Context-aware computing

Introduction

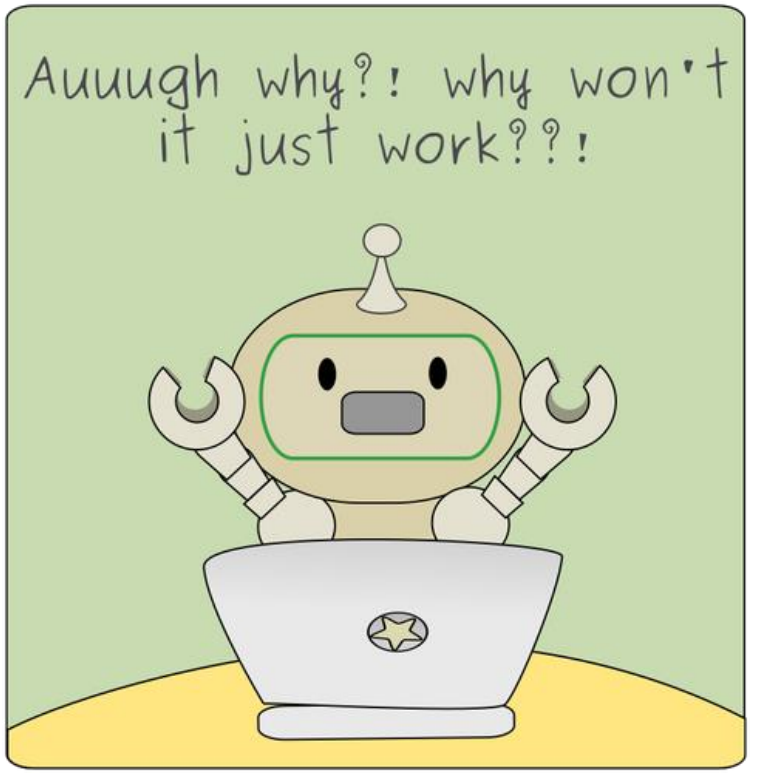
- When humans speak with humans, they are able to use information apparent from the current situation, or context, to increase the conversational bandwidth

- This ability to convey ideas does not transfer well when humans interact with computers

Copyright:
http://arnoldzwicky.s3.amazonaws.com/BrowmanTalk.jpg

Introduction

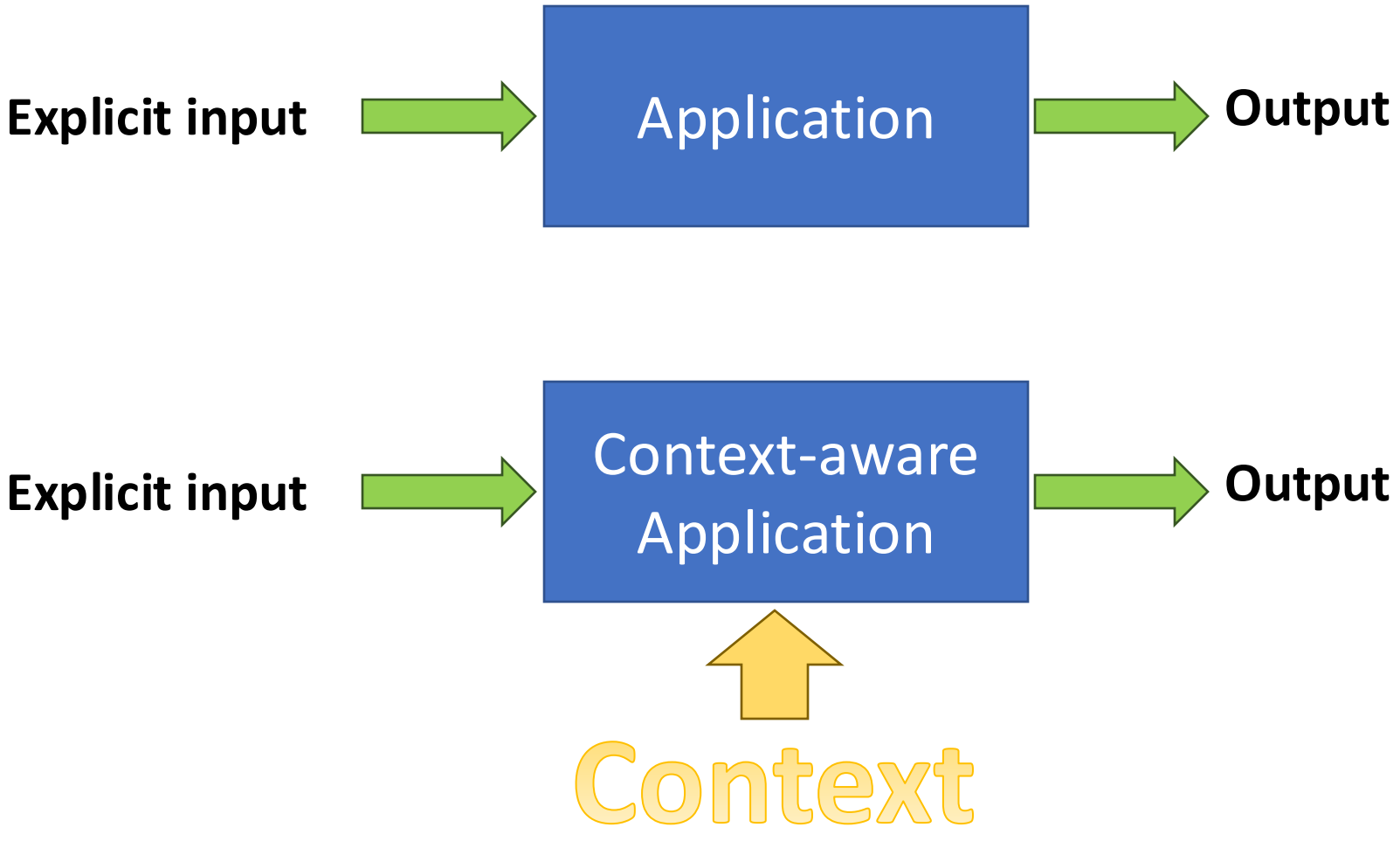
- **Context-aware computing**: using context as an implicit cue to enrich the impoverished interaction from humans to computers, making it easier to interact with computers
- A step towards *calm* technology: an approach to ubiquitous computing, where computing moves back and forth between the center and periphery of the user's attention



Introduction

- Applications that use context, whether on a desktop or in a mobile or ubiquitous computing environment, are called ***context-aware***
- Availability of commercial, off-the-shelf sensing technologies is making it more viable to sense context in a variety of environments
- Mobility creates situations where the user's context, such as his or her location and the people and objects around him/her, is more dynamic

Introduction



Introduction

- Means for the services to adapt appropriately, in order to best support the human–computer and human–environment interactions
- Context-awareness is useful in
  - smart homes
    - Turn heating off if the user is far from home
  - healthcare
    - Detecting falls of elderly people or athletes
  - smart industry
    - Provide instruction to workers when assembling complex systems
  - quantified self
    - Number of steps walked
  - smart transport
    - Detect traffic jams

What is context?

We all have a general and maybe vague idea of what context is

- *Schilit and Theimer (1994):* refer to context as location, identities of nearby people and objects, and changes to those objects
- *Brown et al. (1997):* define context as location, identities of the people around the user, the time of day, season, temperature, etc
- *Ryan et al. (1998):* define context as the user's location, environment, identity, and time
- *Dey (1998):* enumerated context as the user's emotional state, focus of attention, location and orientation, date and time, and objects and people in the user's environment
- These definitions define context by example and are difficult to apply
- Some consider context to be the user's environment, whereas others consider it to be the application's environment

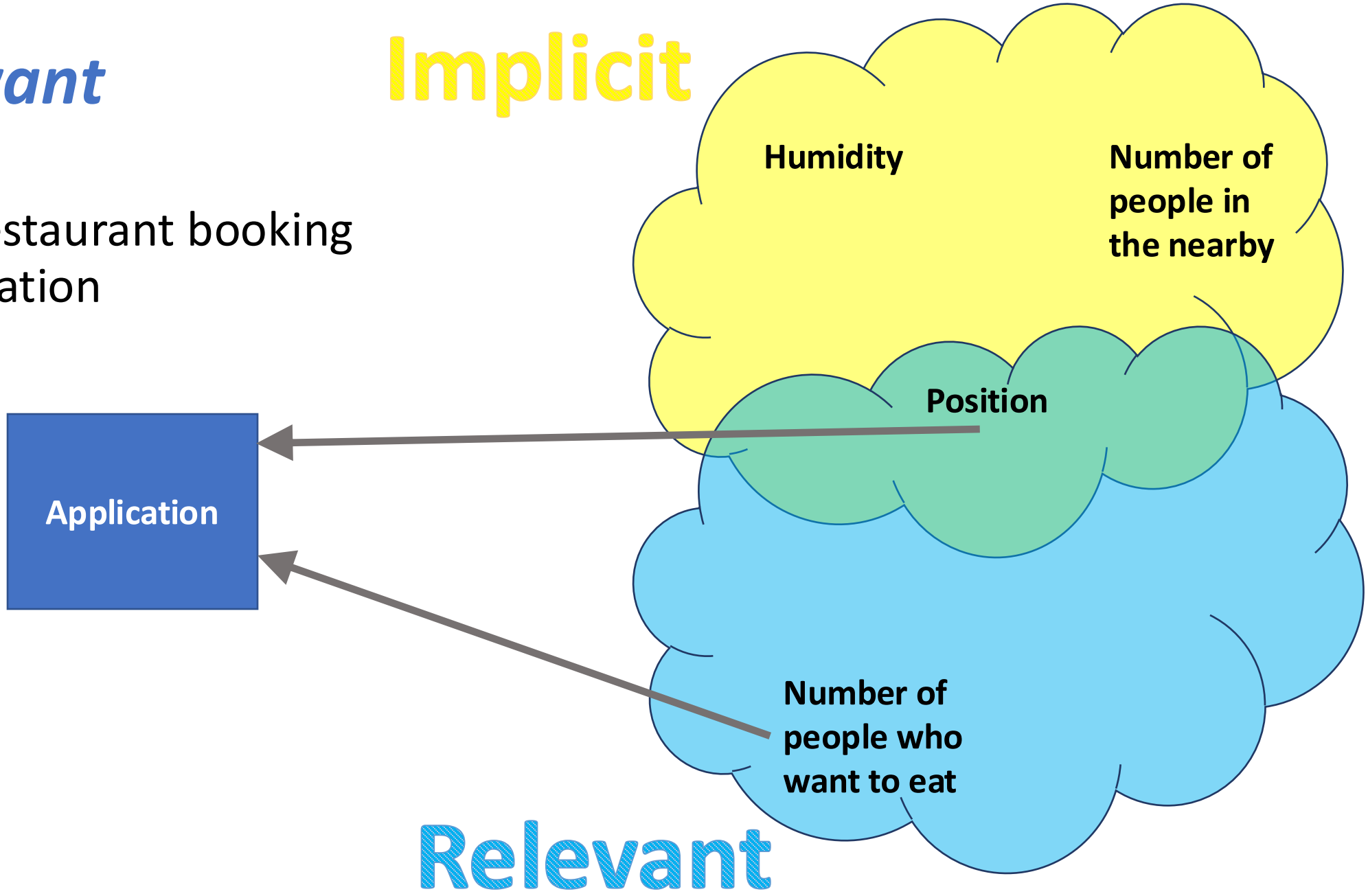
Context, a definition

“Any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.”

Dey and Abowd (2000a)

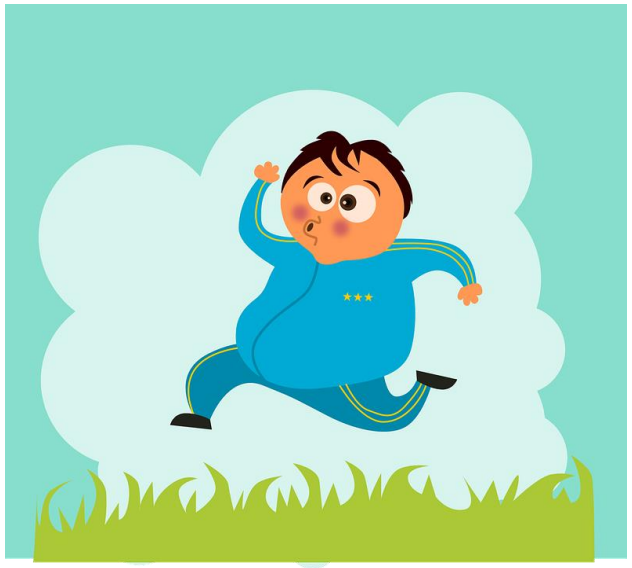
Relevant

- In a restaurant booking application

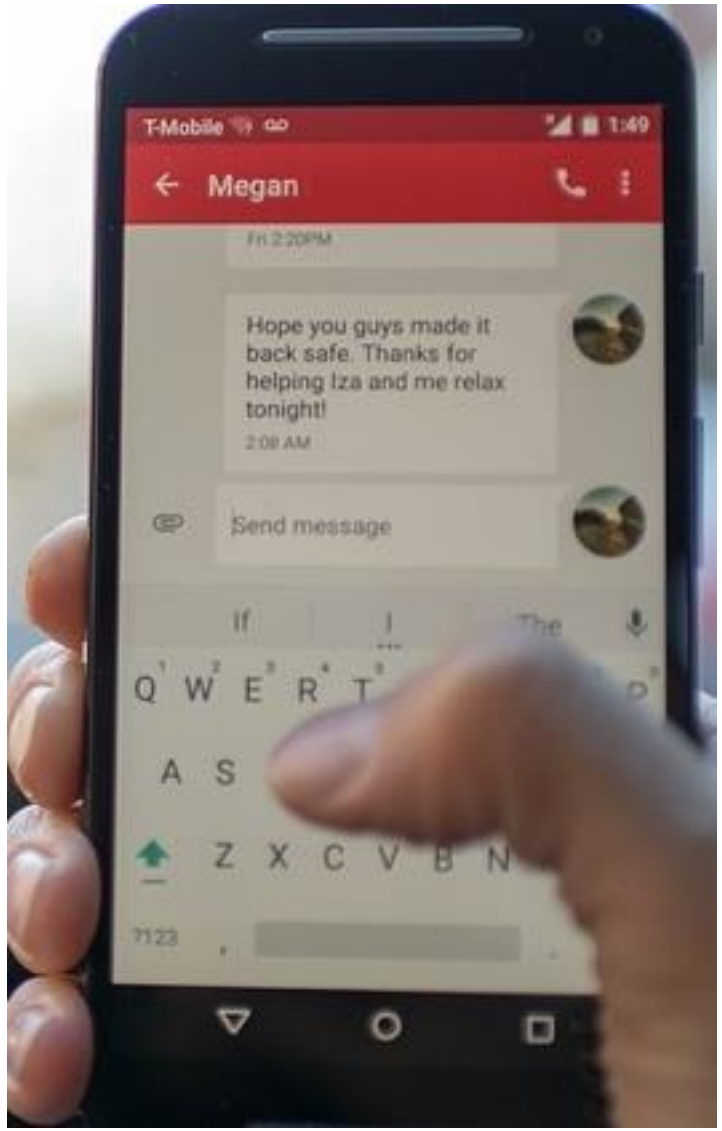


Context

- Environmental sensing is relatively straight-forward
  - Use sensors for temperature, humidity, pressure, etc
- Human sensing is a little harder (ranked easy to hard)
  - **When**: time (Easiest)
  - **Where**: location
  - **Who**: identification
  - **What**: running, eating, cooking (meta task)
  - **Why**: reason for actions (extremely hard!)
  - ***How***: (Mood) happy, sad, bored (gesture recognition, cameras)



- Certain types of context that are, in practice, more important than others **location** (where), **identity** (who), **time** (when), and **activity** (what). Can be used as pointers to access other information:
  - given an entity's location, we can determine what other objects or people are near the entity

