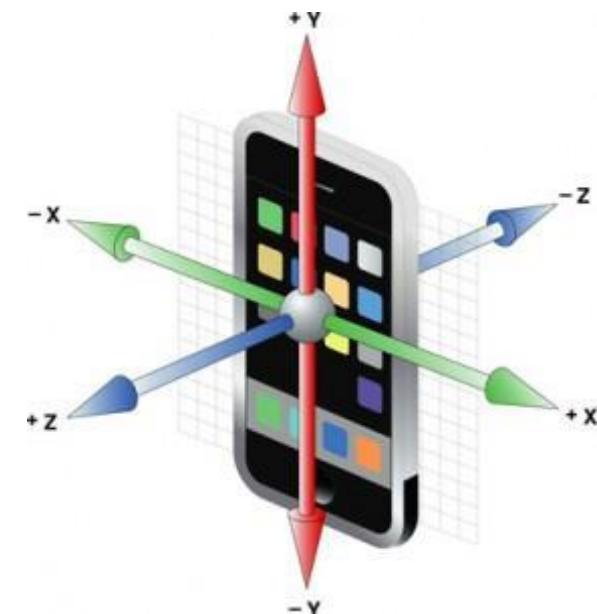
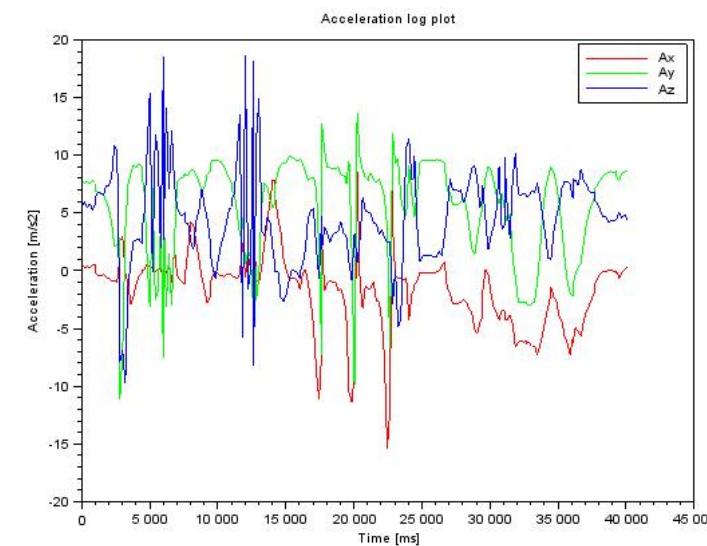
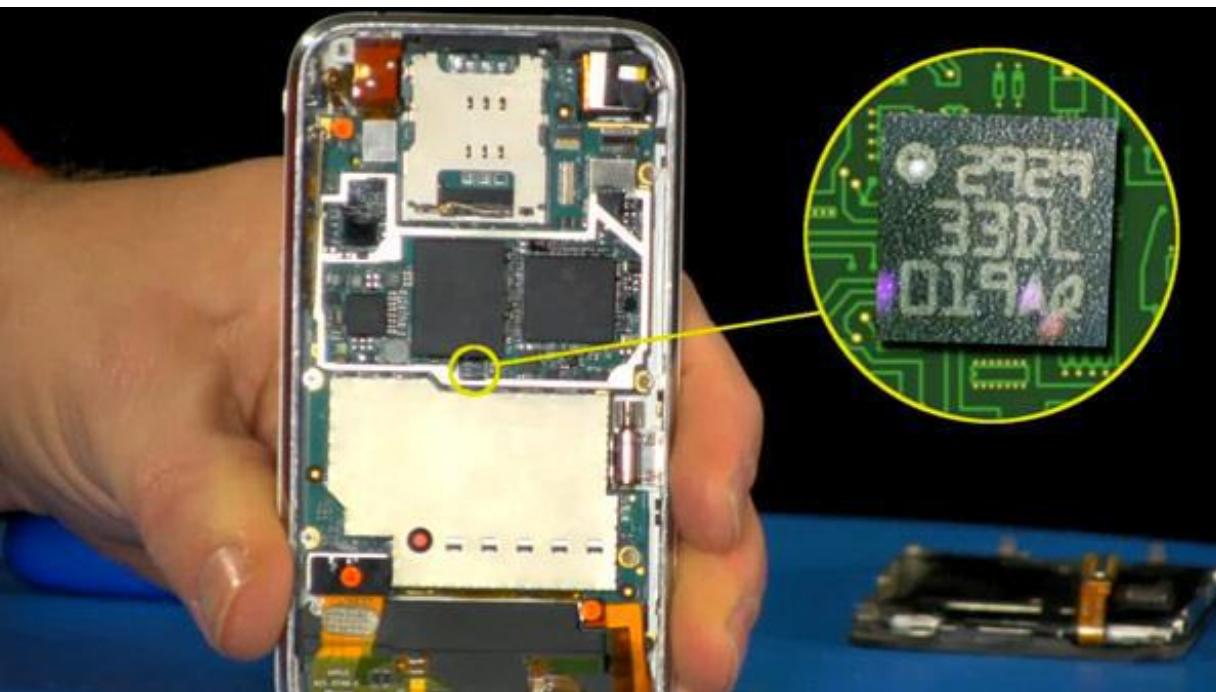


CSE 162 Mobile Computing Mobile Sensing

Hua Huang

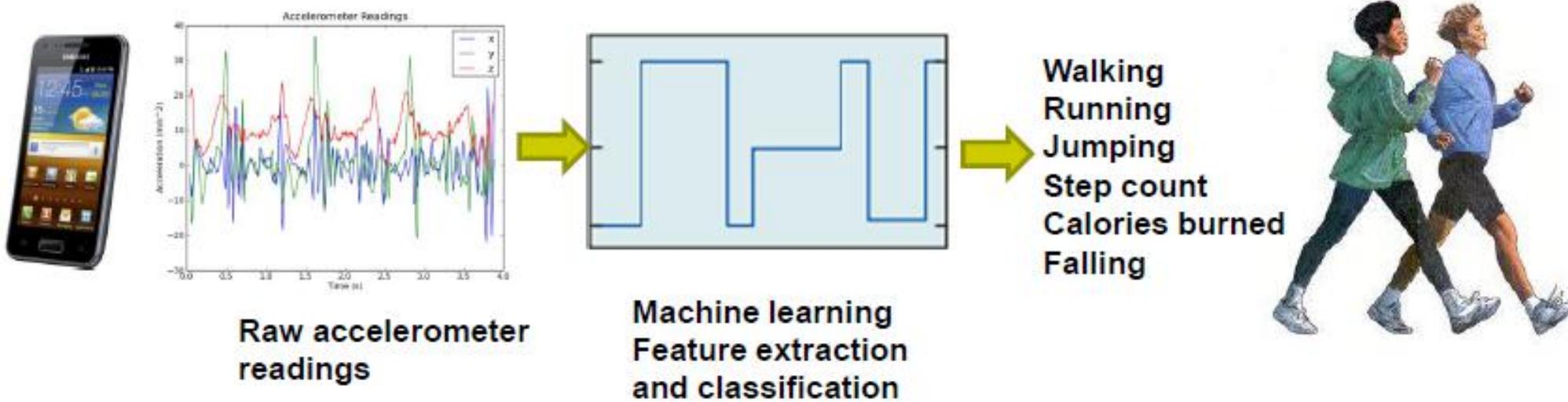
What is a Sensor?

- Converts physical quantity (e.g. light, acceleration, magnetic field) into a signal
- Example: accelerometer converts acceleration along X,Y,Z axes into signal



So What?

- Raw sensor data can be processed into useful info
- **Example:** Raw accelerometer data can be processed/classified to infer user's activity (e.g. walking running, etc)
- Voice samples can be processed/classified to infer whether speaker is nervous or not



Why are we talking about sensors?

- Sensors have been used in cellphones since they were invented ...
 - Microphone, number keys
- What about smartphones?
 - accelerometers, gyroscopes, GPS, cameras, etc ...
- Allowed cellphones explode into different markets
 - R.I.P: Garmin, Tomtom, Kodak, and more
- Instead of carrying around 10 separate devices, now you just need 1

Sensor Applications

Give some examples of sensor use

- Cars
- Computers
- Retail, logistics:
- Buildings
- Environment monitoring
- Industrial sensing & diagnostics

Definitions

- A **sensor** is a device that converts physical quality into an electrical signal.
- It is the interface between the physical world and electrical systems.
- Sensors are required to produce data that the computing system can process.
 - E.g., opening a washing machine stops the washing cycle.
 - Opening of a house door results in activation of a house alarm.

Definitions

- A **transducer** is the device that takes one form of input (energy or signal) and changes into another form.
- A transducer can be part of our earlier defined sensors.
 - Many times the terms sensor and transducer are used interchangeably
 - Sensors measure the change in physical environment and produce electrical signals using a transducer,
 - The transducer takes the measured change in the physical environment and transforms it into a different form of energy (such as an electrical signal)

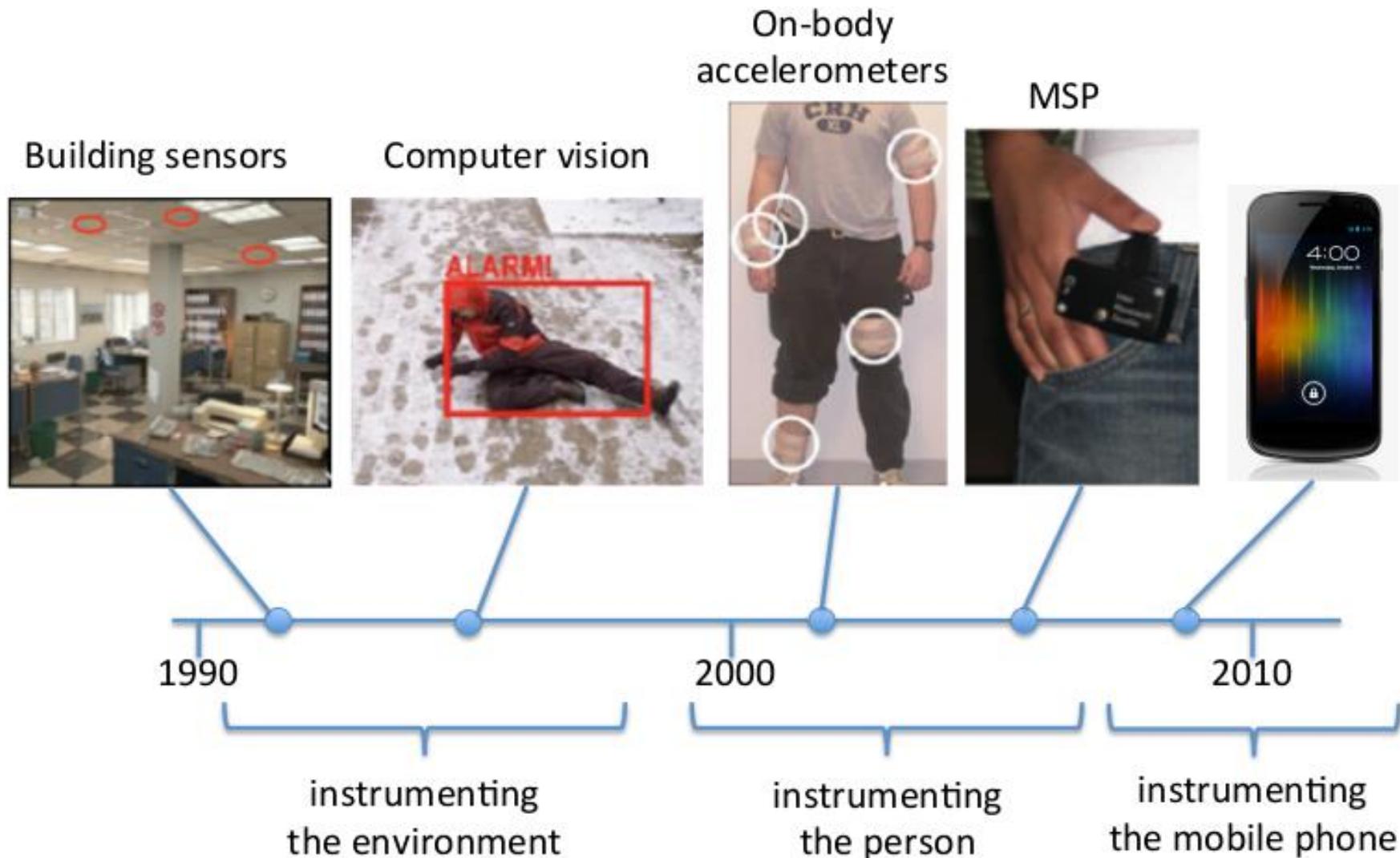
Definitions

- An **actuator** is a transducer that takes one form of energy as input and produces some form of motion, movement, or action.
 - For example, an electrical motor in an elevator converts electrical energy into the vertical movement of going from one floor to another floor of the building.

Common Sensors

- What are some sensors we use every day?
 - Thermometers
 - Radar guns
 - Automatic door openers

History of Sensing Platforms



Android Sensors

- Microphone (sound)
- Camera
- Temperature
- Location (GPS, A-GPS)
- Accelerometer
- Gyroscope (orientation)
- Proximity
- Pressure
- Light



Hardware and software sensors

- Sensors can be hardware- or software-based.
- Hardware-based sensors
 - physical components built into a handset or tablet device.
 - Examples: light sensor, proximity sensor, magnetometer, accelerometer
- Software-based sensors are not physical components, although they mimic hardware-based sensors.
 - Derive their data from one or more of the hardware-based sensors and manipulate it
 - Are sometimes called *virtual sensors* or *composite sensors*
 - Examples: linear acceleration, **orientation**.

Android Sensor Programming

Android Sensor Framework

- Enables apps to:
 - Access sensors available on device and
 - Acquire raw sensor data
- With the Android sensor framework you can:
 - Determine **which sensors are available** on a device.
 - Determine an individual **sensor's capabilities**, such as its maximum range, manufacturer, power requirements, and resolution.
 - **Acquire raw sensor data** and define the minimum rate at which you acquire sensor data.
 - **Register and unregister sensor event listeners** that monitor sensor changes.

Contd.

The following classes are the key parts of the Android sensor framework:

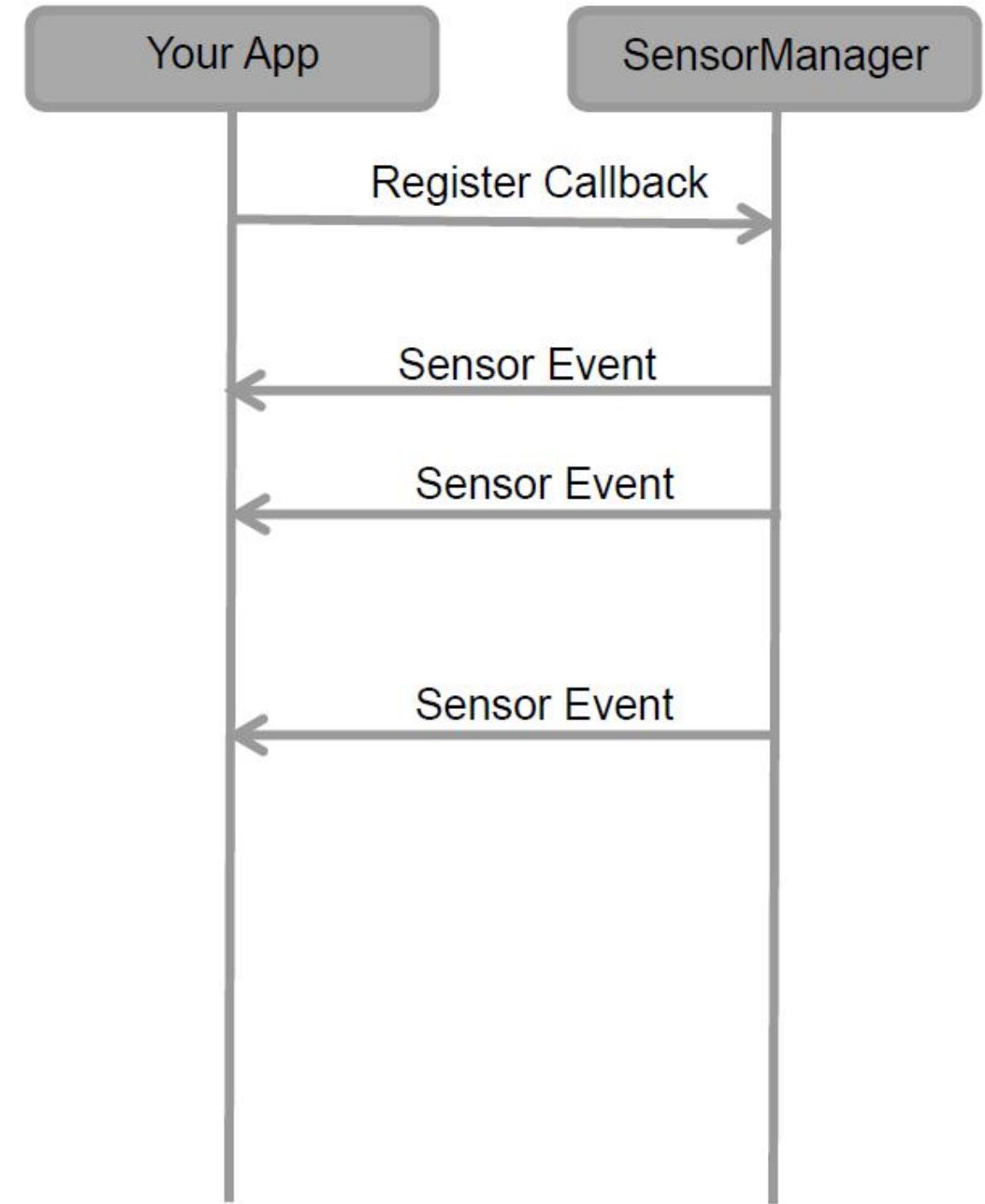
- **SensorManager**
 - Channel between your classes and sensors
- **Sensor**
 - Abstract representation of sensors on device
- **SensorEvent**
 - Represents information about a sensor event
- **SensorEventListener**
 - Register with SensorManager to listen for events from a sensor

Using Sensors

- Obtain the SensorManager and Sensor object
- Create a SensorEventListener for Sensor Events
 - Logic that responds to Sensor Event
 - Varying amounts of data from sensor depending on the type of sensor
- Register the sensor listener with a Sensor using SensorManager
- Unregister when done
 - A good thing to be done in the onPause or onStop method
- Override callback methods

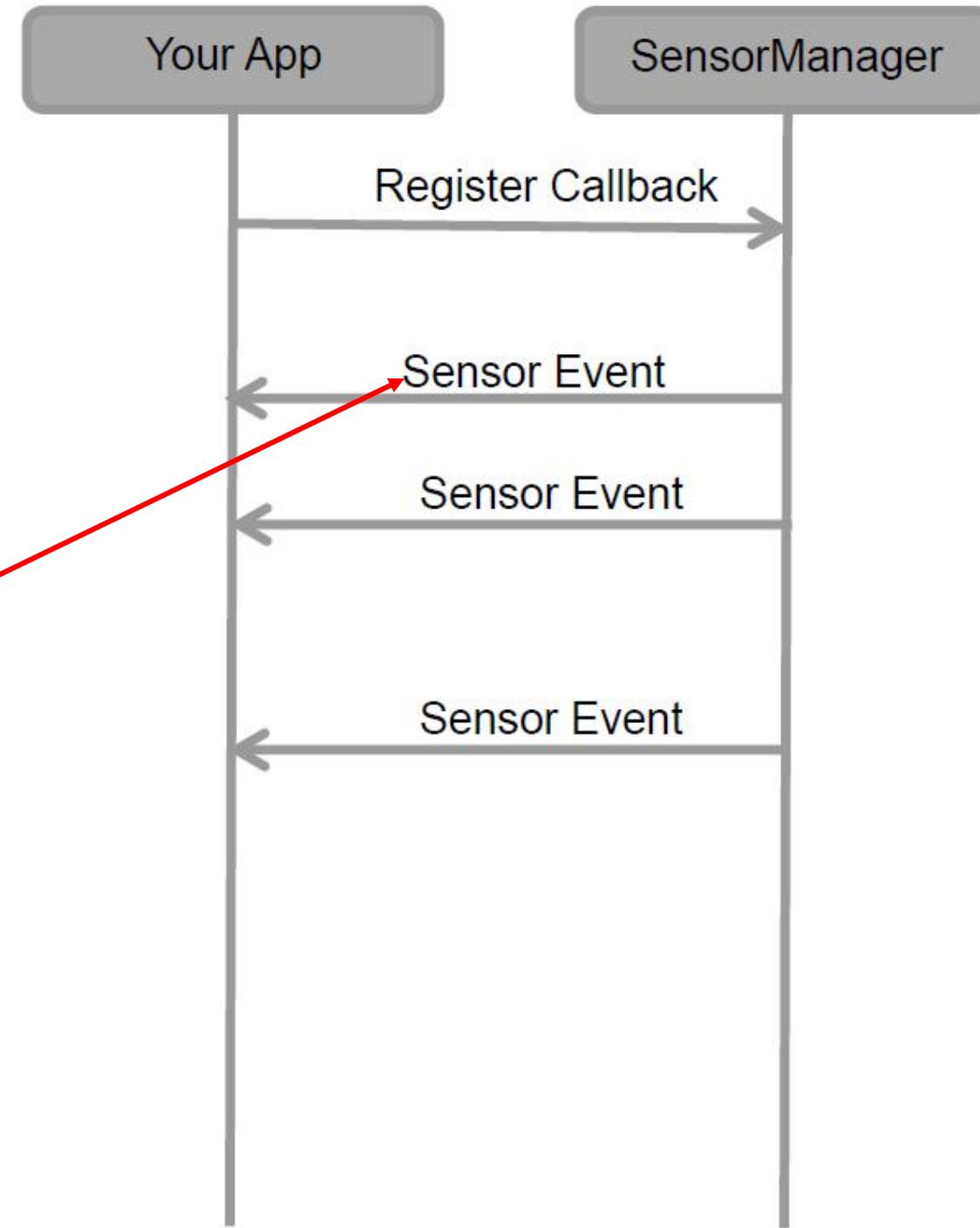
Sensor Events and Callbacks

- Sensors send events to sensor manager asynchronously, when new data arrives
- General approach:
 - App registers callbacks
 - **SensorManager** notifies app of sensor event whenever new data arrives (or accuracy changes)



Sensor Class

- A class that can be used to create instance of a specific sensor
- Has methods used to determine a sensor's capabilities
- Included in sensor event object



Sensor Availability

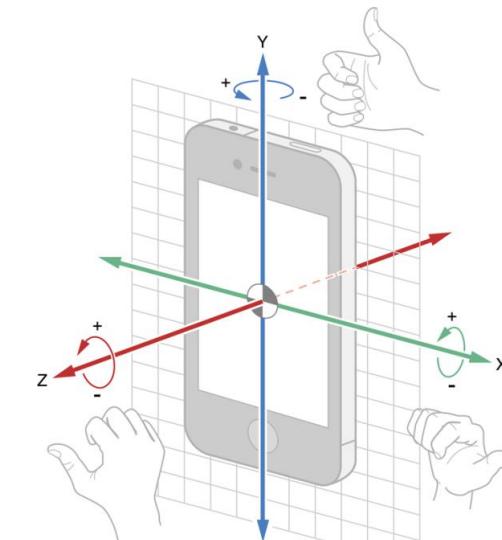
- Sensor availability varies from device to device, it can also vary between Android versions
 - Most devices have accelerometer and magnetometer
 - Some devices have barometers or thermometers
 - Device can have more than one sensor of a given type
 - Availability varies between Android versions

Show all sensors available in a device

```
public class MainActivity extends AppCompatActivity {
    private SensorManager mSensorManager;
    TextView tv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tv = findViewById(R.id.tv);
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
        for(int i=0; i<deviceSensors.size(); i++){
            tv.append("\n"+deviceSensors.get(i).getName()+"\n"+deviceSensors.get(i).getVendor());
        }
    }
}
```

Sensor Types Supported by Android

- **TYPE_PROXIMITY**
 - Measures an **object's proximity to device's screen**
 - **Common uses:** determine if handset is held to ear
- **TYPE_GYROSCOPE**
 - Measures device's **rate of rotation** around X,Y,Z axes in rad/s
 - **Common uses:** rotation detection (spin, turn, etc)



Available Sensors in Most Android Devices

Sensor	Used for
TYPE_ACCELEROMETER	Motion detection (shake, tilt, and so on).
TYPE_AMBIENT_TEMPERATURE	Monitoring air temperature.
TYPE_GRAVITY	Motion detection (shake, tilt, and so on).
TYPE_GYROSCOPE	Rotation detection (spin, turn, and so on).
TYPE_LIGHT	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Monitoring acceleration along a single axis.
TYPE_MAGNETIC_FIELD	Creating a compass.
TYPE_PRESSURE	Monitoring air pressure changes.
TYPE_PROXIMITY	Phone position during a call.
TYPE_RELATIVE_HUMIDITY	Monitoring ambient humidity, and dew point.
TYPE_TEMPERATURE	Monitoring temperatures.

Sensors

- TYPE_ACCELEROMETER
 - Hardware
 - Acceleration force in m/s^2
 - x,y,z axis
 - Includes gravity
- TYPE_AMBIENT_TEMPERATURE
 - Hardware
 - Room temperature in degree Celsius
- TYPE_GRAVITY
 - Software
 - Just gravity
 - If phone at rest same as TYPE_ACCELEROMETER

Sensors

- TYPE GYROSCOPE
 - Hardware
 - Measures device rate of rotation in radians/seconds around 3 axis
- TYPE LIGHT
 - Hardware
 - Light level in lx
 - Lux is SI measures illuminance in luminous flux per unit area
- TYPE LINEAR ACCELERATION
 - Software
 - Measures acceleration force applied to device in 3 axes excluding the force of gravity

Sensors

- TYPE MAGNETIC FIELD
 - Hardware
 - Ambient geomagnetic field in all 3 axes
 - μT Micro Teslas
- TYPE PRESSURE
 - Hardware
 - Ambient air pressure in hPa or mbar
 - Force per unit area

Sensors

- TYPE_PROXIMITY
 - Hardware
 - Proximity of an object in cm relative to the view screen of a device
 - Typically used to determine if handset is being held to person's ear during a call
- TYPE_RELATIVE_HUMIDITY
 - Hardware
 - Ambient humidity in percent (0 to 100)
- TYPE_TEMPERATURE
 - Hardware
 - Temperature of the device in degree Celcius

- **TYPE_STEP_DETECTOR**
 - Triggers sensor event each time user takes a step (**single step**)
 - Delivered event has value of 1.0 + timestamp of step
- **TYPE_STEP_COUNTER**
 - Also triggers a sensor event each time user takes a step
 - Delivers total ***accumulated number of steps since this sensor was first registered by an app,***
 - Tries to eliminate false positives
- **Common uses:** step counting, pedometer apps
- Requires hardware support, available in Nexus 5

Sensor Capabilities

Various methods in Sensor Class to get capabilities of Sensor

- Minimum Delay (in MicroSeconds) - `getMinDelay()`
- Power Consumption (in MicroAmpere) - `getPower()`
- Maximum Range - `getMaximumRange()`
- Resolution - `getResolution()`

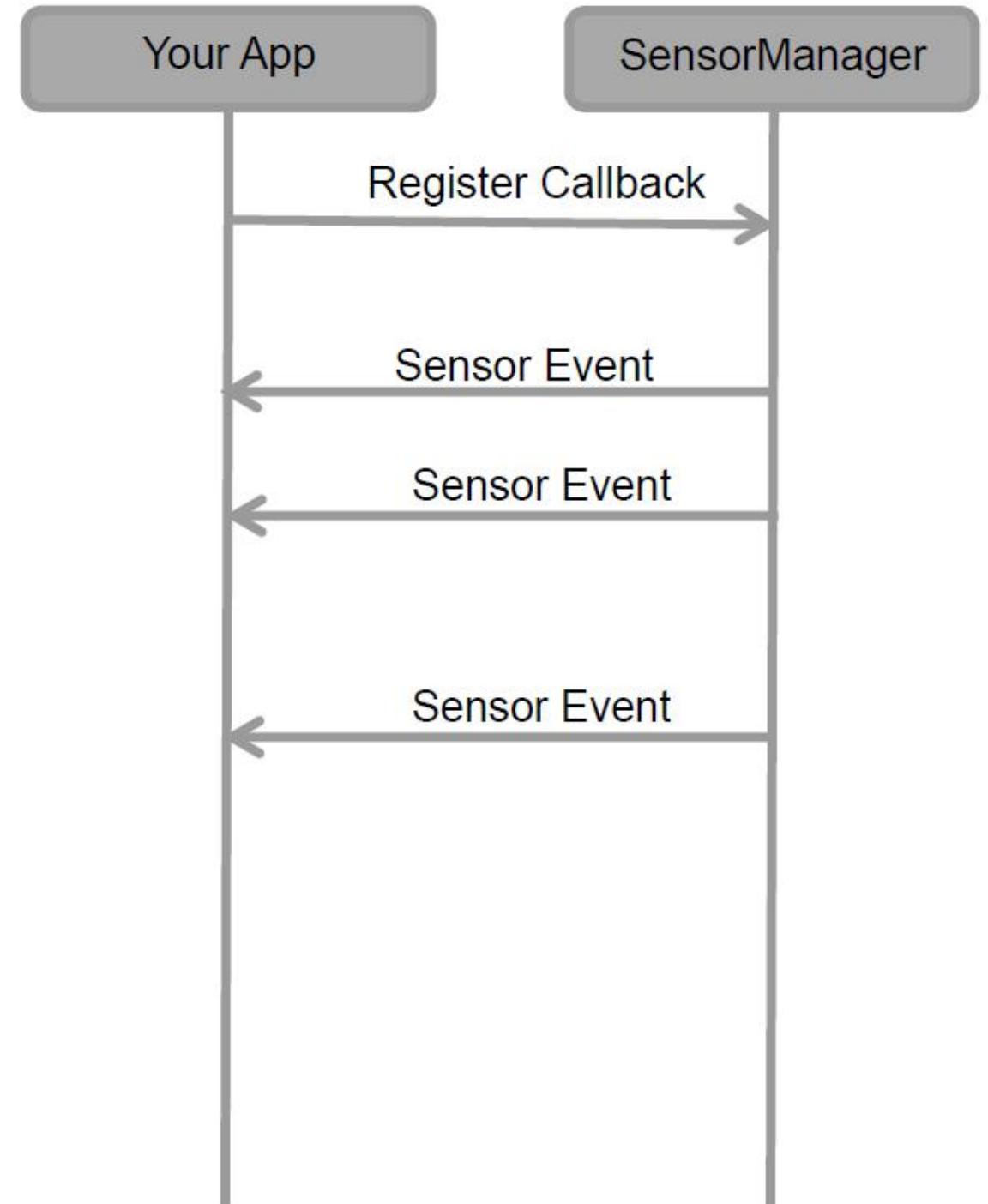
Managing Sensor Accuracy

Accuracy is represented by one of four status constants:

- **SENSOR_STATUS_ACCURACY_HIGH**
 - Indicates that this sensor is reporting data with maximum accuracy
- **SENSOR_STATUS_ACCURACY_LOW**
 - Indicates that this sensor is reporting data with low accuracy, calibration with the environment is needed
- **SENSOR_STATUS_ACCURACY_MEDIUM**
 - Indicates that this sensor is reporting data with an average level of accuracy, calibration with the environment may improve the readings
- **SENSOR_STATUS_UNRELIABLE**
 - Indicates that the values returned by this sensor cannot be trusted, calibration is needed or the environment doesn't allow readings

SensorEvent

- Android system sensor event information as a **sensor event object**
- **Sensor event object** includes:
 - **Sensor**: Type of sensor that generated the event
 - **Values**: Raw sensor data
 - **Accuracy**: Accuracy of the data
 - **Timestamp**: Event timestamp



Sensor Delay

- Data should begin to come in at the rate you specified as an argument
- The rate can be SENSOR_DELAY_NORMAL
- SENSOR_DELAY_UI (For basic UI interaction)
- SENSOR_DELAY_GAME (A high rate that many games require)
- SENSOR_DELAY_FASTEST (without any delay)
- You can also specify the delay as an absolute value (in microseconds).

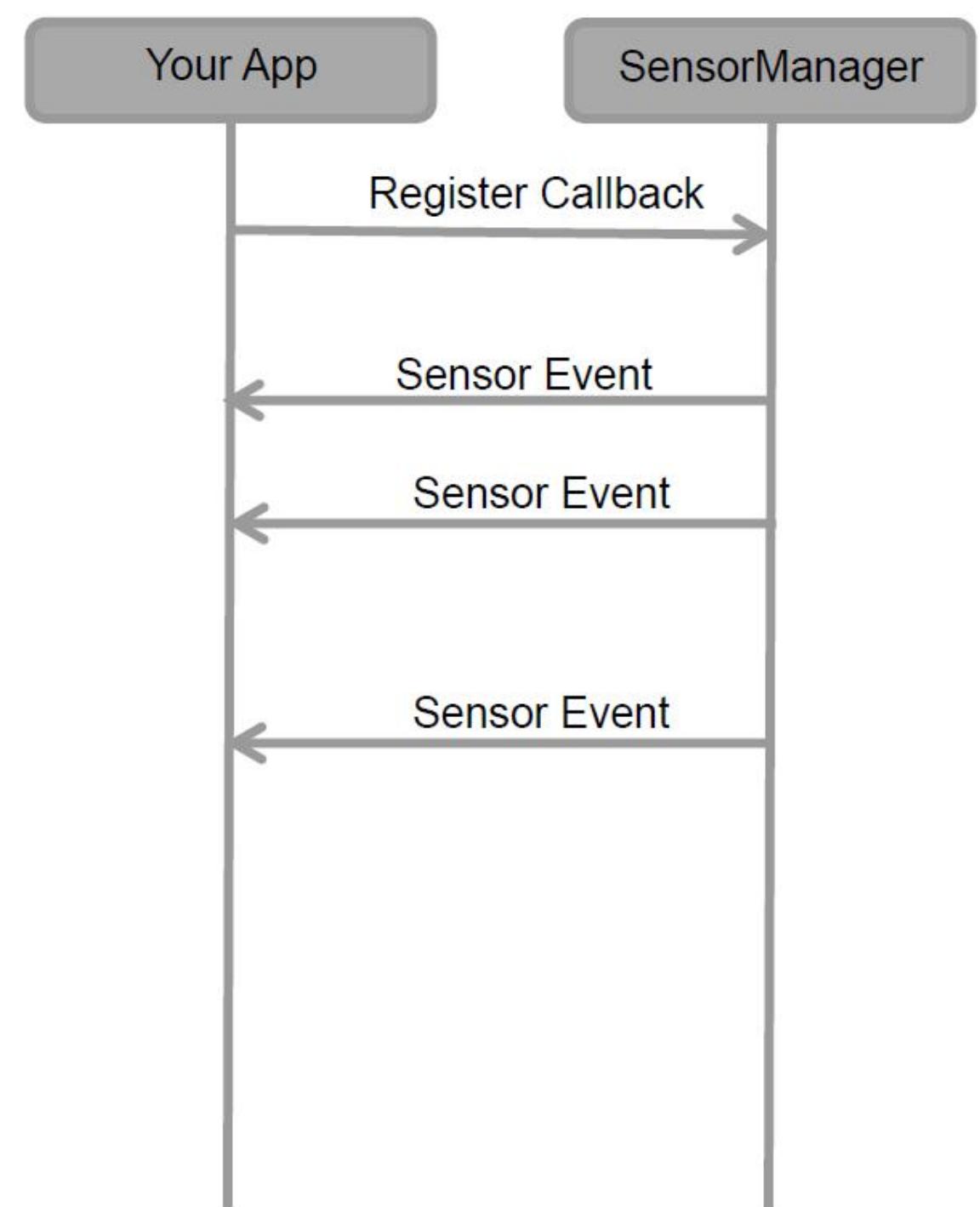
- Sensor Values Depend on Sensor Type

Sensor	Sensor event data	Description	Units of measure
TYPE_ACCELEROMETER	SensorEvent.values[0]	Acceleration force along the x axis (including gravity).	m/s ²
	SensorEvent.values[1]	Acceleration force along the y axis (including gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (including gravity).	
TYPE_GRAVITY	SensorEvent.values[0]	Force of gravity along the x axis.	m/s ²
	SensorEvent.values[1]	Force of gravity along the y axis.	
	SensorEvent.values[2]	Force of gravity along the z axis.	
TYPE_GYROSCOPE	SensorEvent.values[0]	Rate of rotation around the x axis.	rad/s
	SensorEvent.values[1]	Rate of rotation around the y axis.	
	SensorEvent.values[2]	Rate of rotation around the z axis.	
TYPE_GYROSCOPE_UNCALIBRATED	SensorEvent.values[0]	Rate of rotation (without drift compensation) around the x axis.	rad/s
	SensorEvent.values[1]	Rate of rotation (without drift compensation) around the y axis.	
	SensorEvent.values[2]	Rate of rotation (without drift compensation) around the z axis.	
	SensorEvent.values[3]	Estimated drift around the x axis.	
	SensorEvent.values[4]	Estimated drift around the y axis.	
	SensorEvent.values[5]	Estimated drift around the z axis.	

