator

Dart

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

class SensorViewModel extends ChangeNotifier {
  StreamSubscription<AccelerometerEvent>? _accelSub;
  StreamSubscription<Position>? _gpsSub;

  void startSensors() {
    _accelSub = accelerometerEventStream(
      samplingPeriod: SensorInterval.gameInterval,
    ).listen((e) {
      accel = e; notifyListeners();
    });
    _gpsSub = Geolocator.getPositionStream(
      locationSettings: const LocationSettings(
        accuracy: LocationAccuracy.high,
        distanceFilter: 5, // min 5m zmiana
      )
    ).listen((pos) {
      position = pos; notifyListeners();
    });
  }

  @override
  void dispose() {
    _accelSub?.cancel(); _gpsSub?.cancel();
    super.dispose();
  }
}
```

Cross-Platform: sensory w React Native

React Native dostarcza sensory przez JavaScript bridge (JSI w nowej architekturze) i ekosystem bibliotek community, w tym Expo SDK.

Biblioteki dla sensorów (React Native)

react-native-sensors
Accelerometer, Gyroscope, Magnetometer: observables RxJS. setUpdateIntervalForType(SensorTypes.accelerometer, 100ms).

react-native-geolocation-service
Precyzyjny GPS foreground+background. navigator.geolocation kompatybilny. Android FusedLocation pod spodem.

expo-sensors
Expo SDK: Accelerometer, Gyroscope, Magnetometer, Barometer, Pedometer, LightSensor, DeviceMotion.

react-native-vision-camera
VisionCamera v3: Frame Processors: natywny Kotlin/Swift kod per klatka bez bridge. < 5 ms ML latencja.

Nowa Architektura JSI (RN 0.71+)
JavaScript Interface zastępuje bridge: synchroniczne wywołania native, brak serializacji JSON. Turbo Modules+Fabric Renderer. Sensory bezpośrednio w wątku JS.

Expo vs bare workflow
Expo: szybki start, managed permissions, zero natywnej konfiguracji. Bare: pełna kontrola gdy potrzebujesz własnych wrappers lub obscure platform APIs.

Implementacja RxJS + VisionCamera

JavaScript/TS

```
import { Accelerometer } from 'react-native-sensors';
import { setUpdateIntervalForType, SensorTypes }
    from 'react-native-sensors';
import { map, filter, throttleTime } from 'rxjs/operators';

setUpdateIntervalForType(
    SensorTypes.accelerometer, 200);  // ms

const subscription = new Accelerometer({ updateInterval: 200 })
  .pipe(
    map(({ x, y, z }) => ({
      x, y, z,
      magnitude: Math.sqrt(x*x + y*y + z*z)
    })),
    filter(({ magnitude }) => magnitude > 1.2),
    throttleTime(50),
  )
  .subscribe(data => setAccelData(data));

// VisionCamera Frame Processor
const frameProcessor = useFrameProcessor(frame => {
  'worklet';
  const barcodes = scanBarcodes(frame, [BarcodeFormat.QR]);
  runOnJS(setBarcodes)(barcodes);
}, []);

return () => subscription.unsubscribe();
```

Wearables i sensory peryferyjne: Wear OS, watchOS i IoT

Smartwatch jako hub sensoryczny; bliżej ciała, ciągłe monitorowanie zdrowia i nowe formy interakcji niedostępne dla smartfonów.

Wear OS: Health Services API

Health Services (Wear OS 3.0)
Unified API: HeartRateSource, SpO_2, VO2max, EDA, temperatura skóry. ExerciseClient dla śledzenia treningów z automatyczną detekcją aktywności.

PassiveMonitoringClient
Ciągłe zbieranie danych w tle: kroki, tętno, aktywność. CHRE: zawsze aktywny. Dane do HealthKit/HealthConnect.

AmbientMode i Always-On Display
onAmbientModeChanged(): zredukuj animacje. Aktualizuj UI co minutę. Watchface API wyświetla dane sensorów stale.

Data Layer API (Phone↔Watch)
ChannelClient, MessageClient, DataClient: synchronizacja danych sensorów w czasie rzeczywistym. WearableListenerService.