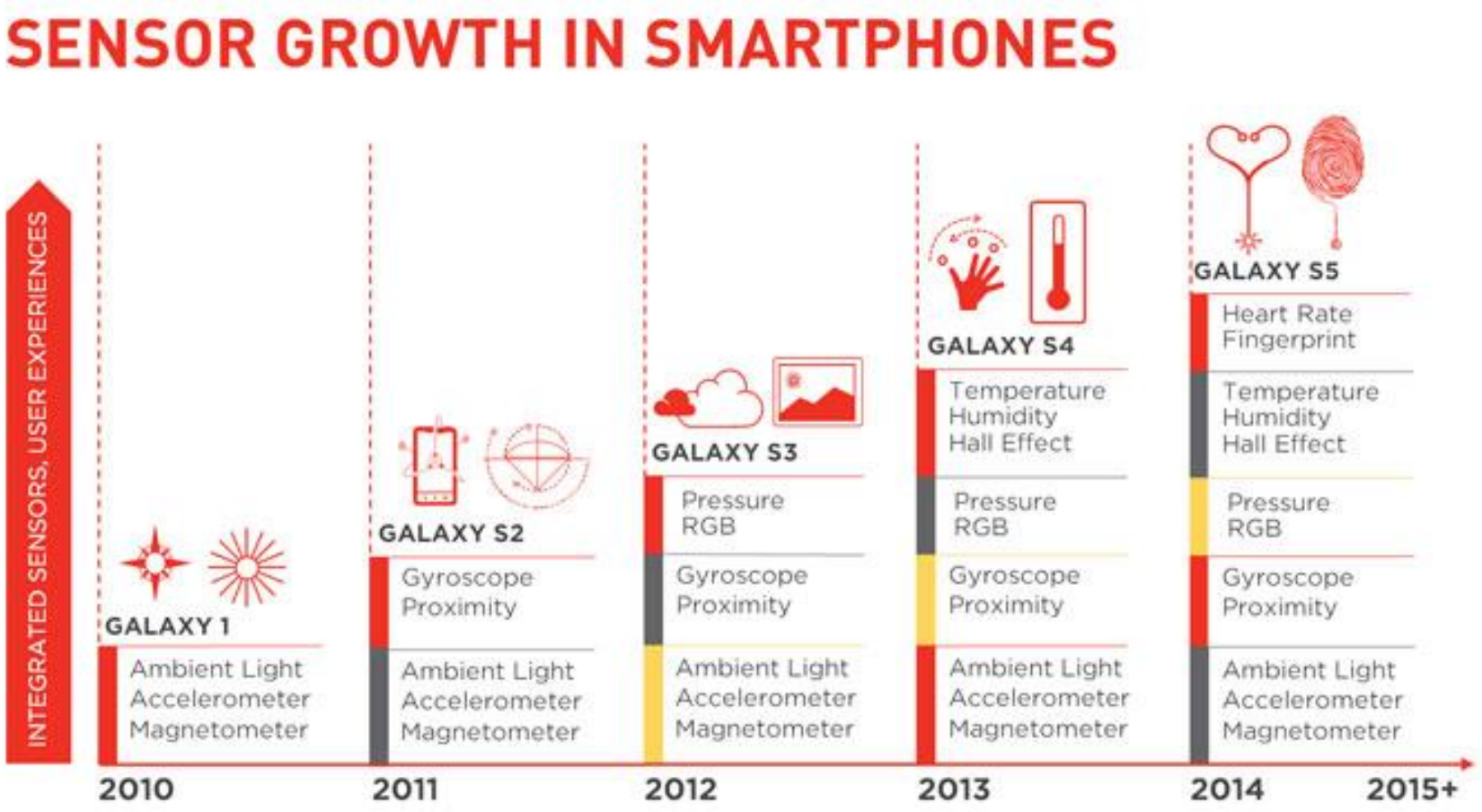

T-Mobile
1:49
Megan
Fri 2:20PM
Hope you guys made it back safe. Thanks for helping Iza and me relax tonight!
2:08 AM
Send message


T-Mobile
1:49
Megan
Fri 2:20PM
Hope you guys made it back safe. Thanks for helping Iza and me relax tonight!
2:08 AM
Send message

Context acquisition

- **Physical sensors**
  - Hardware sensors
  - Examples: humidity, pollution level, noise, position, light level, …
- **Virtual sensors**
  - Data published similarly to HW-originated data
  - Example: number of tweets containing some keywords as an estimation of the number of tourists
- **Logical sensors**
  - Information produced by physical and virtual sensors is fused
  - Examples: acceleration + position -> transportation mode

Sensing (context) with smartphones



Source: qualcomm.com

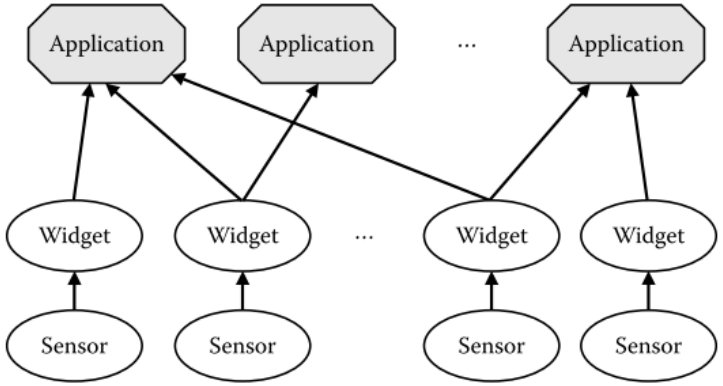
Building context-aware applications

Three main alternatives used for building applications:

- no support
- a widget- or object-based system
- a blackboard-based system

Building context-aware applications: widgets

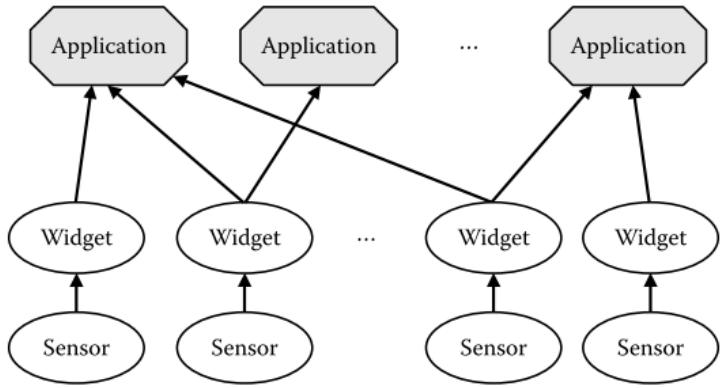
- **Context widget**: a software component that provides applications with access to context information
- GUI widgets insulate applications from presentation concerns, similarly context widgets insulate applications from context acquisition concerns by wrapping sensors with a uniform interface
- Separation of concerns: they hide the complexity of the actual sensors used from the application
  - Whether the presence of people is sensed using IR sensors, floor sensors, video image processing, or a combination of these should not impact the design of the application
- Abstract context information to suit the expected needs of applications
  - Widget that tracks the location of a user within a building notifies the application only when the user moves from one room to another, and does not report less significant moves to the application



From «Ubiquitous Computing Fundamentals».

Building context-aware applications: widgets

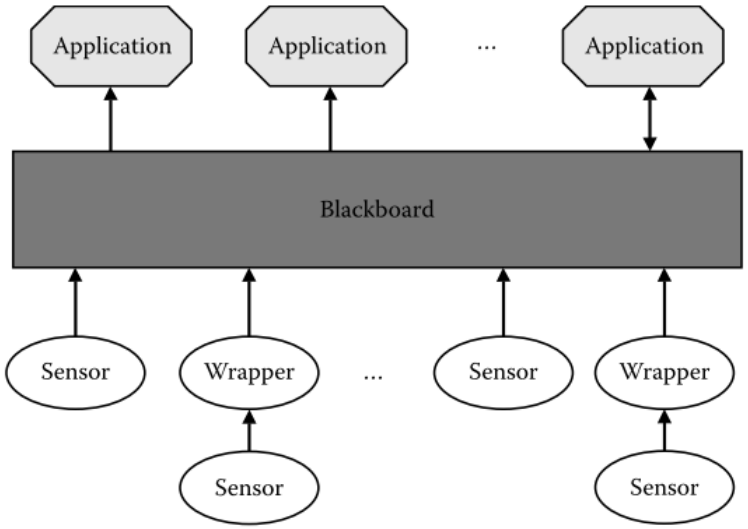
- Access to context data through querying and notification mechanisms
- Reusable and customizable building blocks of context sensing
  - Widget that tracks the location of a user can be used by a variety of applications, from tour guides to industrial automation
- Context widgets can be combined in ways similar to GUI widgets
  - E.g. a *presence* widget senses the presence of people in a room. A *meeting* widget may be built on top of a *presence* widget and assume a meeting is beginning when two or more people are present



From «Ubiquitous Computing Fundamentals».

Building context-aware applications: blackboard

- Blackboards originated from the Linda programming language and tuple space model from the early 1980s
- Components can place information into the storage system, and read or remove this information
- Blackboards may have the ability to notify components when information of interest has been added to the blackboard
- Blackboard approaches sometimes inefficient particularly as the amount of data in the tuple space grows and searching for specific tuples becomes harder



Representing context

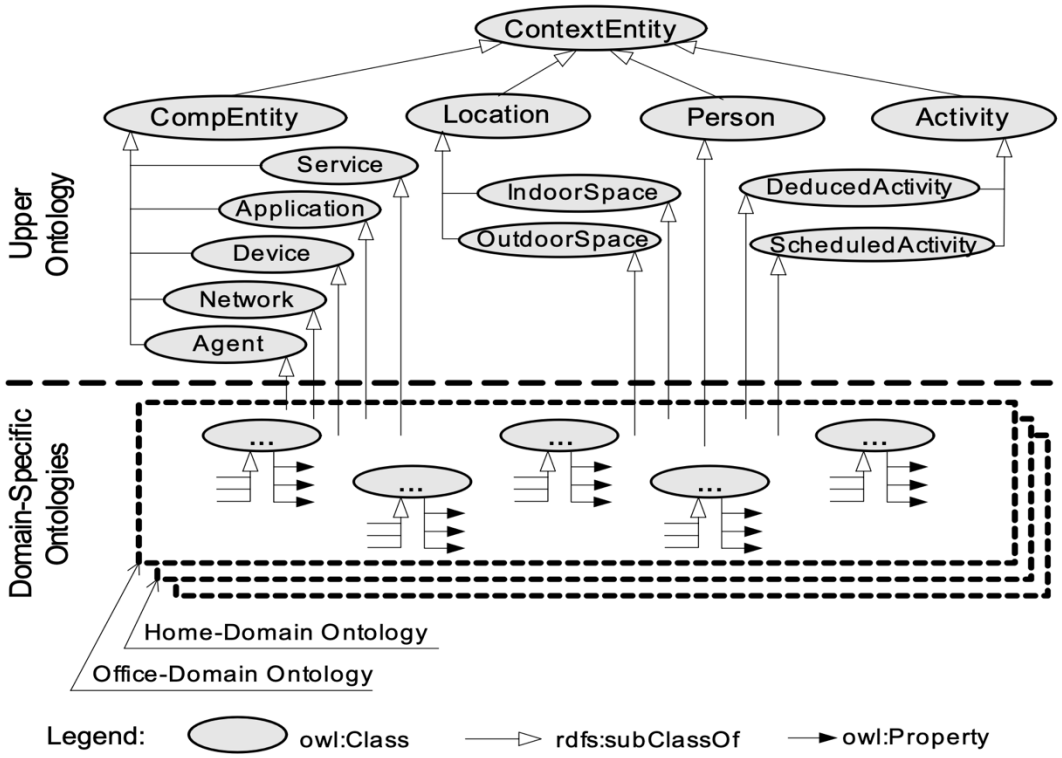
- There is no standardized approach
- Some common ways
  - key-value pairs: *activity=running*, *temperature=23*
  - XML-based representation:

```
<contextML><ctxEls><ctxEl> <contextProvider id="ActivityCP" v="0.0.1"/>
 <entity id="john" type="username"/> <requestEntity id="john" type="username"/>
 <scope>activity</scope>
 <timestamp>2010-02-08T16:25:55+01:00</timestamp>
 <expires>2010-02-08T16:26:55+01:00</expires>
 <dataPart>
  <parS n="activity">
   <par n="working">0.5946</par>      <par n="shopping">0.2396</par>
   <par n="dining">0.0829</par>       <par n="partying">0.0829</par>
  </parS>
  <parS n="mood">
   <par n="frustrated">0.6650</par>  <par n="happy">0.3350</par>
  </parS>
 </dataPart>
</ctxEl>
</ctxEls></contextML>
```

ContextML: A light-weight context representation and context management schema

Representing context

- Ontology-based modeling (OWL)
  - Semantic Web: set of standards for exchanging machine-understandable information



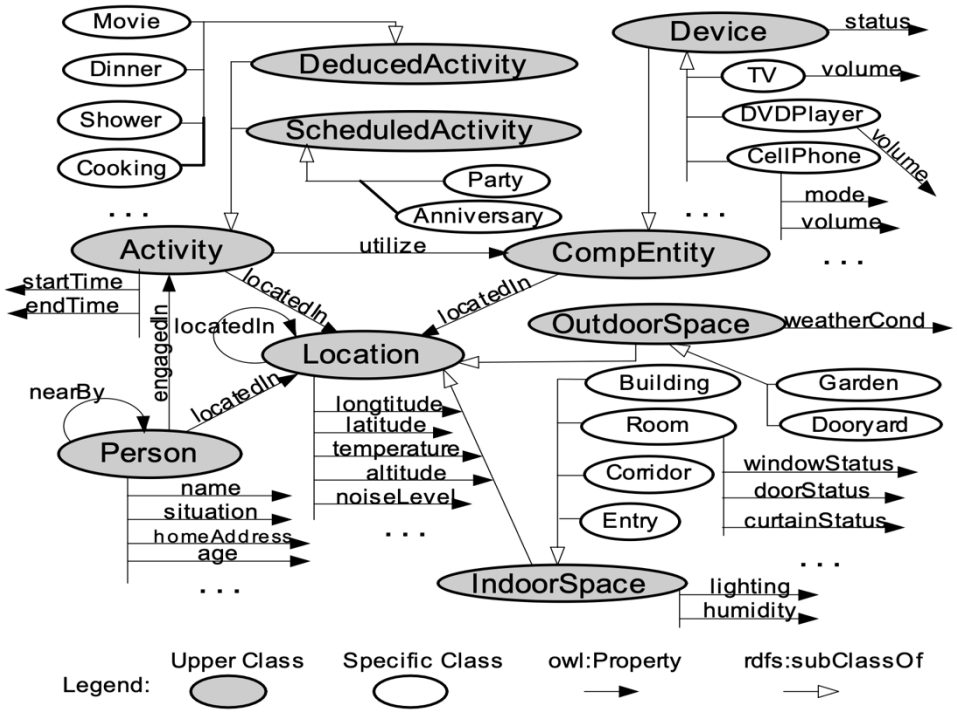


Figure 2. Partial definition of a specific ontology for home domain

Representing context

- With OWL, context can be processed with logical reasoning mechanisms

Table 1. Parts of OWL ontology reasoning rules

Transitive-Property	(?P rdf:type owl:TransitiveProperty) $\wedge$ (?A ?P ?B) $\wedge$ (?B ?P ?C) $\Rightarrow$ (?A ?P ?C)
subClassOf	(?a rdfs:subClassOf ?b) $\wedge$ (?b rdfs:subClassOf ?c) $\Rightarrow$ (?a rdfs:subClassOf ?c)

Table 2. Reasoning about location using ontology

INPUT	DL Reasoning Rules	(?P rdf:type owl:TransitiveProperty) $\wedge$ (?A ?P ?B) $\wedge$ (?B ?P ?C) $\Rightarrow$ (?A ?P ?C)
		(?P owl:inverseOf ?Q) $\wedge$ (?X ?P ?Y) $\Rightarrow$ (?Y ?Q ?X)
	Explicit Context	<owl:ObjectProperty rdf:ID="locatedIn"> <rdf:type="owlTransitiveProperty"/> <owl:inverseOf rdf:resource="#contains"/> </owl:ObjectProperty> <Person rdf:ID="Wang"> <locatedIn rdf:resource="#Bedroom"/> </Person > < Room rdf:ID="Bedroom"> < locatedIn rdf:resource="#Home"/> </ Room>
OUTPUT	Implicit Context	<Person rdf:ID="Wang"> <locatedIn rdf:resource="#Home"/> </Person > <Building rdf:ID="Home"> < contains rdf:resource="#Bedroom"/> < contains rdf:resource="#Wang"/> </Building> <Room rdf:ID="Bedroom"> < contains rdf:resource="#Wang"/> </Room>