TYPE_LINEAR_ACCELERATION	SensorEvent.values[0]	Acceleration force along the x axis (excluding gravity).	m/s ²
	SensorEvent.values[1]	Acceleration force along the y axis (excluding gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (excluding gravity).	
TYPE_ROTATION_VECTOR	SensorEvent.values[0]	Rotation vector component along the x axis ($x * \sin(\theta/2)$).	Unitless
	SensorEvent.values[1]	Rotation vector component along the y axis ($y * \sin(\theta/2)$).	
	SensorEvent.values[2]	Rotation vector component along the z axis ($z * \sin(\theta/2)$).	
	SensorEvent.values[3]	Scalar component of the rotation vector ($(\cos(\theta/2))$). ¹	
TYPE_SIGNIFICANT_MOTION	N/A	N/A	N/A
TYPE_STEP_COUNTER	SensorEvent.values[0]	Number of steps taken by the user since the last reboot while the sensor was activated.	Steps
TYPE_STEP_DETECTOR	N/A	N/A	N/A

SensorEventListener

- Interface used to create 2 callbacks that receive notifications (sensor events) when:
 - Sensor values change (**onSensorChange()**) or
 - When sensor accuracy changes (**onAccuracyChanged()**)



Sensor API Tasks

- **Sensor API Task 1: Identifying sensors and their capabilities**
- Why identify sensor and their capabilities at runtime?
 - Disable app features using sensors not present, or
 - Choose sensor implementation with best performance
- **Sensor API Task 2: Monitor sensor events**
- Why monitor sensor events?
 - To acquire raw sensor data
 - Sensor event occurs every time sensor detects change in parameters it is measuring

Identifying Sensors and Sensor Capabilities

- First create instance of **SensorManager** by calling **getSystemService()** and passing in **SENSOR_SERVICE** argument

```
private SensorManager mSensorManager;  
...  
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

- Then list sensors available on device by calling **getSensorList()**

```
List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

- To list particular type, use **TYPE_GYROSCOPE**, **TYPE_GRAVITY**, etc

Checking if Phone has at least one of particular Sensor Type

- Device may have multiple sensors of a particular type.
 - E.g. multiple magnetometers
- If multiple sensors of a given type exist, one of them must be designated “the default sensor” of that type
- To determine if specific sensor type exists use **getDefaultSensor()**
- **Example:** To check whether device has at least one magnetometer

```
private SensorManager mSensorManager;  
...  
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){  
    // Success! There's a magnetometer.  
}  
else {  
    // Failure! No magnetometer.  
}
```

Example: Monitoring Light Sensor Data

- Goal: Monitor light sensor data using **onSensorChanged()**, display it in a **TextView** defined in main.xml

Create instance of Sensor manager

Get default Light sensor

Called by Android system when accuracy of sensor being monitored changes

```
public class SensorActivity extends Activity implements SensorEventListener {  
    private SensorManager mSensorManager;  
    private Sensor mLight;  
  
    @Override  
    public final void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
        mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);  
    }  
  
    @Override  
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {  
        // Do something here if sensor accuracy changes.  
    }  
}
```

Example: Monitoring Light Sensor Data (Contd)

```
@Override  
public final void onSensorChanged(SensorEvent event) {  
    // The light sensor returns a single value.  
    // Many sensors return 3 values, one for each axis.  
    float lux = event.values[0];  
    // Do something with this sensor value.  
}  
  
@Override  
protected void onResume() {  
    super.onResume();  
    mSensorManager.registerListener(this, mLIGHT, SensorManager.SENSOR_DELAY_NORMAL);  
}  
  
@Override  
protected void onPause() {  
    super.onPause();  
    mSensorManager.unregisterListener(this);  
}
```

Called by Android system to report new sensor value

Provides SensorEvent object containing new sensor data

Get new light sensor value

Register sensor when app becomes visible

Unregister sensor if app is no longer visible to reduce battery drain

Handling Different Sensor Configurations

- Different phones have different sensors built in
 - E.g. Motorola Xoom has pressure sensor, Samsung Nexus S doesn't
- If app uses a specific sensor, how to ensure this sensor exists on target device?
- Two options
- **Option 1:** Detect device sensors at runtime, enable/disable app features as appropriate
- **Option 2:** Use AndroidManifest.xml entries to ensure that only devices possessing required sensor can see app on Google Play
 - E.g. following manifest entry in AndroidManifest ensures that only devices with accelerometers will see this app on Google Play

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

Detecting Sensors at Runtime

```
private SensorManager mSensorManager;  
...  
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
if (mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE) != null){  
    // Success! There's a pressure sensor.  
}  
else {  
    // Failure! No pressure sensor.  
}
```

Using Sensors

- Recall basics for using a Sensor:
 - Obtain the *SensorManager* object
 - create a *SensorEventListener* for *SensorEvents*
 - logic that responds to sensor event
 - Register the sensor listener with a *Sensor* via the *SensorManager*

Sensor Best Practices

- Unregister sensor listeners
 - when done with Sensor or activity using sensor paused (onPause method)
 - `sensorManager.unregisterListener(sensorListener)`
 - otherwise data still sent and battery resources continue to be used

Sensors Best Practices

- verify sensor available before using it
- use getSensorList method and type
- ensure list is not empty before trying to register a listener with a sensor

Sensors Best Practices

- Avoid deprecated sensors and methods
- **TYPE_ORIENTATION** and **TYPE_TEMPERATURE** are deprecated as of Ice Cream Sandwich / Android 4.0

Sensors Best Practices

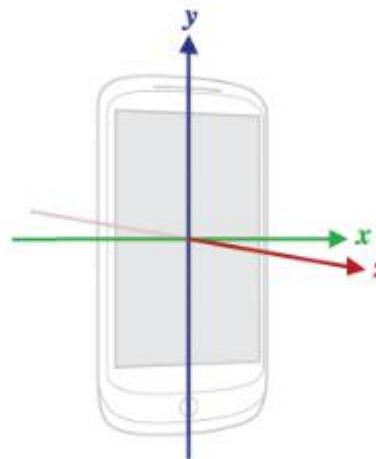
- Don't block the `onSensorChanged()` method
 - 50 updates a second for `onSensorChange` method not uncommon
 - if necessary save event and do work in another thread or asynch task

Sensor Best Practices

- Testing on the emulator

Sensor Coordinate System

- In general, the sensor framework uses a standard 3-axis coordinate system to express data values.
- For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation
 - $+x$ to the right
 - $+y$ up
 - $+z$ out of the front face



CSE 162 Mobile Computing

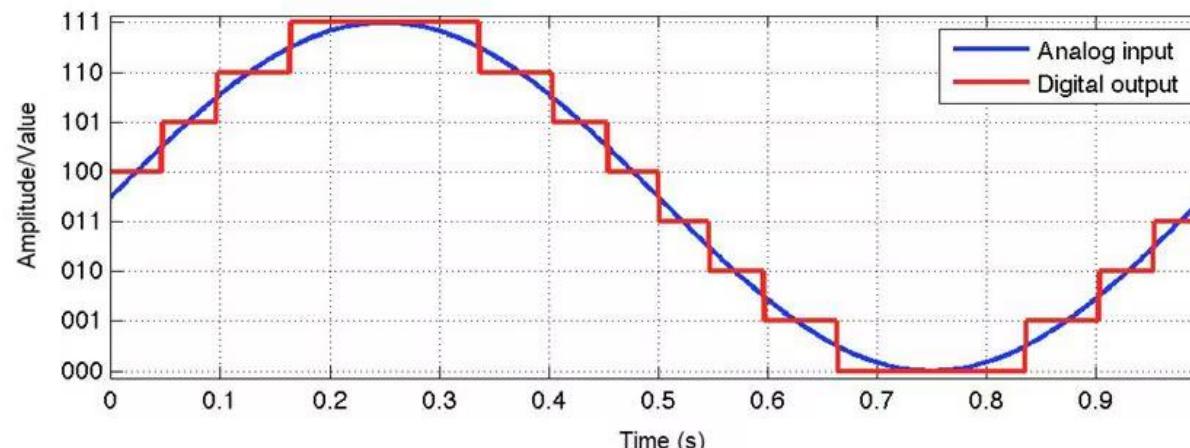
Mobile Sensors

Hua Huang

Analog to Digital Converter

How does an ADC work?

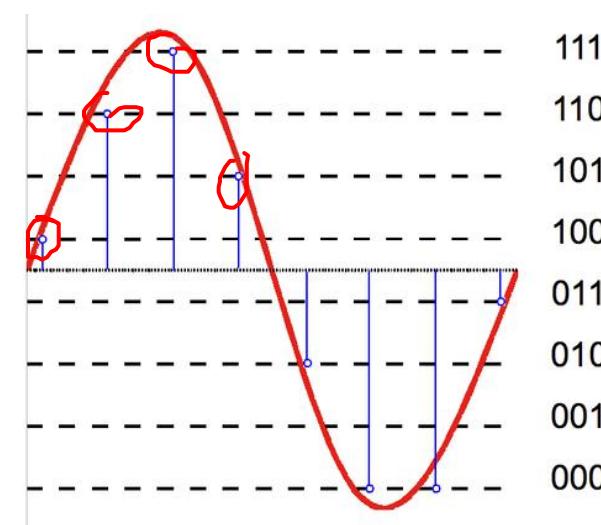
- Analog signals are signals that have a continuous sequence with continuous values
- Digital signals are represented by a sequence of discrete values broken down into sequences
- Computers can't read values unless it's digital data.
 - They can only see “levels” of the voltage



- An analog-to-digital converter (ADC) can be modeled as two processes: sampling and quantization.
 - Sampling converts a time-varying voltage signal into a discrete-time signal, a sequence of real numbers.
 - Quantization replaces each real number with an approximation from a finite set of discrete values.

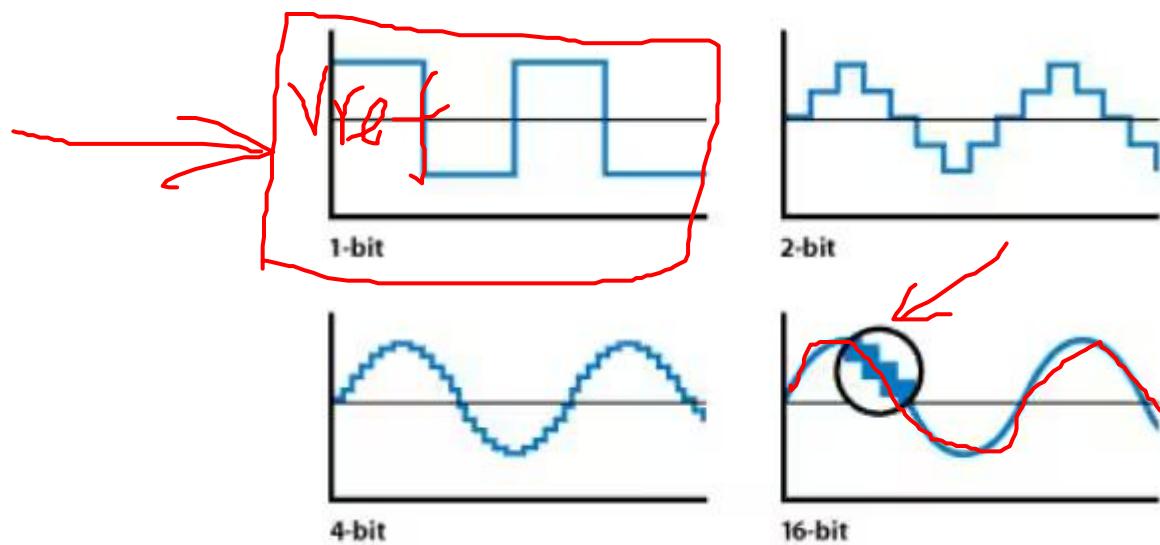


11
10
01
00



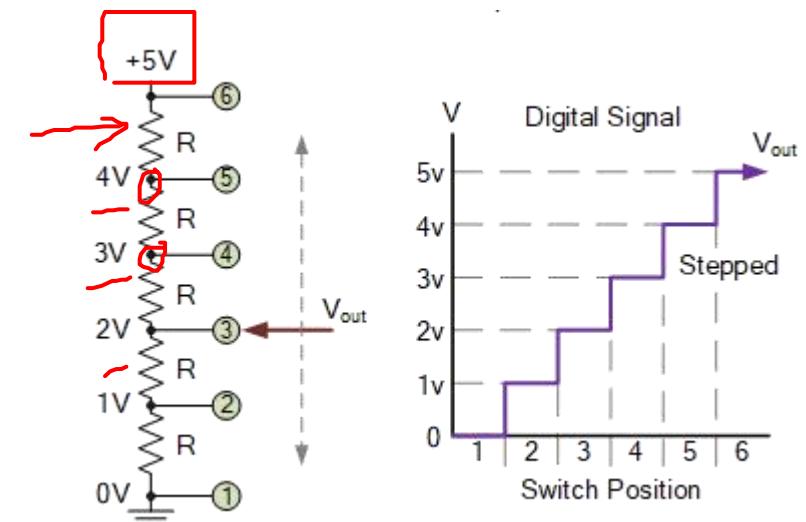
Resolution of ADC

- The resolution of the ADC can be determined by its bit length.
 - 1-bit only has two “levels”.
 - As the bit length increases, the levels increase making the signal more closely represent the original analog signal.



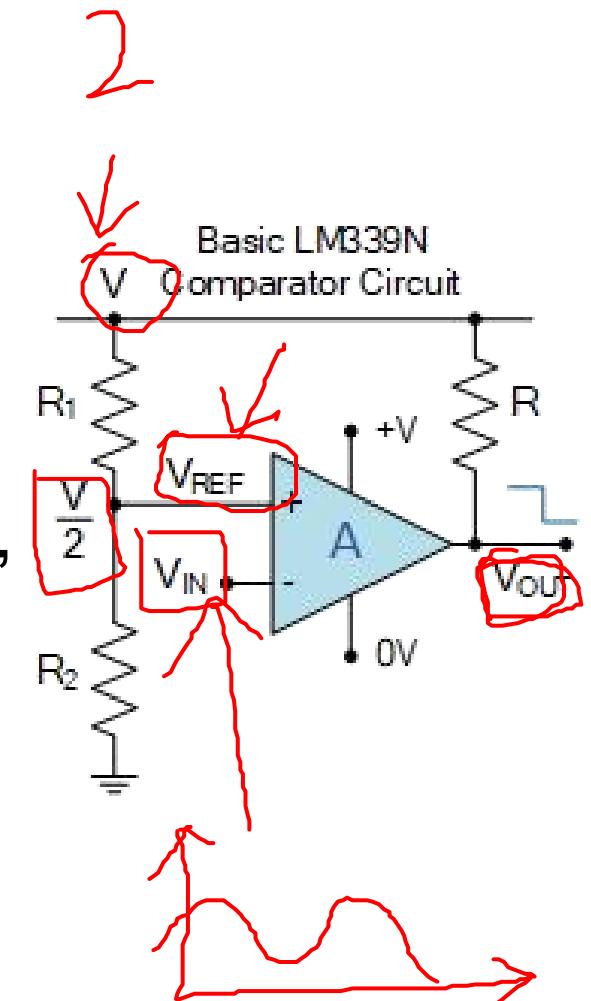
ADC Building blocks

- How to generate the Vref?
- Series resistor chain, forms a basic potential divider network.
 - When the switch is rotated from one position (or node) to the next the output voltage, Vref changes quickly in discrete and distinctive voltage steps representing multiples of 1.0 volts on each switching action as shown.

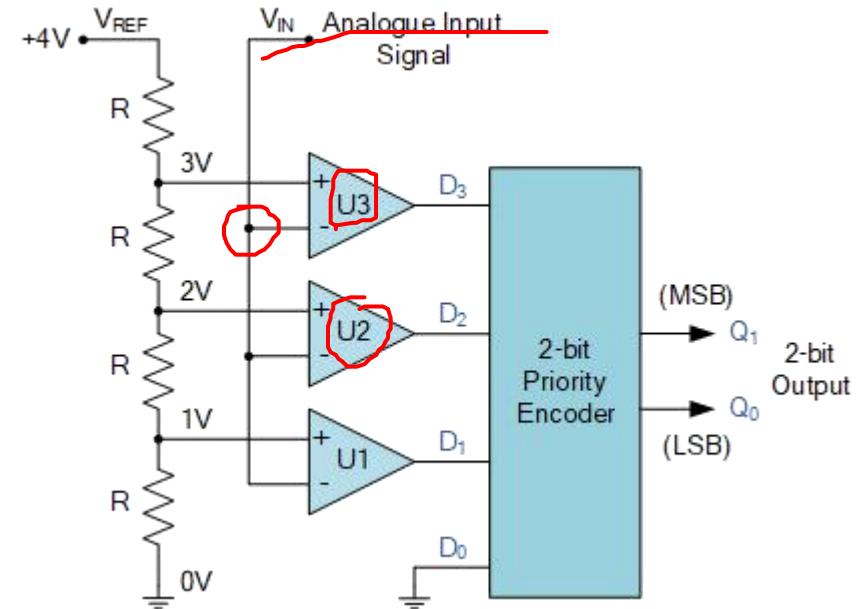


ADC Building blocks

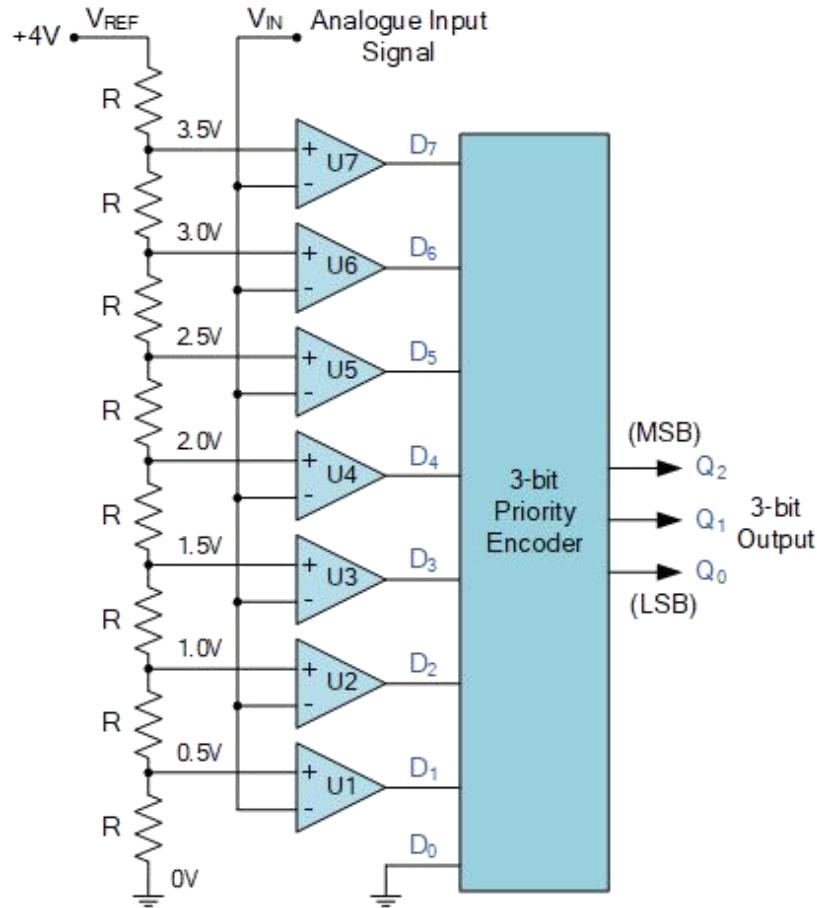
- An analogue comparator has two analogue inputs, one positive and one negative
 - Can compare the magnitudes of two different voltage levels.
- A voltage input, (V_{IN}) signal is applied to one input of the comparator, while a reference voltage, (V_{REF}) to the other.
- A comparison of the two voltage levels at the comparator's input is made to determine the comparators digital logic output state, either a “1” or a “0”.



- By adding more resistors to the voltage divider network we can effectively “divide” the supply voltage by an amount determined by the resistances of the resistors.
- In general, $2^n - 1$ comparators would be required for conversion of an “n”-bit binary output, where “n” is typically in the range from 8 to 16.
- If we now create a 2-bit ADC, then we will need $2^2 - 1$ which is “3” comparators as we need four different voltage levels corresponding to the 4 digital values required for a 4-to-2 bit encoder circuit as shown.



Analogue Input Voltage (V_{IN})	Comparator Outputs				Digital Outputs	
	D ₃	D ₂	D ₁	D ₀	Q ₁	Q ₀
0 to 1 V	0	0	0	0	0	0
1 to 2 V	0	0	1	X	0	1
2 to 3 V	0	1	X	X	1	0
3 to 4 V	1	X	X	X	1	1



Analogue Input Voltage (V_{IN})	Comparator Outputs								Digital Outputs		
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0 to 0.5 V	0	0	0	0	0	0	0	0	0	0	0
0.5 to 1.0 V	0	0	0	0	0	0	1	X	0	0	1
1.0 to 1.5 V	0	0	0	0	0	1	X	X	0	1	0
1.5 to 2.0 V	0	0	0	0	1	X	X	X	0	1	1
2.0 to 2.5 V	0	0	0	1	X	X	X	X	1	0	0
2.5 to 3.0 V	0	0	1	X	X	X	X	X	1	0	1
3.0 to 3.5 V	0	1	X	X	X	X	X	X	1	1	0
3.5 to 4.0 V	1	X	X	X	X	X	X	X	1	1	1

Where again "X" is a "don't care", that is either a logic "0" or a logic "1" input condition.

Micro-electromechanical systems (MEMS)

MEMS

- Accelerometer+Gyroscope= 6 DoF Inertia Motion Units(IMU)
- Very small and low-cost
 - less than 6 mm × 6 mm in footprint can weigh less than a gram
 - MEMS vendors have been shipping high volumes for mobiles, tablets, and automotive applications since 1990.
- Robust to environmental changes
 - The shock specifications of today's generation of devices are stated to 10,000 g, but in reality can tolerate much higher
 - Normal humans can withstand no more than 9 g's



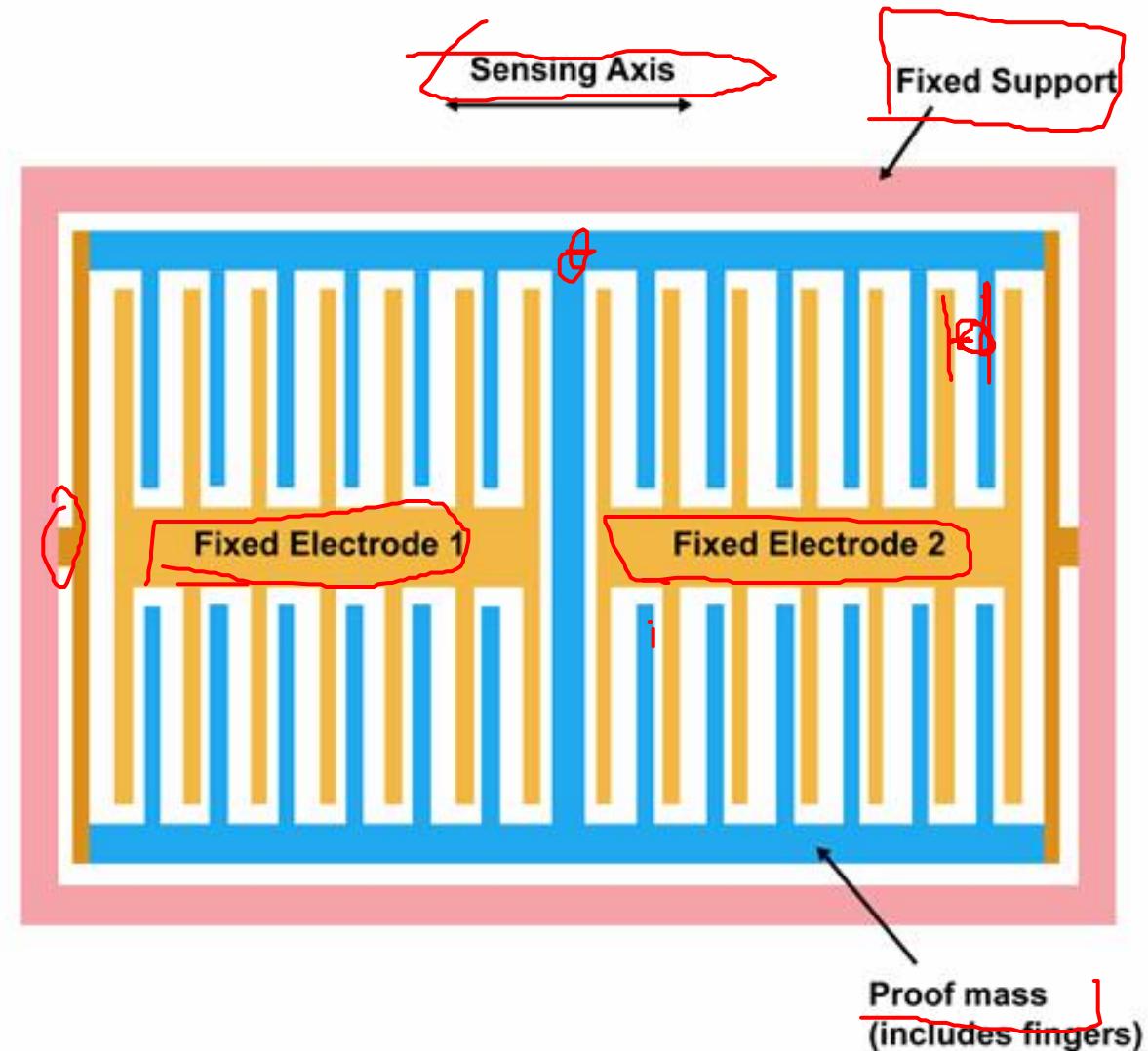
MEMS

- Where do you see these?
 - Wii Nunchuks
 - Orientation sensing in smartphones
 - Image Stabilization in cameras
 - Collision detection in cars
 - Pedometers
 - Monitoring equipment for failure (vibrations in ball bearings, etc)



MEMS: Accelerometer

- Accelerometers:
 - Measures change in velocity in x y z axes
- How does it work?
 - The “proof mass” shown above is allowed to move in a plane.
 - The attached fingers form a capacitor with the two plates around it.
 - The rate of change of the capacitance is measured and translated into an acceleration



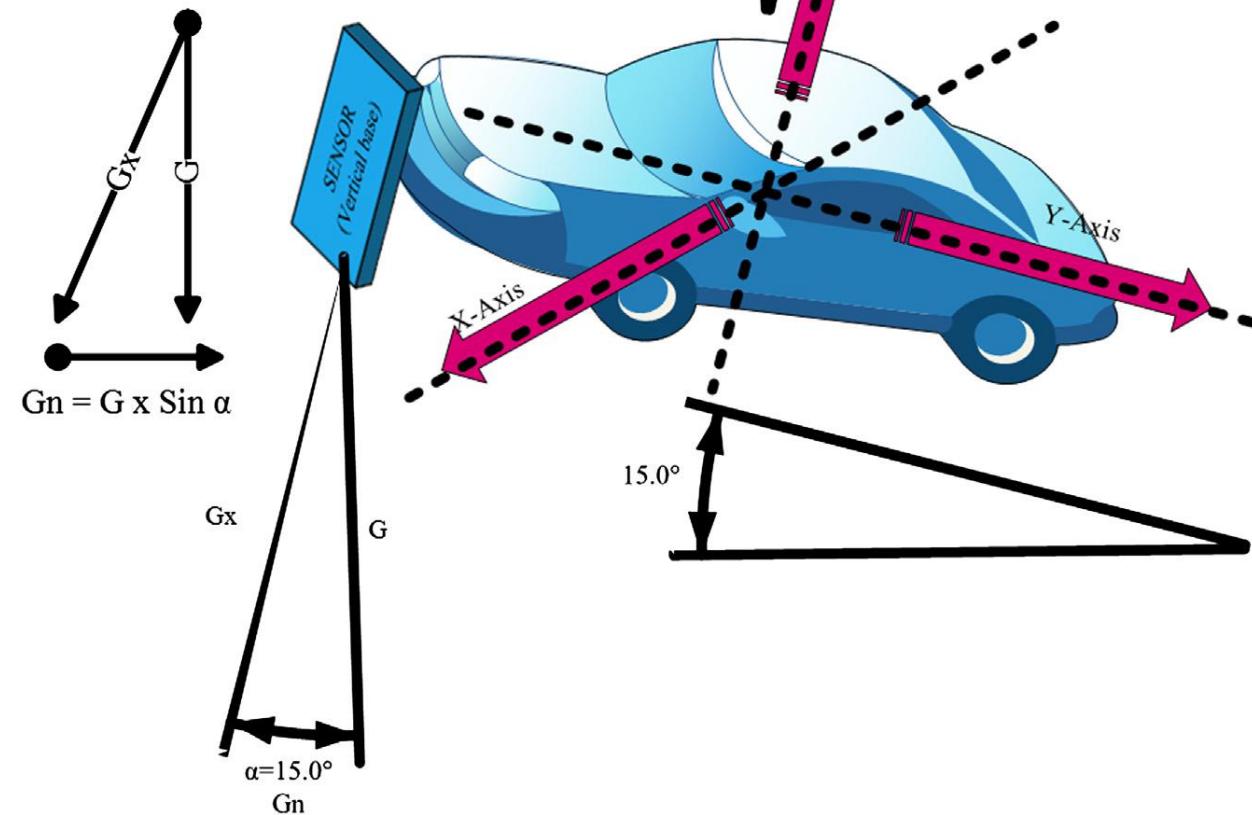
- With the device flat on a surface what, roughly, will be the magnitude of the largest acceleration?
 - 0 m/s²
 - 1 m/s²
 - 5 m/s²
 - 10 m/s²
 - 32 m/s²

Influence of Gravity

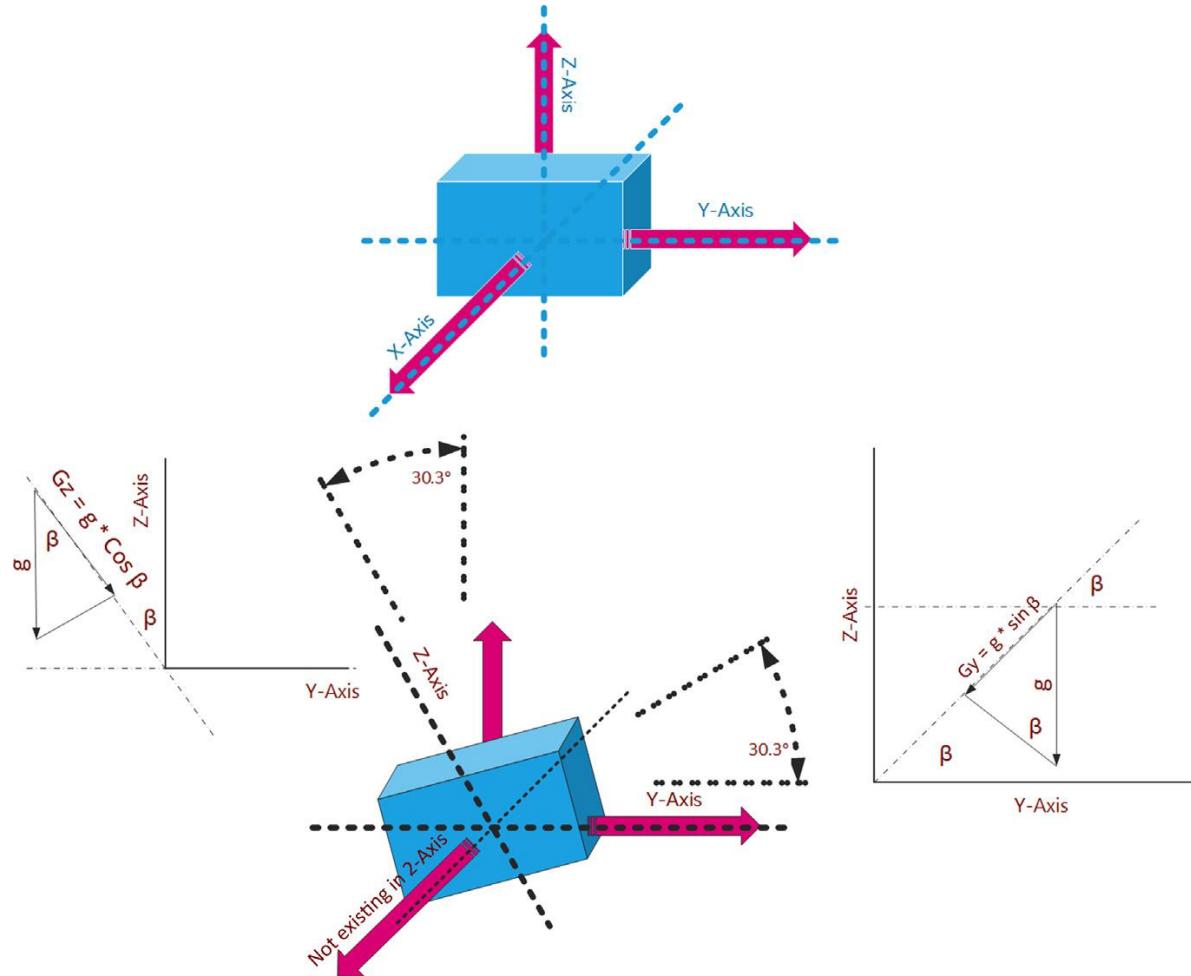
- An accelerometer at rest will not have a zero measurement. Instead, it measures $9.8m/s^2$
- A free-falling sensor will measure 0