```swift
// watchOS: HealthKit sensor access
let hrType = HKObjectType.quantityType(
    forIdentifier: .heartRate)!
hkStore.requestAuthorization(toShare: nil,
    read: [hrType]) { ok, _ in startHRQuery() }
```

BLE i zewnętrzne sensory IoT

BLE Medical Devices (SIG Profiles)
GATT: Heart Rate (0x180D), Blood Pressure (0x1810), Glucose (0x1808), Thermometer (0x1809). Interoperabilność bez custom protokołu.

ANT+ Fitness Sensors
ANT+ (Garmin ecosystem): HR strap, cadence, power meter. Standard w sporcie wyczynowym. Mniejsza społeczność niż BLE ale stały protokół.

Polar H10 / Shimmer3 (research)
ECG badawczy: 12-lead przez BLE, raw IMU 512 Hz. Open SDK. Popularny w akademickich studiach aktywności i HCI.

Environmental IoT (MQTT)
Stacje pogodowe, PM2.5, CO_2, TVOC: BLE lub Wi-Fi + MQTT broker. Telefon jako agregator. InfluxDB + Grafana wizualizacja.

Matter *(Apple/Google/Amazon):* to standard w IoT. Sensory smart home: temperatura, drzwi, ruch, jakość powietrza: telefon jako controller hub.

Case studies: zaawansowane zastosowania sensorów

Analiza rzeczywistych systemów pokazuje jak połączenie sensorów tworzy wartość niemożliwą do uzyskania z jednego czujnika.

Google Maps: Nawigacja piesza

Fuzja: GPS (5m) + PDR akcelerometr + magnetometr (kierunek) + barometr (piętro) + Wi-Fi scan (indoor) → pozycja < 2m outdoor, < 5m indoor. Krok bazowy: gdy GPS słaby (canyon) → PDR przejmuje. Kroki z STEP_DETECTOR → dystans modelem Weinberga → azymut z fuzji mag+gyro. Indoor: Google Indoor Maps fingerprinting Wi-Fi RSSI: lotniska, centra handlowe. Barometr wyznacza piętro ±1.

Apple Crash Detection: iOS 16 / Pixel

Sensory: akcelerometr high-g (do 256g) + żyroskop + barometr + GPS (prędkość→0) + mikrofon (dźwięk uderzenia) → klasyfikator ML. Wzorzec: nagłe przyspieszenie > 2.5g → zatrzymanie → cisza lub airbag deploy sound → GPS prędkość 0 → brak aktywności → Emergency SOS. Wyzwanie false positives: rollercoaster, sporty ekstremalne → explicit 'I'm safe' button z 10s oknem potwierdzenia.

AR Nawigacja Chirurgiczna: iPad Pro LiDAR

iPad Pro 2020+ LiDAR: ARKit MeshAnchor rekonstruuje anatomię pacjenta 3D → nakładanie modelu CT/MRI z precyzją < 3 mm.
Pipeline: CT DICOM → segmentacja AI → model STL → ARKit anchor do fizycznej anatomii przez LiDAR registration.
Wyniki kliniczne: intraoperative guidance ortopedii i neurochirurgii. Kilka systemów w FDA clearance pipeline (2024).

Pytania 6

Które problemy architektoniczne są szczególnie istotne przy projektowaniu aplikacji mobilnych w porównaniu z aplikacjami desktopowymi?

A. brak wielozadaniowości systemu operacyjnego
B. brak systemów plików w urządzeniach mobilnych
C. niestabilność połączenia sieciowego
D. ograniczona dostępność energii

Które czynniki mają największy wpływ na zużycie energii przez aplikację mobilną?

A. liczba bibliotek Gradle
B. liczba klas w projekcie
C. intensywne użycie GPU
D. częste operacje sieciowe

Które elementy są typowymi komponentami architektury aplikacji mobilnej opartej na Clean Architecture?

A. warstwa kernel
B. warstwa HAL
C. warstwa data
D. warstwa domain

Które czynniki mają największy wpływ na fragmentację platformy Android z punktu widzenia projektowania aplikacji?

A. otwartość kodu źródłowego systemu Android