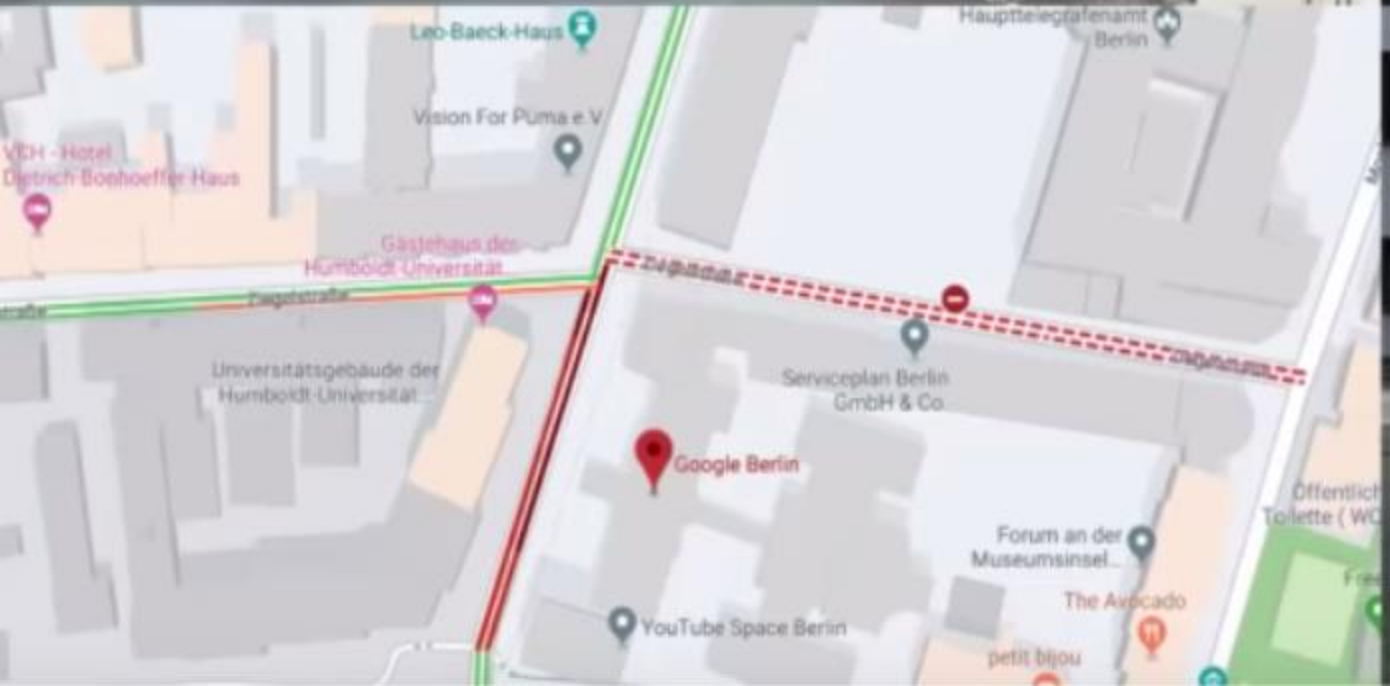
Table 3. User-defined context reasoning rules

Situation	Reasoning Rules
Sleeping	(?u locatedIn Bedroom) $\wedge$ (Bedroom lightLevel LOW) $\wedge$ (Bedroom drapeStatus CLOSED) $\Rightarrow$ (?u situation SLEEPING)

Issues when building contex-aware applications

Context is a proxy for human intent

- Context awareness is about understanding human intent
- Applications would use this human intent to adapt appropriately by providing information or taking actions
- However, context information is only a **proxy** for this intent
- Museum tour guide: a user standing in front of a particular painting may not represent interest in the artpiece, maybe user has her back turned to the exhibit and is having a conversation with a friend there
- Solutions:
  - Additional context is required to understand human intent
  - Applications should express a certainty in their beliefs about their sensed information and, if that certainty is not above an appropriate threshold, ask for confirmation of this information before taking action


Leo-Baeck-Haus
Vision For Puma e.V.
VCH - Hotel Dietrich Bonhoeffer Haus
Gästehaus der Humboldt-Universität
Universitätsgebäude der Humboldt-Universität
Serviceplan Berlin GmbH & Co.
Google Berlin
Forum an der Museumsinsel
The Avocado
YouTube Space Berlin
petit bijou

Issues when building contex-aware applications

Context ambiguity

- Many sources of ambiguity or errors: context sensors can sense incorrectly, fail, or be unsure about what they sensed; context inferencing systems can inaccurately reach conclusions about a situation
- One approach to dealing with context ambiguity is to combine multiple disparate sources of the same type of context to improve the accuracy or dependability of the provided context
- This is commonly known as sensor fusion
- Another possibility: context does not change completely randomly, but instead shows some coherence over time

Issues when building contex-aware applications

Privacy

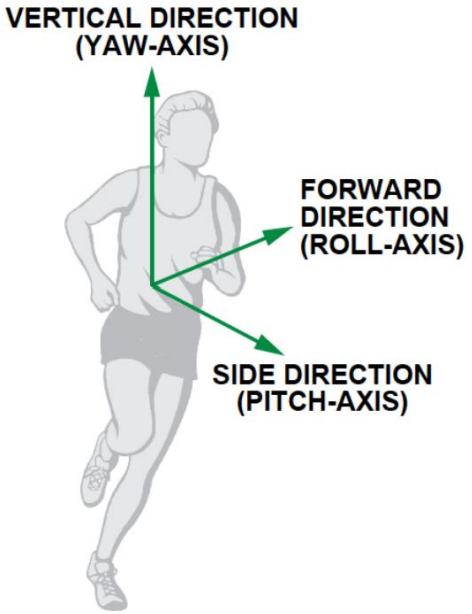
- There is a real concern that context information could either be disseminated inappropriately or disseminated to a component that is not completely trustworthy, and may disseminate the information further
- Developers of context-aware applications need to ensure that a user's privacy is maintained and that information is not being used inappropriately, or in a manner that the user feels inappropriate
- Make clear **why** some specific information is needed
  - E.g. the position of the user is needed to better estimate the amount of burned calories
  - Some systems (Android) show the user the permissions required by apps: statistics show that the ones which better explain why sensitive information is needed are less likely disinstalled

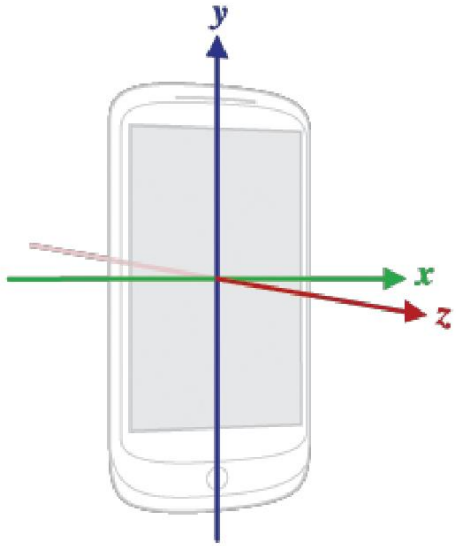
Activity recognition and sensors

How does a pedometer count steps

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

- As example of processing accelerometer data
- Walking or running results in motion along the 3 body axes (forward, vertical, side)
- Smartphone has similar axes
  - Alignment depends on phone orientation

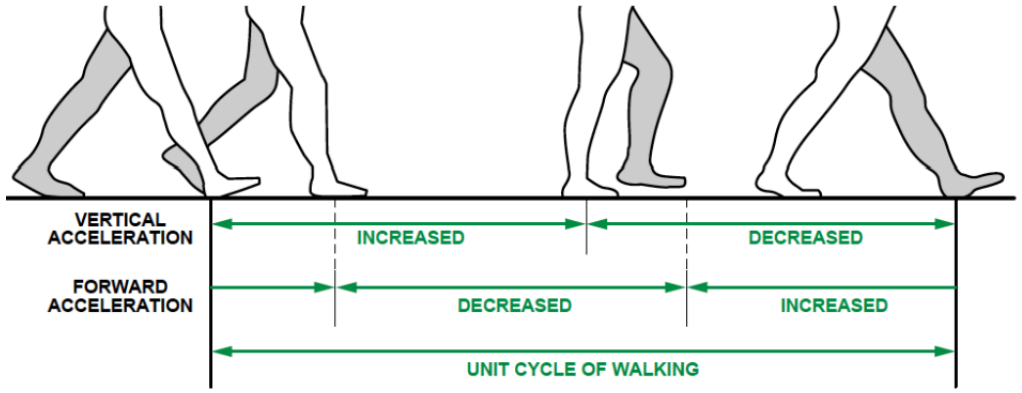


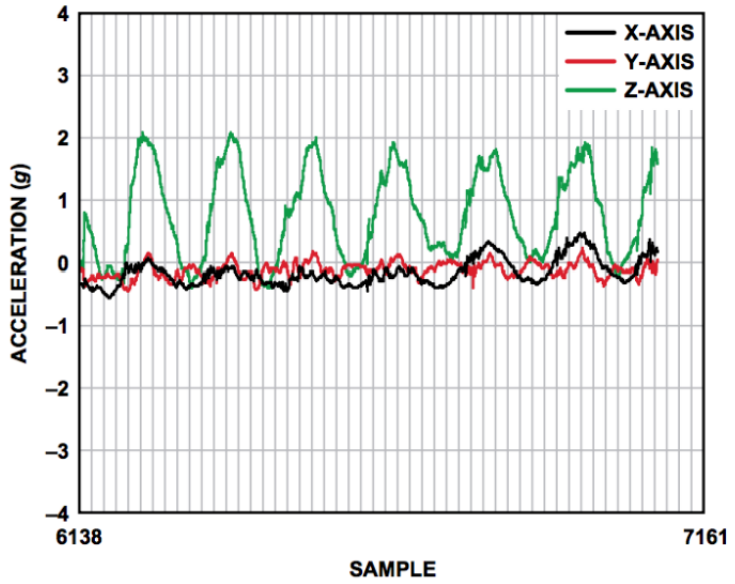


The nature of walking

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

- Vertical and forward acceleration increases/decreases during different phases of walking
- Walking causes a large periodic spike in one of the accelerometer axes
- Which axis (*x*, *y* or *z*) and magnitude depends on phone orientation

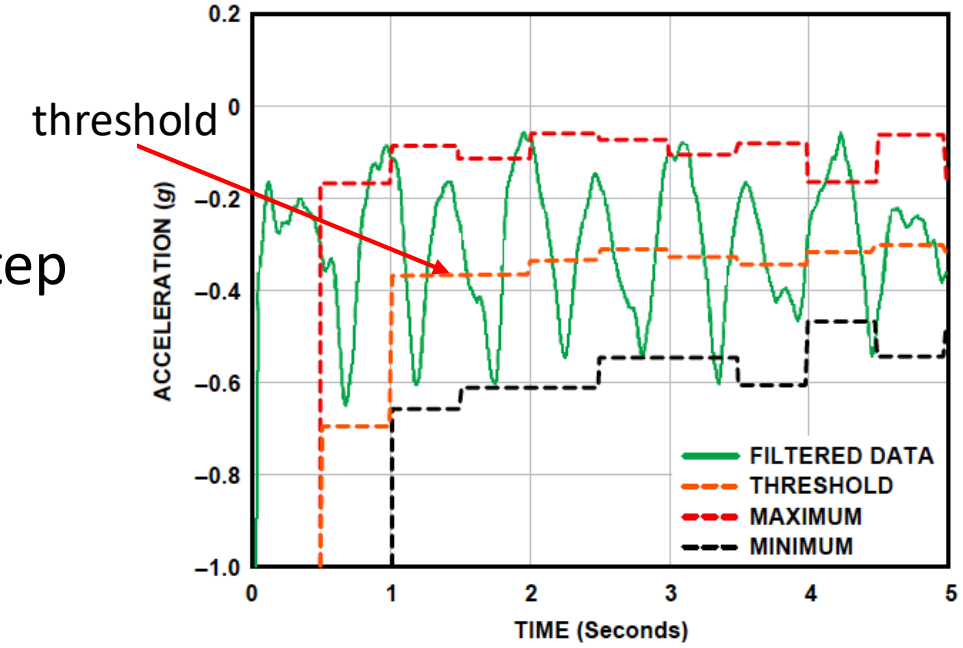




Step detection algorithm

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

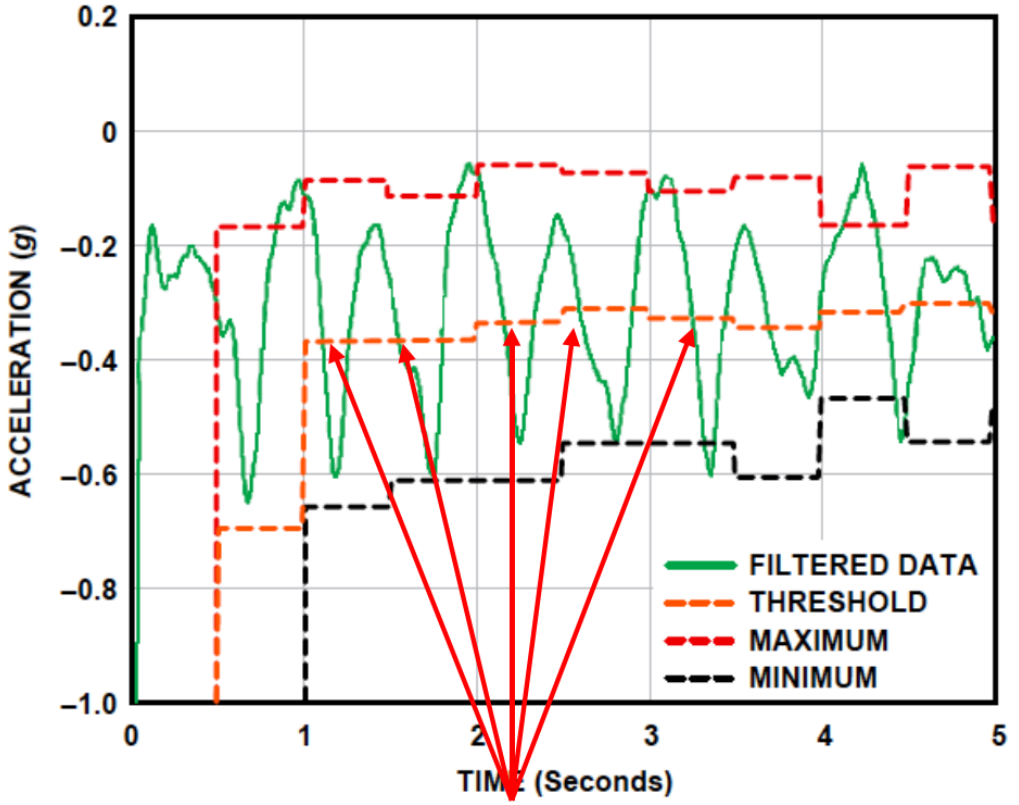
- **Step 1: smoothing**
  - Signal looks noisy
  - Smooth by replacing each sample with average of current, prior, and the one before the prior one (moving average with window size = 3, or more if needed)
- **Step 2: dynamic threshold detection**
  - Focus on the accelerometer axis with largest peak
  - Would like a threshold such that each crossing is a step
  - Threshold cannot be fixed (magnitude depends on phone orientation and walking style)
  - Track *min*, *max* values observed every 50 samples
  - Compute dynamic threshold: *(max + min)/2*