Influence of Tilt Angle

- $a_z = g * \cos(\alpha)$
- $a_y = g * \sin(\alpha)$

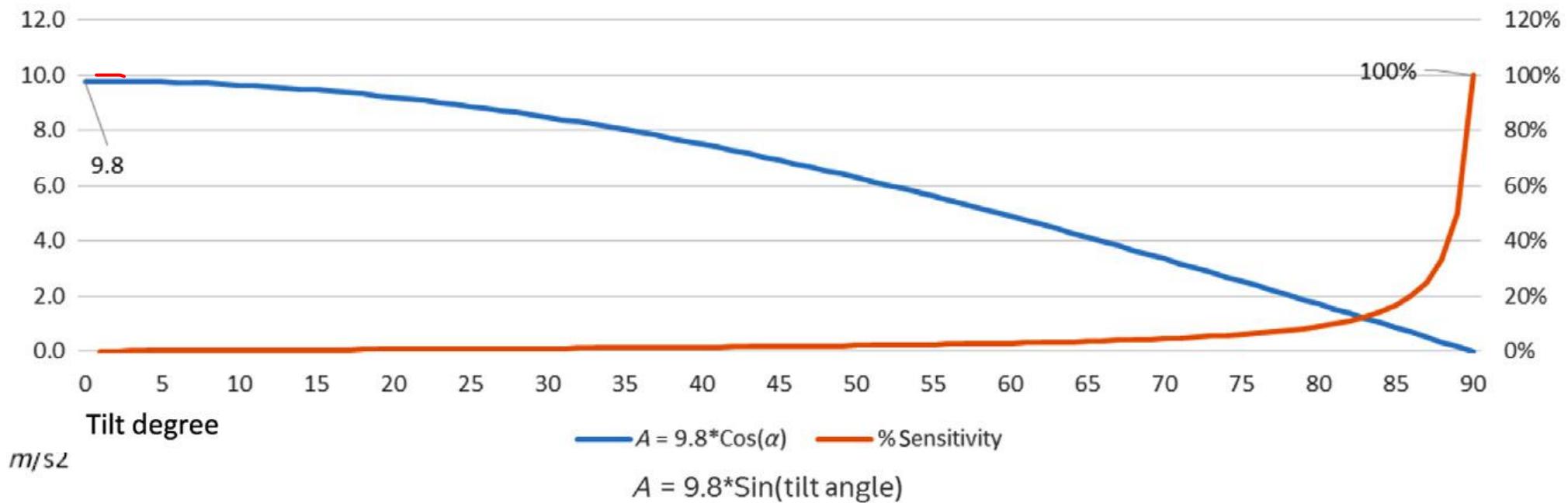


Sensitivity to Tilt Angle



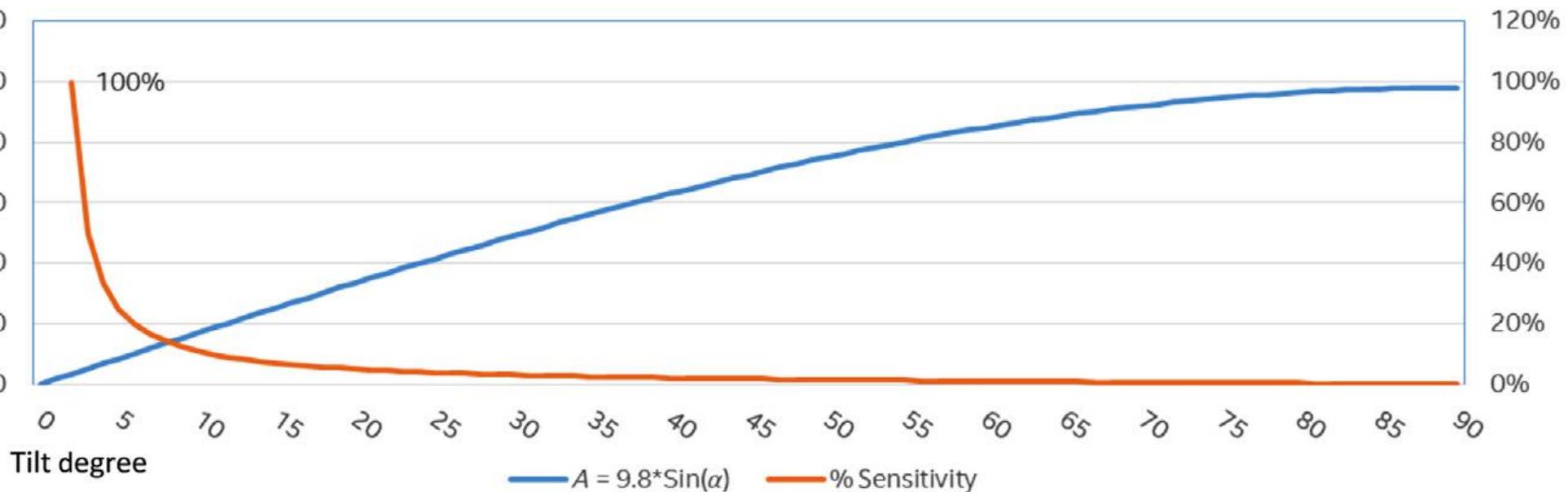
m/s²

$$A = 9.8 \cdot \cos(\text{tilt angle})$$



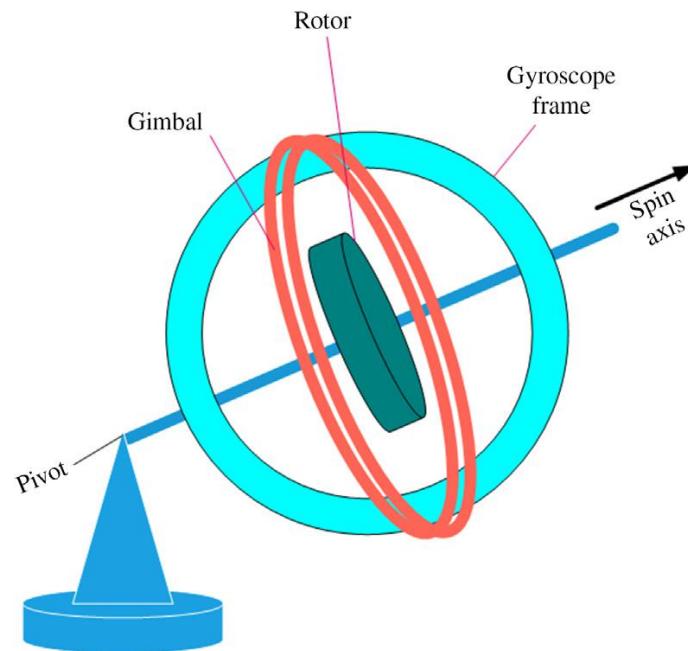
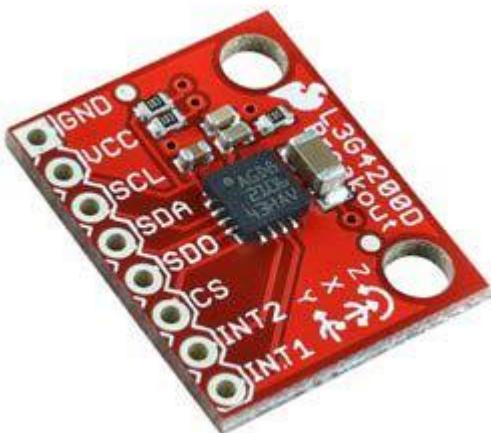
m/s²

$$A = 9.8 \cdot \sin(\text{tilt angle})$$

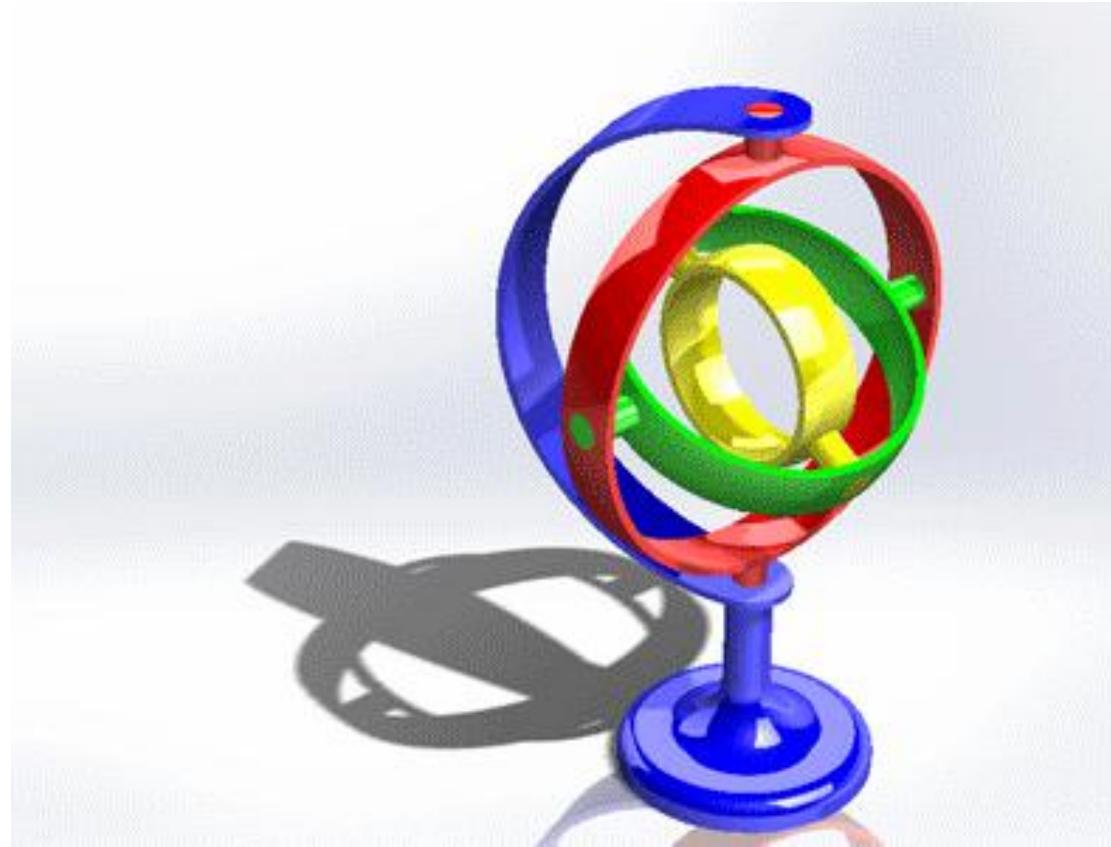


Gyroscope

- Measures rotation speed

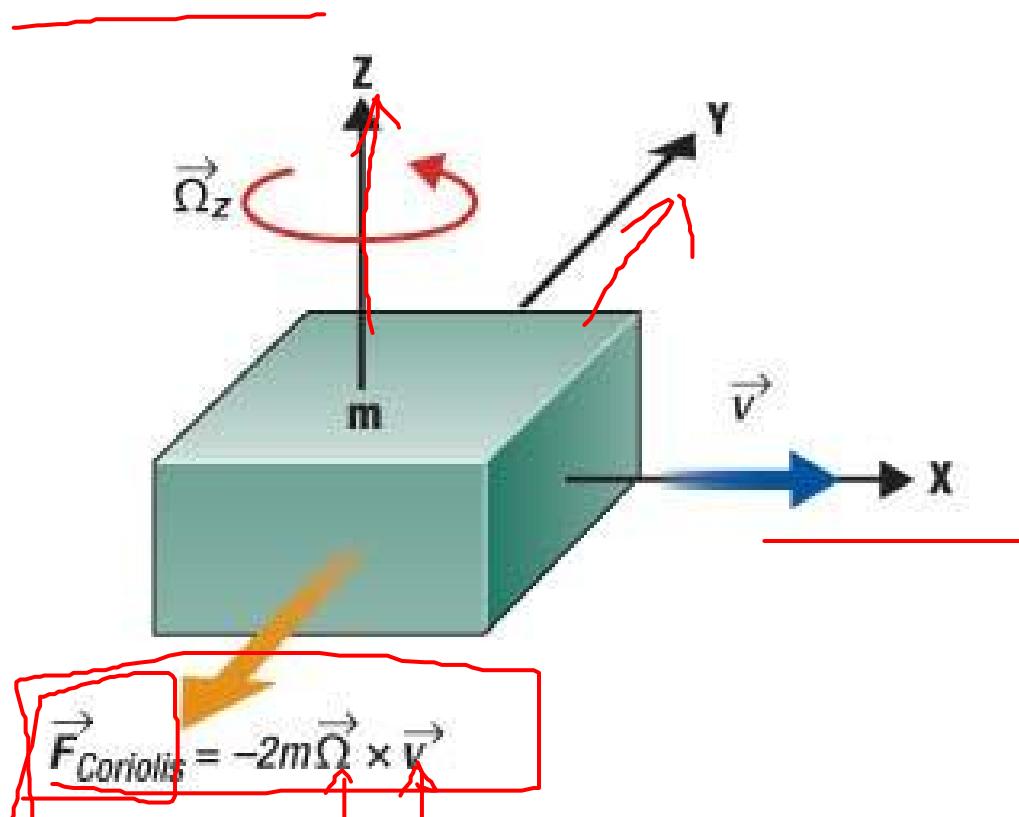


Mechanical Gyroscope



How MEMS Gyroscope works?

- The Coriolis force
 - If an object is moving along one axis, and it is rotated above another, it will feel a Coriolis force in the third axial direction

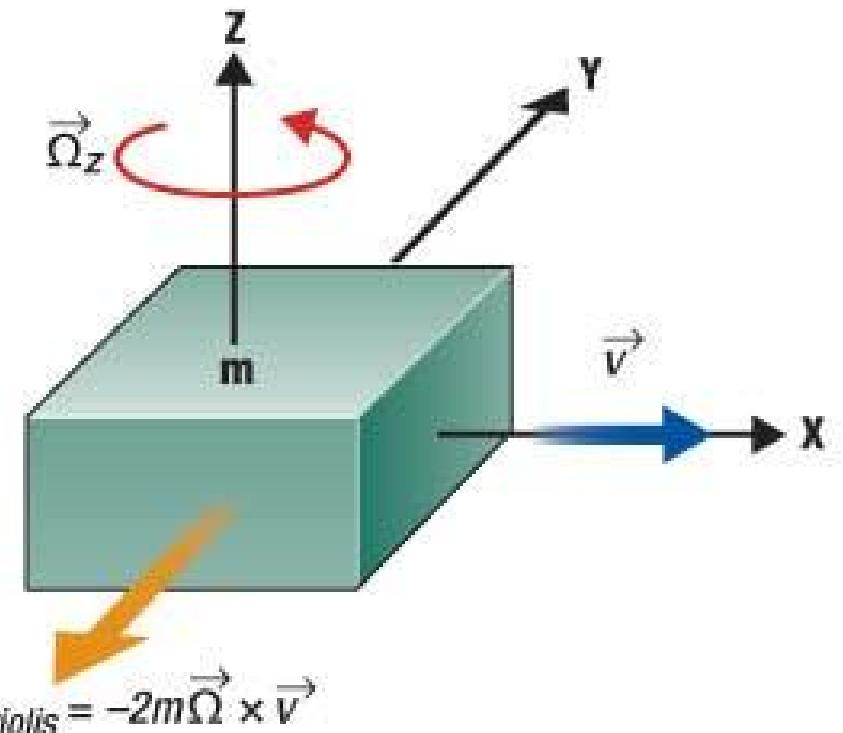


The Direction of Coriolis Force

- To memorize using the right hand rules.

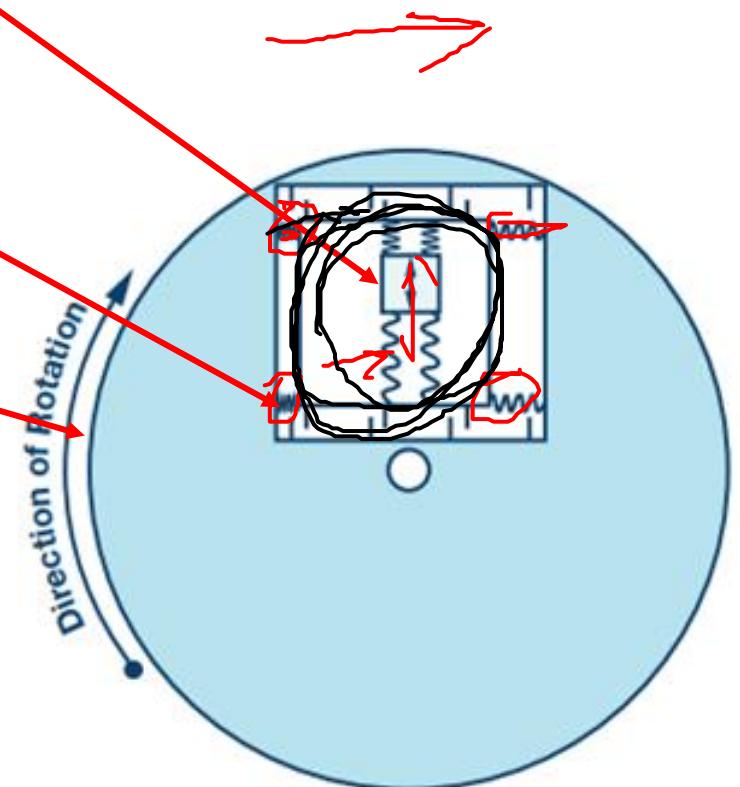
Steps:

1. Find the rotation axis using the right hand rule (z).
2. Cross product the movement direction(x) and rotation axis (z), and get the force direction. Using the right hand rule



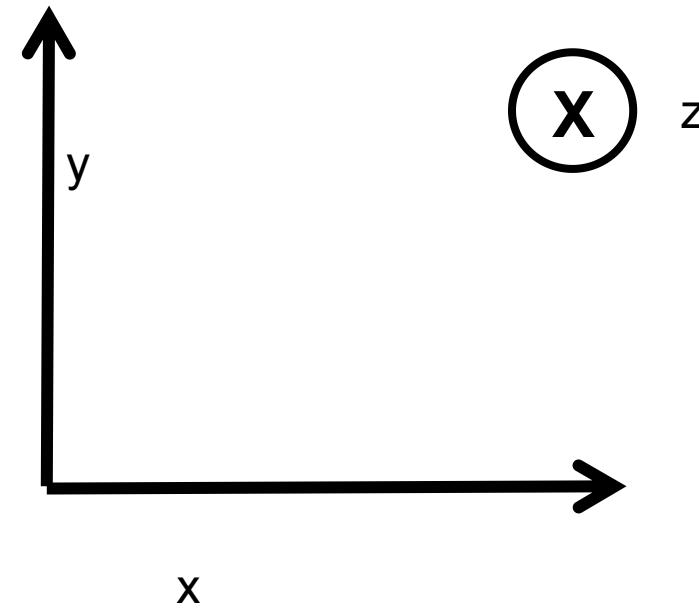
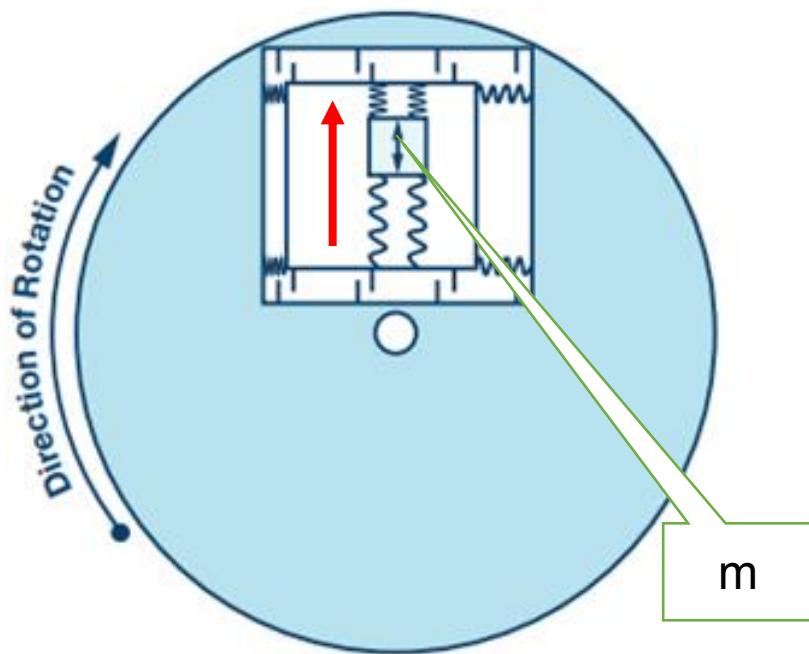
How MEMS Gyroscope works?

- A gyroscope will have a mass oscillating back and forth along the first axis. Easy to implement using a minuscule mass.
- This oscillating mass is then placed on a second spring controlled pads. The capacitance is proportional to their positions
- These structures are placed on a disc that is free to rotate.
- When a rotation is detected around the second direction, the capacitance changes



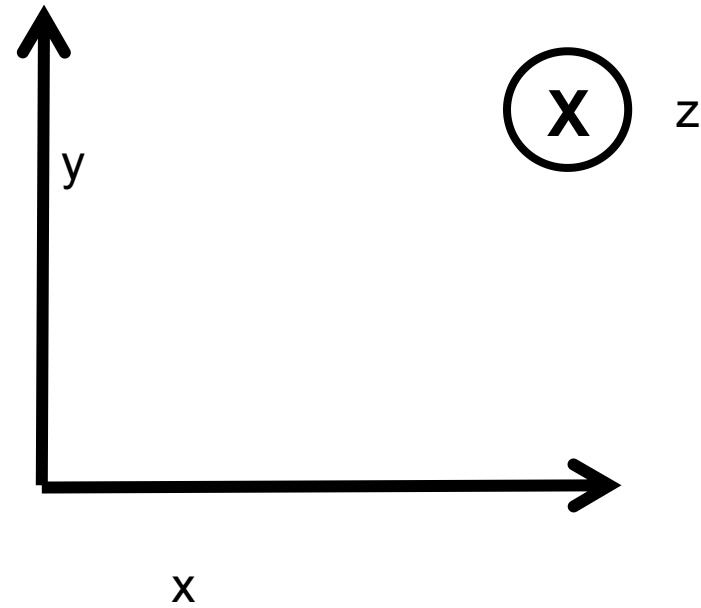
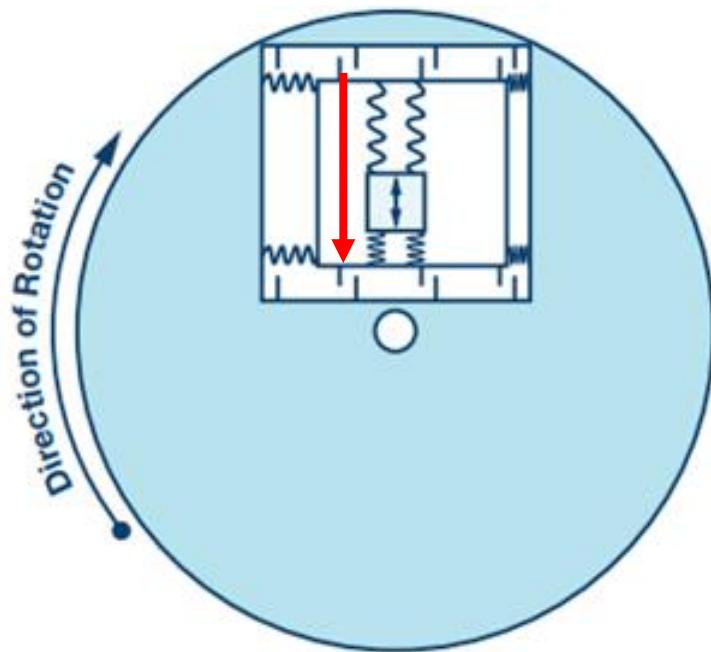
Inside an MEMS gyroscope

- A mass, m , vibrating along y axis
- When the mass moves $+y$ direction, rotating along z clockwise
 - What is Coriolis force direction?
 - $-x$



Exercise

- Explain the direction of Coriolis force when object moves along $-y$ direction



CSE 162 Mobile Computing

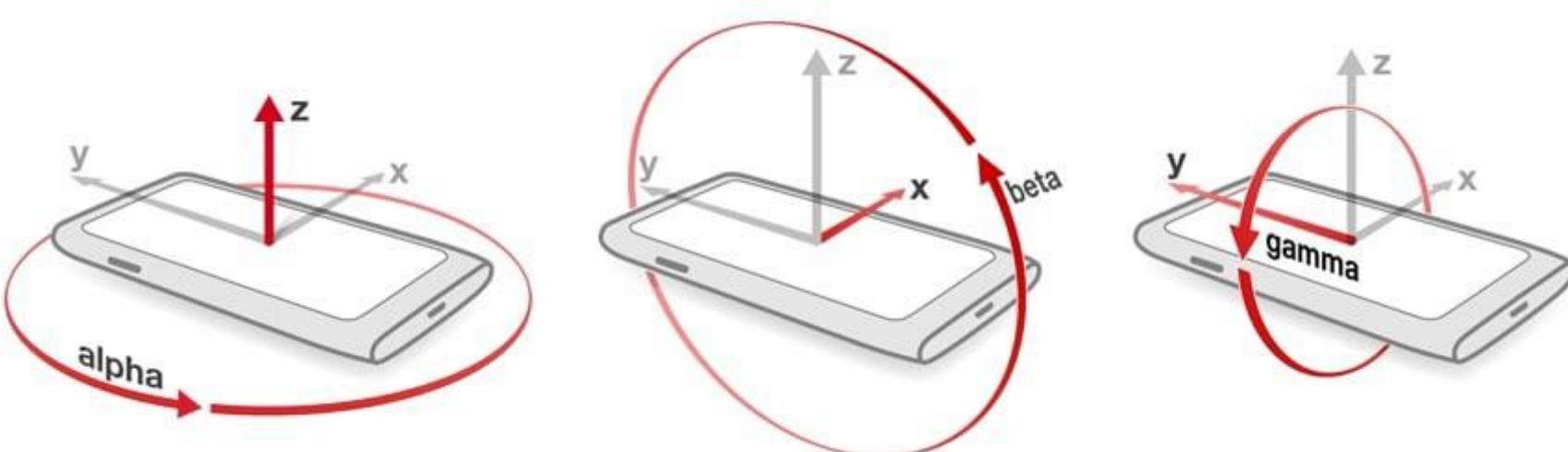
Orientation and Step Counting

Hua Huang

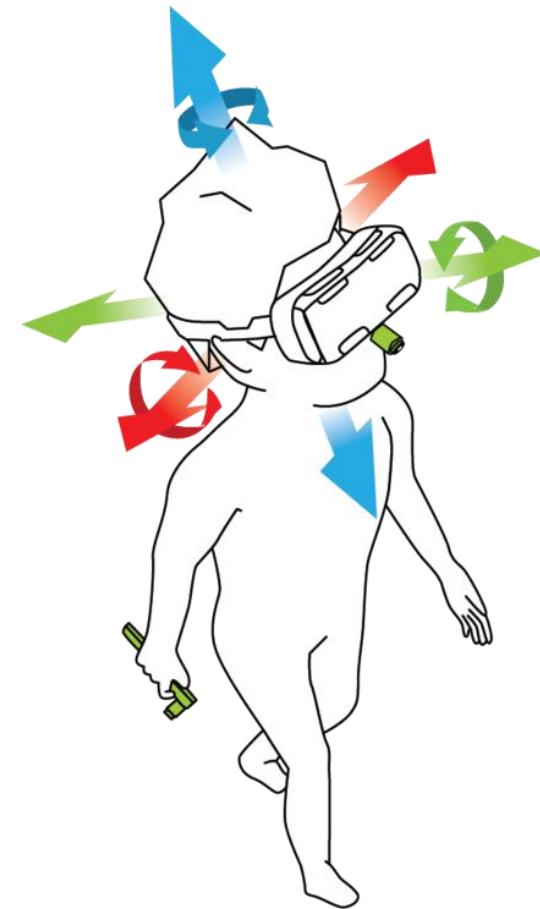
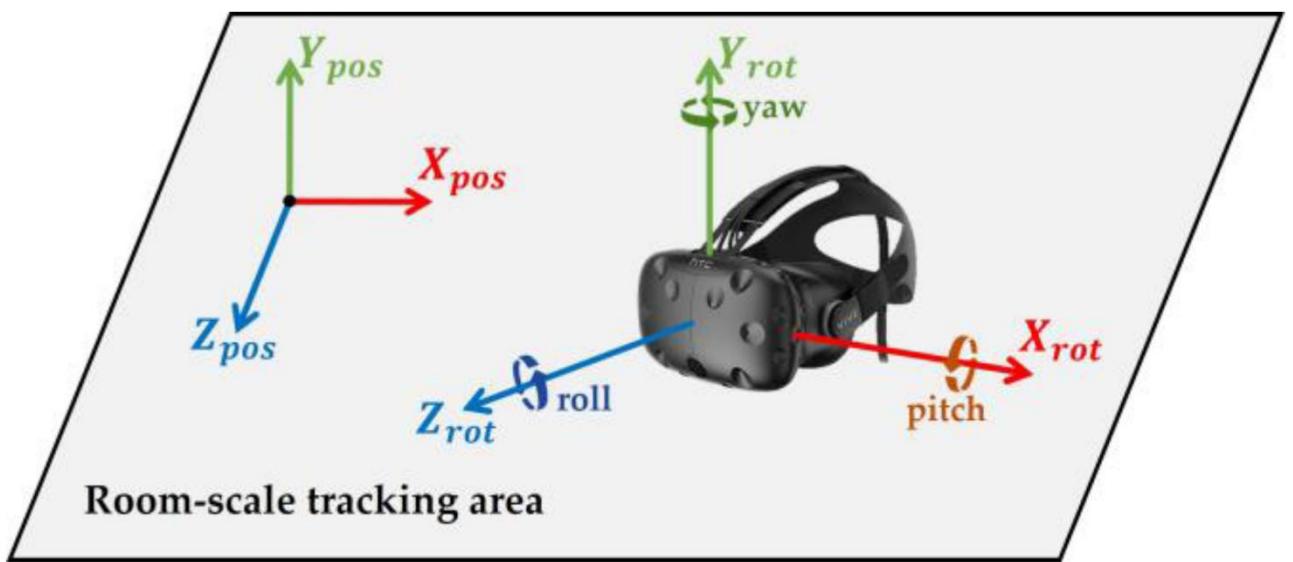
Virtual Sensor: 3D Orientation

Question:

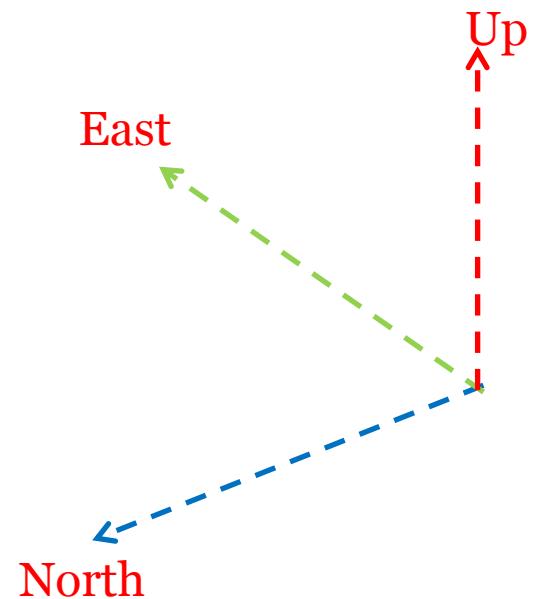
How do we know the orientation of the phone?



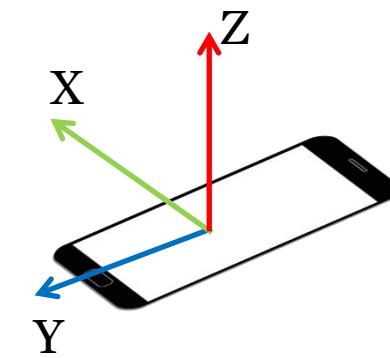
- Orientations of the VR goggle?



Coordinate frames

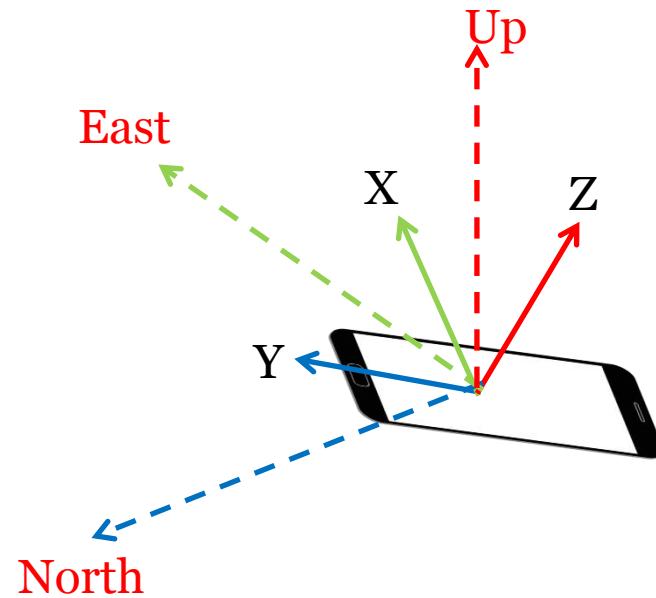


Global Frame



Local Frame

Consider a phone in a random orientation

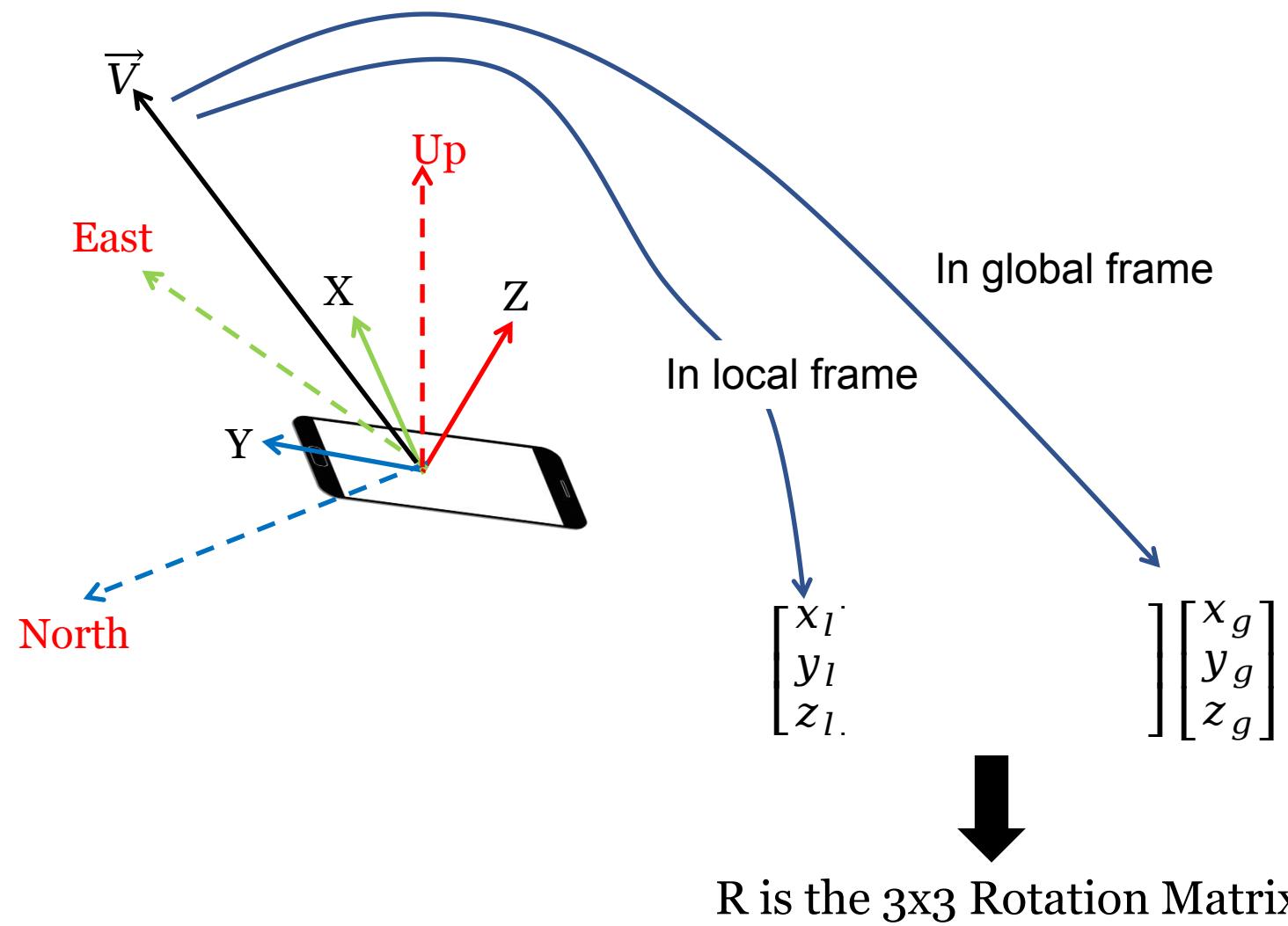


3D Orientation captures the **misalignment** between **global** and **local** frames

Gravity Sensor

- A virtual sensor
- Calculated using accelerometer
- Always points to the earth

Rotation Matrix



3x3 Rotation matrix captures the full 3D orientation

How can we estimate rotation matrix?

Key idea use globally known reference vectors
which can also be measured in the local frame of reference

- Gravity
- Magnetic North

Gravity equation

Gravity globally known, measurable in local frame with gravity sensor

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \begin{bmatrix} 0 \\ M \\ 0 \end{bmatrix}$$

Magnetic north, globally known, measurable in local frame with magnetometer

6 equations and 9 unknowns (3x3 rotation matrix) can we solve ?

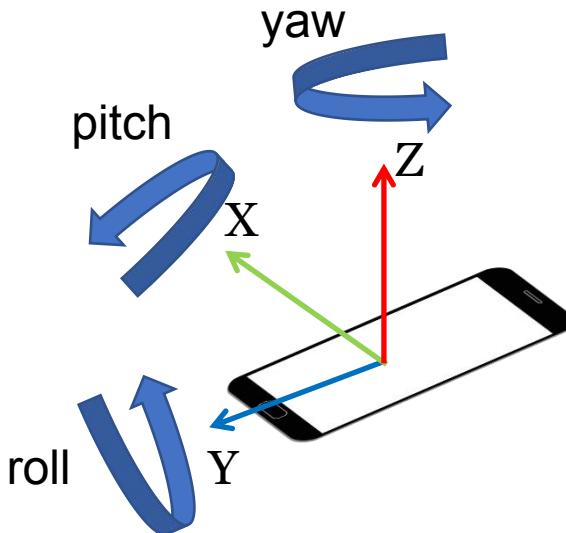
Yes, these 9 unknowns are all not independent (rotation matrix satisfies special properties)

- It does not change length of a vector
- Columns are orthogonal unit vectors

The above 6 equations are sufficient to solve the rotation matrix

Gravity Sensor and Magnetometer can be used to determine the rotation matrix (3D orientation)

Decomposing the rotation matrix



$$\boxed{\begin{bmatrix} \text{3x3 Rotation} \\ \text{Matrix } R \end{bmatrix}} = \begin{bmatrix} \cos(pitch) & 0 & -\sin(pitch) \\ 0 & 1 & 0 \\ \sin(pitch) & 0 & \cos(pitch) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(roll) & \sin(roll) \\ 0 & -\sin(roll) & \cos(roll) \end{bmatrix} \begin{bmatrix} \cos(yaw) & -\sin(yaw) & 0 \\ \sin(yaw) & \cos(yaw) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Orientation can be represented as 3D yaw, pitch, roll

Estimating yaw, pitch, roll will determine the orientation

Gravity equation

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \text{Rotation Matrix } R \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} \cos(pitch) & 0 & -\sin(pitch) \\ 0 & 1 & 0 \\ \sin(pitch) & 0 & \cos(pitch) \end{bmatrix} \begin{bmatrix} 0 \\ \cos(roll) & \sin(roll) \\ -\sin(roll) & \cos(roll) \end{bmatrix} \begin{bmatrix} \cos(yaw) & -\sin(yaw) \\ \sin(yaw) & \cos(yaw) \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

Gravity equation

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \text{Rotation Matrix } R \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} \cos(pitch) & 0 & -\sin(pitch) \\ 0 & 1 & 0 \\ \sin(pitch) & 0 & \cos(pitch) \end{bmatrix} \begin{bmatrix} 0 \\ \cos(roll) \\ \sin(roll) \end{bmatrix} \begin{bmatrix} 0 \\ \sin(roll) \\ \cos(roll) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

Gravity output does not depend on yaw!