B. obecność wielu języków programowania
C. różnorodność konfiguracji sprzętowych urządzeń
D. różnorodność producentów sprzętu i konfiguracji hardware

Które cechy architektury offline-first są kluczowe dla aplikacji mobilnych?

A. blokowanie UI w przypadku braku połączenia z siecią
B. przechowywanie danych wyłącznie w pamięci RAM
C. mechanizmy synchronizacji konfliktów danych
D. lokalna baza danych synchronizowana z serwerem

Które cechy architektury aplikacji mobilnych poprawiają skalowalność i testowalność kodu?

A. globalne zmienne stanu aplikacji
B. bezpośredni dostęp UI do bazy danych
C. dependency injection
D. separacja warstw (domain / data / presentation)

Które elementy architektury aplikacji mobilnej zmniejszają sprzężenie między warstwą UI, a logiką biznesową?

A. bezpośrednie wywoływanie zapytań do bazy danych z Activity
B. umieszczanie logiki biznesowej w klasach widoku
C. zastosowanie architektury MVVM
D. wykorzystanie warstwy repozytorium (repository pattern)

Które czynniki wpływają na decyzję o wyborze native development zamiast cross-platform?

A. brak interfejsu użytkownika w aplikacji
B. brak potrzeby publikacji w sklepach aplikacji
C. maksymalizacja wydajności aplikacji
D. potrzeba dostępu do zaawansowanych API sprzętowych

Które mechanizmy systemowe ograniczają aktywność aplikacji w tle w Androidzie?

A. Garbage collector
B. Fragment lifecycle
C. Doze mode
D. App Standby buckets

Które mechanizmy systemu Android umożliwiają komunikację między komponentami aplikacji lub między aplikacjami?

A. ViewModel
B. Fragment transactions
C. Binder IPC
D. Intents

Które elementy procesu CI/CD są szczególnie istotne dla projektów mobilnych?

A. ograniczona pamięć ROM
B. brak wielowątkowości w systemie operacyjnym
C. operacje sieciowe o wysokiej latencji
D. automatyczne budowanie aplikacji dla wielu architektur CPU

Które mechanizmy pozwalają aplikacjom Android korzystać z kodów natywnych (C/C++)?

A. Binder IPC
B. ART bytecode verifier
C. Android NDK
D. JNI (Java Native Interface)

Podsumowanie

Generatywna AI + sensory

LLM + IMU/GPS/kamera → kontekstowe odpowiedzi live. Gemini Nano on-device.

Continuous Health Monitoring

Nieinwazyjny glukometr przez skórę *(Samsung R&D)*, EKG przez watch, FDA clearance 2024.

Neural IMU *(Sensor Fusion ML)*

Deep learning zastępuje filtry: on-device transformer < 2ms, wyższa dokładność orientacji.

Privacy-First On-Device ML

Dane sensorów nie wychodzą z urządzenia. Apple Private Relay, local processing jako USP.

Spatial Computing (Vision Pro)

6DoF + kamera + IMU + LiDAR = nowa era spatial UX. M2 Motion Coprocessor jako model.

UWB Precise Positioning

U1/U2 chip: centimetrowa precyzja indoor. Keyless entry, AirTag, shared AR handoff.

Kluczowe wnioski wykładu

- Żaden pojedynczy sensor nie jest wystarczający: fuzja (Kalman, komplementarny) to produkcyjny standard każdej aplikacji mobilnej
- Prywatność i bateria to ograniczenia niefunkcjonalne pierwszej klasy: projektuj z nimi od dnia 1, nie na końcu
- IMU dostępne BEZ uprawnień runtime: potencjalny wektor ataku; Android 12+ ogranicza, ale nie eliminuje ryzyka
- CHRE / Motion Coprocessor zmienił paradygmat: Always-On sensing nie kosztuje baterii tyle co kiedyś: sensory ciągłe są możliwe

KONIEC