Step detection algorithm

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

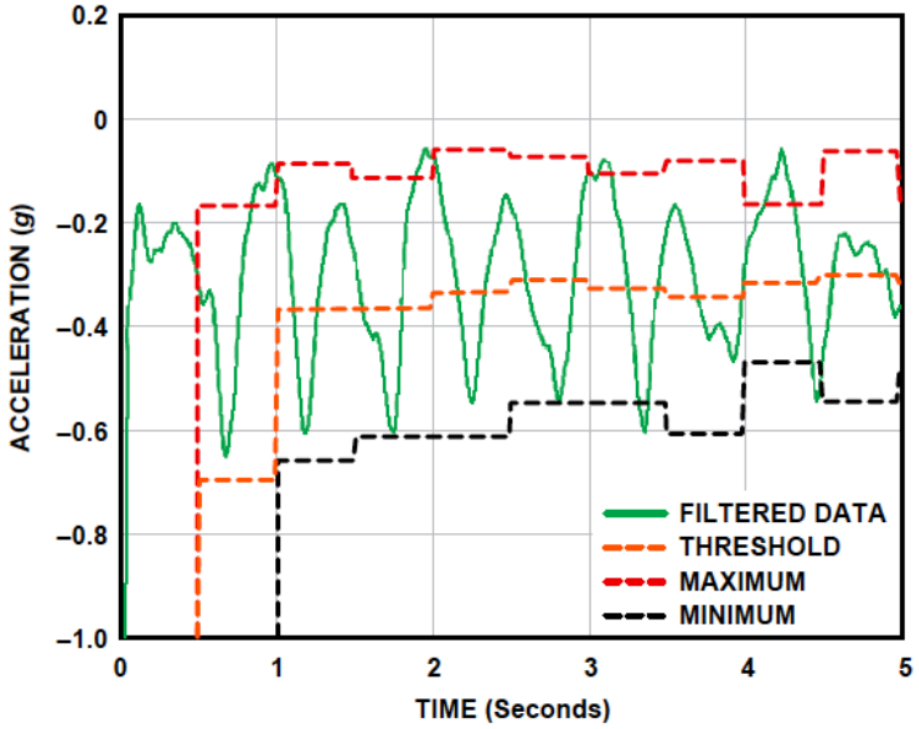
- A step is
  - indicated by crossings of dynamic threshold
  - defined as negative slope ($sample_i < sample_{i-1}$) when smoothed waveform crosses dynamic threshold



Step detection algorithms

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

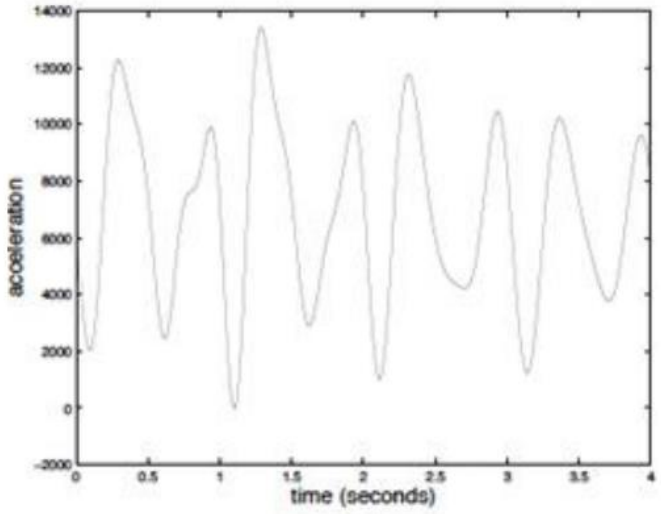
- Problem: vibrations (e.g. mowing lawn, car engine) could be counted as steps
- Periodicity of walking/running can be useful
- Assume people can:
  - Run: 4 steps per second => 0.25 seconds per step
  - Walk slowly: 1 step every 1 second => 1 second per step
  - Eliminate “negative crossings” that occur outside period [0.25 – 1 seconds] (e.g. vibrations)



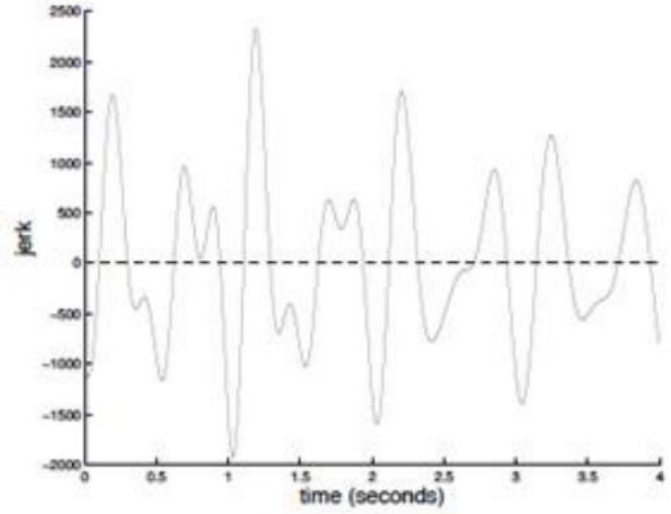
Step Detection Algorithms

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

- Previous step detection algorithm is simple
- Can use more sophisticated signal processing algorithms for smoothing
- Frequency domain processing (e.g. low-pass filter)



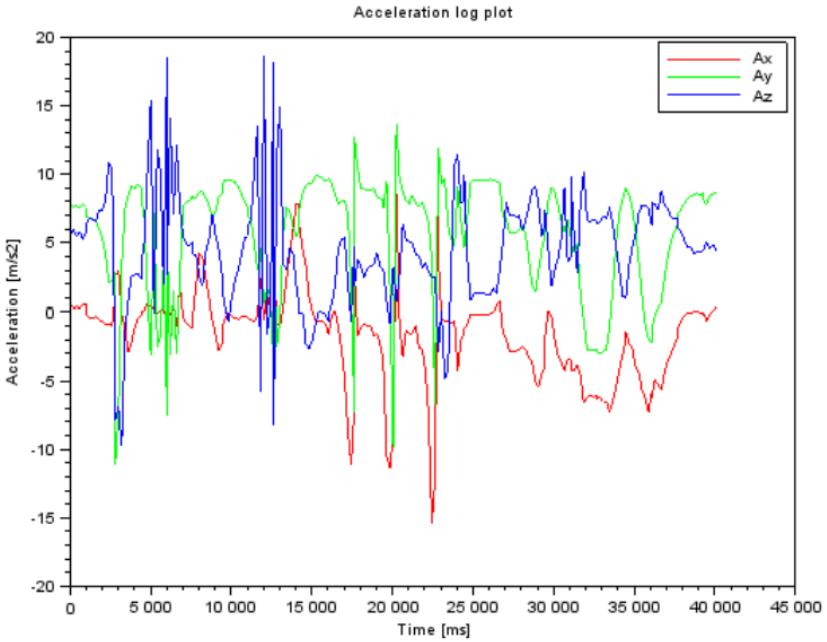
(c) Output of the low-pass filter.



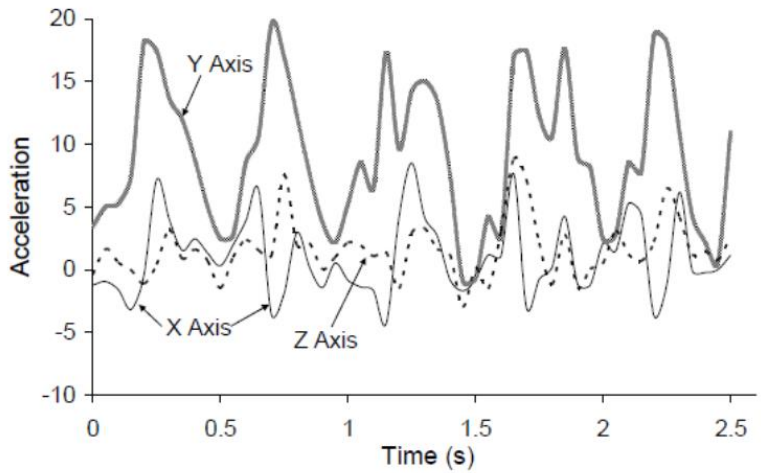
(d) Derivative of the low-pass filter.

Activity recognition

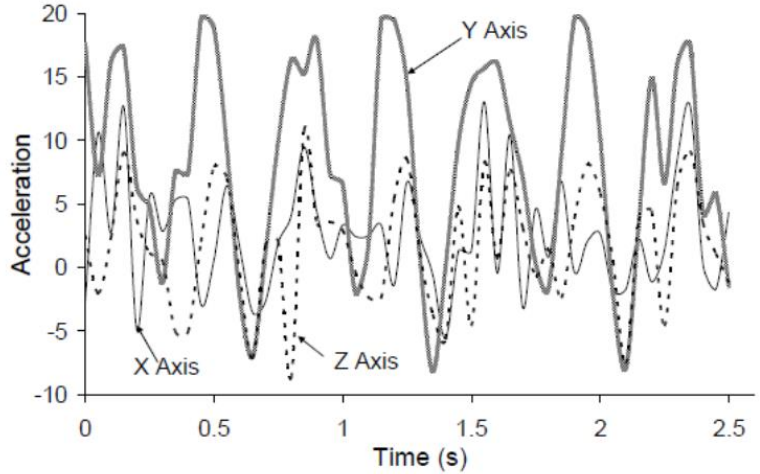
- Goal: Want our app to detect what activity the user is doing?
- Classification task: which of these 6 activities is user doing?
  - *Walking*
  - *Jogging*
  - *Ascending stairs*
  - *Descending stairs*
  - *Sitting*
  - *Standing*
- Typically, use machine learning classifers to classify user's accelerometer signals



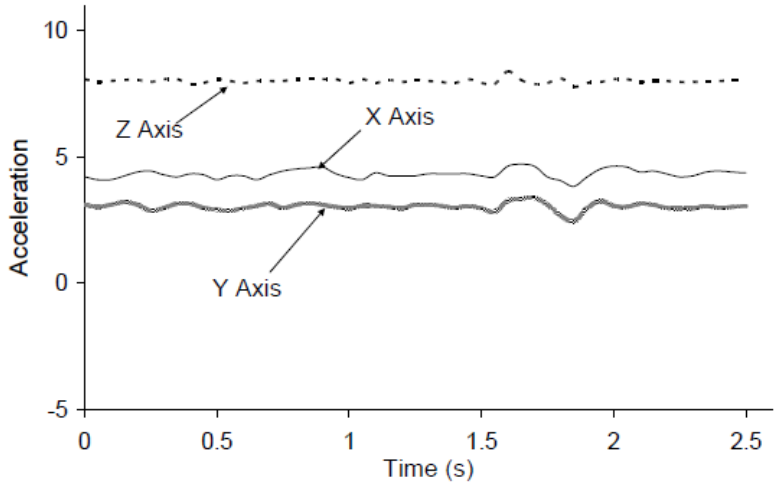
Acceleration for the different activities