Hence, yaw cannot be estimated using gravity

Accelerometer equation

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \boxed{\text{Rotation Matrix } R} \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} -\sin(pitch) \cdot \cos(roll) \\ -\sin(roll) \cdot g \\ -\cos(pitch) \cdot \cos(roll) \end{bmatrix} g$$

The above equations estimate pitch and roll

Magnetometer equation

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \text{Rotation Matrix } R \begin{bmatrix} 0 \\ M \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \begin{bmatrix} \cos(pitch) & 0 & -\sin(pitch) \\ 0 & 1 & 0 \\ \sin(pitch) & 0 & \cos(pitch) \end{bmatrix} \begin{bmatrix} 0 \\ \cos(roll) & \sin(roll) \\ 0 & -\sin(roll) & \cos(roll) \end{bmatrix} \begin{bmatrix} \cos(yaw) & -\sin(yaw) \\ \sin(yaw) & \cos(yaw) \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ M \\ 0 \end{bmatrix}$$

Magnetometer equation

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \text{Rotation Matrix } R \begin{bmatrix} 0 \\ M \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \begin{bmatrix} \cos(\text{pitch}) & 0 & -\sin(\text{pitch}) \\ 0 & 1 & 0 \\ \sin(\text{pitch}) & 0 & \cos(\text{pitch}) \end{bmatrix} \begin{bmatrix} 0 \\ \cos(\text{roll}) & \sin(\text{roll}) \\ -\sin(\text{roll}) & \cos(\text{roll}) \end{bmatrix} \begin{bmatrix} \cos(\text{yaw}) & -\sin(\text{yaw}) \\ \sin(\text{yaw}) & \cos(\text{yaw}) \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ M \\ 0 \end{bmatrix}$$

Pitch, roll known from accelerometer

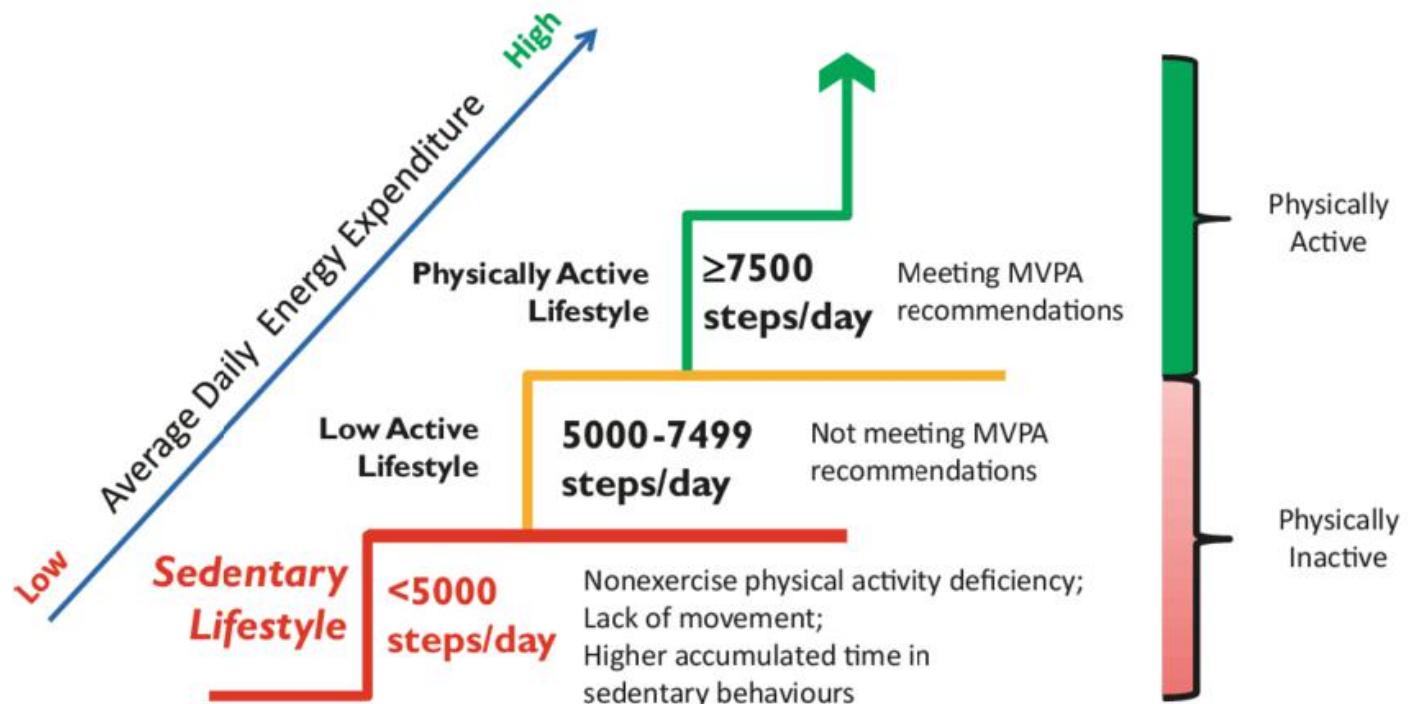
Unknown yaw can be determined from above equations

yaw, pitch, roll together determine the rotation matrix (3D orientation) of a system

Virtual Sensor: Step Counting

Sedentary Lifestyle

- Sedentary lifestyle
 - Increases risk of diabetes, heart disease, dying earlier, etc
 - Kills more than smoking!!
- Categorization of sedentary lifestyle based on step count:
 - “A step-defined sedentary lifestyle index: < 5000 steps/day” (2013)



Step Count Mania

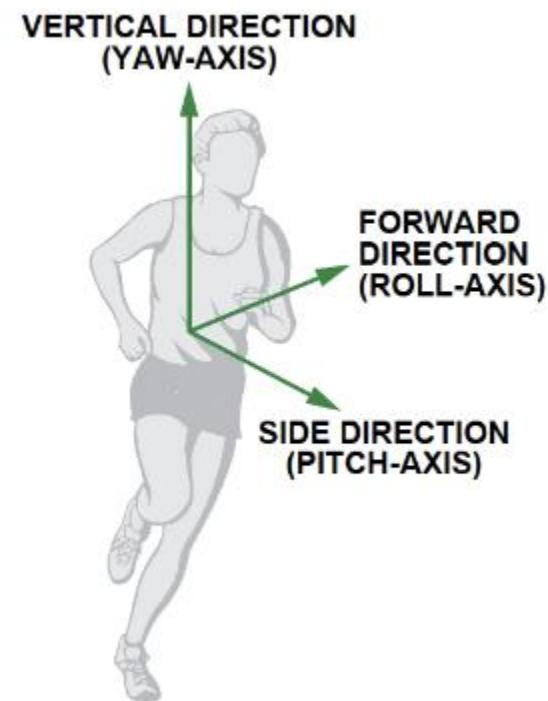
- Everyone is crazy about step count these days
- Pedometer apps, pedometers, fitness trackers, etc
- Tracking makes user aware of activity levels, motivates them to exercise more



Benefits mobile step counters

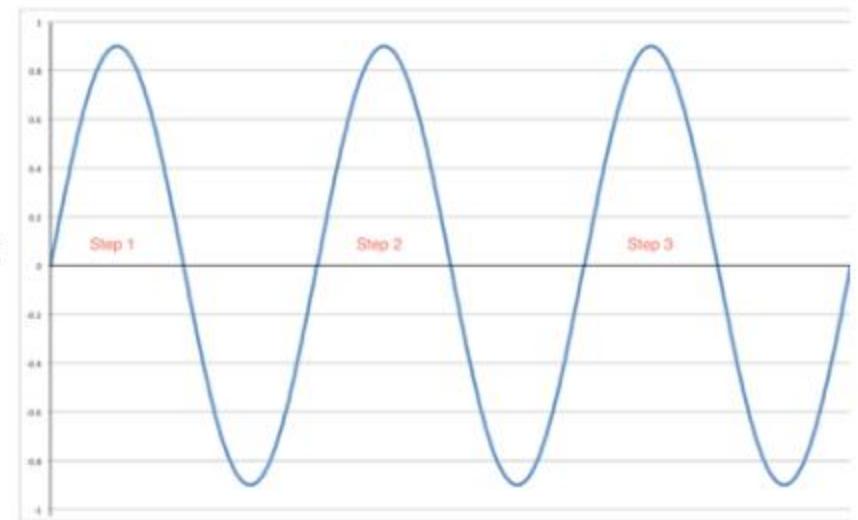
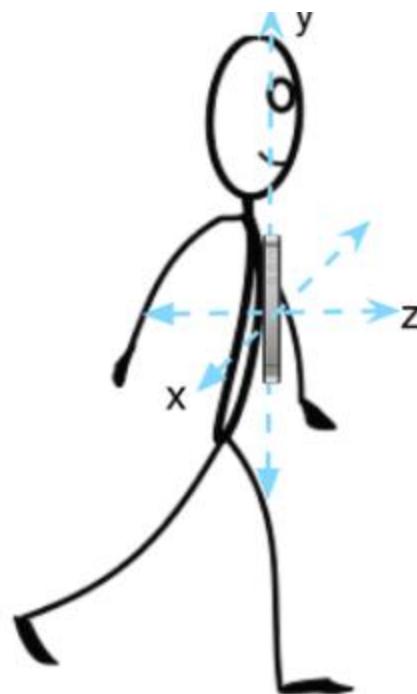
- Always on. Everywhere. Continuous monitoring.
- Low-cost
- Better privacy than computer vision

Definition of each axis



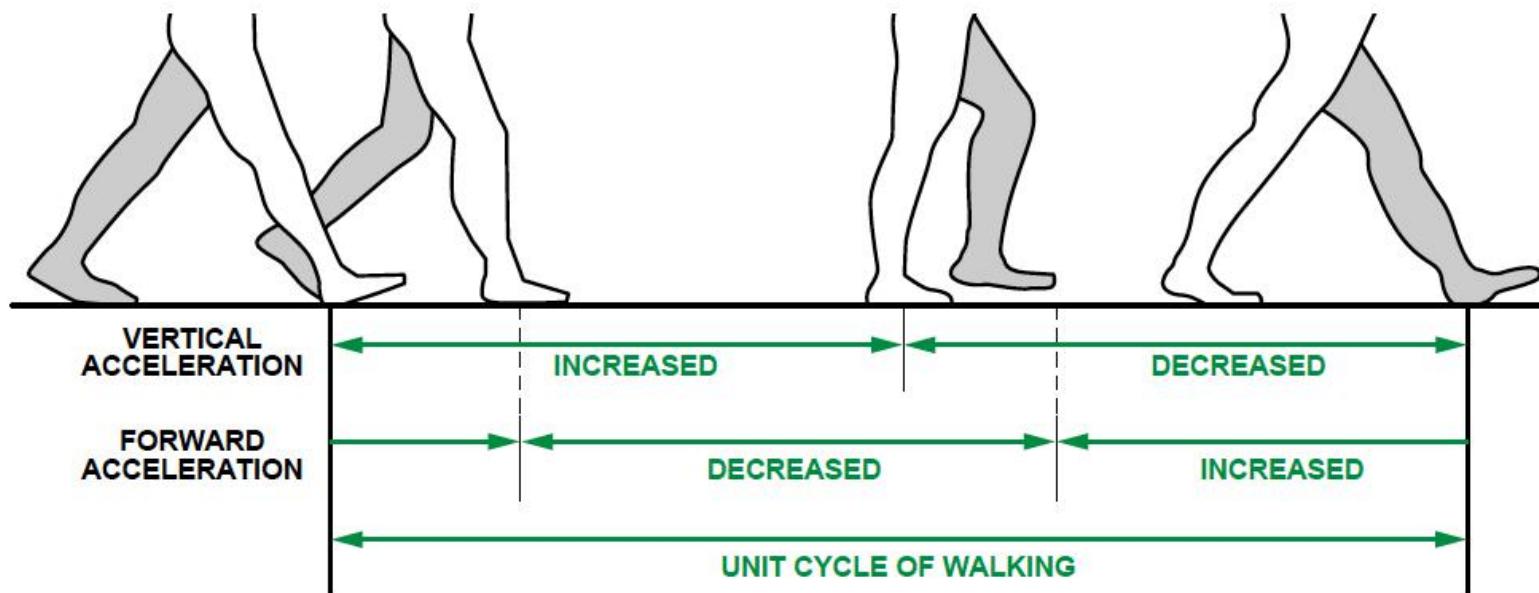
Ideal sensor data

- In an ideal situation, you would like to see a signal like this.
 - One of the axes of the phone is along the direction of gravity, and the steps can be clearly observed in the signal.
- But in practice, the data often deviates from the ideal case



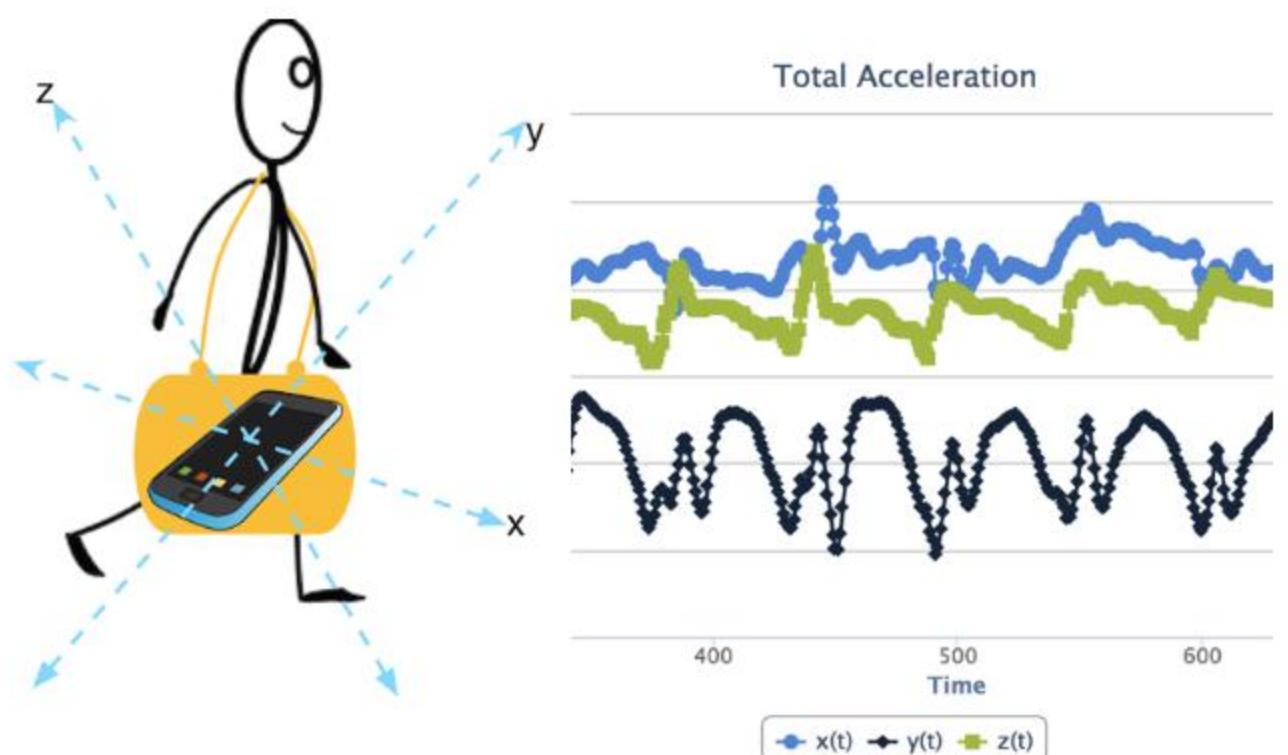
The Nature of Walking

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



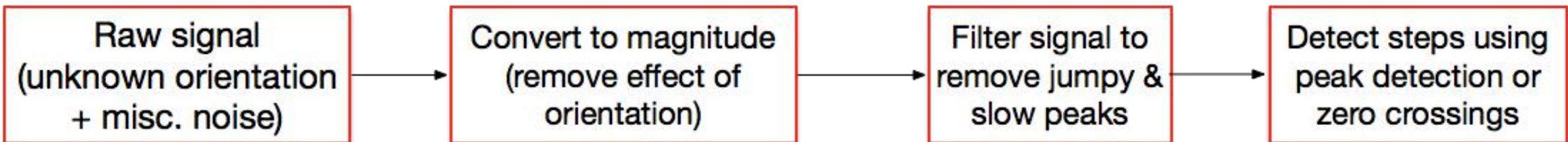
Accelerometer signal in a real-world situation

- A more realistic signal is shown
 - some component of the user acceleration and gravity is present along all three axes.
 - The measurements are influenced by the phone orientation
- we need to design an *orientation-independent* algorithm to detect steps.



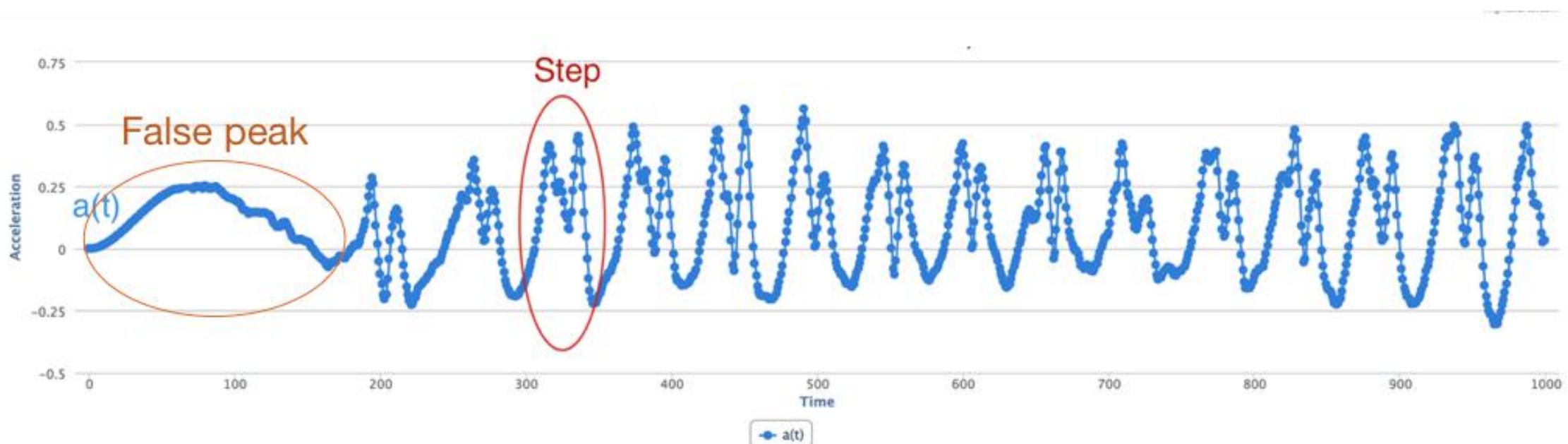
Step Detection Algorithm

- The key insight in our method is to convert the 3-axis signal into a one axis magnitude signal, and then extract steps from this signal.



Step 1: Extract Signal Magnitude

- take the magnitude of the entire acceleration vector i.e. $\sqrt{x^2 + y^2 + z^2}$, where x, y, and z are the readings of the accelerometer along the three axes.
- The signal is not dependent on phone orientation now



Step 2: Filter the signal to remove noise

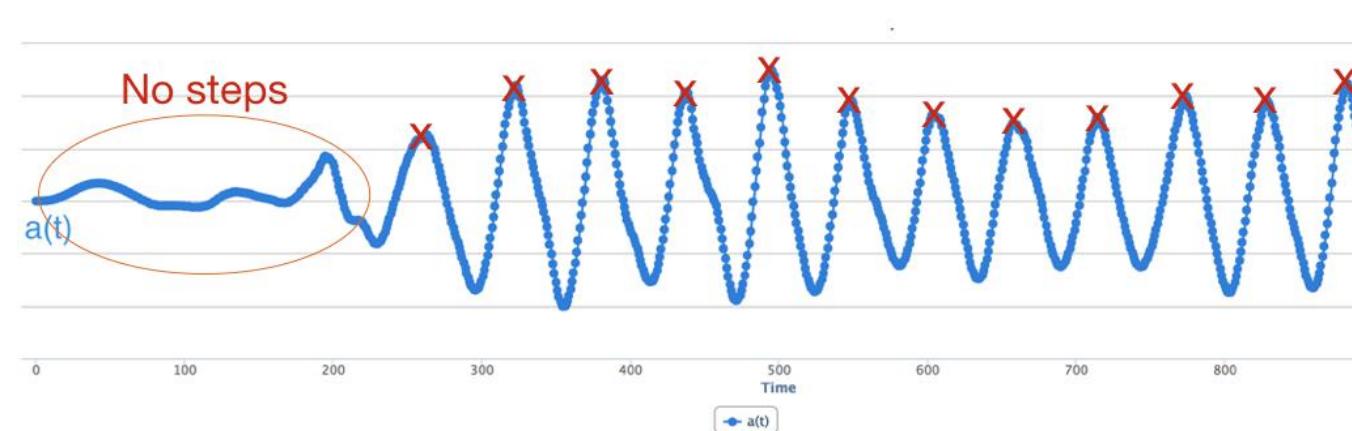
- What noises exist in the data?
 - **Jumpy peaks:** Since the phone is often carried in a pocket/purse, it can jiggle a little with each step. Also, some users have a bounce in their step, so even though they are taking a single step, the phone can bounce multiple times within this step.
 - **Short peaks:** Small peaks can occur when a user is using a phone (e.g. making a call or using an app).
 - **Slow peaks:** Slow peaks can occur when the phone is moved or due to movements of the leg while sitting (if the phone is in the pant pocket)

Filtering

- To remove these sources of noise, we are going to use frequency-domain noise removal.
- Notice that we need to remove high frequency variations like jumpy peaks and low frequency variations like slow peaks.
- A simple solution is to use a filter that keeps only frequencies relating to walking and removes the rest.

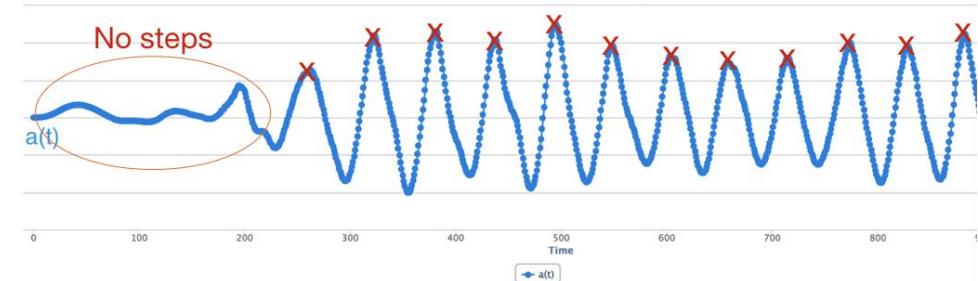
Filtering

- Typical walking pace may be under three steps a second (3 Hz) and over half step a second (0.5Hz), so we remove all frequencies above 5 Hz and below 0.5 Hz (just to give some margin for error)
- Even after we remove low and high frequency peaks, we may be left with some short peaks.
 - A simple way to deal with this is to look only for large peaks and ignore small peaks.



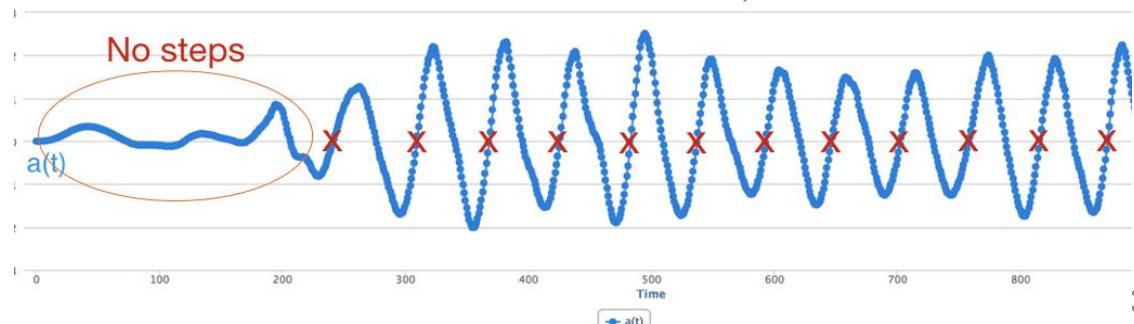
Step 3: Detect Steps

- Approach 1: Find signal peaks



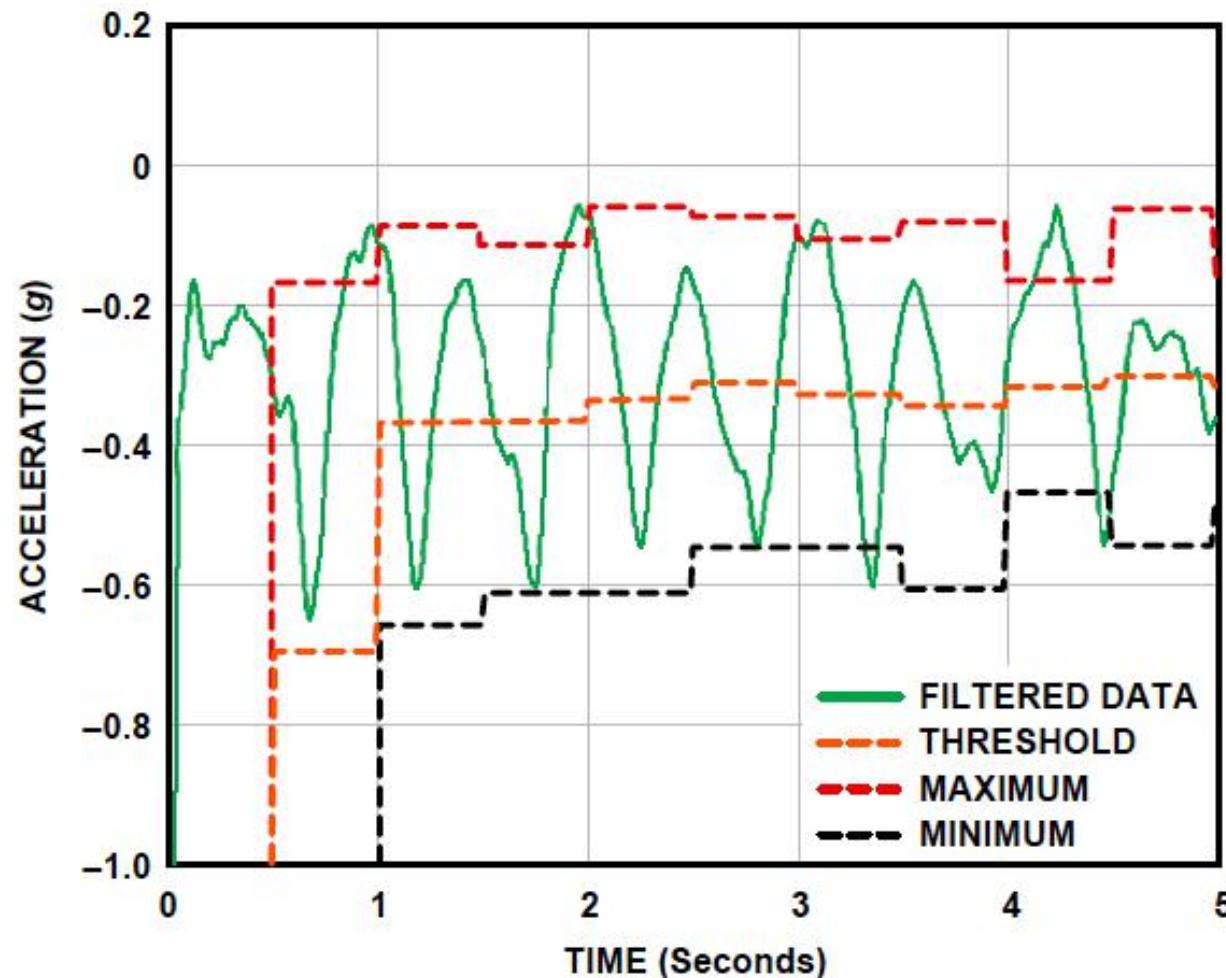
- Approach 2: Zero crossing:

- Subtract the mean for each window and look at zero crossings i.e. times when the signal crosses from the negative to positive in the upward direction



Dynamic Threshold-based Step Detection

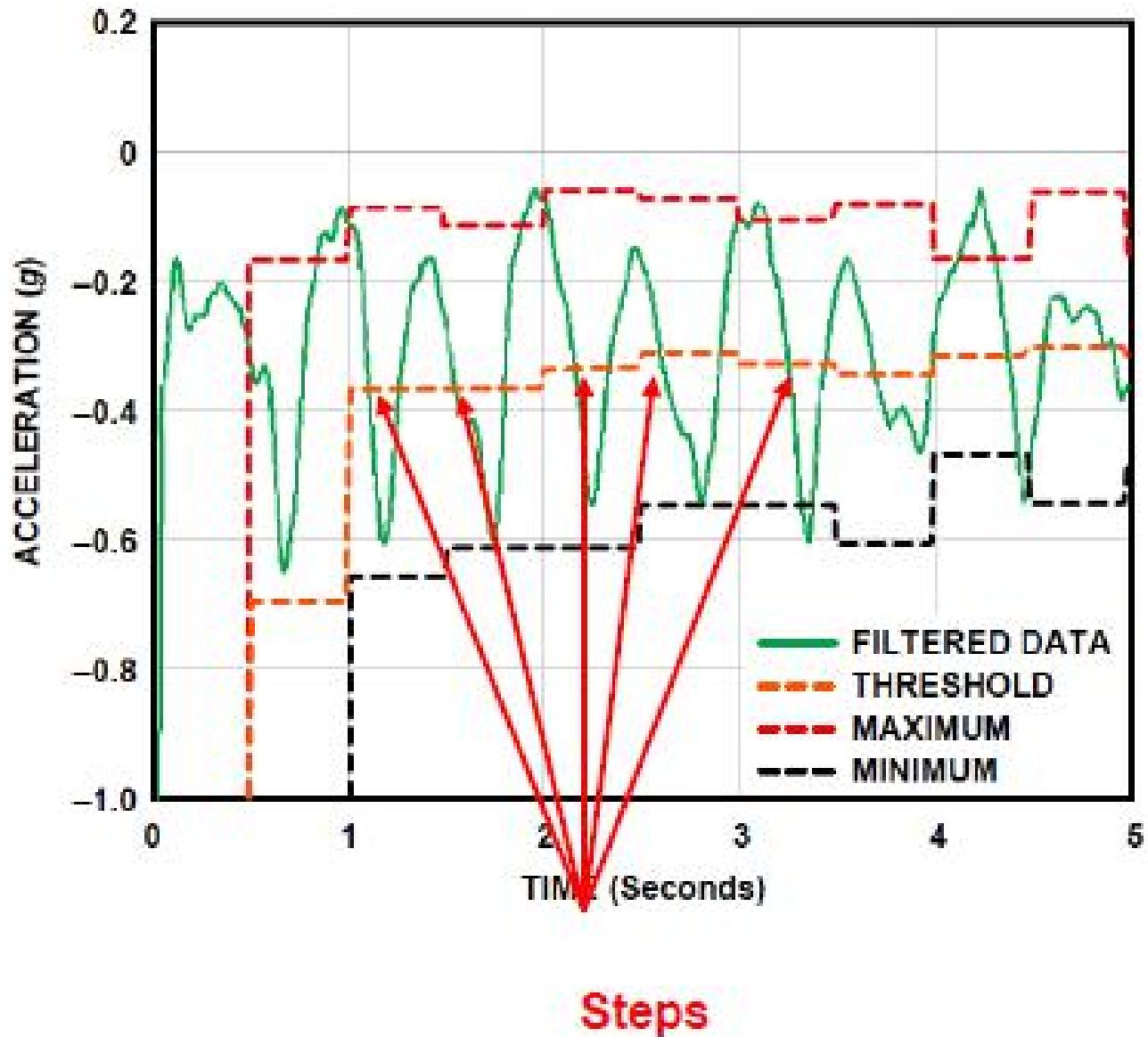
- Focus on accelerometer axis with largest peak
- Would like a threshold such that each crossing is a step
- Track min, max values observed every 50 samples
- Compute ***dynamic threshold***: $(\text{Max} + \text{Min})/2$



Step Detection Algorithm

A step is

- Indicated by crossings of dynamic threshold
- Defined as negative slope ($\text{sample_new} < \text{sample_old}$) when smoothed waveform crosses dynamic threshold



Distance Estimation

- Calculate distance covered based on number of steps taken
 - $Distance = number\ of\ steps \times distance\ per\ step$
- Distance per step (stride) depends on user's height (taller people, longer strides), and step frequency
- Using person's height, can estimate their stride, then number of steps taken per 2 seconds

Steps per 2 s	Stride (m/s)
0~2	Height/5
2~3	Height/4
3~4	Height/3
4~5	Height/2
5~6	Height/1.2
6~8	Height
≥ 8	$1.2 \times Height$

Calorie Estimation

- To estimate speed, remember that speed = distance/time. Thus,
 - $Speed \text{ (in m/s)} = (\text{no. steps per 2 s} \times \text{stride (in meters)})/2\text{s}$
- Calorie expenditure, which depends on many factors
 - Body weight, workout intensity, fitness level, etc
- Empirical simplified equation:
 - $Calories \text{ (C/kg/h)} = 1.25 \times speed \text{ (m/s)} \times 3600/1000 = 4.5 \times speed \text{ (m/s)}$

Limitations and Future Work

- Strong assumptions on how the users walk.
 - What about short-interval, high intensity exercise?
 - What about the other calorie expenditures? Standing vs sitting
- Currently, dedicated system for each activities. General activity recognition is still under research.



CSE 162 Mobile Computing

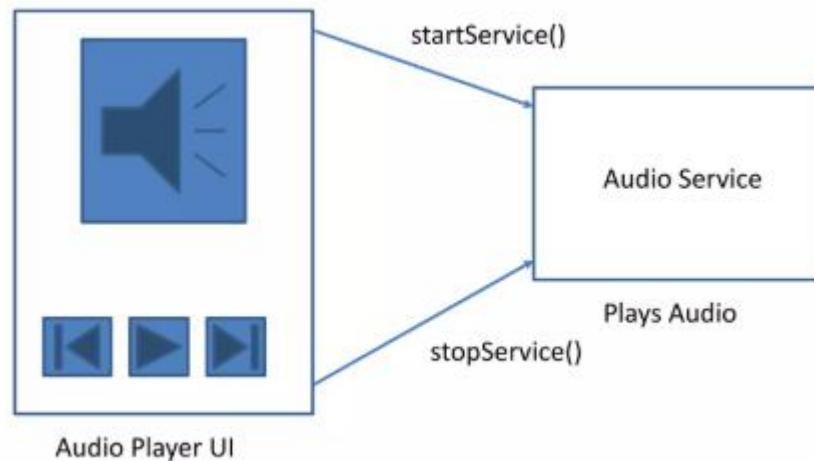
Mobile Activity Recognition

Hua Huang

Services

What is Android Service?

- Services are codes that run in the background
- They can be started and stopped
- Services doesn't have UI



What a Service is NOT?

- There are some confusions:
- A Service is not a separate process.
 - The Service object itself does not imply it is running in its own process; unless otherwise specified, it runs in the same process as the application it is part of.
- A Service is not a thread.
 - It is not a means itself to do work off of the main thread (to avoid Application Not Responding errors).

Main Features of Service

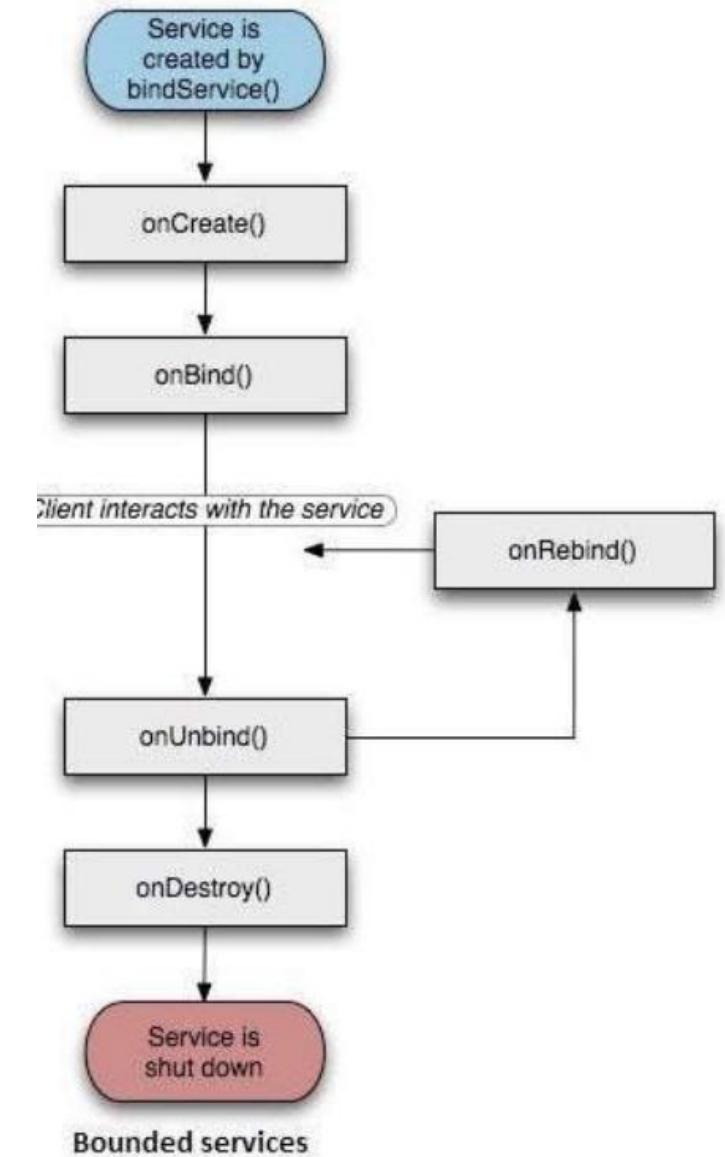
- To tell the system about something it wants to be doing in the background (even when the user is not directly interacting with the application).
- To call `Context.startService()`, which asks the system to schedule work for the service, to be run until the service or someone else explicitly stops it.

When to use Services

- Activities are short-lived, can be shut down anytime (e.g when user presses back button)
- Services keep running in background
- Similar to Linux/Unix CRON job
- Example uses of services:
 - Periodically check/update device's GPS location
 - Check for updates to RSS feed
 - Independent of any activity, minimal interaction
- Typically an activity will control a service --start it, pause it, get data from it
- Services in an App are sub-class of Android's **Services** class

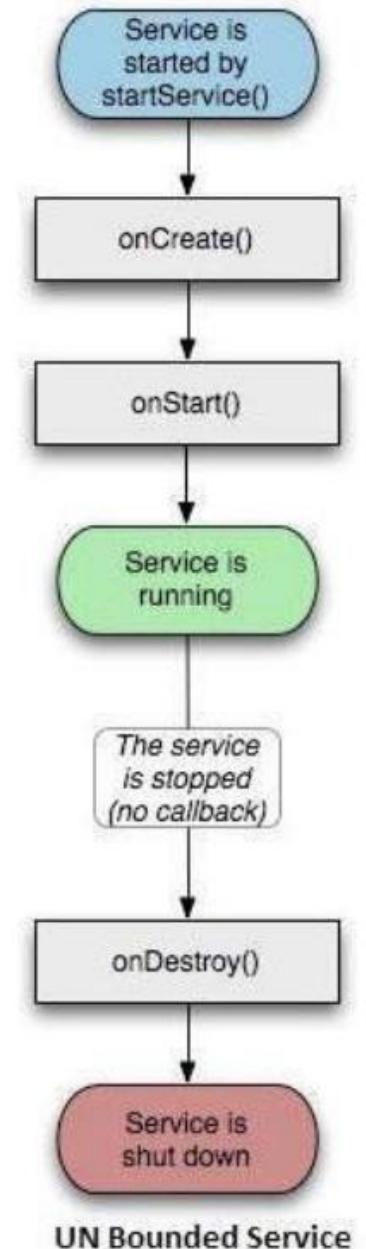
Bound Service

- An Android component may bind itself to a Service using `bindService()`.
- A bound service would run as long as the other application components are bound to it.
- As soon as they unbind, the service destroys itself.



Unbound Service

- In this case, an application component starts the service, and it would continue to run in the background, even if the original component that initiated it is destroyed.
- For instance, when started, a service would continue to play music in the background indefinitely.



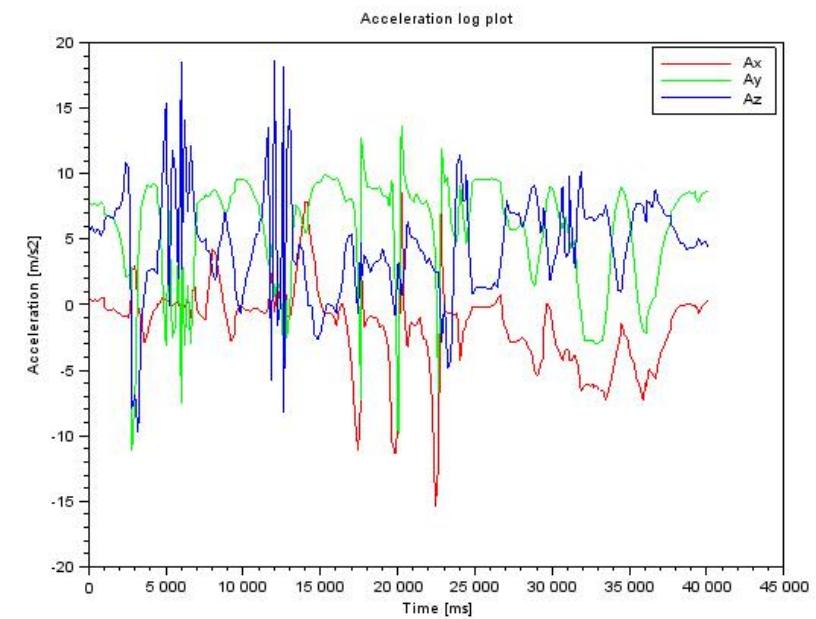
Android Platform Services

- Android Services can either be on:
 - On smartphone or Android device (local)
 - Remote, on Google server/cloud
- Android platform local services examples (on smartphone):
 - **LocationManager:** location-based services.
 - **ClipboardManager:** access to device's clipboard, cut-and-paste content
 - **DownloadManager:** manages HTTP downloads in background
 - **FragmentManager:** manages the fragments of an activity.
 - **AudioManager:** provides access to audio and ringer controls.

An Introduction to Human Activity Recognition

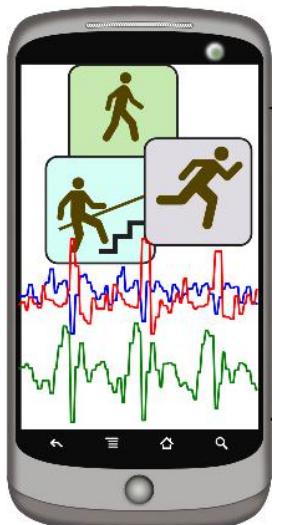
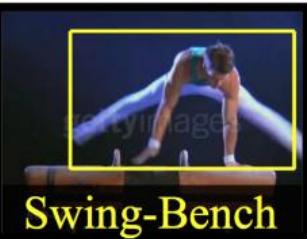
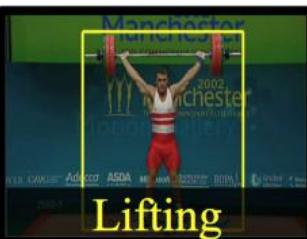
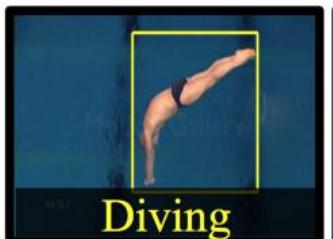
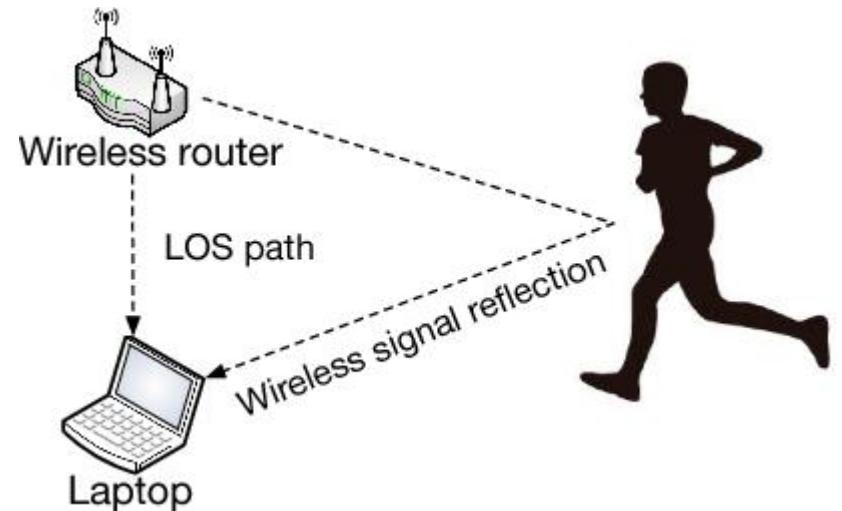
Activity Recognition

- 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 classifiers to classify user's accelerometer signals

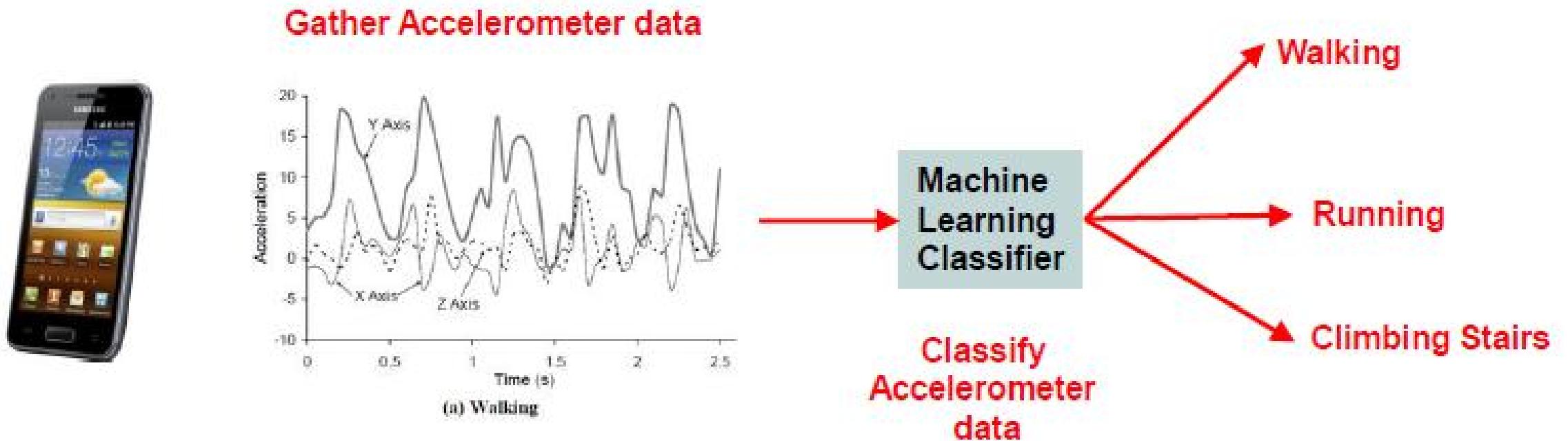


Activity Recognition Systems

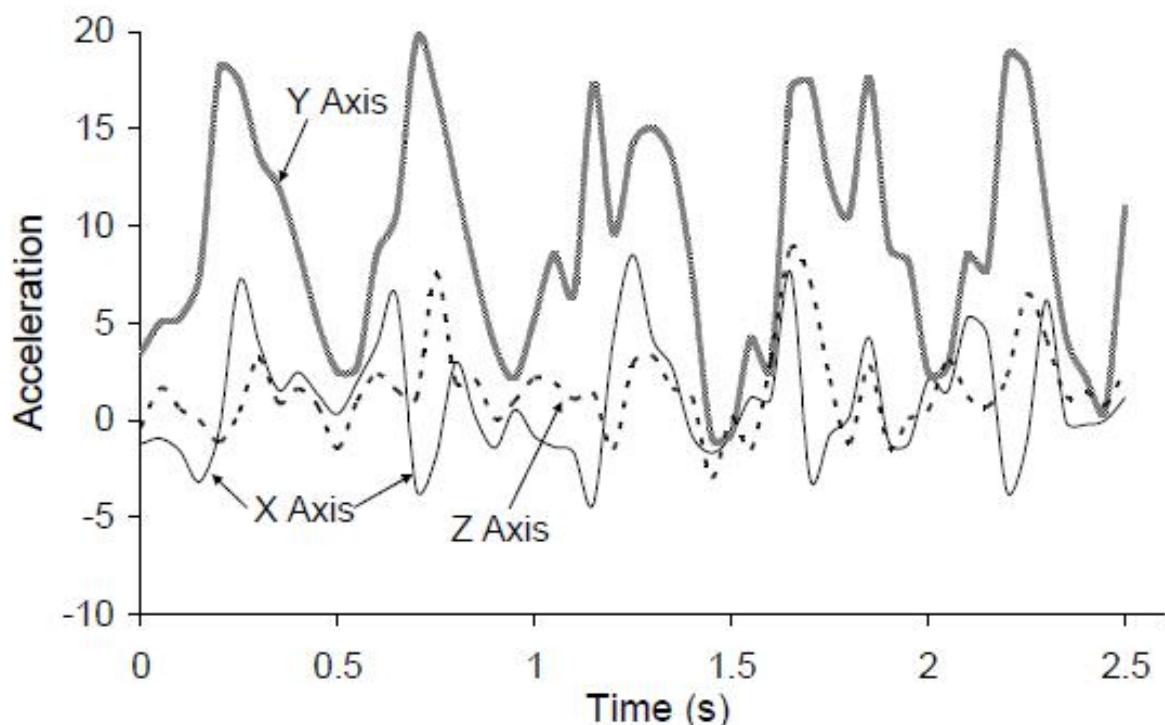
- Computer vision based
- Environmental sensor based
- Mobile sensor based



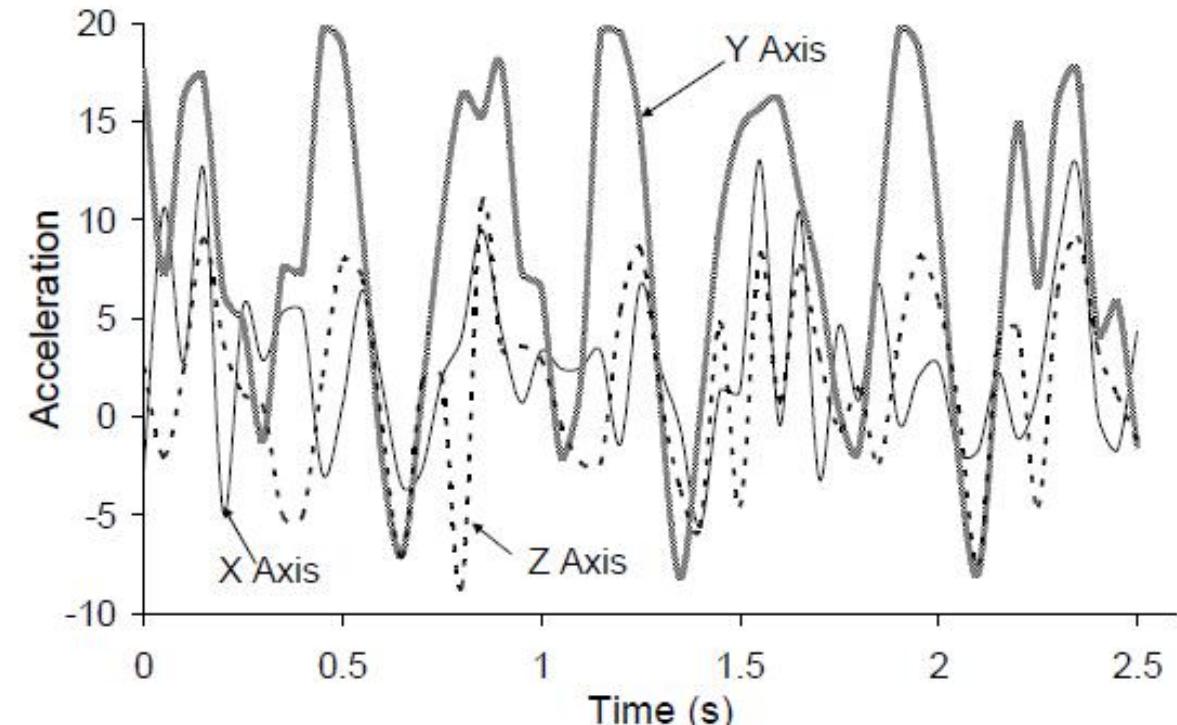
Mobile Activity Recognition Architecture



Sample Accelerometer during activities

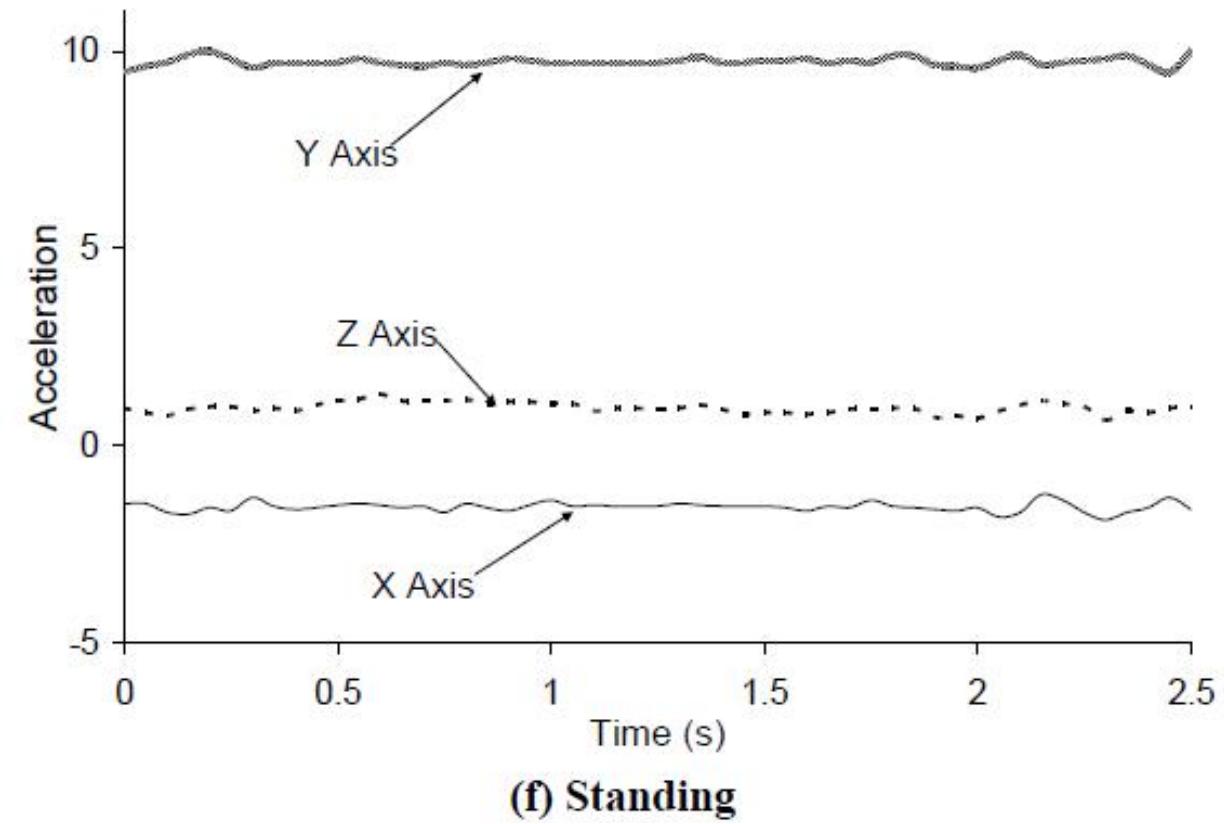
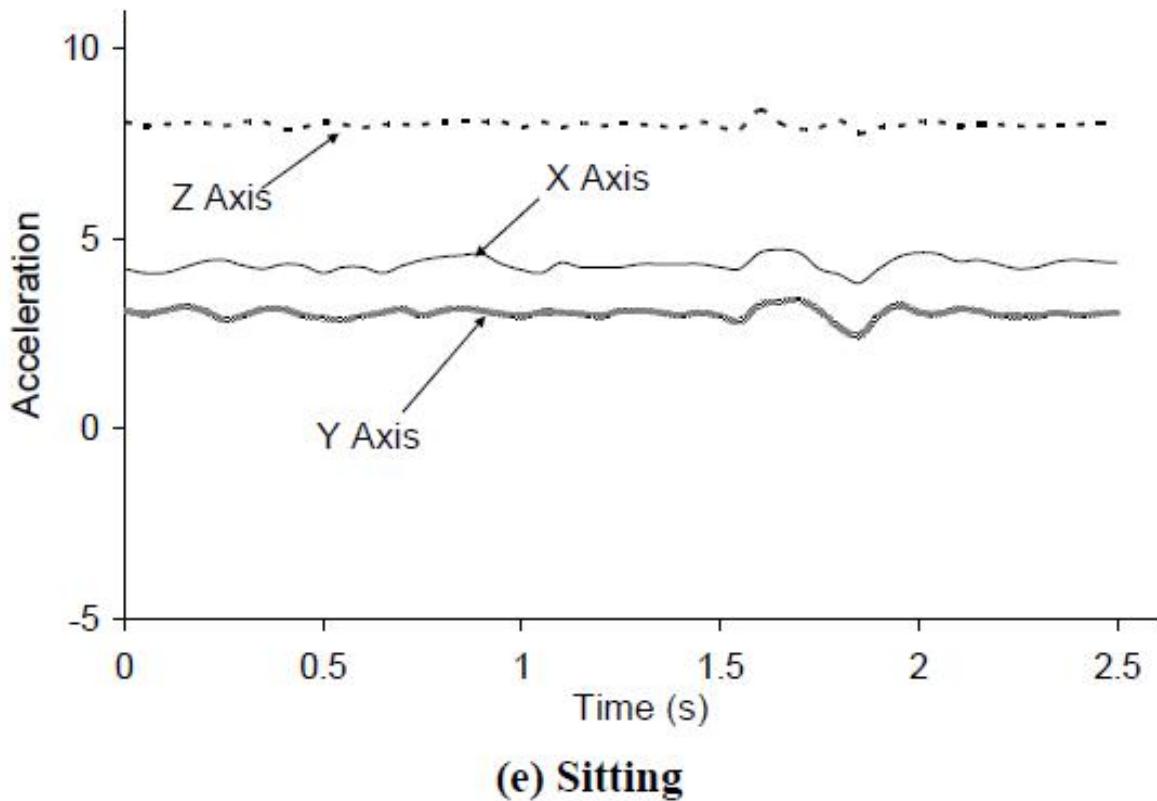


(a) Walking



(b) Jogging

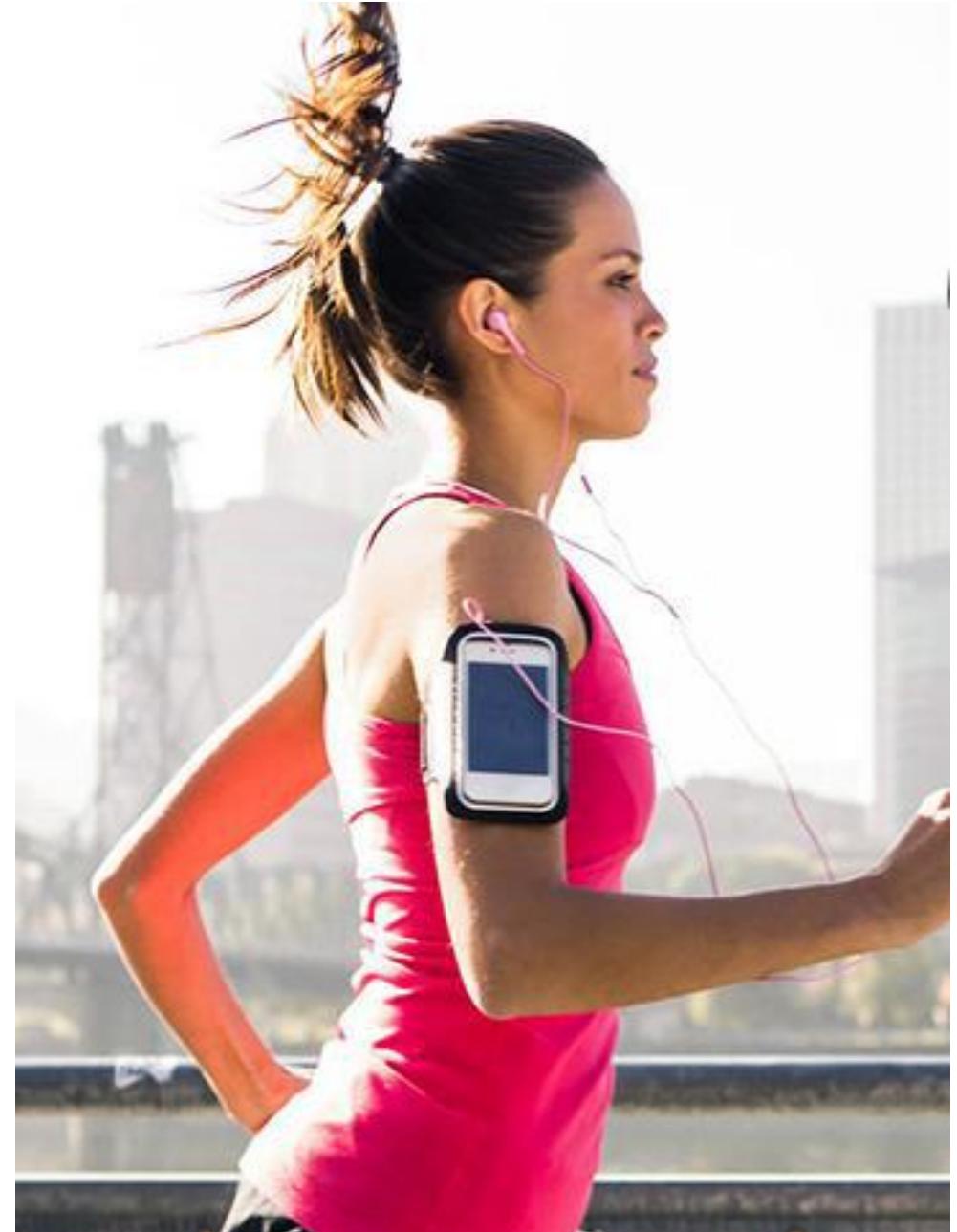
Sample Accelerometer during activities



Activity Recognition Applications

Fitness Tracking

- **Initially:**
 - Physical activity type,
 - Distance travelled,
 - Calories burned
- **Newer features:**
 - Stairs climbed,
 - Physical activity (duration + intensity)
 - Activity type logging + context
 - Sleep tracking
 - Activity history



Health Monitoring

- Make clinical monitoring pervasive, continuous, real world!!
 - Gather context information (e.g. what makes condition worse/better?)
 - E.g. timed up and go test
- Show patient contexts that worsen condition => Change behavior
 - E.g. walking in narrow hallways worsens gait freeze



Gait Freezing

**Question: What data would you need to build PD gait classifier?
From what types of subjects?**

Fall Detection

- A leading cause of death for seniors
- Smartphone/watch, wearable detects senior who has fallen, alert family
 - Text message, email, call relative



Context-aware Behavior

- Study found that messages delivered when transitioning between activities better received
 - In-meeting? => Phone switches to silent mode
 - Exercising? => Play song from playlist, use larger font sizes for text
 - Arrived at work? => download email
- Adaptive Systems to Improve User Experience:
 - Walking, running, riding bike? => Turn off Bluetooth, WiFi (save energy)
 - an increase battery life up to 5x

Smart Home

- Smart Thermostat
 - Determines if the users are at home
 - Recognize where the users are and adjust temperature accordingly
- Smart TV
 - Turns on TV when the user is at the couch



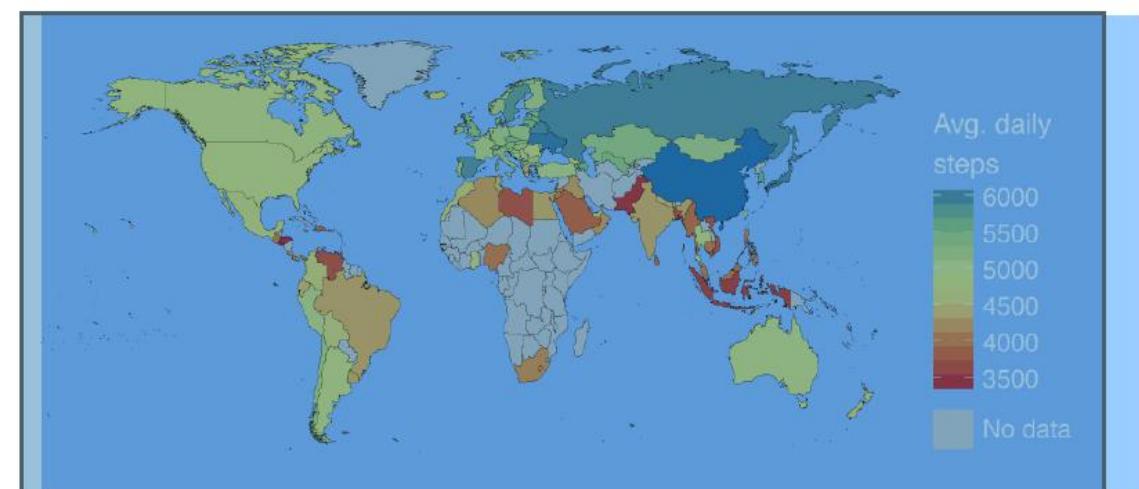
Targeted Advertisements

- User runs a lot => Get exercise clothing ads
- Goes to pizza places often + sits there => Get pizza ads



Research Platforms for Data Collection

- E.g. public health officials want to know how much time various people (e.g. students) spend sleeping, walking, exercising, etc
- Mobile AR: inexpensive, automated data collection
- E.g. Stanford Inequality project: Analyzed physical activity of 700k users in 111 countries using smartphone AR data
 - <http://activityinequality.stanford.edu/>



Track, manage staff on-demand

- E.g. at hospital, determine “availability of nurses”, assign them to new jobs/patients/surgeries/cases

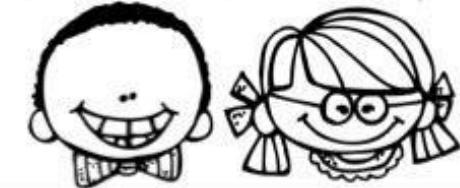


Activity-Based Social Networking

- Automatically connect users who do same activities + live close together

Find a friend who . . .

name _____



has a pet dog 	has black hair 	likes to play soccer 	has a blue backpack 
has a brother 	likes to color 	has a summer birthday 	likes chocolate ice cream 
likes to eat pizza 	can play an instrument 	has a sister 	likes to swim 
has brown eyes 	is wearing white shoes 	likes the color red 	has a pet cat 

©2014 First Grade Schoolhouse

Activity Recognition using Google API

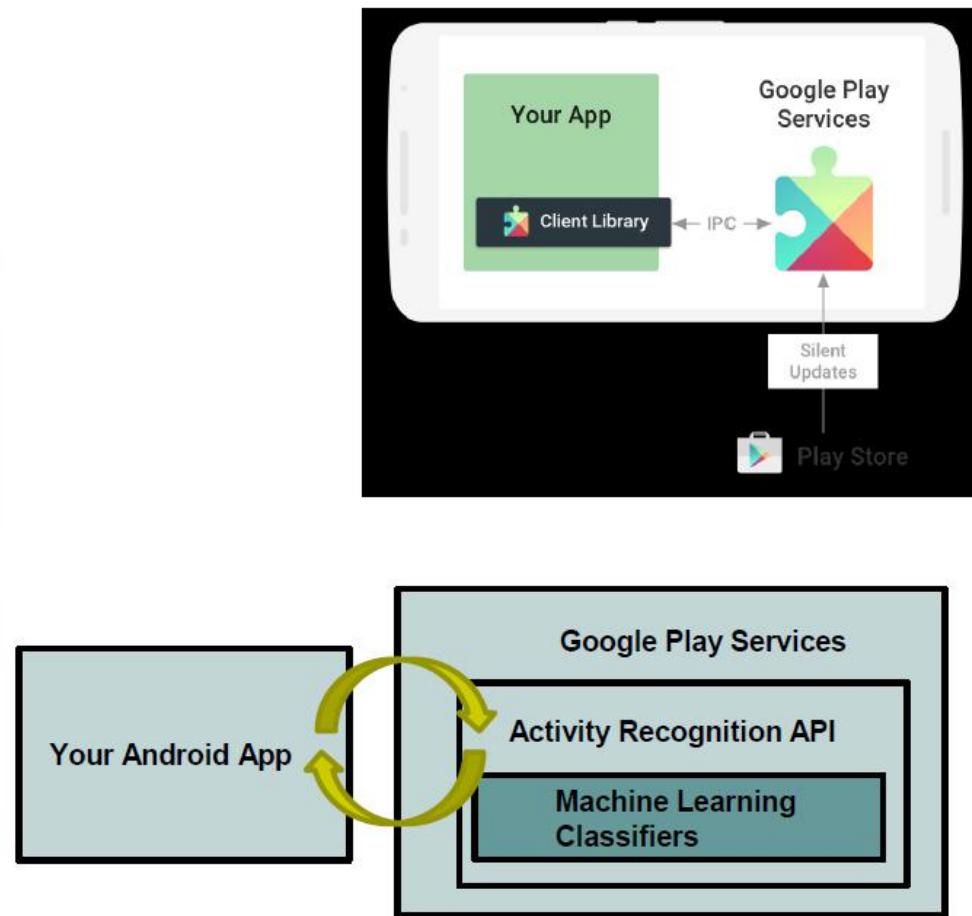
- We can adapt the mobile behaviors based on the user's behaviors
 - E.g., If the user is driving, don't send notifications

Google Activity Recognition API

- API to detect smartphone user's current activity
- Programmable, can be used by your Android app
- Currently detects 8 states:
 - In vehicle
 - On Bicycle
 - On Foot
 - Running
 - Walking
 - Still
 - Tilting
 - Unknown

Google Activity Recognition API

- Deployed as part of Google Play Services



Activity Recognition API

- Understanding what users are doing in the physical world allows your app to be smarter about how to interact with them.
 - For example, an app can start tracking a user's heartbeat when she starts running,
 - another app can switch to *car mode* when it detects that the user has started driving.
- The Activity Recognition API is built on top of the sensors available in a device.
 - Device sensors provide insights into what users are currently doing.

- The Activity Recognition API automatically detects activities by periodically reading short bursts of sensor data and processing them using machine learning models.
- To optimize resources, the API may stop activity reporting if the device has been still for a while, and uses low power sensors to resume reporting when it detects movement.

Features of Google API

- Receive information about activities using minimal resources
 - Get notified when your user starts or ends a particular activity
 - Perform an action when your app receives activity information
 - Receive detected activities that include a confidence grade

Get notified when your user starts or ends a particular activity

- For example, a mileage tracking app could start tracking miles when a user starts driving, or a messaging app could mute all conversations until the user stops driving.
- The Activity Recognition Transition API detects changes in the user's activity.
- The app subscribes to a transition in activities
 - The API notifies your app only when needed.
- No need to implement complex heuristics to detect when an activity starts or ends.

Perform an action when your app receives activity information

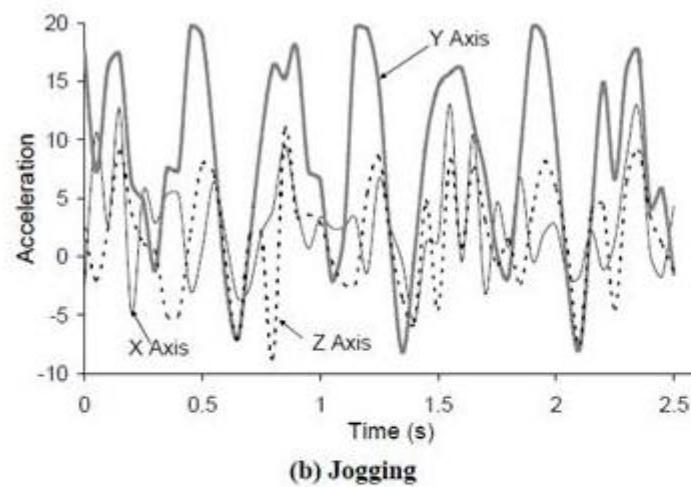
- The Activity Recognition API delivers its results to a callback, which is usually implemented as an IntentService
- The results are delivered at intervals that you specify,
 - or your app can use the results requested by other clients without consuming additional power itself.
- You can tell the API how to deliver results by using a PendingIntent
 - It removes the need to have a service constantly running in the background for activity detection purposes.
 - The app receives the corresponding Intents from the API, extracts the detected activities, and decides if it should take an action.

Receive detected activities that include a confidence grade

- The Activity Recognition API does the heavy lifting by processing the signals from the device to identify the current activities.
- Your app receives a list of detected activities, each of which includes confidence and type properties.
 - The confidence property indicates the likelihood that the user is performing the activity represented in the result.
 - The type property represents the detected activity of the device relative to entities in the physical world, for example, the device is on a bicycle or the device is on a user who is running.