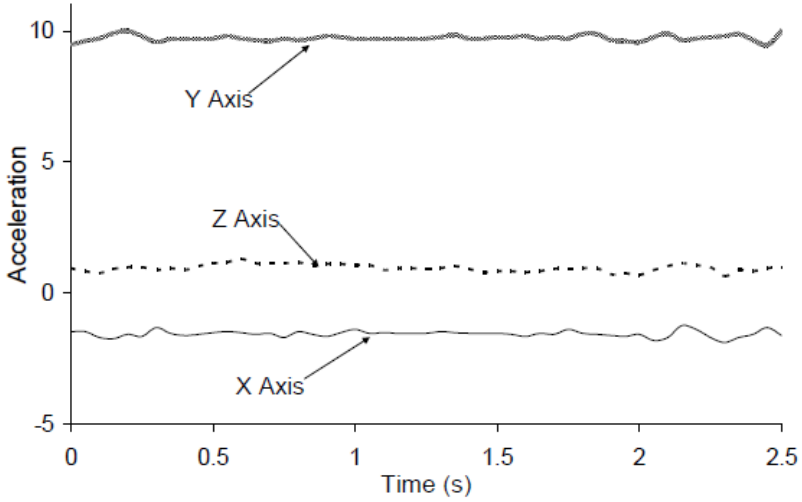
(a) Walking



(b) Jogging

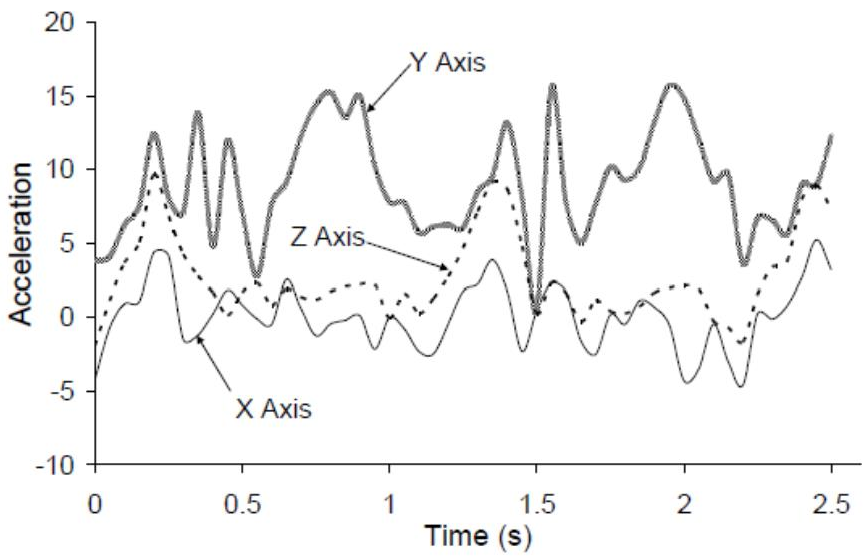


(e) Sitting

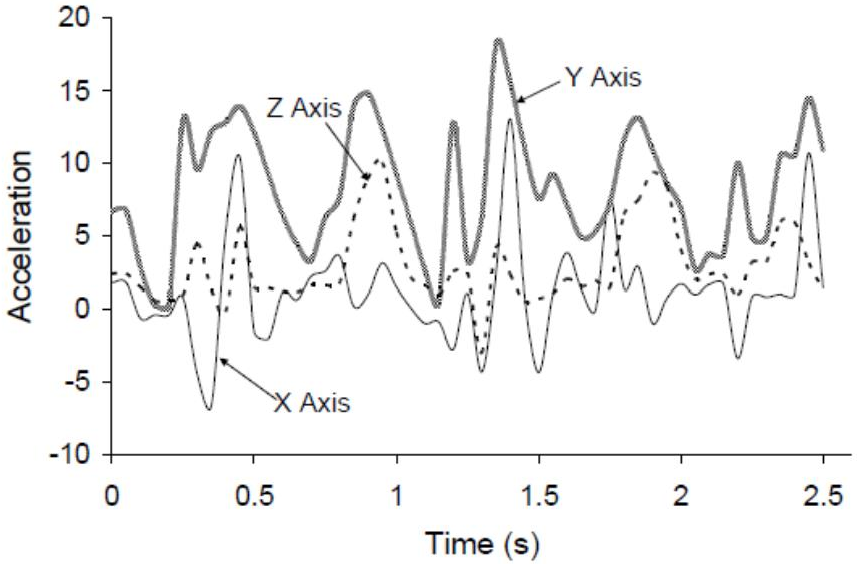


(f) Standing

Acceleration for the different activities



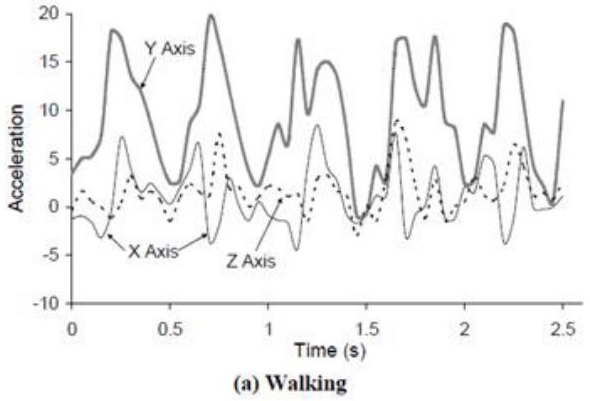
(c) Ascending Stairs



(d) Descending Stairs

Activity recognition overview

Gather Accelerometer data



(a) Walking

Machine Learning Classifier

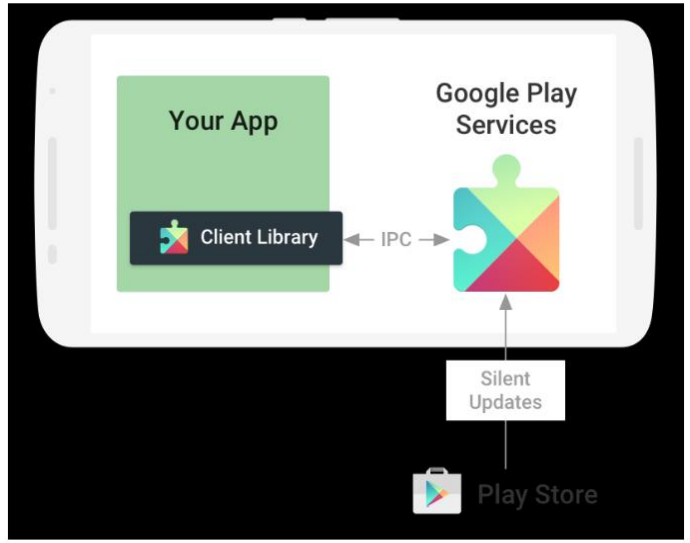
Classify Accelerometer data

Walking

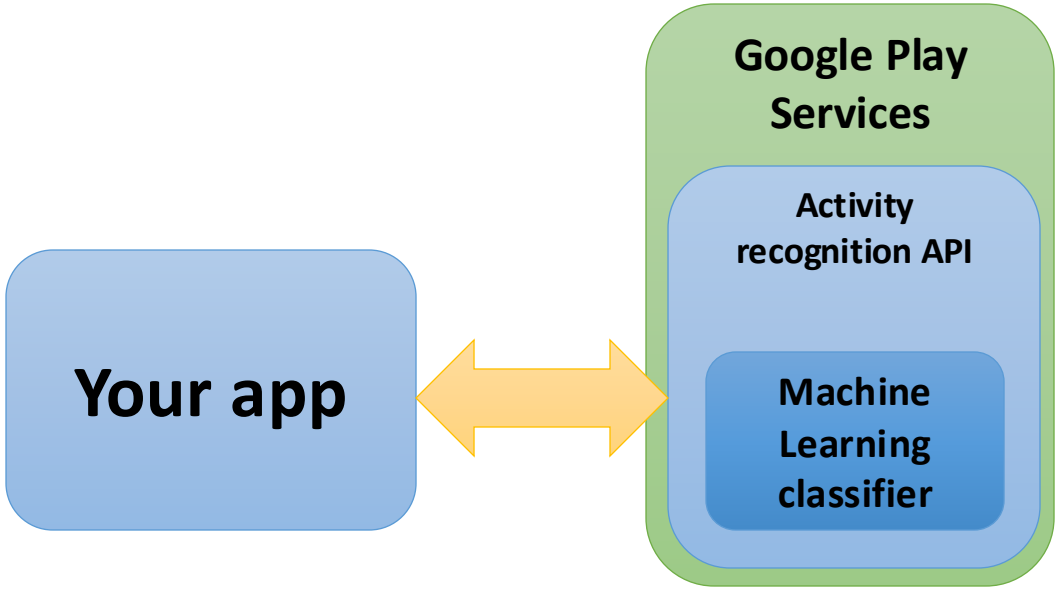
Running

Climbing Stairs

Google APIs for activity recognition



- API to detect smartphone user's current activity
- Can be used by your Android app
- Currently detects 8 states:
  - *In vehicle*
  - *On Bicycle*
  - *On Foot*
  - *Running*
  - *Walking*
  - *Still*
  - *Tilting*
  - *Unknown*
- Two forms:
  - Transition based: app receives intents when activity starts/stops
  - Periodic: app receives intents periodically



Google APIs for activity recognition

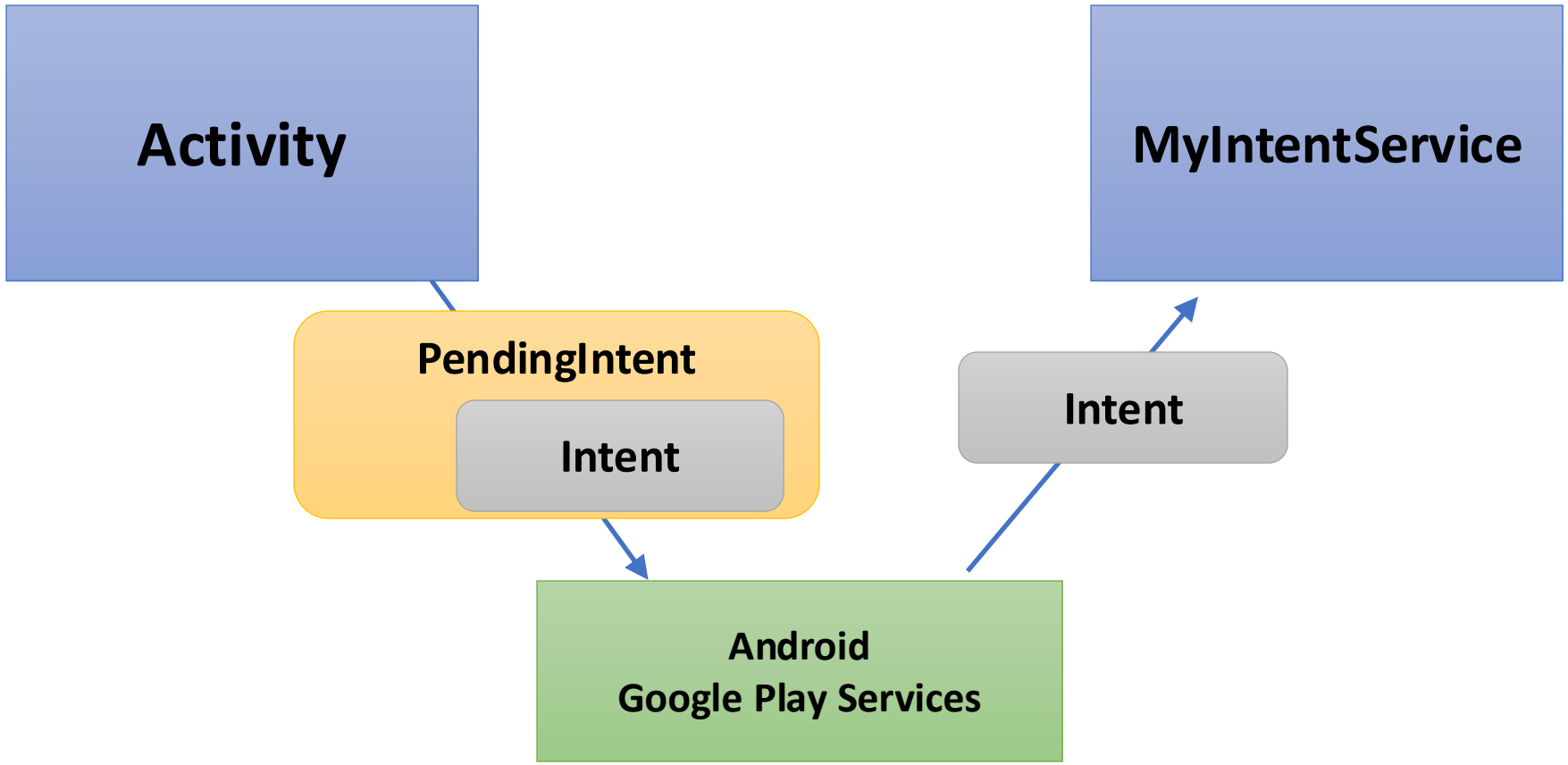
- *Google Play Services* is a component containing closed source code
- It provides an interface to several APIs
- Example: we want to log the activity the user is performing (no GUI to make it simple)
- In main Activity (for example in *onCreate()*):

Client for Google API

```
val PERIOD = 1000L
val arc = ActivityRecognition.getClient(this)
val i = Intent(this, MyIntentService::class.java)
val pi = PendingIntent.getService(this,1, i, PendingIntent.FLAG_MUTABLE)
if (ActivityCompat.checkSelfPermission(
        this,
        Manifest.permission.ACTIVITY_RECOGNITION
    ) != PackageManager.PERMISSION_GRANTED
) {
    val perms = arrayOf(Manifest.permission.ACTIVITY_RECOGNITION)
    ActivityCompat.requestPermissions(this, perms, 1)
    //...
}
val t = arc.requestActivityUpdates(PERIOD, pi)
t.addOnSuccessListener {Log.d("Example", "Successful registration")}
t.addOnFailureListener{Log.d("Example", "Failure in registration")}
```

A container for an intent that will be fired in the future

Google APIs for activity recognition



Google APIs for activity recognition

Code of MyIntentService

Note: *Services* are executed by the same main thread that executes activities methods

```
override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
    Log.d("Example", "Service started")
    super.onStartCommand(intent, flags, startId)
    if(intent == null)
        return START_NOT_STICKY
    val arr = ActivityRecognitionResult.extractResult(intent)
    val detectedActivities = arr?.probableActivities
    if (detectedActivities != null) {
        for (d in detectedActivities) {
            val str = convertToString(d)
            Log.i("Example", "Detected activity: " + str + ", " + d.confidence)
        }
    }
    return START_NOT_STICKY
}
```

```
private fun convertToString(d: DetectedActivity): String{
    when(d.type){
        DetectedActivity.ON_BICYCLE -> return "on bicycle"
        DetectedActivity.IN_VEHICLE -> return "in vehicle"
        ...
```

Google APIs for activity recognition

```
<uses-permission android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION" />
or
<uses-permission android:name="android.permission.ACTIVITY_RECOGNITION" />
```

```
dependencies {
    implementation 'com.google.android.gms:play-services-location:17.0.0'
}
```

- Output:

```
.579-1579  Example              it.unipi.dii.masssarexample      D  Service started
.579-1579  Example              it.unipi.dii.masssarexample      I  Detected activity: still, 96
.579-1579  Example              it.unipi.dii.masssarexample      I  Detected activity: on foot, 2
.579-1579  Example              it.unipi.dii.masssarexample      I  Detected activity: walking, 2
.579-1579  Example              it.unipi.dii.masssarexample      I  Detected activity: in vehicle, 1
.579-1579  Example              it.unipi.dii.masssarexample      I  Detected activity: unknown, 1
```

Awareness API

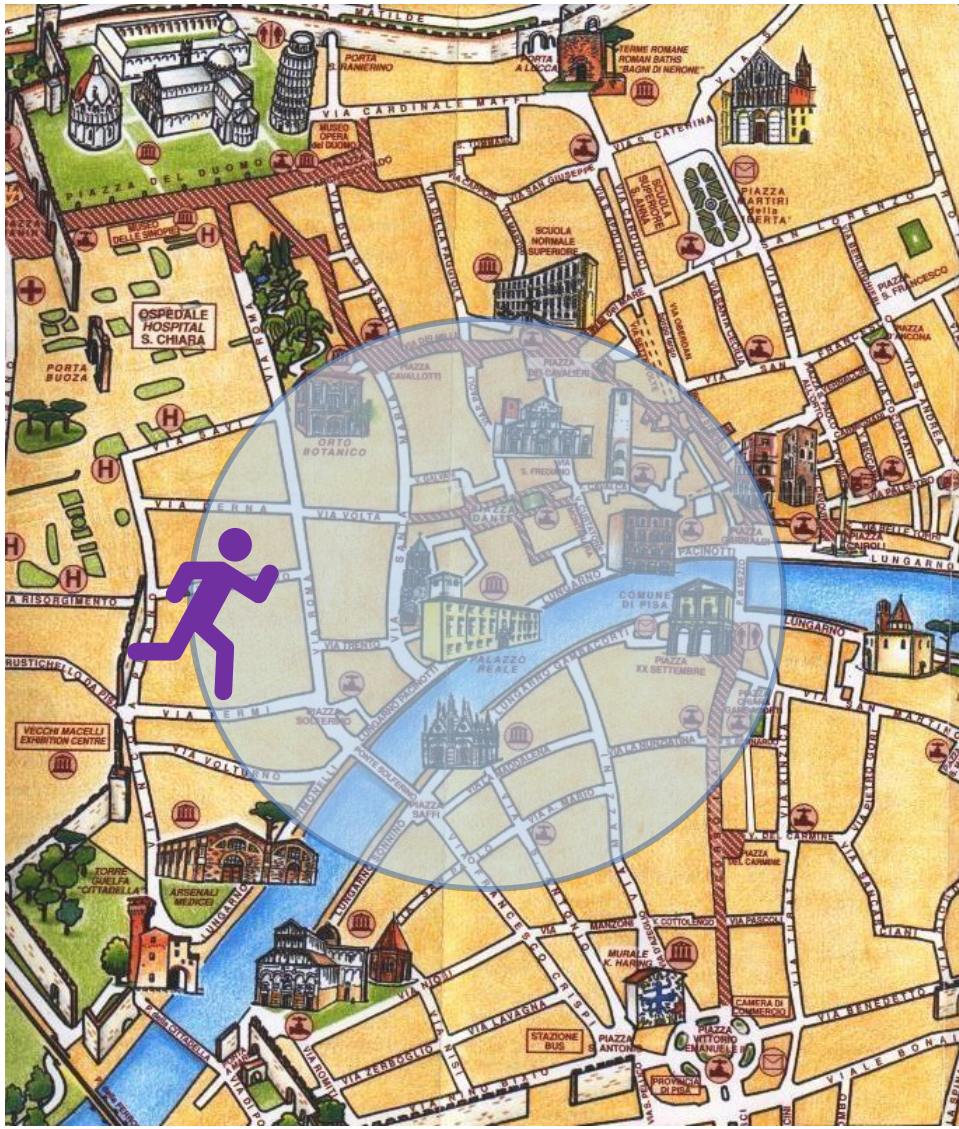
https://developers.google.com/awareness/overview

- Single Android API for context awareness
- Combines some other existing APIs (*Activity*, *Location*)

Context type	Example
Time	Current local time
Location	Latitude and longitude
Activity	Detected user activity, like walking, running, or biking
Beacons	Nearby beacons that match the specified namespace
Headphones	Status of whether headphones are plugged in, or not

Awareness API

- Snapshot API:
  - Return cached values of *Location*, *Activity*, *Time* (Morning, Afternoon, Weekend, etc)
  - Optimized for battery and power consumption
- Fences API:
  - Used to set conditions to trigger events
  - E.g. `if(user enters a geoFence && Activity == running)` notify my app



Android sensors

- Android provides an API to access sensors and obtain data with uniform interface
- In general, apps
  1. Obtain the list of available sensors
  2. Register listeners to one (or more)
  3. Process data received through callbacks
- Correspond to the widget-based context support
- Sensors provide abstractions
  - Drift compensation, calibration
  - Some are SW sensors, fuse data originated by HW ones

Android sensors

- Android sensors can be categorized as
  - Environmental sensors
  - Motion/orientation sensors
- The real sensors are different from device to device, and their availability depends on API level
- *SensorManager* provides methods for
  - Obtaining the list of sensors
  - Registering listeners

Android sensors

- List sensors and info:

```
fun f(){
    val sm: SensorManager? =
        getSystemService(android.content.Context.SENSOR_SERVICE) as SensorManager?
    val list = sm?.getSensorList(Sensor.TYPE_ALL)
    if (list == null)
        return
    for(x in list)
        Log.i(TAG, "Type: " + x.getStringType() +
                ", name: " + x.getName() +
                ", vendor: " + x.getVendor() +
                ", current (mA): " + x.getPower() +
                ", is wakeup: " + x.isWakeUpSensor() +
                ", FIFO: " + x.getFifoMaxEventCount())
}
```