Activity Recognition System Design

Activity recognition on Android



New accelerometer
Sample in real time

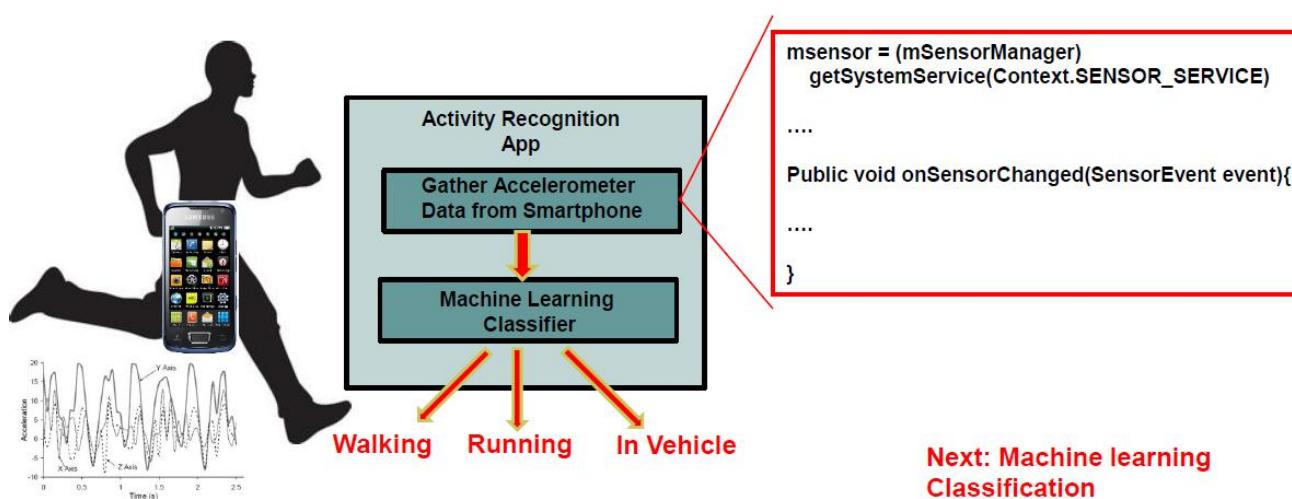


Classifier in
Android app

Activity
(e.g. Jogging)

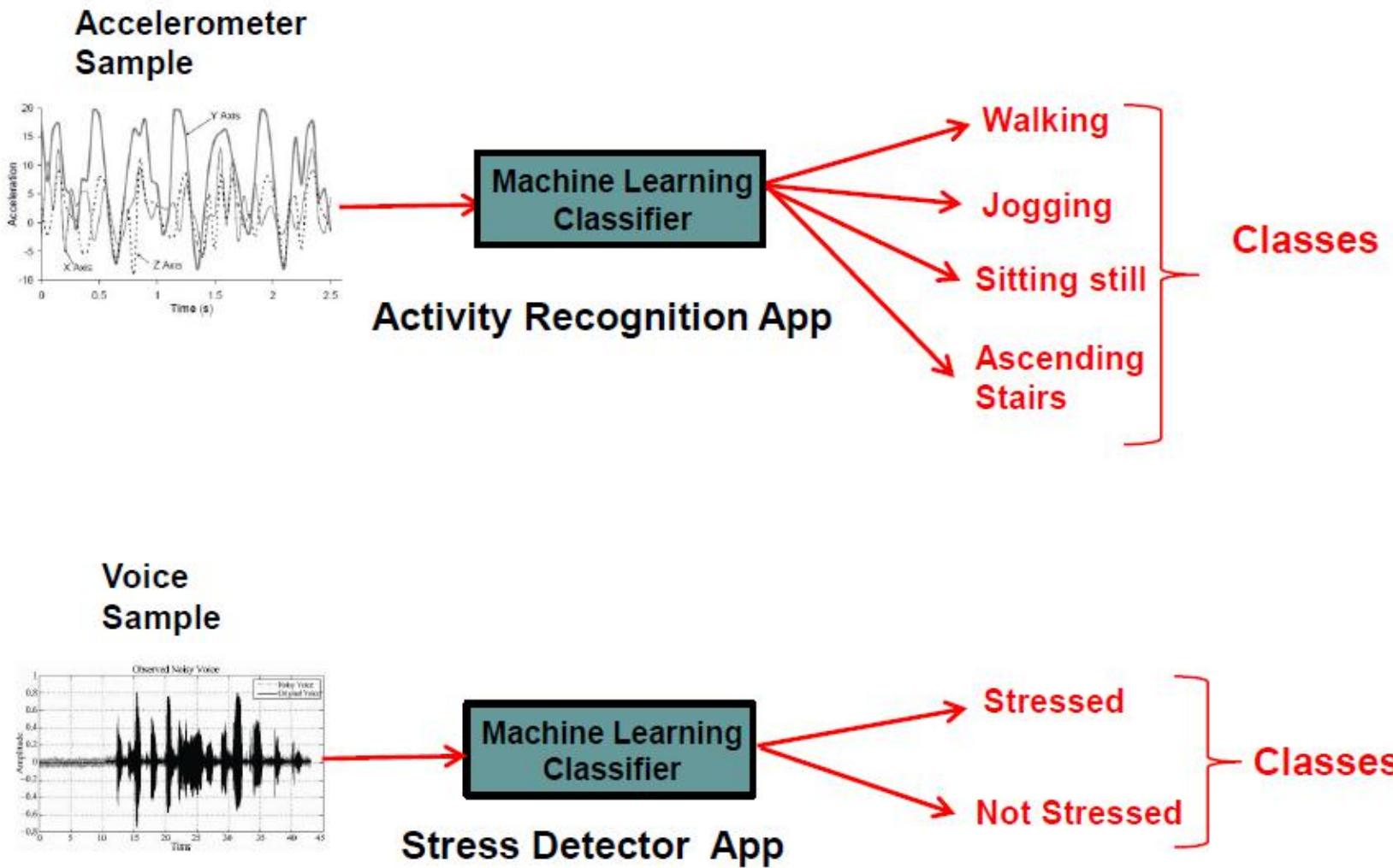
Activity Recognition (AR) Android App

- As user performs an activity, AR app on user's smartphone
 - Gathers accelerometer data
 - Uses **machine learning classifier** to determine what activity (running, jumping, etc) accelerometer pattern corresponds to
- **Classifier:** Machine learning algorithm that guesses what activity **class**(or type) accelerometer sample corresponds to

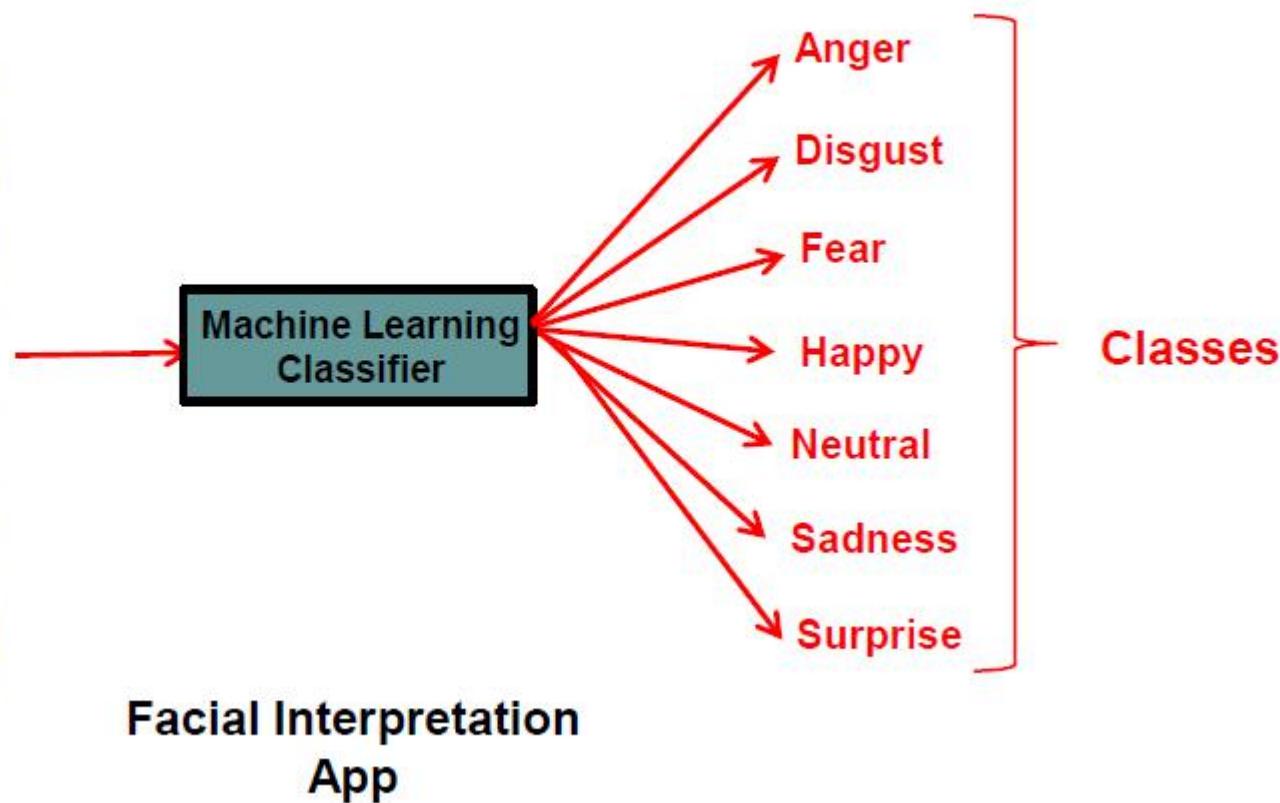
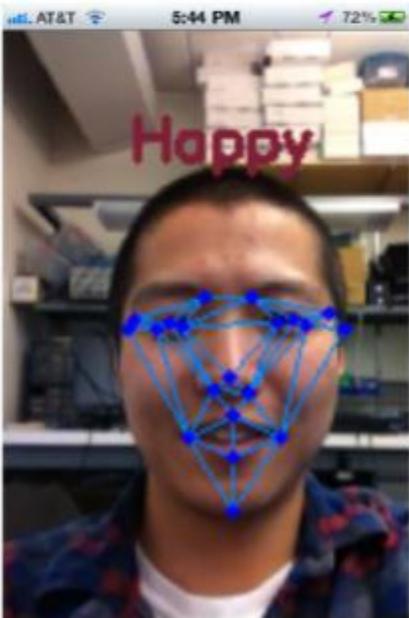


Classification

- Classification?
determine which **class** a sample
(e.g. snippet of accelerometer data) belongs to.
Examples:



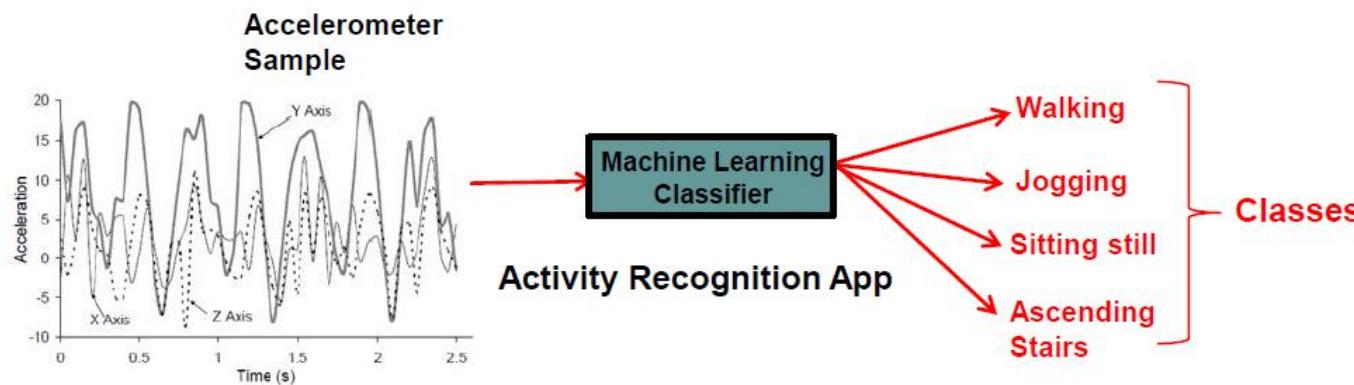
**Image showing
Facial Expression**



Classifier

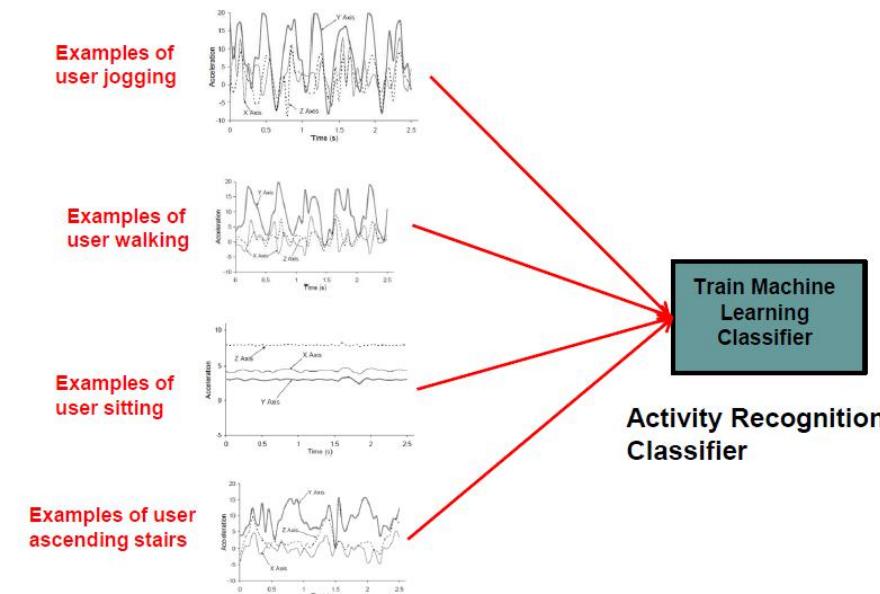
- Analyzes new sample, guesses corresponding class
- Intuitively, can think of classifier as set of rules for classification
- Example rules for classifvina accelerometer signal in Activity Recognition

```
If ((Accelerometer peak value > 12 m/s)
    and (Accelerometer average value < 6 m/s)) {
    Activity = "Jogging";
}
```



Training a Classifier

- Created using example-based approach (called training)
- **Training a classifier:** Given examples of each class => generate rules to categorize new samples
- **E.g:** Analyze 30+ Examples (from 30 subjects) of accelerometer signal for each activity type (walking, jogging, sitting, ascending stairs) => generate rules (classifier) to classify future activities



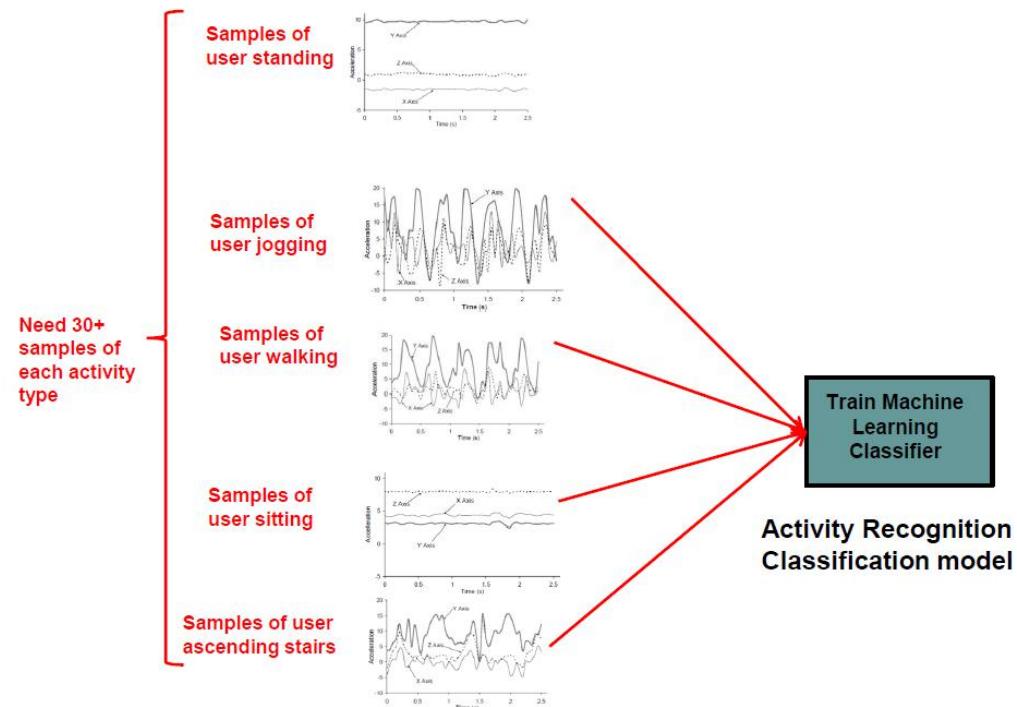
Training a Classifier: Steps

Steps for Training a Classifier

1. Gather data samples + label them
2. Import accelerometer samples into classification library (e.g. python, MATLAB)
3. Pre-processing (segmentation, smoothing, etc)
4. Extract features
5. Train classifier
6. Export classification model as JAR file
7. Import into Android app

Step 1: Gather Sample data + Label them

- Need many samples of accelerometer data corresponding to each activity type (jogging, walking, sitting, ascending stairs, etc)



Step 1: Gather Sample data + Label them

- Conduct a study to gather sample accelerometer data for each activity class
 - Recruit 30+ subjects
 - Run program that gathers accelerometer sensor data on subject's phone
 - Each subject:
 - Perform each activity (walking, jogging, sitting, etc)
 - Collect accelerometer data while they perform each activity (walking, jogging, sitting, etc)
 - Label data. i.e. tag each accelerometer sample with the corresponding activity
- Now have 30+ examples of each activity

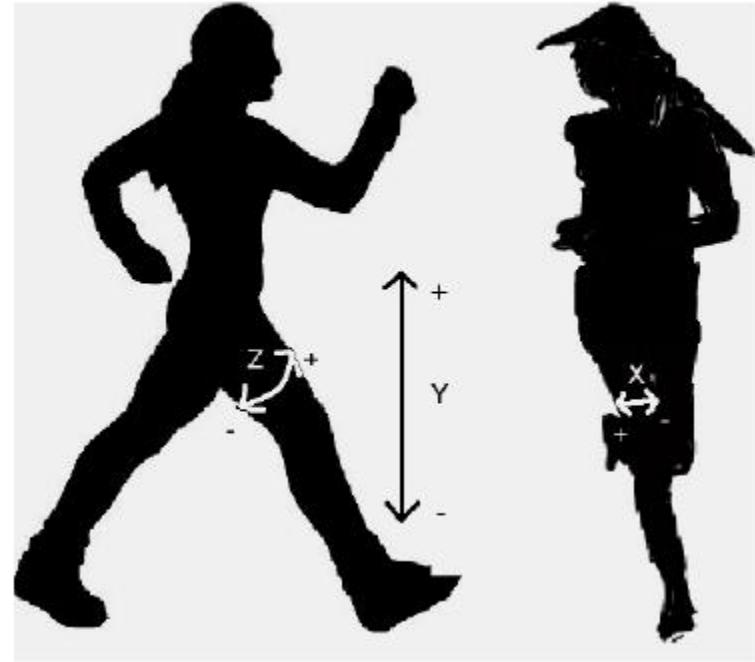
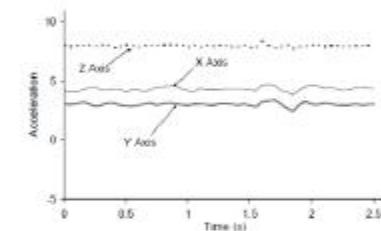
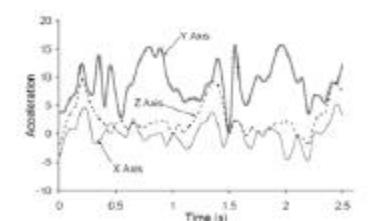


Figure 1: Axes of Motion Relative to User

30+
Samples of
user sitting



30+ Samples of
user ascending
stairs



Step 1: Gather Sample data + Label them

Program to Gather Accelerometer Data

- **Option 1:** Can write sensor program app that gathers accelerometer data while user is doing each of 6 activities (1 at a time)

```
msensor = (mSensorManager)
getSystemService(Context.SENSOR_SERVICE)

...
Public void onSensorChanged(SensorEvent event){

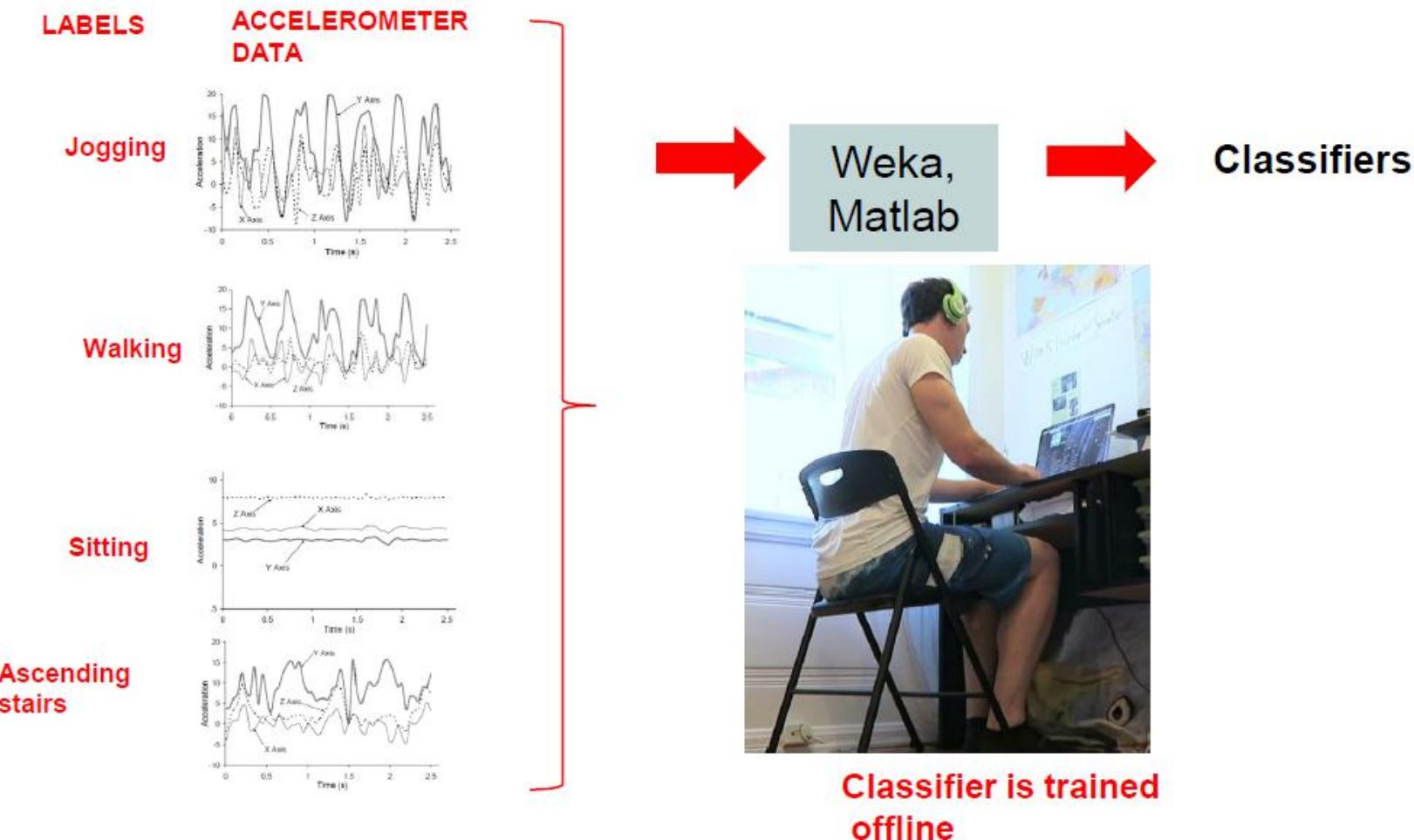
...
}
```

Step 1: Gather Sample data + Label them

Program to Gather Accelerometer Data

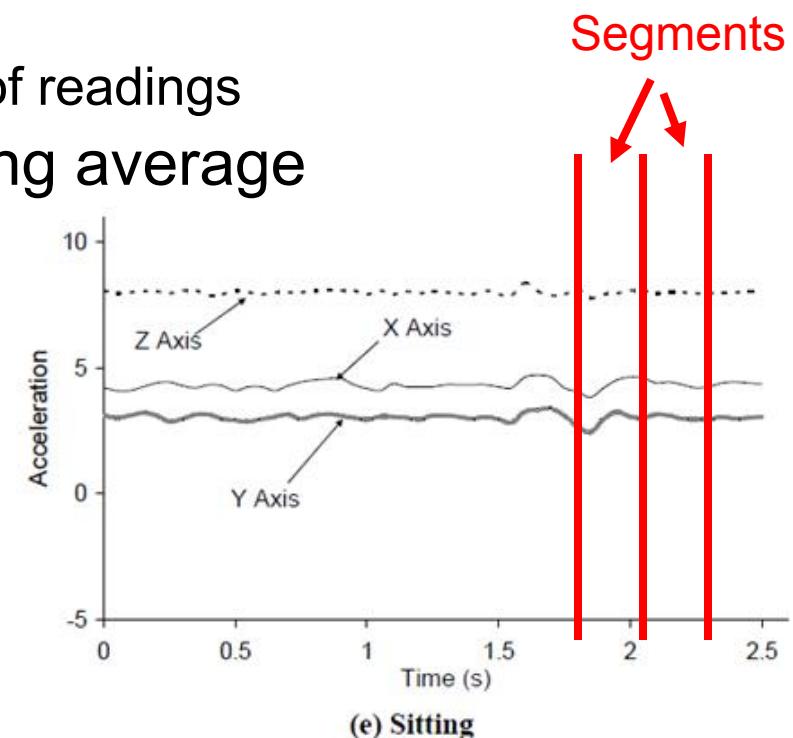
- **Option 2:** Use 3rdparty app to gather accelerometer
 - 2 popular ones: **Funf** and **AndroSensor**
 - Just download app,
 - Select sensors to log (e.g. accelerometer)
 - Continuously gathers sensor data in background

Step 2: Import accelerometer samples into classification library (e.g. Python, MATLAB)



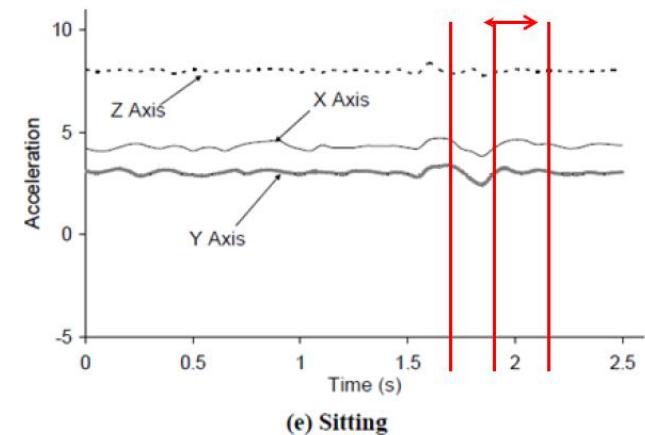
Step 3: Pre-processing (segmentation, smoothing, etc) Segment Data (Windows)

- Pre-processing data may include segmentation, smoothing, etc
 - **Segment:** Divide data into smaller chunks. E.g. divide 60 seconds of raw time-series data into 5 second chunks
 - Note: 5 seconds of accelerometer data could be 100s of readings
 - **Smoothing:** Replace groups of values with moving average



Step 4: Compute (Extract) Features

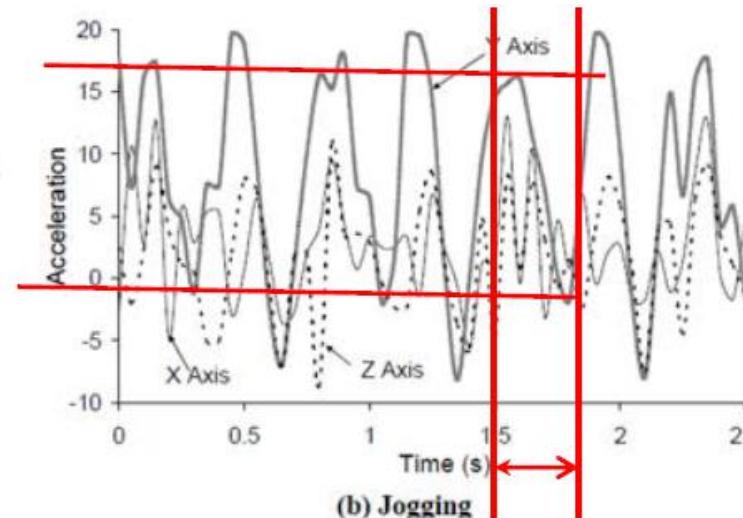
- For each 5-second segment (batch of accelerometer values) compute features
- **Features:** Formulas computed to quantify attributes of accelerometer data, captures accelerometer characteristics
- **Examples:** min-max of values within each segment, largest magnitude, standard deviation



Step 4: Compute (Extract) Features

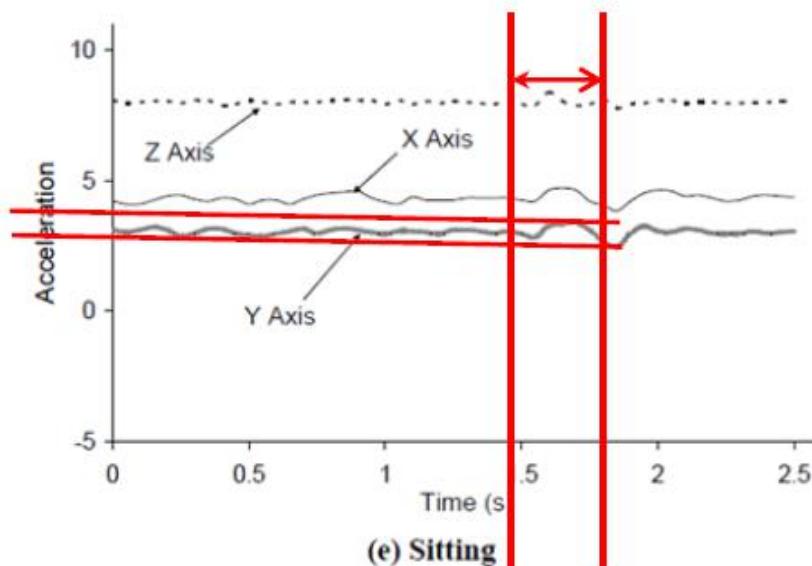
- **Important:** Ideally, values of features different for, distinguish each activity type (class)
- E.g: Min-max range feature

Large min-max
for jogging



(b) Jogging

Small min-max
for sitting



(e) Sitting

Step 4: Compute (Extract) Features

Calculate
many
different
features

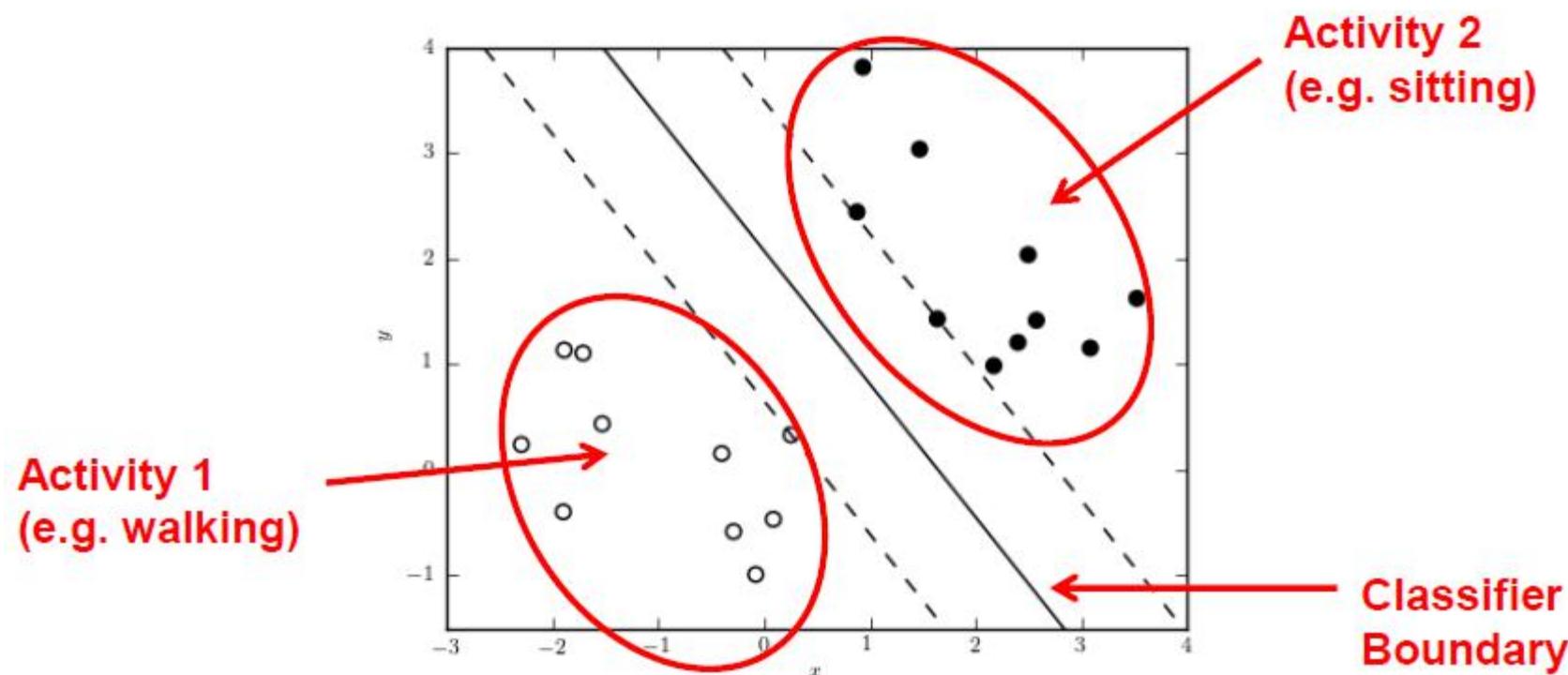
- Average[3]: Average acceleration (for each axis)
- Standard Deviation[3]: Standard deviation (for each axis)
- Average Absolute Difference[3]: Average absolute difference between the value of each of the 200 readings within the ED and the mean value over those 200 values (for each axis)
- Average Resultant Acceleration[1]: Average of the square roots of the sum of the values of each axis squared $\sqrt{x_i^2 + y_i^2 + z_i^2}$ over the ED
- Time Between Peaks[3]: Time in milliseconds between peaks in the sinusoidal waves associated with most activities (for each axis)
- Binned Distribution[30]: We determine the range of values for each axis (maximum – minimum), divide this range into 10 equal sized bins, and then record what fraction of the 200 values fell within each of the bins.

Step 5: Train classifier

- Features are just numbers (e.g. values of features for different subjects, activities)
- Different values for different activities
- **Training classifier:**figures out feature values corresponding to each activity
- Python, MATLAB already programmed with different classification algorithms (SVM, Naïve Bayes, Random Forest, J48, logistic regression, SMO, etc)
- Try different ones, compare accuracy

Step 5: Train classifier

- SVM example



Step 6: Export Classification model as JAR file

Step 7: Import into Android app

- Export classification model (most accurate classifier type + data threshold values) as Java JAR file
- Import JAR file into Android app
- In app write Android code to
- Gather accelerometer data, segment, extract feature, classify using classifier in JAR file
- Classifies new accelerometer patterns while user is performing activity => Guess (infer) what activity

CSE 162 Mobile Computing

Lecture 12: Localization

Hua Huang

Department of Computer Science and Engineering

University of California, Merced

An introduction to localization

What is localization?

- Get the location of a mobile device
 - Some devices, e.g., cell phones, are a proxy of a person's location
- Get the location of wireless signal source
 - Wireless emitter
- Used to help derive the context and activity information
 - Location based services
 - Privacy problems

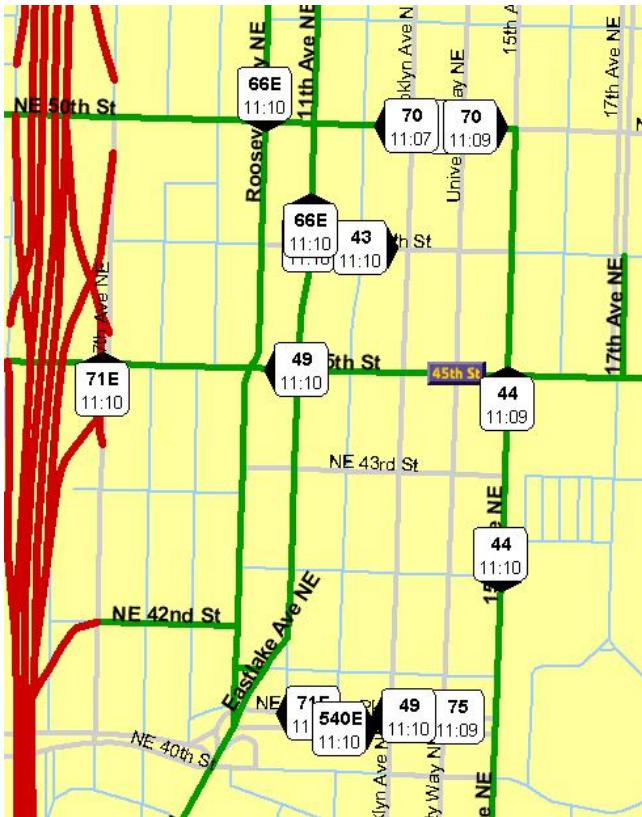
Localization

- Well studied topic (3,000+ PhD theses??)
- Application dependent
- Research areas
 - Technology
 - Algorithms and data analysis
 - Evaluation

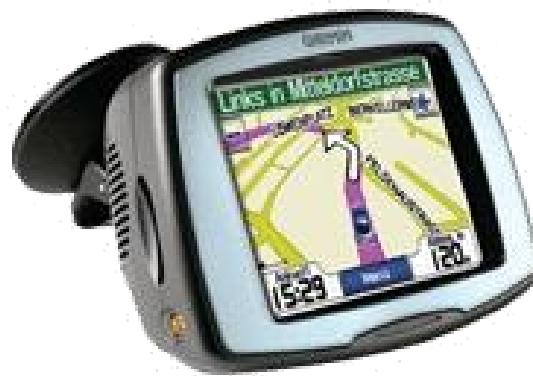
Representing Location Information

- Absolute
 - Geographic coordinates (Lat: 33.98333, Long: -86.22444)
- Relative
 - 1 block north of the main building
- Symbolic
 - Home, road, bedroom, work

Some outdoor applications



E-911



Car Navigation



Child tracking

Some indoor applications



Elder care



Indoor navigation:
mall, airport, museum, etc



Contact Tracing

No one size fits all!