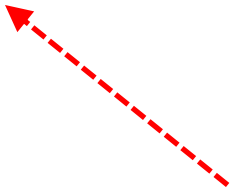
Type: **accelerometer**, name: **LSM6DSM Accelerometer**, vendor: STMicroelectronics, current (mA): **0.15**, is wakeup: false, FIFO: 40960
Type: **magnetic_field**, name: **AK09915 Magnetometer**, vendor: AKM, current (mA): **1.8**, is wakeup: false, FIFO: 40960
Type: **gyroscope**, name: **LSM6DSM Gyroscope**, vendor: STMicroelectronics, current (mA): **0.45**, is wakeup: false, FIFO: 40960
Type: **proximity**, name: **TMx490x PROX**, vendor: AMS TAOS, current (mA): **0.1**, is wakeup: true, FIFO: 42922
Type: **elmyra.raw**, name: **Elmyra - Raw data**, vendor: Google, current (mA): **2.2**, is wakeup: false, FIFO: 40960
Type: **step_detector**, name: **LSM6DSM Step Detector**, vendor: **STMicroelectronics**, current (mA): **0.9**, is wakeup: false, FIFO: 40960
and 20+ more on a Pixel 2 XL

Android sensors

- To register a listener and receive events

```
val s: android.hardware.Sensor? =
    sm.getDefaultSensor(android.hardware.Sensor.TYPE_LINEAR_ACCELERATION)
sm.registerListener(this, s, SensorManager.SENSOR_DELAY_GAME)
```

Object that implements the *SensorEventListener* interface



Requested sampling rate

```
public override fun onSensorChanged(event: SensorEvent) {
    val m: FloatArray = event.values
    Log.i("Example", m.contentToString())
}
public override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
    Log.i(TAG, "Accuracy changed")
}
```

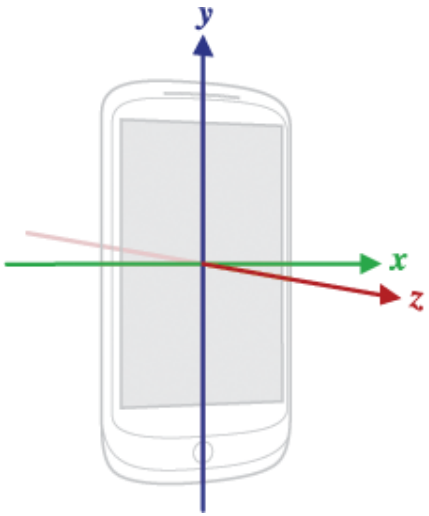
Android sensors: environment

- Environmental sensors:
  - Ambient temperature
  - Barometer
    - Can be useful to detect if the user moves to the upper/lower floor in a building
    - May be subject to large fluctuations
  - Light
  - Humidity

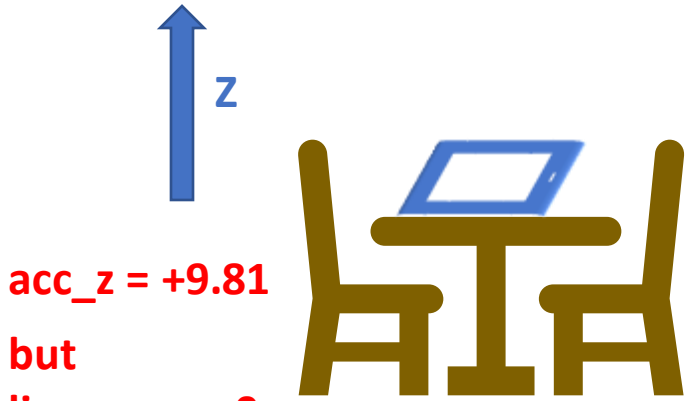
Android sensors: motion/orientation

- Used to detect movements such as tilt, rotation, shake, swing
- Sensors
  - **Accelerometer**: returns acceleration
  - **Gyroscope**: returns angular velocity
  - **Gravity**: returns direction of gravity, useful for understanding device orientation
  - **Linear acceleration**: returns acceleration - gravity
  - **Rotation vector**: returns device orientation
  - **Significant motion**: used as a trigger to understand device is picked up
  - **Step detection**: an event for each step
  - **Step counter**: returns the cumulative number of steps since reboot
  - **Magnetic field**: amount of geomagnetic field along 3 axes
- Accel. and gyro. are always HW, others can be HW or SW

Android sensors: motion

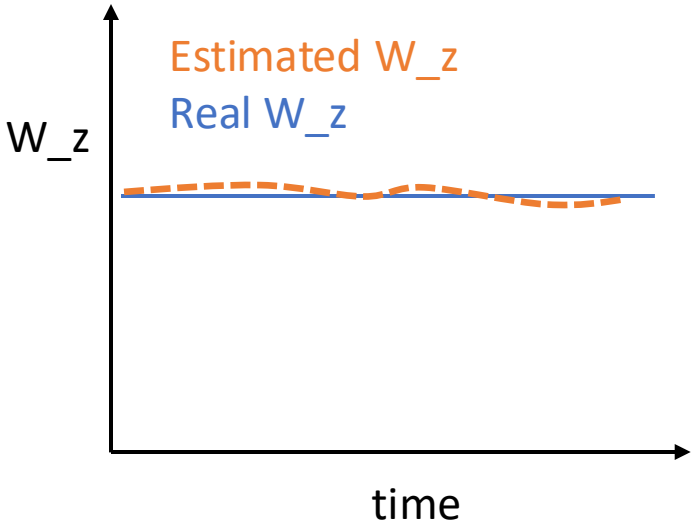


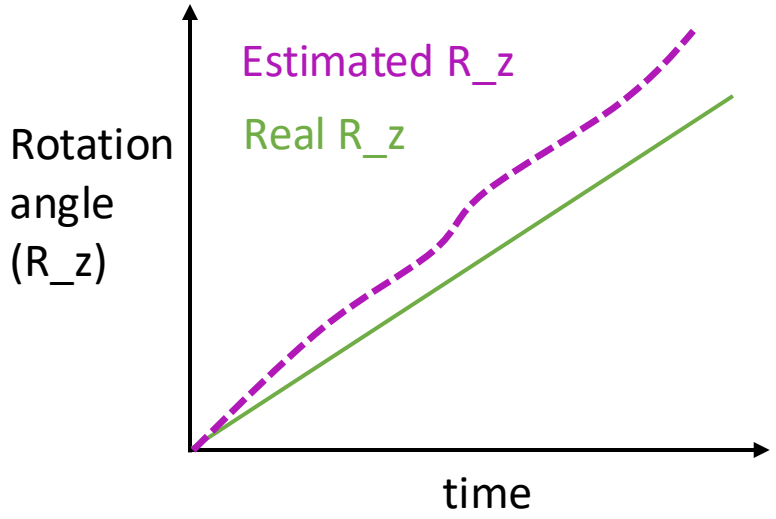
- **Acceleration**: includes gravity
  - Standing still on a table, face up: *acc_z = +9.8m/s^2*
  - Free fall*: acc_x=0, acc_y=0, acc_z=0*
- **Gravity***:* estimated using a low-pass filter or more complex techniques possibly involving also other sensors
- **Linear acceleration**: gravity is subtracted
  - Standing still on a table: 0 on all axes
  - Gravity is first estimated, then removed
  - In many cases, gravity can be a confounding factor: if so, use linear acceleration
    - E.g. understand if device is shaken
- **Linear acceleration = acceleration - gravity**



Android sensors: orientation

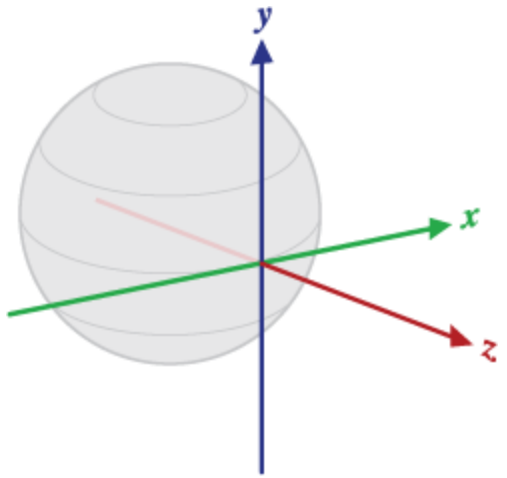
- **Gyroscope**
  - Returns angular velocity
  - To compute rotated angle, velocity must be integrated
  - Errors are accumulated: drift in computing angles

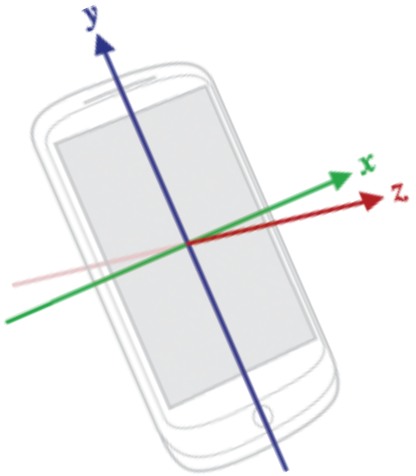




Android sensors: orientation

- **Rotation vector**
  - Goal: compute azimuth, pitch, roll
    - *Azimuth*: angle around z-axis
    - *Pitch*: angle around x-axis
    - *Roll*: angle around y-axis
  - Based on sensor fusion: gyroscope, accelerometer, geomagnetic field
    - Accelerometer can provide information avout pitch and roll because of gravity, but not about azimuth
    - Geomagnetic field can provide information about azimuth because of magnetic north
    - Gyroscope provide info, drift eliminated by means of the other sensors
  - Returns orientation with respect to global reference frame



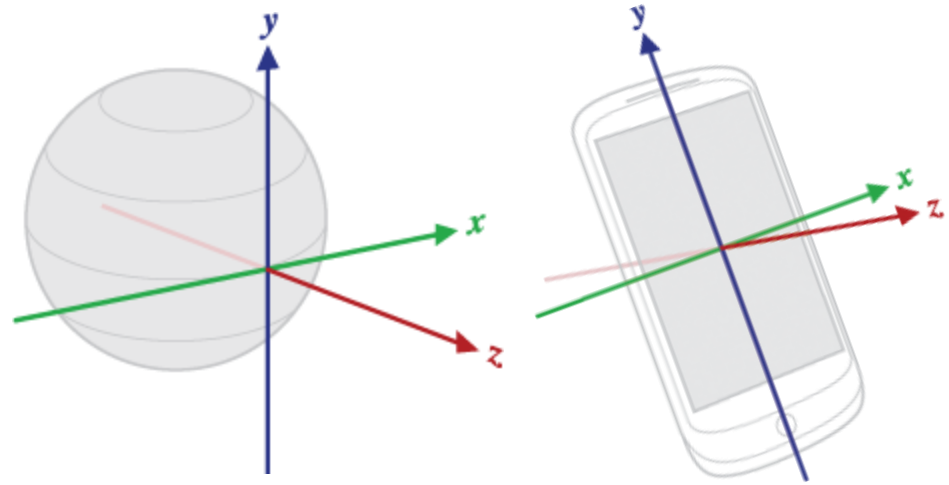


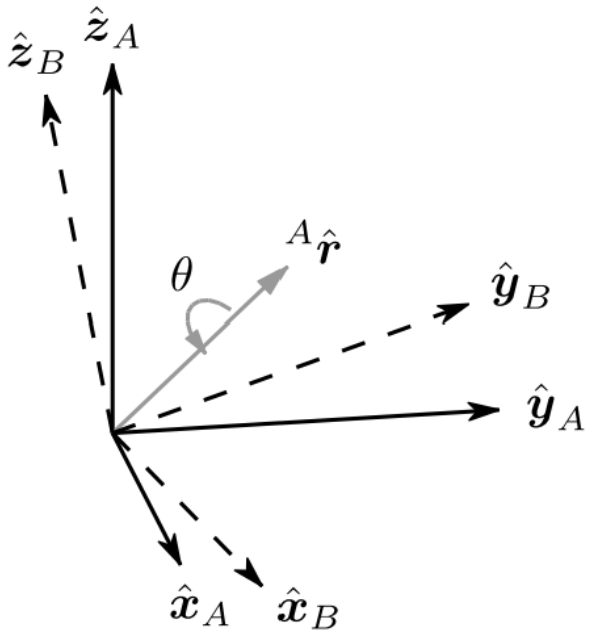
Android sensors: orientation

- **Rotation vector**
  - Orientation: rotation needed to transform a reference system into the other one
  - Orientation information can be represented as a vector (x, y, z) which identifies the rotation axis + an angle
  - Android's rotation vector returns quaternion representing same information

    $(x \cdot \sin(\theta/2), y \cdot \sin(\theta/2), z \cdot \sin(\theta/2))$

  - Commonly used in maps/games





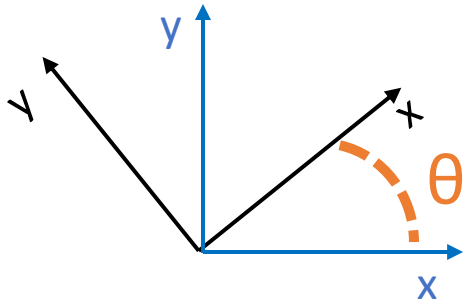
The orientation of frame B is achieved by a rotation, from alignment with frame A, of angle θ around the axis Ar.

Android sensors: orientation

- Suppose phone on table
- Rotation θ = PI/4
- Rotation vector: [0, 0, 1], corresponds to z-axis
- Android returns rotation vector =

  [0, 0, 0.38..]

  0.38 = sin(PI/8)
- In general you need azimuth-pitch-roll angles, android provides a converting function



Android sensor: orientation

```
private var gs: Sensor? = null
private lateinit var sm: SensorManager
private val rotationMatrix = FloatArray(9)
private val orientationAngles = FloatArray(3)
private var tv1: TextView? = null
private var tv2: TextView? = null
private var tv3: TextView? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    tv1 = findViewById<View>(R.id.tv1) as TextView
    tv2 = findViewById<View>(R.id.tv2) as TextView
    tv3 = findViewById<View>(R.id.tv3) as TextView
    sm = getSystemService(SENSOR_SERVICE) as SensorManager
    gs = sm.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR)
}
```

Android sensor: orientation

```
override fun onSensorChanged(event: SensorEvent) {
    if (event.sensor.type == Sensor.TYPE_ROTATION_VECTOR) {
        SensorManager.getRotationMatrixFromVector(rotationMatrix, event.values)
        SensorManager.getOrientation(rotationMatrix, orientationAngles)
        Log.d(
            "APR", String.format(
                "Quaternion components: %.2f %.2f %.2f",
                event.values[0], event.values[1], event.values[2]
            )
        )
        tv1!!.text = String.format(
            "Az: %.0f", Math.toDegrees(
                orientationAngles[0]
                    .toDouble()
            )
        )
        tv2!!.text = String.format(
            "Pi: %.0f", Math.toDegrees(
                orientationAngles[1]
                    .toDouble()
            )
        )
    ...
```

Location and maps

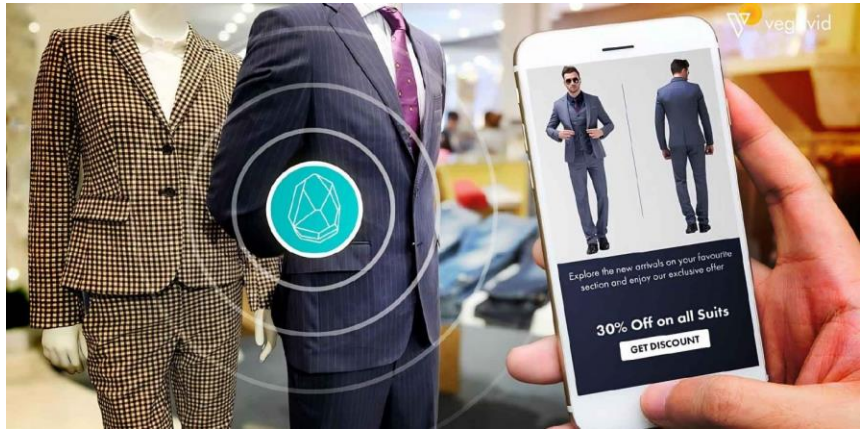
Localization

- Localization methods can be evaluated in terms of
  - **Accuracy**: the distance between the estimated location and the true location
  - **Precision**: consistency or repeatability of measurements
  - **Delay**: time needed to obtain a position fix
  - **Time costs**: time needed for installation and maintenance
  - **Energy**: the energy needed to operate the system
  - **Monetary costs**: additional equipment needed for localization
  - **Robustness**: still operational despite faults or input unavailability
  - **Scalability**: how many devices, covered surface, how much hardware needed when increasing coverage

Localization

- Depending on the application, some criteria are more relevant than others.

- Accuracy
- Precision
- Delay
- Energy

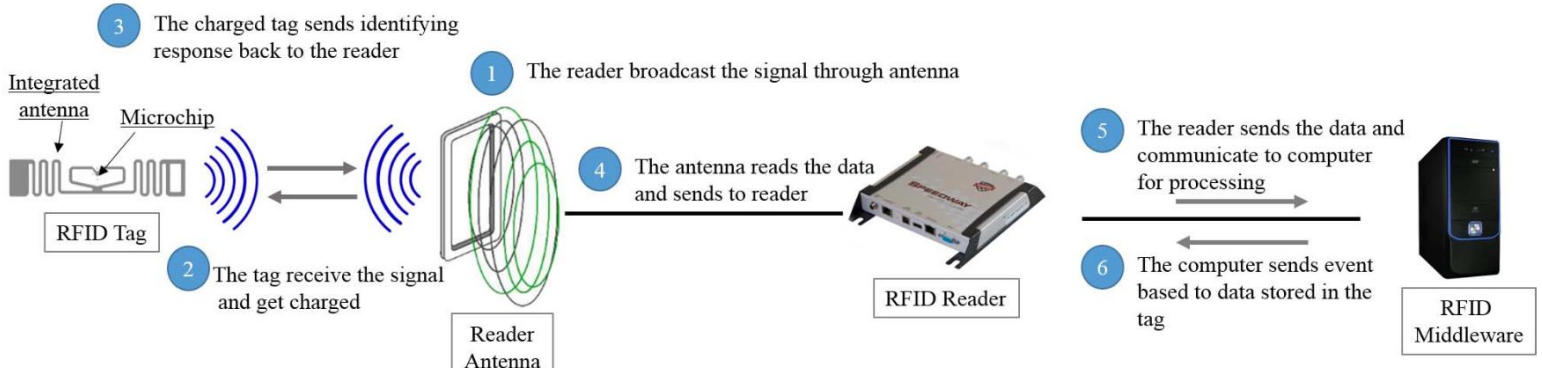


- Energy
- €
- Scalability

Localization

- Outdoor: Global Navigation Satellite System (GPS, Galileo, GLONASS), cell identification
- Indoor:
  - Vision:
    - Fixed camera systems: mobile target is tracked and the location is computed from cameras' positions
    - Mobile camera systems: mobile target is equipped with a camera, landmarks are placed at known positions
  - Infrared: user wears a badge that emits a unique ID at regular intervals. IR receivers placed at known positions are used to compute location. Signal is automatically confined within rooms. LoS is required.
  - Ultrasound: synchronization between devices is obtained via radio signals, and distance is estimated using the time an audio signal takes to reach the other device (~330 m/s).

- Wi-Fi: can be used in different ways.
  - Cell of Origin: target is located where the strongest AP is located.
  - Triangulation: RSSIs are converted into distances then triangulation is applied
  - Fingerprint: a map of RSS is first built, then observed RSSIs are used to find the location according to a similarity metric
- Bluetooth: beacons with known location emit a unique ID, target is located according to the closest beacon. More energy efficient than Wi-Fi.
- RFID:



A Survey on Indoor Positioning Systems (Mainetti et al.)

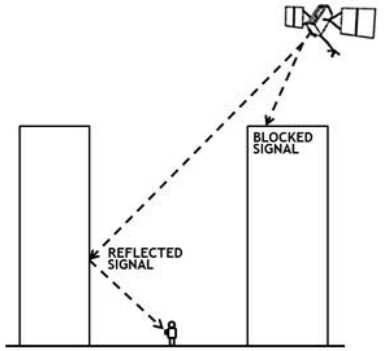
Localization

TABLE I. COMPARISON AMONG INDOOR LOCALIZATION TECHNOLOGIES

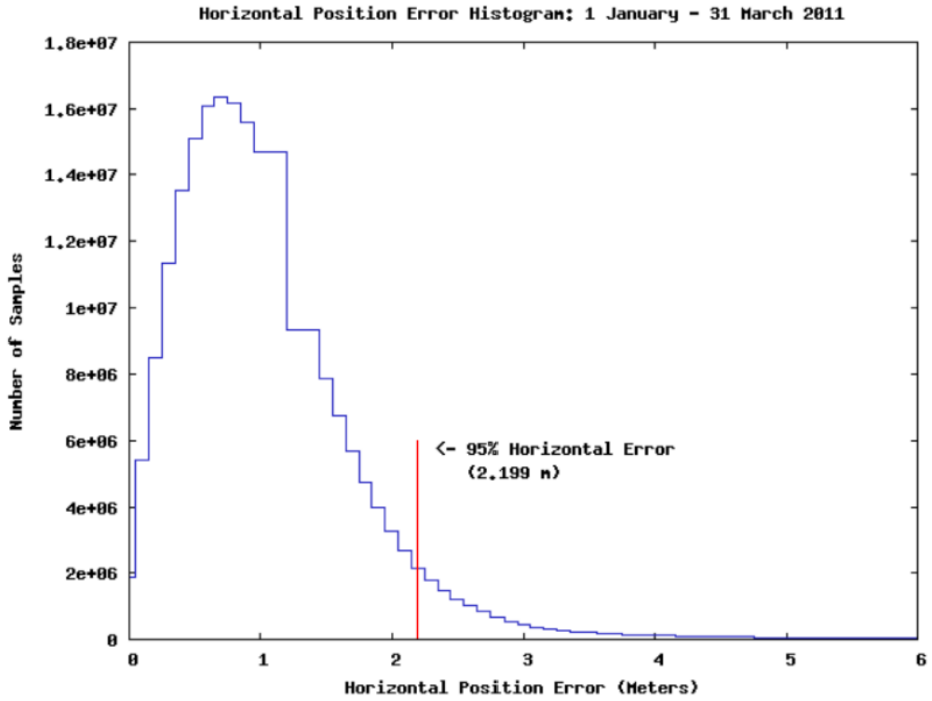
		Parameters				
		Accuracy[m]	***Coverage [m]***	***Cost***	***Complexity***	***Typical Environment***
Technologies	***Vision***	$10^{-3} \div 10^{-1}$	1-10	High	High	Indoor
	Infrared	$10^{-2} \div 1$	1-5	Medium/High	Low	Indoor
	Ultrasound	10^{-2}	2-10	Medium	Low	Indoor
	Wi-Fi	$1 \div 10$	20-50	Medium/Low	Low	Indoor/Outdoor
	RFID	$10^{-1} \div 1$	1-10	Low	Low	Indoor
	Bluetooth	$1 \div 10$	1-30	Low	Low	Indoor/Outdoor

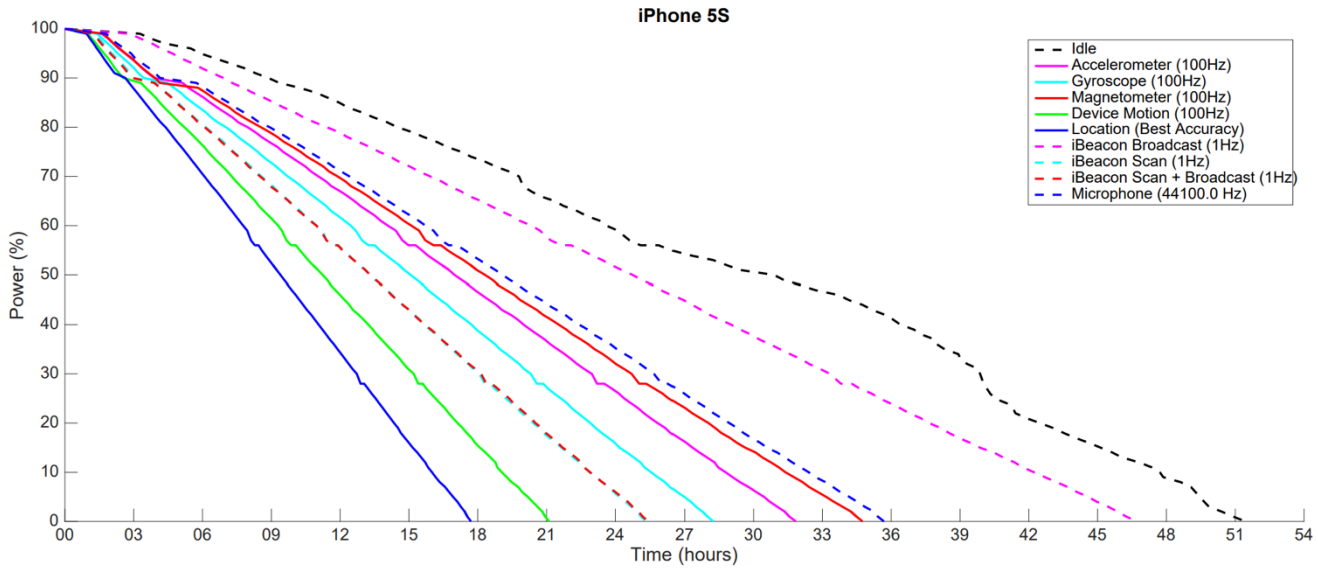
A Survey on Indoor Positioning Systems (Mainetti et al.)

GPS



- Location as *<latitude, longitude>*
- E.g. School of Engineering, Pisa <43.721182, 10.389891>
- GPS reasonably accurate but
  - Requires line-of-sight between satellite and receiver
  - Only works outdoors (signals don't penetrate buildings)
  - Drains battery power
- Alternative: Use Wi-Fi location sensing indoors





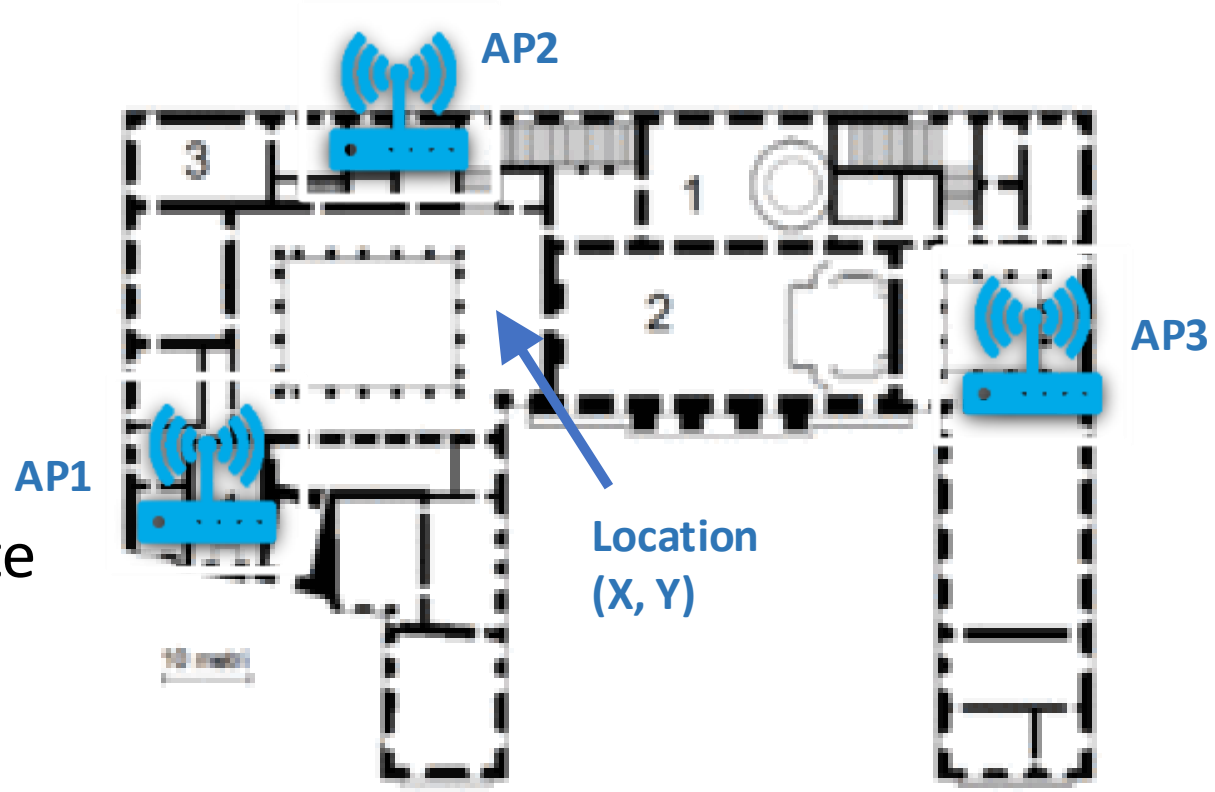
Source: "SensingKit: Evaluating the Sensor Power Consumption in iOS devices", Katevas et al.

WiFi location fingerprinting

- **Basic idea**: At each (X,Y) location the set of WiFi APs observed and their signal strengths is unique

Location	AP1	AP2	AP3
(X, Y)	55	43	65

- **WiFi location fingerprinting**: estimate device's location based on combination of Wi-Fi access points visible + signal strengths



Location estimation using Wi-Fi fingerprinting

- Need a table containing pre-recorded tuples
- Google builds and stores this database (APs + Signal Strength) at each X,Y location
- Observed values are not exactly the same, find the one with min distance

Pre-recorded signal strength

Location (X, Y)	AP1	AP2	AP3	AP4
(10, 33)	32	43	54	65
(42, 21)	76	11	4	32
(5, 12)	-	22	7	43
(65, 3)	16	19	-	5
(8, 21)	2	39	25	17
…	…	…	…	…

Observed signal strength

AP1	AP2	AP3	AP4
74	11	5	31

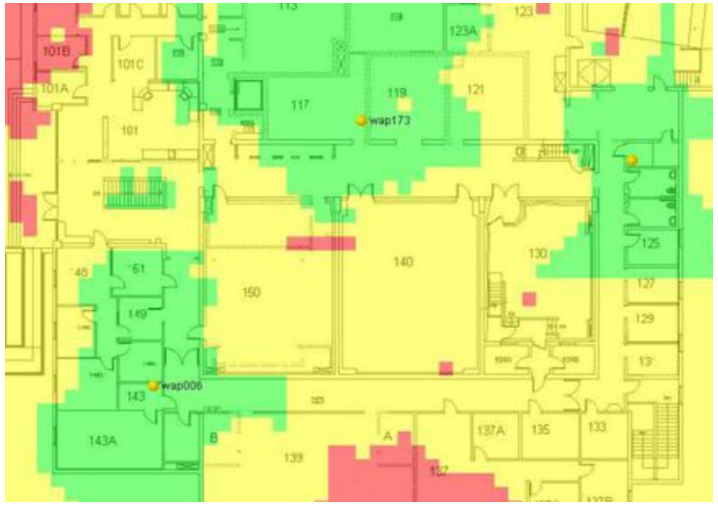
$$argmin \sum_{i} (r_i - s_i)^2$$

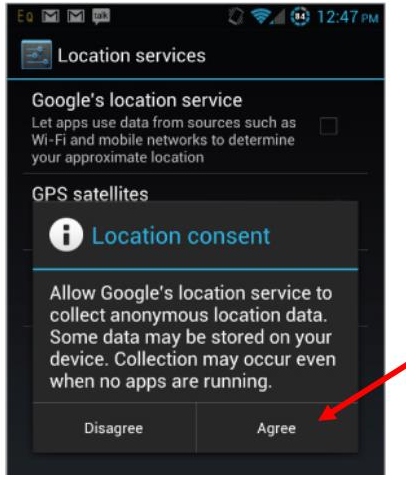
r_i : RSSI of APi offline stage

s_i : RSSI of APi on-line stage

Building tables

- How to Build table of APs observed at different locations?
- Devices (e.g. smartphone) with GPS and WiFi turned on simultaneously build table
- Send data to third party repositories (e.g. Wigle.net) or Google
- This process is known as *war driving*
- Can record cell tower signal strength instead of AP





Location (X, Y)	AP1	AP2	AP3	AP4
(10, 33)	32	43	54	65
(42, 21)	76	11	4	32
(5, 12)	-	22	7	43
(65, 3)	16	19	-	5
(8, 21)	2	39	25	17
…	…	…	…	…