- Accurate
 - Low-cost
 - Easy-to-deploy
 - Ubiquitous
-
- Application needs determine technology

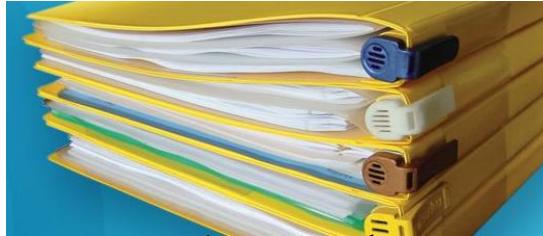
Lots of technologies!



GPS



WiFi Beacons



Ultrasound



Floor pressure



VHF Omni Ranging



Ad hoc signal strength



Laser range-finding



Stereo camera



Ultrasonic time of flight



Infrared proximity

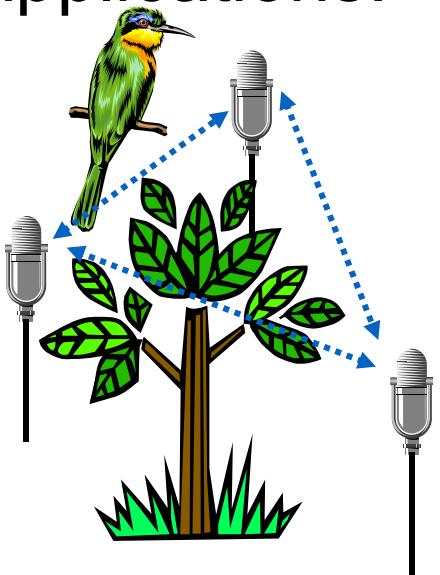


Array microphone

Localization Applications Design Principles

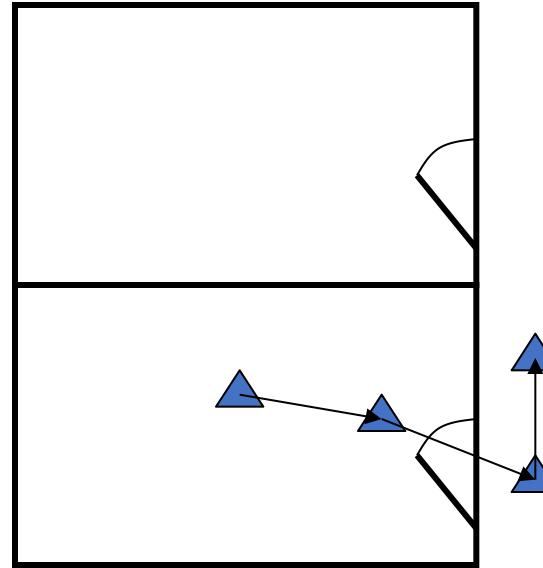
Variety of Applications

- Two applications:



habitat monitoring:

Where is the bird?
What kind of bird is it?



Asset tracking:

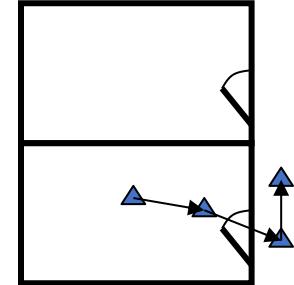
Where is the projector?
Why is it leaving the room?

Variety of Application Requirements

■ Very different requirements!

- Outdoor operation
 - Weather problems
- Bird is not tagged
- Birdcall is characteristic but not exactly known
- Accurate enough to photograph bird
- Infrastructure:
 - Several acoustic sensors, with known relative locations; coordination with imaging systems

- Indoor operation
 - Multipath problems
- Assets are tagged
- Signals from asset tags can be engineered
- Accurate enough to track through building
- Infrastructure:
 - Room-granularity tag identification and localization; coordination with security infrastructure



Multidimensional Requirement Space

- Granularity & Scale
- Accuracy & Precision
- Relative vs. Absolute Positioning
- Dynamic vs. Static (Mobile vs. Fixed)
- Cost & Form Factor
- Infrastructure & Installation Cost
- Environmental Sensitivity
- Cooperative or Passive Target

Axes of Application Requirements

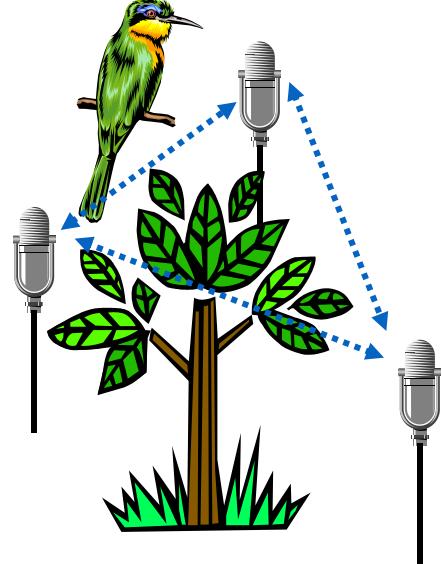
- Granularity and scale of measurements:
 - What is the smallest and largest measurable distance?
 - e.g. 50m (acoustics) vs. 25000km (GPS)
- Accuracy and precision:
 - How close is the answer to “ground truth” (accuracy)?
 - How consistent are the answers (precision)?
- Relation to established coordinate system:
 - GPS? Campus map? Building map?
- Dynamics:
 - Refresh rate? Motion estimation?

Axes of Application Requirements

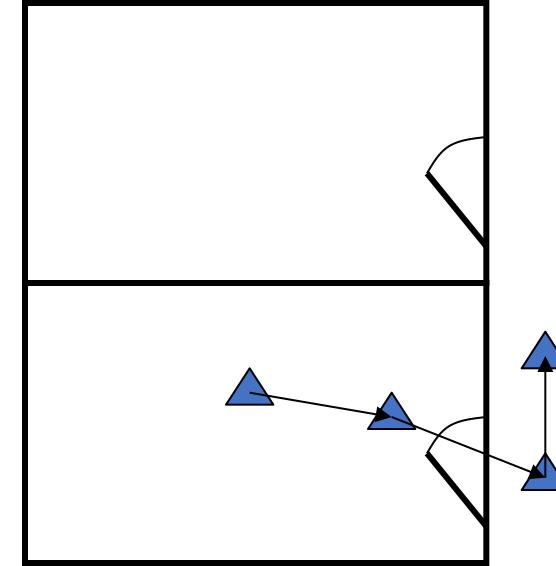
- Cost:
 - Node cost? Power? \$? Time?
 - Infrastructure cost? Installation cost?
- Form factor:
 - How big
 - Think about tracking tags on wild animals
- Communications Requirements:
 - Network topology: cluster head vs. local determination
 - What kind of coordination among nodes?
- Environment:
 - Indoor? Outdoor? On Mars?
- Is the target known? Is it cooperating?

Returning to our two Applications...

- Choice of mechanisms differs:



Passive habitat monitoring:
Minimize environ. interference
No two bird songs are the same



Asset tracking:
Controlled environment
We know exactly what tag is like

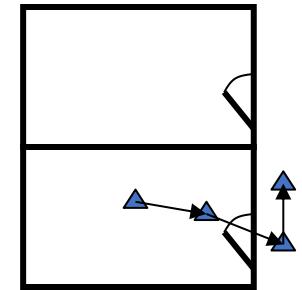
Variety of Localization Mechanisms

n Very different mechanisms indicated!

- Bird is not tagged
 - Passive detection of bird presence
- Birdcall is characteristic but not exactly known
- Bird does not have radio; Acoustic based ranging
- **Passive target localization**
 - Requires
 - Sophisticated detection
 - Coherent beamforming
 - Large data transfers



- Asset is tagged
 - Projector might know it had moved
- Signals from asset tag can be engineered
- Tag can use radio signal for ranging
- **Cooperative Localization**
 - Requires
 - Basic correlator
 - Simple triangulation
 - Minimal data transfers



Wireless Technologies for Localization

Name	Effective Range	Pros	Cons
GSM	35km	Long range	Very low accuracy
LTE	30km-100km		
Wi-Fi	50m-100m	Readily available; Medium range	Low accuracy
Ultra Wideband	70m	High accuracy	High cost
Bluetooth	10m	Readily Available; Medium accuracy	Short range
Ultrasound	6-9m	High accuracy	High cost, not scalable
RFID & IR	1m	Moderate to high accuracy	Short range, Line-Of-Sight (LOS)
NFC	<4cm	High accuracy	Very short range

Localization Algorithms

Algorithms to obtain locations

- Range-based algorithms
- Range-free algorithms
- Fingerprinting

Range Based Algorithms

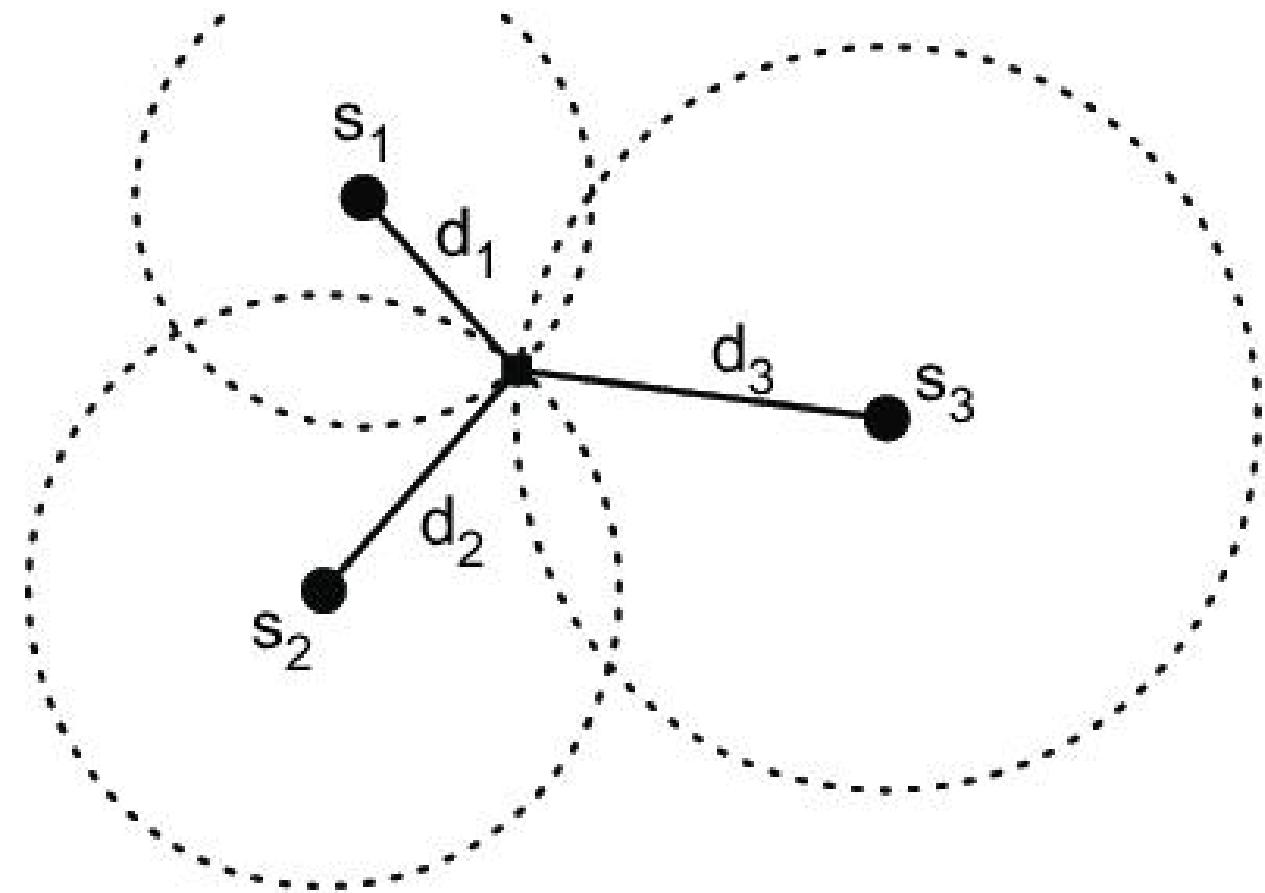
- Rely on the distance (angle) measurement between nodes to estimate the target location
- Approaches
 - Proximity
 - Lateration
 - Hyperbolic Lateration
 - Angulation
- Distance estimates
 - Time of Flight
 - Signal Strength Attenuation

Approach: Proximity

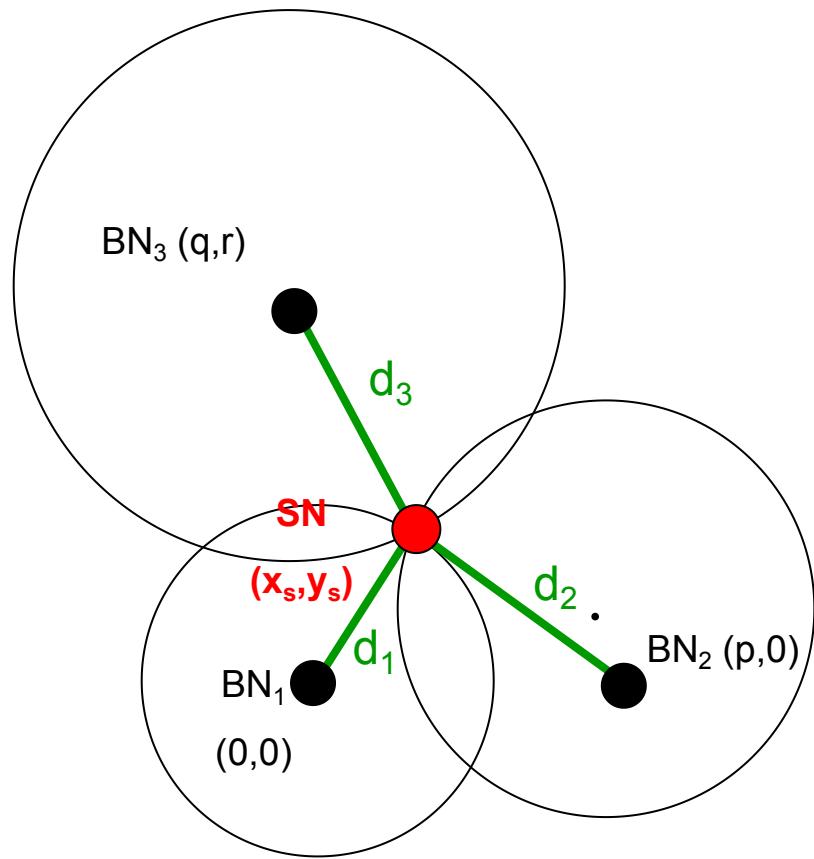
- Simplest positioning technique
- Closeness to a reference point
- Based on loudness, physical contact, etc
- Examples
 - RFID Door Access Control System

Approach: Lateration

- Method: Measure distance between device and reference points
 - s_1, s_2, s_3 locations are known
 - Measure the distances d_1, d_2 , and d_3
 - Search for the most-likely location given the distances
- 3 reference points needed for 2D and 4 for 3D



- Assuming accurate distance measurements between nodes, apply the trilateration technique to determine the SN coordinates (unknown) using the three BNs coordinates and the r distances.
- Without loss of generality, let the coordinates of the references be $(0,0), (0,p), (q,r)$



By definition:

$$d_1^2 = x^2 + y^2$$

$$d_2^2 = (x - p)^2 + y^2$$

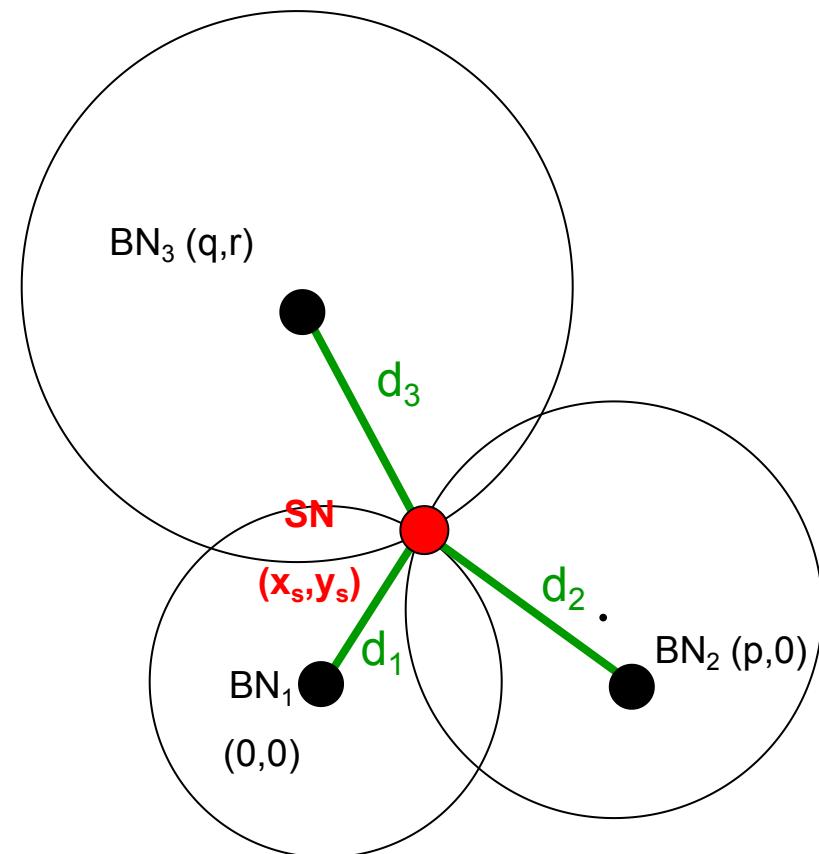
$$d_3^2 = (x - q)^2 + (y - r)^2$$

- By subtracting the second equation from the first, x is attained.

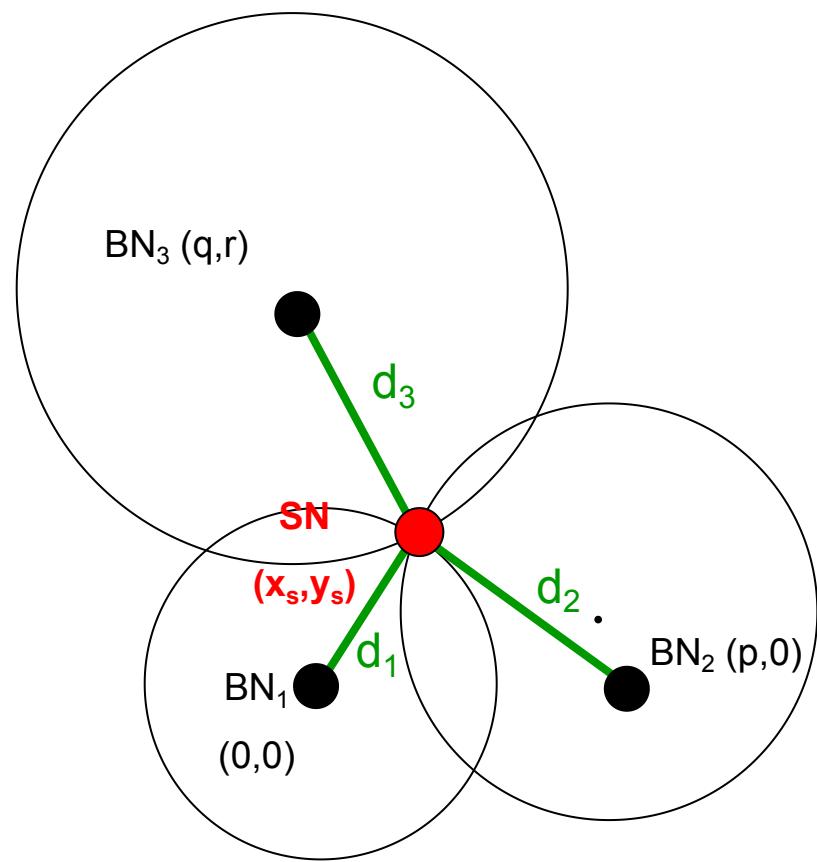
$$x = \frac{d_1^2 - d_2^2 + p^2}{2p}$$

- Substituting this value back into the first equation will result in values for y .

$$y = \pm \sqrt{d_1^2 - \left(\frac{d_1^2 - d_2^2 + p^2}{2p} \right)^2}$$

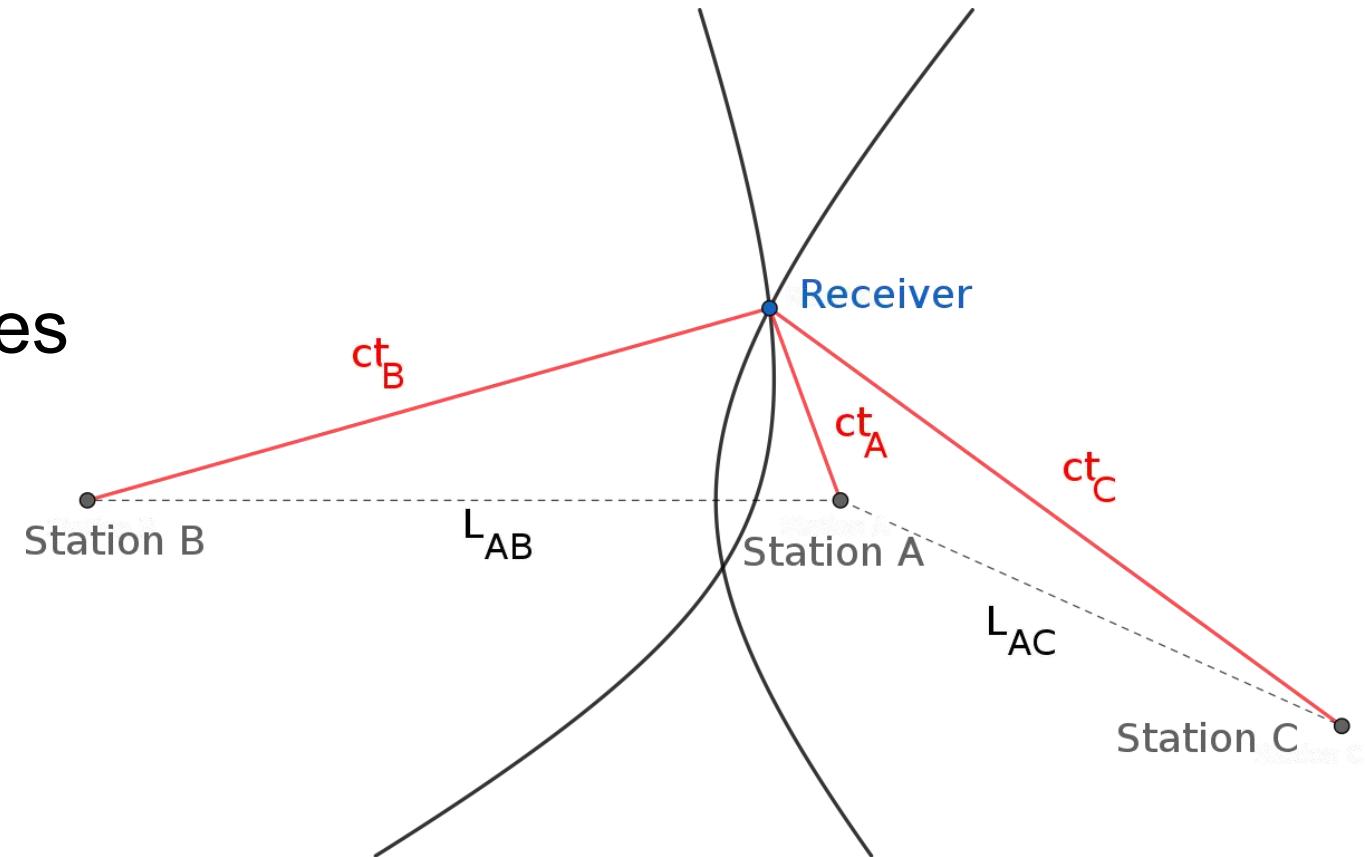


To determine which value of y correctly describes the point in question, x and y are substituted into the third equation.



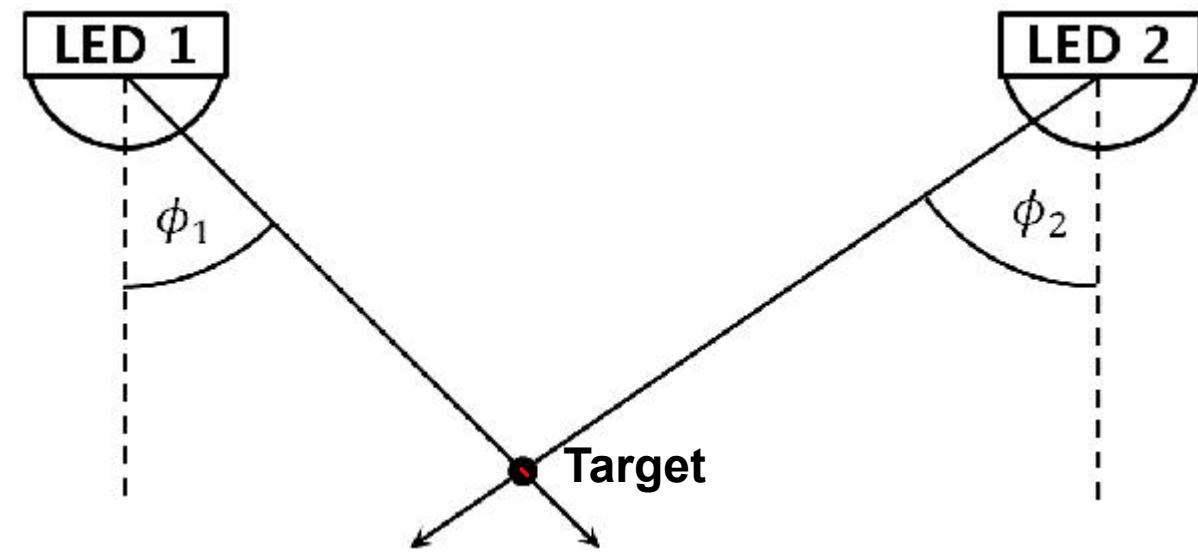
Approach: Hyperbolic Lateration

- Known: Station A, B, C locations
- Unknown: The signal traveling times to the receiver: ct_A, ct_B, ct_C
- Known: $ct_A - ct_B, ct_A - ct_C, ct_B - ct_C$
- Each time difference locates the receiver on a branch of a hyperbola



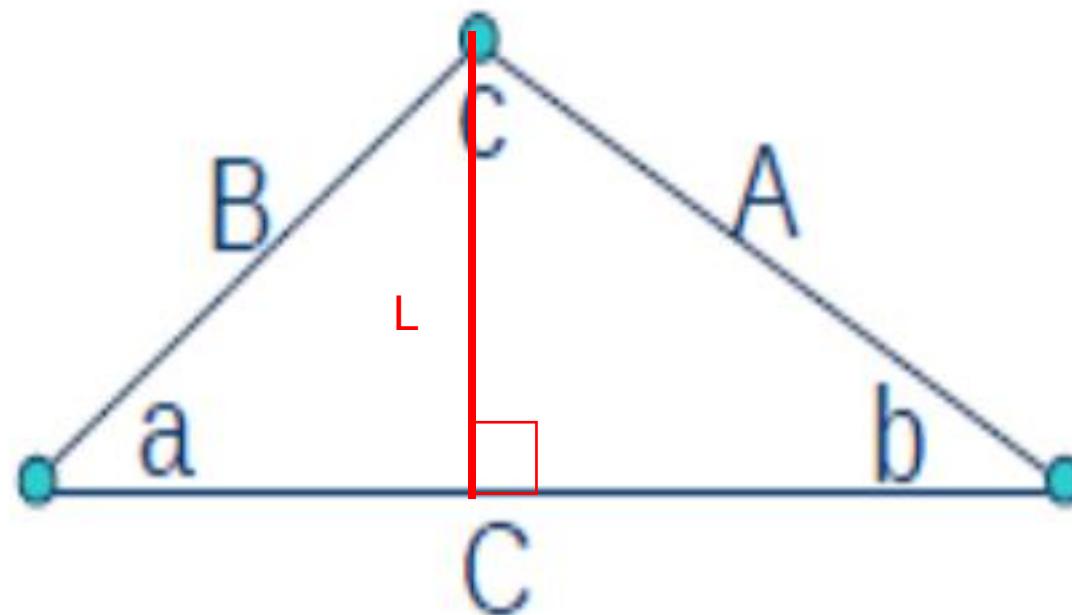
Approach: Triangulation

- Use the location of the signal sources and the relative angles
- Uniquely identify the location of the target
- How to do it mathematically?



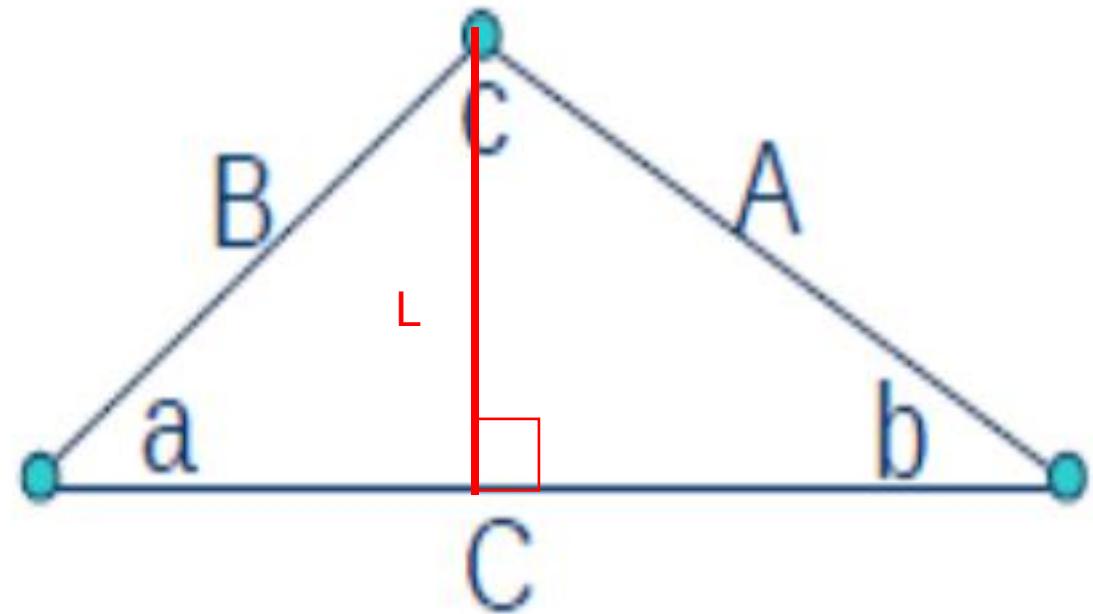
Question

- Angles a and b are known. Distance C is known.
- What is L ?



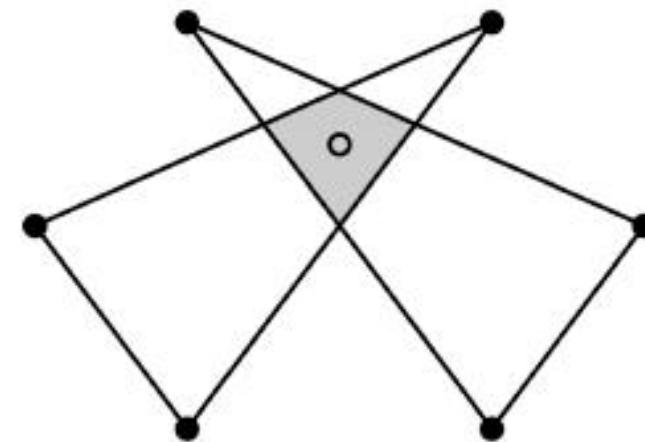
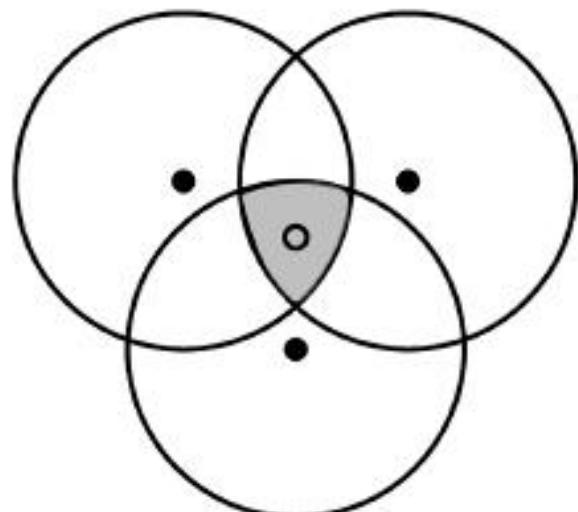
Question

- $L/\tan(a) + L/\tan(b) = C$
- $L = C/(1/\tan(a) + 1/\tan(b))$



Range Free Algorithms

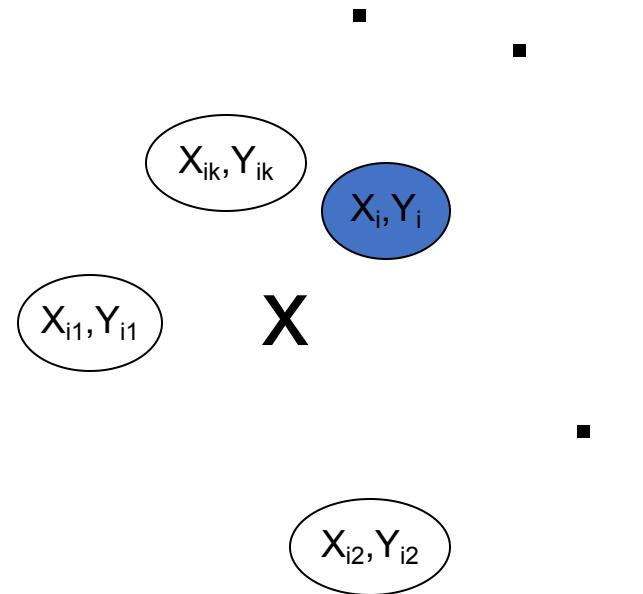
- Rely on target object's proximity to anchors with known positions
 - Neighborhood: single/multiple closest BS
 - Area estimation:



Range-free Localization Techniques

Centroid Algorithm

- Nodes localize themselves to the centroid of their proximate reference points



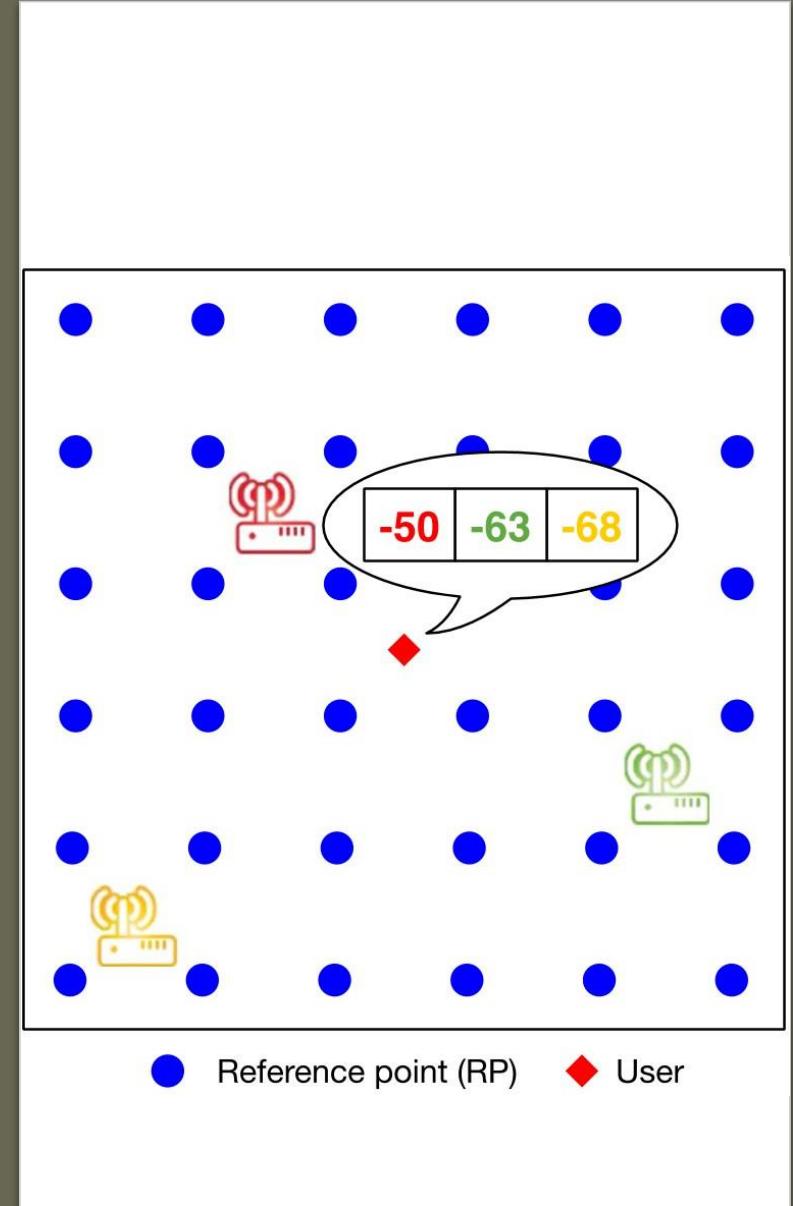
$$(X_{est}, Y_{est}) = \left(\frac{X_{i1} + \dots + X_{ik}}{k}, \frac{Y_{i1} + \dots + Y_{ik}}{k} \right)$$

Fingerprinting

- Mapping solution
- Address problems with multipath
- Better than modeling complex RF propagation pattern

Fingerprinting: Steps

- Step1
 - Use war-driving to build up location fingerprints (i.e. location coordinates + respective RSSI from nearby base stations)
- Step2
 - Match online measurements with the closest *a priori* location fingerprints



Fingerprinting: Example

SSID (Name)	BSSID (MAC address)	Signal Strength (RSSI)
linksys	00:0F:66:2A:61:00	18
starbucks	00:0F:C8:00:15:13	15
newark wifi	00:06:25:98:7A:0C	23

Fingerprinting: Pros and Cons

- Pros
 - Physical model not required
- Cons
 - Requires a dense site survey
- Prerequisite:
 - Spatial differentiability
 - Temporal stability

Concluding Remarks

Pros and Cons

- Two main types of distributed localization algorithms:
 - Range-based
 - Estimating the coordinates based on the collected information of distances or angles among nodes
 - Merit: Relatively high accuracy
 - Drawback: Costly (Hardware, Power consumption)
 - Range-free
 - Estimating the coordinates based on the connectivity relations
 - Merit: Cost-effective
 - Drawback: Not as accurate (But: coarse accuracy is sufficient for some applications)

Hardware/Energy Cost vs Location Precision

Summary of Localization Algorithms

	Measurement Scheme	Accuracy	Special Requirement
Range-based	TOA	Moderate	Synchronization, dense beacons
	TDOA	High	Synchronization, LOS, dense beacons
	AOA	High	Directional antenna
	RSSI	Moderate	No
Range-free	Neighborhood	Low	No
	Area estimation	Moderate	Dense Beacons
Fingerprinting	RSSI	High	No

CSE 162 Mobile Computing

Lecture 13: Distance Estimation Technologies, GPS

Hua Huang

Department of Computer Science and Engineering
University of California, Merced

Distance Estimation Technologies

Distance Estimation

- Multiply the radio signal velocity and the travel time
 - Time of arrival (TOA)
 - Time difference of arrival (TDOA)
- Compute the attenuation of the emitted signal strength
 - RSSI
- Problem: Multipath fading

Distance Estimation: TOA

- Distance
 - Based on one signal's travelling time from target to measuring unit
 - $d = v_{\text{radio}} * t_{\text{radio}}$
- Requirement
 - Transmitters and receivers should be precisely synchronized
 - Timestamp must be labeled in the transmitting signal

Distance Estimation: TDOA

- The transmitter sends a radio and a sound
- Receiver measurements: receiving time t_1 and t_2
- Known parameter: v_r and v_s
- How to estimate the distance?