GPS collects location

WiFi collects identifiers of visible AP and signal strength

Location in Android

- Android now has 2 location APIs
  - **New** location API is provided by Google Play Services (closed source)
  - **Old** Android framework location APIs (*android.location*)
- We will see the new location API, the other is similar
- Increasing restrictions in recent versions of Android
  - Location has a deep impact on user's privacy
  - Energy consumption

The new Location API

- Apps that use Location require permissions:
  - **ACCESS_COARSE_LOCATION**: precision=~city block
  - **ACCESS_FINE_LOCATION**: precision=as much as possible
  - **ACCESS_BACKGROUND_LOCATION**: when app in background
- Example:

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

- If the device is running Android 6.0 or higher, and your app's target SDK is 23 or higher, app also must request those permissions at run time
- Permissions can be granted or removed by the user after installation

Obtaining Location

- The new API fuses information coming from GPS and network for better precision and energy consumption
- App interacts with *FusedLocationProvider* through a *Client* object

```
private lateinit var fusedLocationClient: FusedLocationProviderClient

fun f() {
    fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)
    ...
}
```

Location request

- You have to specify the characteristics of desired location sampling using the *LocationRequest* class

```
myRequest = LocationRequest.Builder(10000L).setPriority(Priority.PRIORITY_HIGH_ACCURACY).build()
```

- Enable/disable updates according to activity lifecycle

```
override fun onResume() {
    super.onResume()
    if (ActivityCompat.checkSelfPermission(this,
            Manifest.permission.ACCESS_FINE_LOCATION
        ) == PackageManager.PERMISSION_GRANTED
    ) {
        val t = fusedLocationClient.requestLocationUpdates(myRequest, myCallback, Looper.getMainLooper()
        )
        t.addOnSuccessListener { Log.d("Example", "Successful registration") }
        t.addOnFailureListener { Log.d("Example", "Failure in registration: " + it.message) }
    }
}
```

```
override fun onPause() {
    super.onPause()
    fusedLocationClient.removeLocationUpdates(myCallback)
}
```

Receiving updates

- An implementation of the *LocationCallback* interface

```
private val myCallback  =  object: LocationCallback() {
    override fun onLocationResult(p0: LocationResult) {
        for (location in p0.locations){
            Log.i("Example", "Loc: " + location.latitude + ", " + location.longitude )
        }
    }
}
```

- Several locations may be returned, each has a timestamp
- Location has methods for getting lat, lon, altitude, accuracy, provider, etc

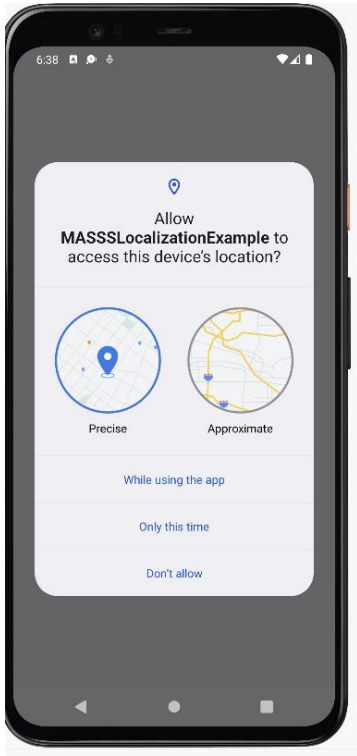
Permissions

- Position is extremely relevant for privacy

```
override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {
    super.onRequestPermissionsResult(requestCode,
                                     permissions, grantResults)
    Log.d("Example", "onReqPermResult: " +
            permissions.contentToString())
    Log.d("Example", "onReqPermResult: " +
            grantResults.contentToString())
}
```

```
if (ActivityCompat.checkSelfPermission(
        this,
        Manifest.permission.ACCESS_FINE_LOCATION
    ) != PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(
        this,
        Manifest.permission.ACCESS_COARSE_LOCATION
    ) != PackageManager.PERMISSION_GRANTED
) {
    ActivityCompat.requestPermissions(this, permissions, 1)
    Log.i("Example", "Requesting permissions")
}
```

app should work also with just coarse if possible
permissions can be removed by the user
and should be checked every time position is requested



Receiving updates in the background

- Previous method is OK in foreground, e.g. in a map
  - also **Foreground services**: a service that shows a notification on the bar, to make its execution clear to users

```
<service
    android:name=".MyForegroundService"
    android:foregroundServiceType="location" ... >
    ...
</service>
```

- If you need to receive location updates in the background
  - **Background**: when the app needs updates also in the absence of user interaction

Receiving updates in the background

- Updates, when app is in background, can be delivered with reduced rate than what the app requested
- Approach is based on *PendingIntents*, which are fired when new updates are delivered

```
val intent = Intent(this, MyBroadcastReceiver::class.java)
intent.setAction(MyBroadcastReceiver.MY_ACTION)
val pi = PendingIntent.getBroadcast(
    this, 0,
    intent, PendingIntent.FLAG_MUTABLE
)
val t = fusedLocationClient.requestLocationUpdates(myRequest, pi)
```

Receiving updates in the background

- The receiver:

```
<receiver
   android:name=".MyBroadcastReceiver"
   android:exported="true">
   <intent-filter>
      <action android:name="MY_ACTION"/>
   </intent-filter>
</receiver>
```

```
class MyBroadcastReceiver: BroadcastReceiver(){
   override fun onReceive(context: Context?, intent: Intent?) {
      if (intent != null) {
         val action = intent.action;
         if (MY_ACTION.equals(action)) {
            val result: LocationResult? = LocationResult.extractResult(intent);
            if (result != null) {
               val locations = result.getLocations();
               for(l in locations)
                  Log.i("Example", "Location: " + l.latitude + " " + l.longitude)
            }
         }
      }
   }

   companion object {
      val MY_ACTION = "MY_ACTION"
   }
}
```

Receiving updates in the background

- Management of permission slightly changes depending on the version of Android
- In 11+:
  - Collect ACCESS_COARSE_PERMISSION and ACCESS_FINE_PERMISSION
  - Check if ACCESS_BACKGROUND_PERMISSION is granted
  - If not, send user to settings to change app permissions
  - if yes, register for updates

Geocoding and reverse geocoding

- Location API represents location as *<latitude, longitude>*
- Symbolic location is better for reasoning about locations
- E.g. Street address (via Diotisalvi 2, Pisa) or (building, floor, room)
- Android supports:
  - **Geocoding**: Convert addresses into longitude/latitude coordinates
  - **Reverse geocoding**: convert longitude/latitude coordinates into human readable address

Places API

- **Place**: physical space that has a name (e.g. local business, point of interest, geographic location)
- E.g Fiumicino airport, place type is airport
- Places API: provides contextual information about places, $ depending on usage
- Information includes: name of place, address, geographical location, place ID, phone number, place type, website URL, etc
- Many types of places
- Get current place: place where device is last known to be located
- GUI element for selecting a place in the nearby

`accounting`

`airport`

`amusement_park`

`aquarium`

`art_gallery`

`atm`

`bakery`

`bank`

`bar`

`beauty_salon`

`bicycle_store`

`book_store`

`bowling_alley`

`bus_station`

`cafe`

`campground`

`car_dealer`

`car_rental`

`car_repair`

`car_wash`

`casino`

`cemetery`

`church`

`city_hall`

`clothing_store`

`convenience_store`

`courthouse`

`dentist`

`department_store`

`doctor`

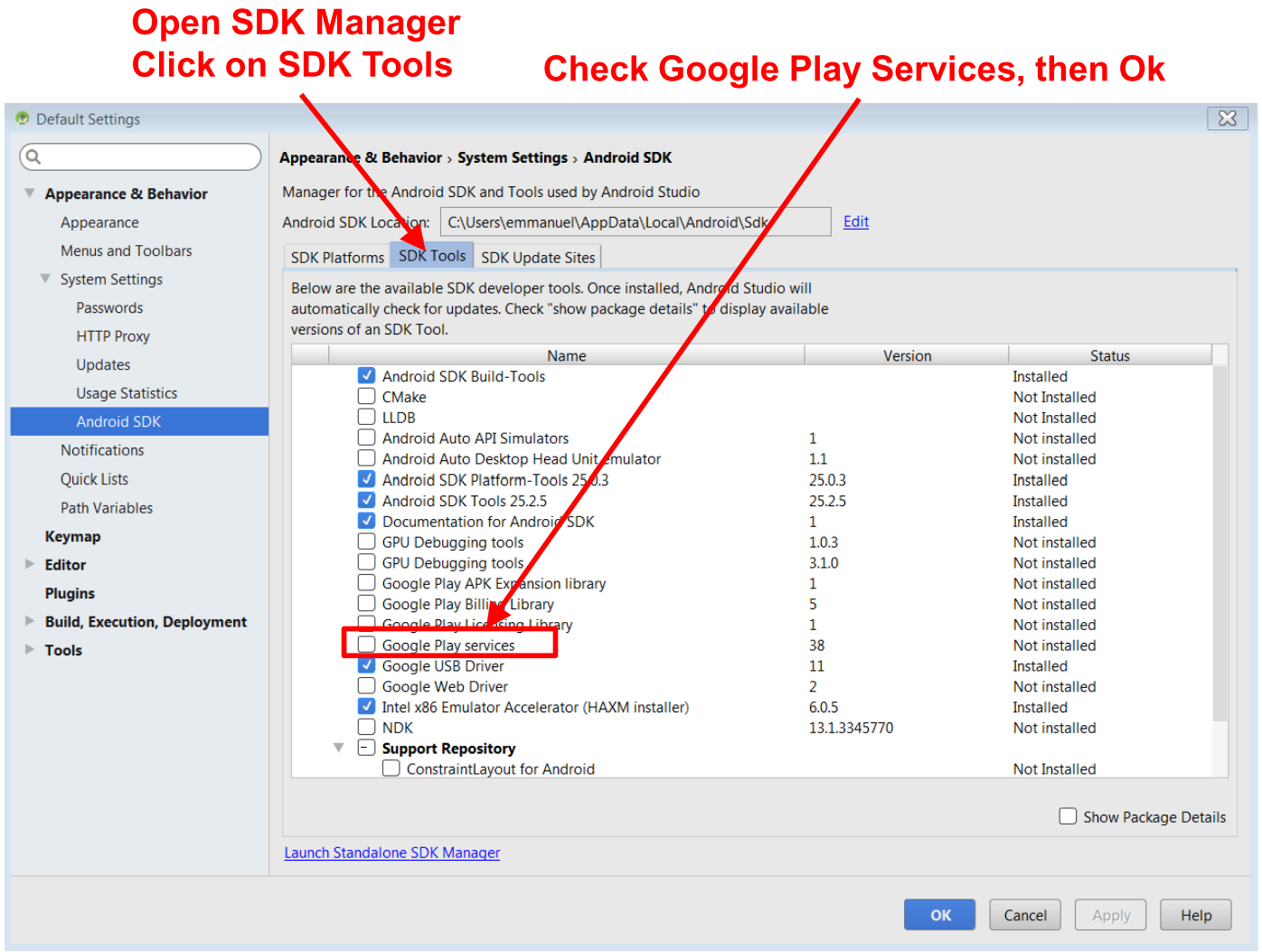
`drugstore`

Other location-related APIs

- **Directions API**: calculates directions between locations (walking, driving, public transportation)
- **Distance Matrix API**: Calculate travel time and distance for multiple destinations
- **Elevation API**: Query locations on earth for elevation information, calculate elevation changes along routes
- **Roads API**: snaps set of GPS coordinates to road user was likely travelling on (best fit)
- Need API key

Maps

- Google Maps is part of Google Play Services SDK
- Use Android Studio SDK manager to download Google Play services



Maps

- There is a fragment class implementing a map
- An interface to be notified that map is ready
- Methods for adding markers, polylines, etc

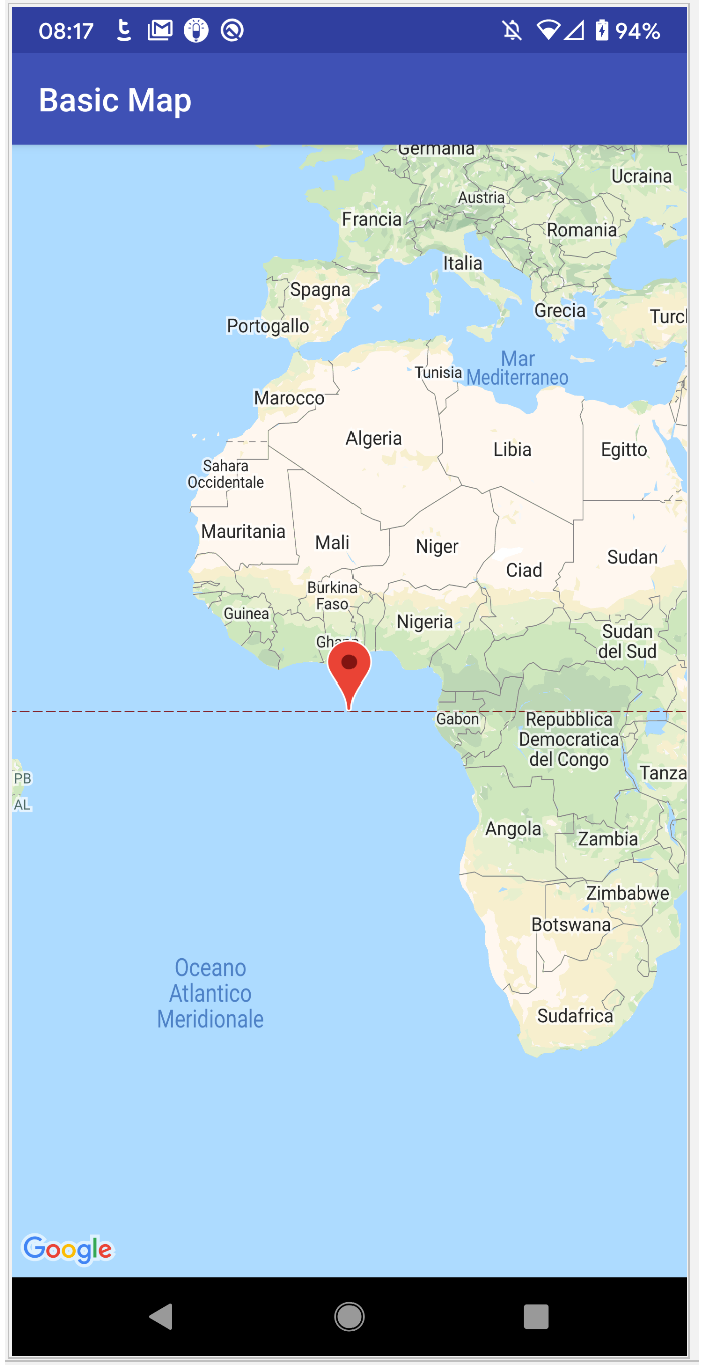
```
class MainActivity : AppCompatActivity(), OnMapReadyCallback {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val mapFragment =
       supportFragmentManager.findFragmentById(R.id.map)
                                as SupportMapFragment
        mapFragment.getMapAsync(this)
    }

    override fun onMapReady(p0: GoogleMap) {
        p0.addMarker(MarkerOptions()
                                .position(LatLng(0.0, 0.0))
                                .title("Marker"))
    }
}
```

Maps

```
<fragment
xmlns:android="http://schemas.android.com/apk/res/android"
     android:id="@+id/map"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     class="com.google.android.gms.maps.SupportMapFragment" />
```

- In manifest include API_KEY
- Obtained in Google developer console
- Free, identify your app, track its resource usage

```
<meta-data
     android:name="com.google.android.geo.API_KEY"
     android:value=«Alaksnasd3ksc893nfa" />
```



References

- CS 528 Mobile and Ubiquitous Computing, WPI
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, University of Texas Austin, Spring 2014
- https://developers.google.com/awareness/overview
- «Activity Recognition using Cell Phone Accelerometers», Jennifer R. Kwapisz, Gary M. Weiss, Samuel A. Moore
- https://developer.android.com/training/location
- Mobile Computing CSE 40814/60814, U. of Notre Dame
- Ubiquitous Computing Fundmantals, Krumm ed, Ch. 8
- CS 528 Mobile and Ubiquitous Computing, WPI