Distance Estimation: TDOA

Let the transmission time be denoted t_0 . We have

$$d = (t_1 - t_0) * v_r$$

$$d = (t_2 - t_0) * v_s$$

Therefore : $\frac{d}{v_s} - \frac{d}{v_r} = t_2 - t_1$

Finally we get: $d = (t_2 - t_1) * v_r * v_s / (v_r - v_s)$

Distance Estimation: RSSI

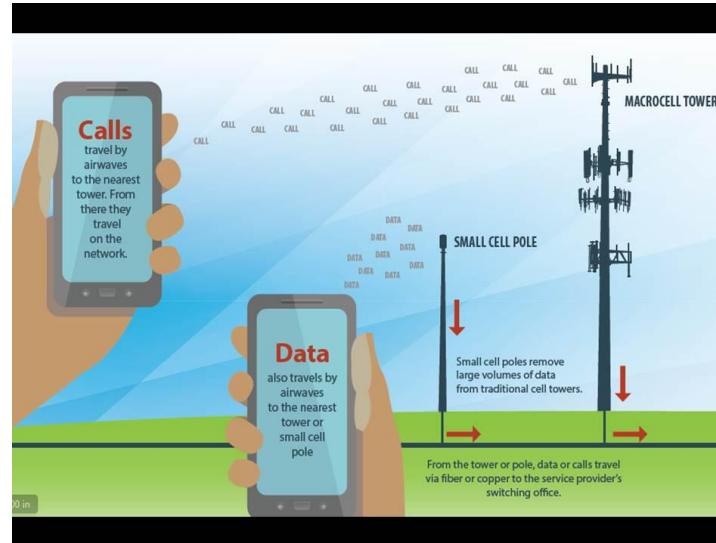
- Distance
 - Based on radio propagation model

$$P(d) = P(d_0) - \eta 10 \log \left(\frac{d}{d_0} \right) + X_\sigma$$

- Requirement
 - Path loss exponent η for a given environment is known

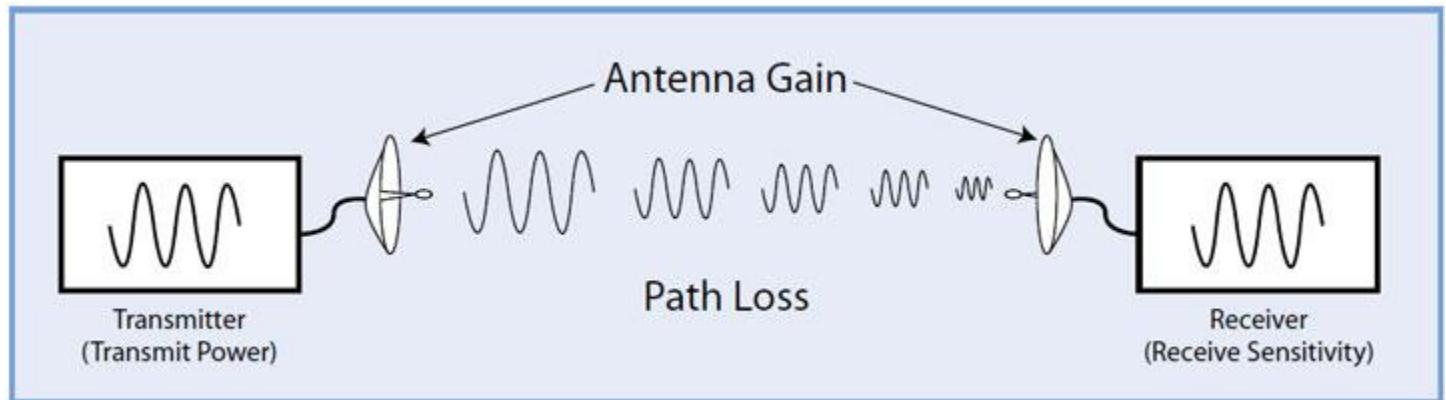
RSSI based distance estimation

- Signal strength attenuates as distance increases

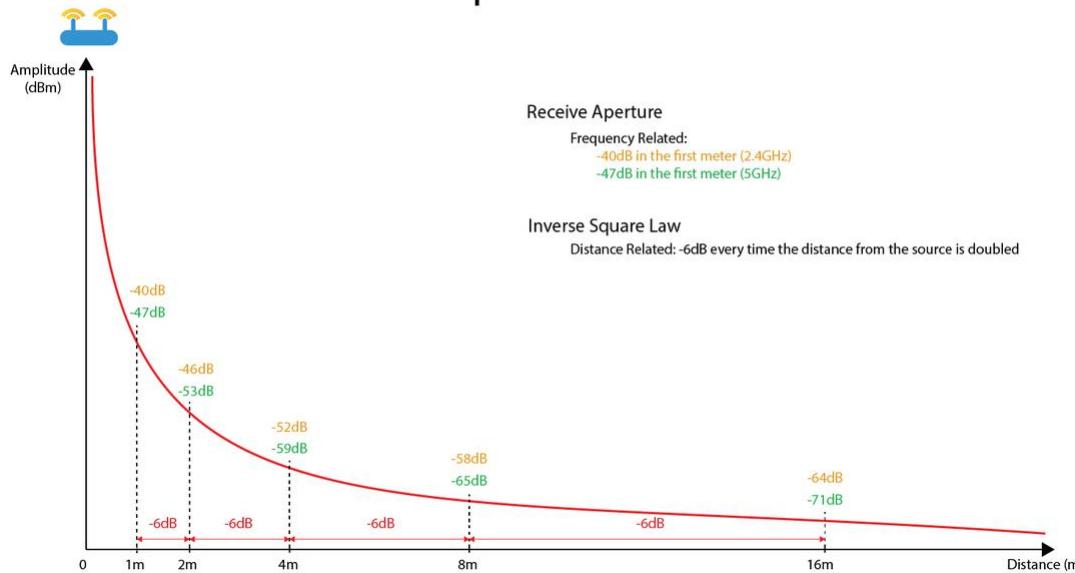


RSSI based distance estimation

- Theoretical model

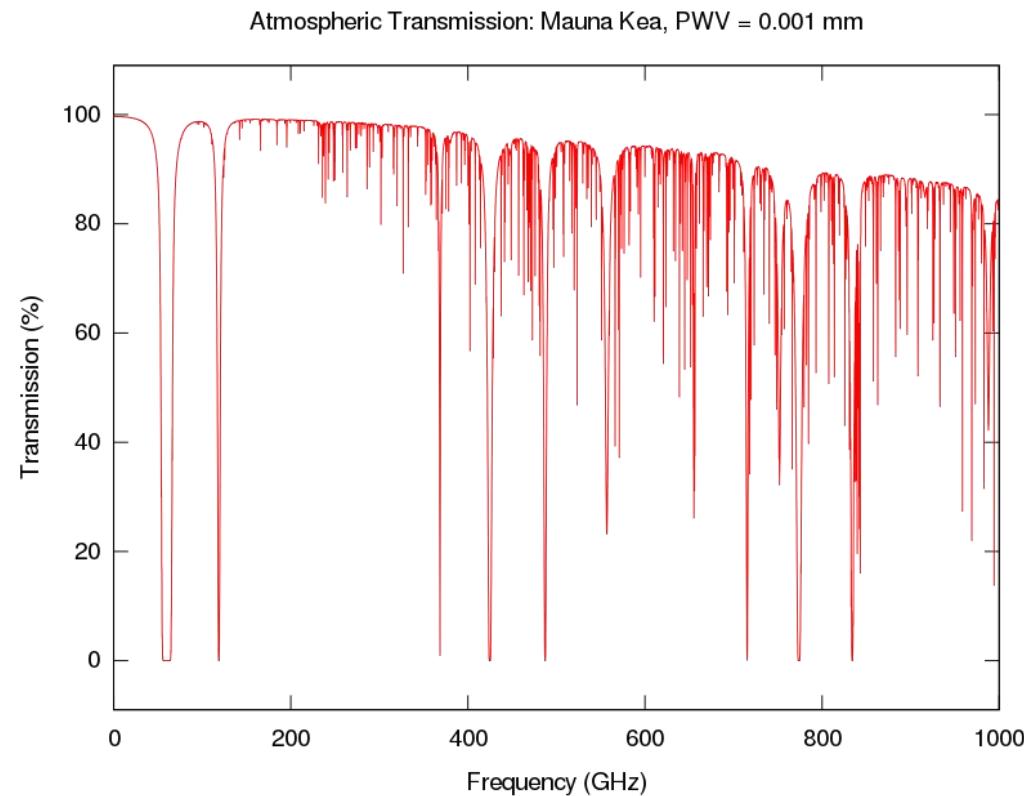


Free Space Path Loss

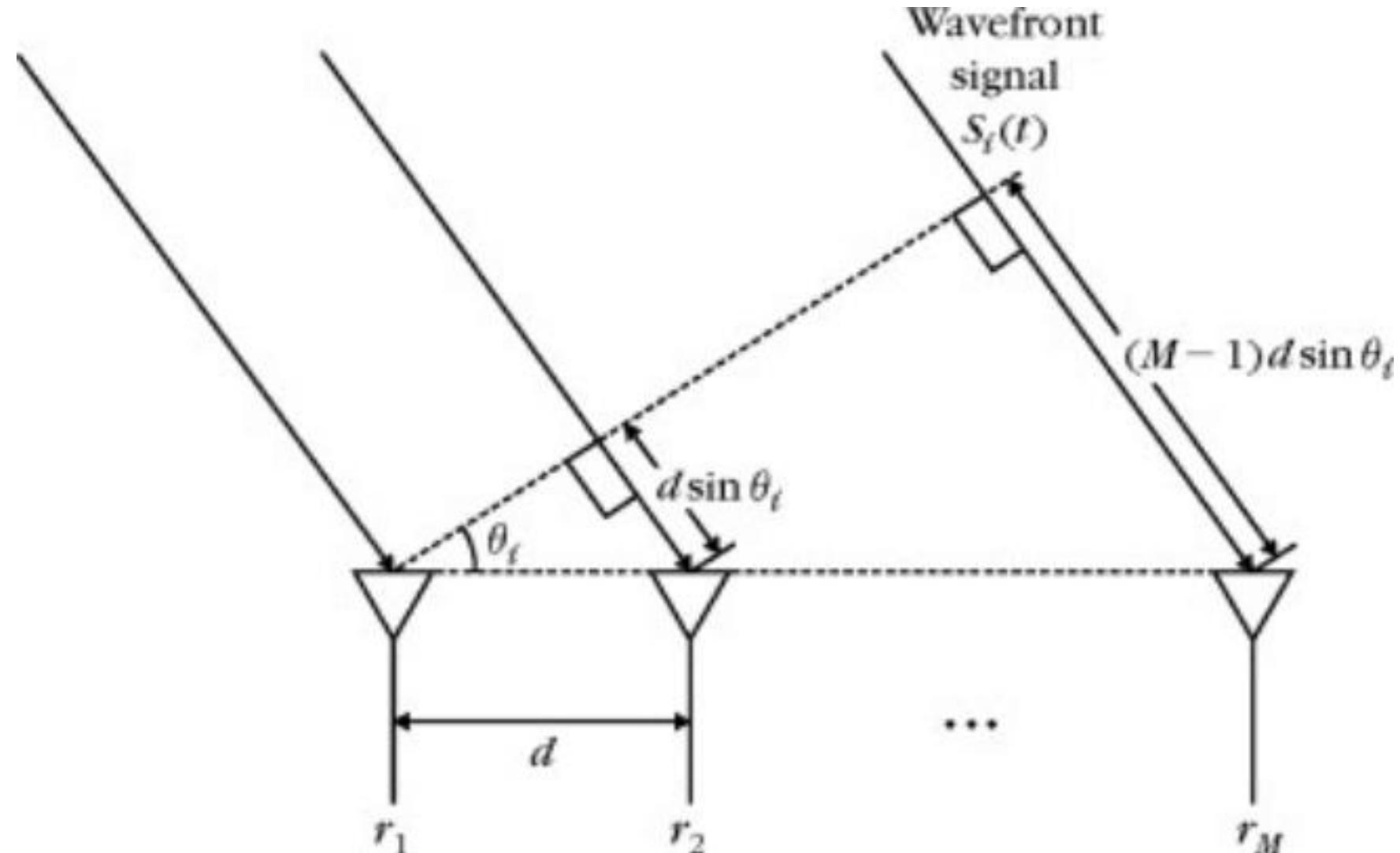


Limitation: Actual rssi is very noisy

- The RSSI and distance are correlated, but can significantly fluctuate



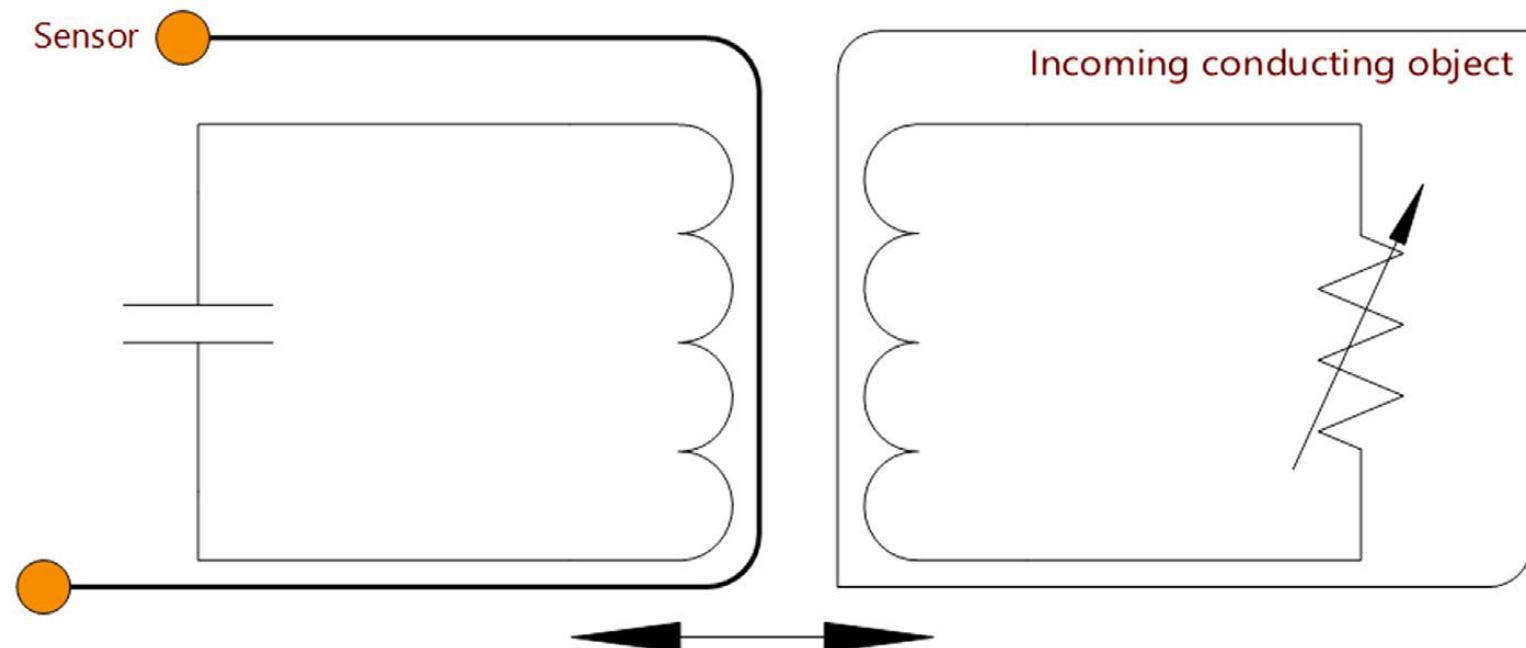
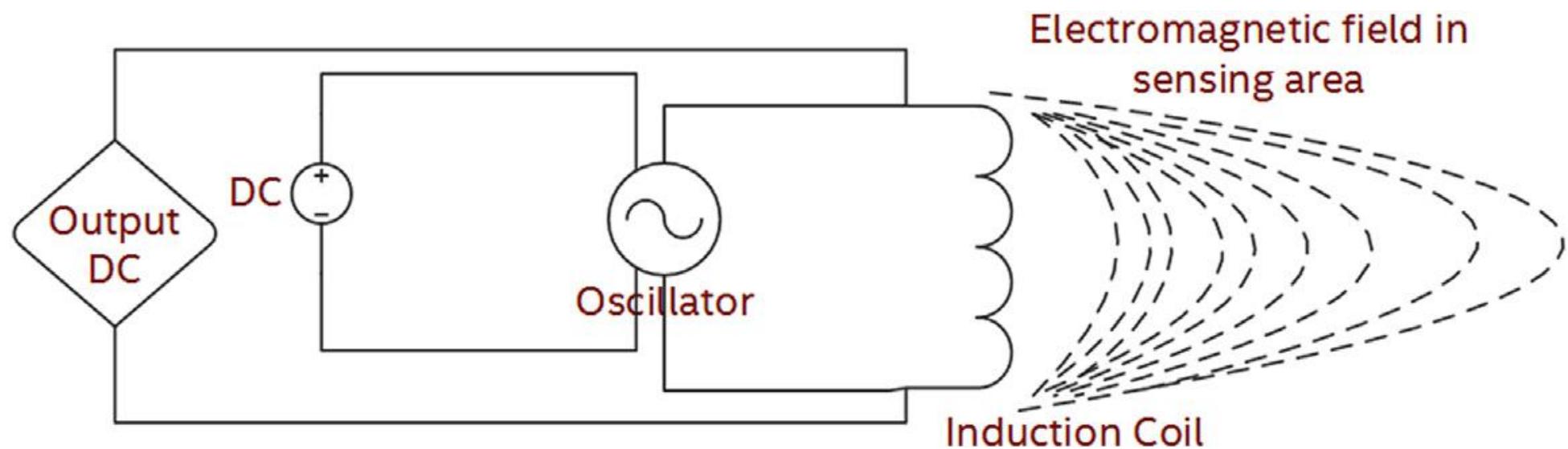
Estimating Angle of Arrival: beamforming



Distance Sensors

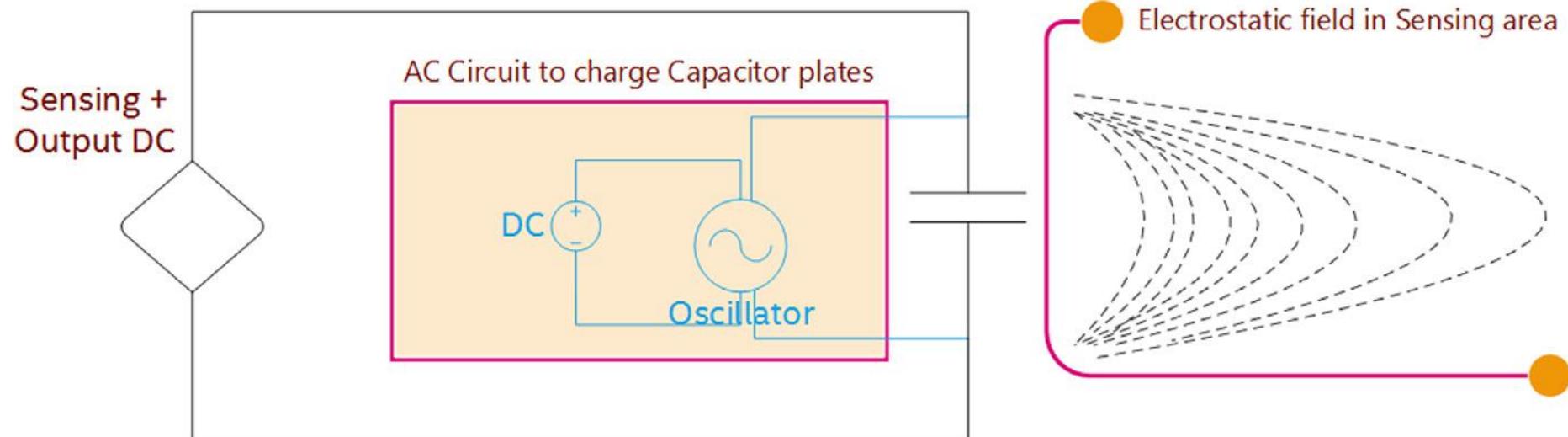
Inductive Sensor

- An inductive proximity sensor mainly consists of a coil, an electronic oscillator, a detection circuit, an output circuit, and an energy source to provide electrical stimulation.
- Working Principle:
 - oscillator generates a changing alternating current
 - the induction coil, it generates a changing electromagnetic field
 - metal object generates a change in impedance due to eddy currents, changing the detection circuit measurements



Capacitive Proximity Sensor

- Use electrostatic field, instead of magnetic field, for detection.
 - Thus, non metallic objects can be sensed.



Distance and Ranging: Infrared Sensors

- Contain an infrared emitter, and an infrared detector
- Works by emitting a certain amount of infrared light, and seeing how much it gets back
- Why infrared?
 - There are not many other infrared sources in everyday life that would interfere with this sensor
 - If visible light were used, light bulbs, computer screens, cellphone screens, etc, would all interfere with the depth reading



Distance and Ranging: Infrared Sensors

- Great at measuring shorter distances (2" –30")
- Where do you see these?
 - Touchless Switches (toilets, faucets, etc)
 - Roomba vacuums
 - Kinect
- Related: Passive Infrared (PIR) Sensors
 - No IR emitter, just detects ambient IR.
 - Detects some normal state (like a wall's IR emissions) and when something moves in front, it detects a change
 - Great for detecting motion (motion sensors for security systems)



Distance and Ranging: Speed detector

- e.g. police radar guns
 - Microwave radars use the Doppler effect (the return echo from a moving object will be frequency shifted).
 - The greater the target speed, the greater the frequency (Doppler) shift



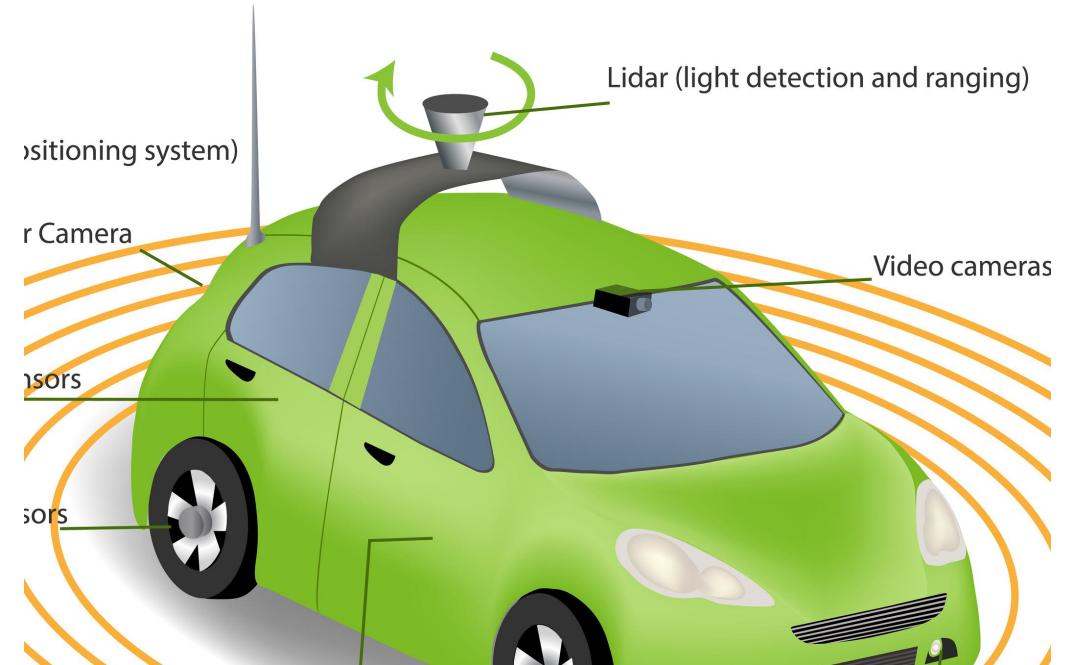
Ultrasonic Sensors

- Contain a high frequency speaker , and a microphone
- Works just like a sonar, emitting a sound, and listening for the echo to determine range
- Why is it called ultrasonic?
 - Very high frequency sound, it is barely at the edge of what humans can hear.
 - This is nice since it is not as annoying to use
- Pros: More accurate than IR sensors at slightly longer distance (typically up to several feet)
- Cons: Almost twice the price



Lidar

- LIDAR — Light Detection and Ranging
- How it works?
 - send pulses of light, and determine the difference in reflection time between consecutive pulses to determine speed
- Application:
 - automobile
 - Indoor mapping robot



CSE 162 Mobile Computing

Lecture 14: GPS

Hua Huang

Department of Computer Science and Engineering

University of California, Merced

The Global Positioning System (GPS)

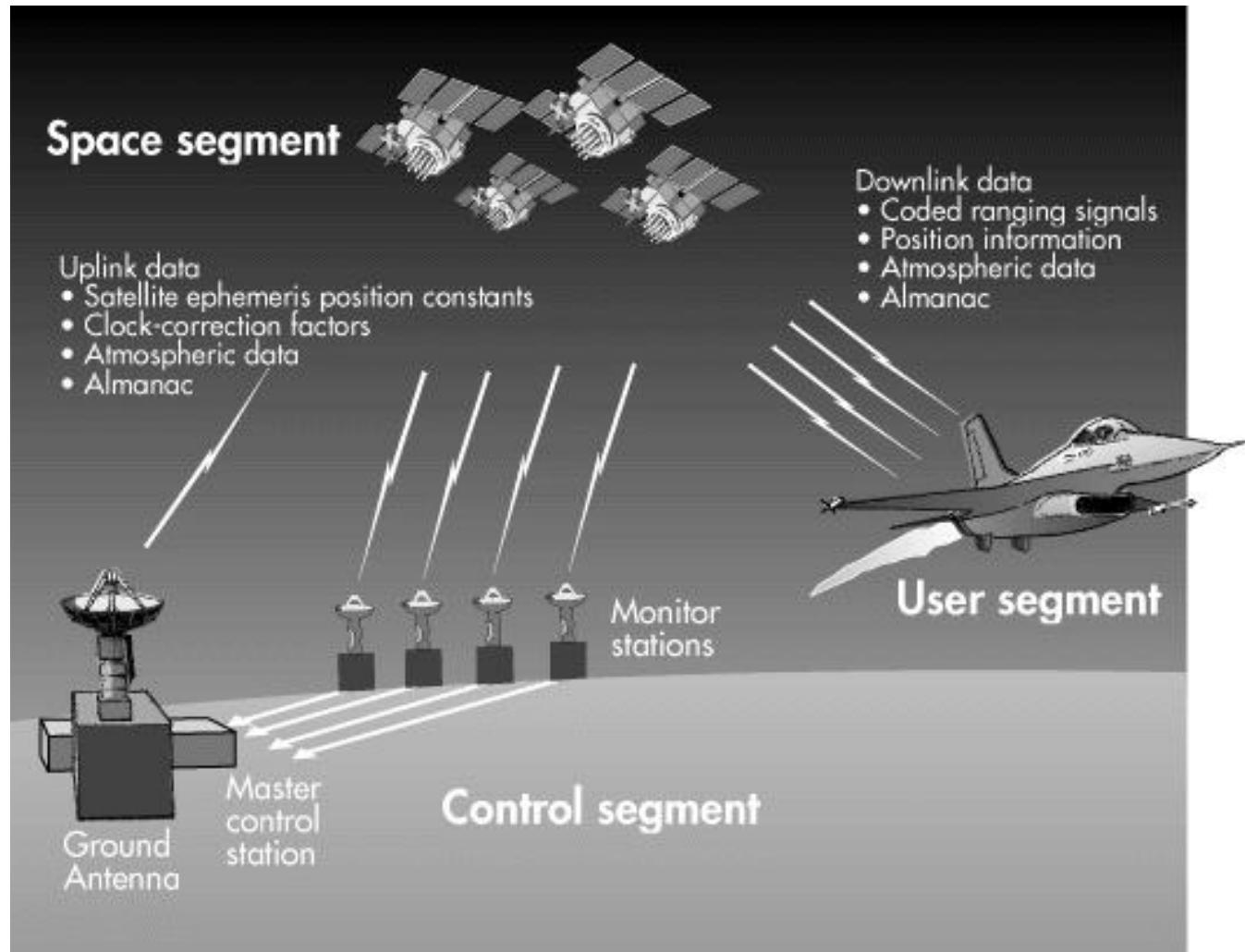
The History of GPS

- Feasibility studies begun in 1960s.
- Pentagon appropriates funding in 1973.
- First satellite launched in 1978.
- System declared fully operational in April 1995.

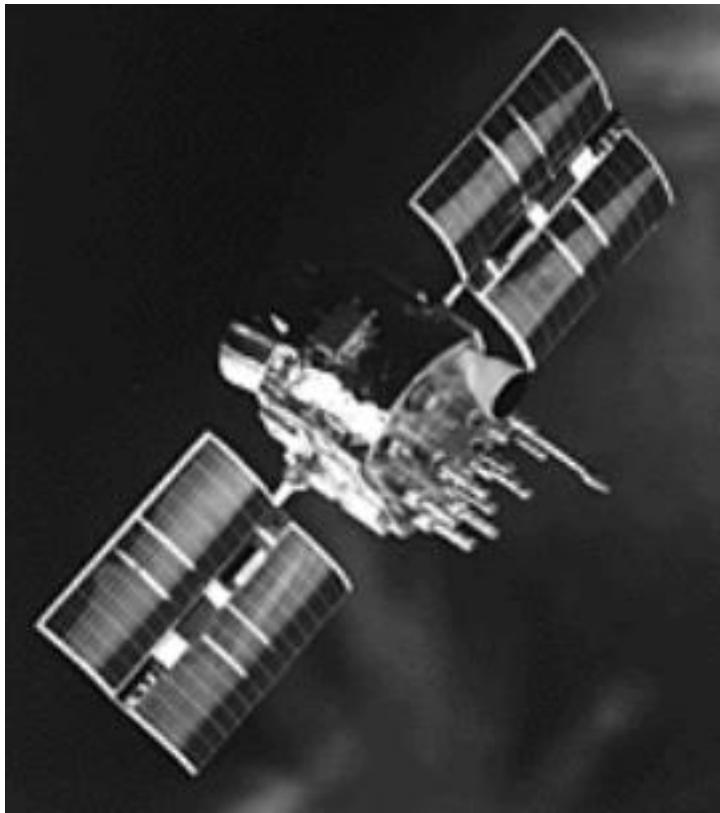


System Components

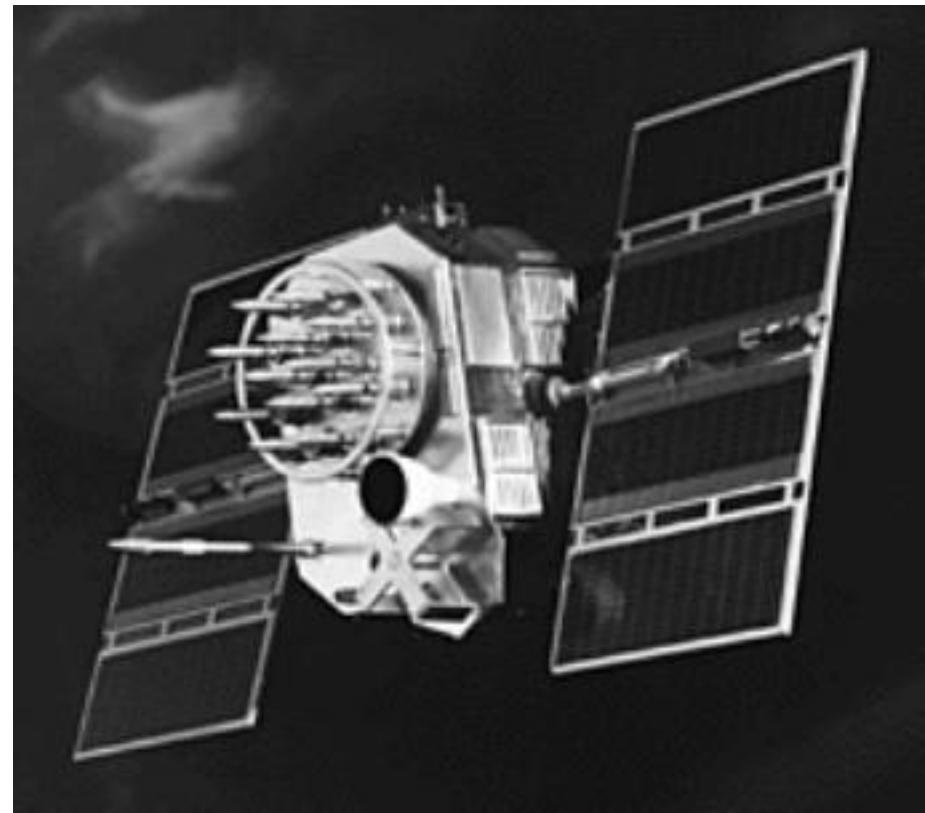
- space (GPS satellite vehicles)
- control (tracking stations)
- users



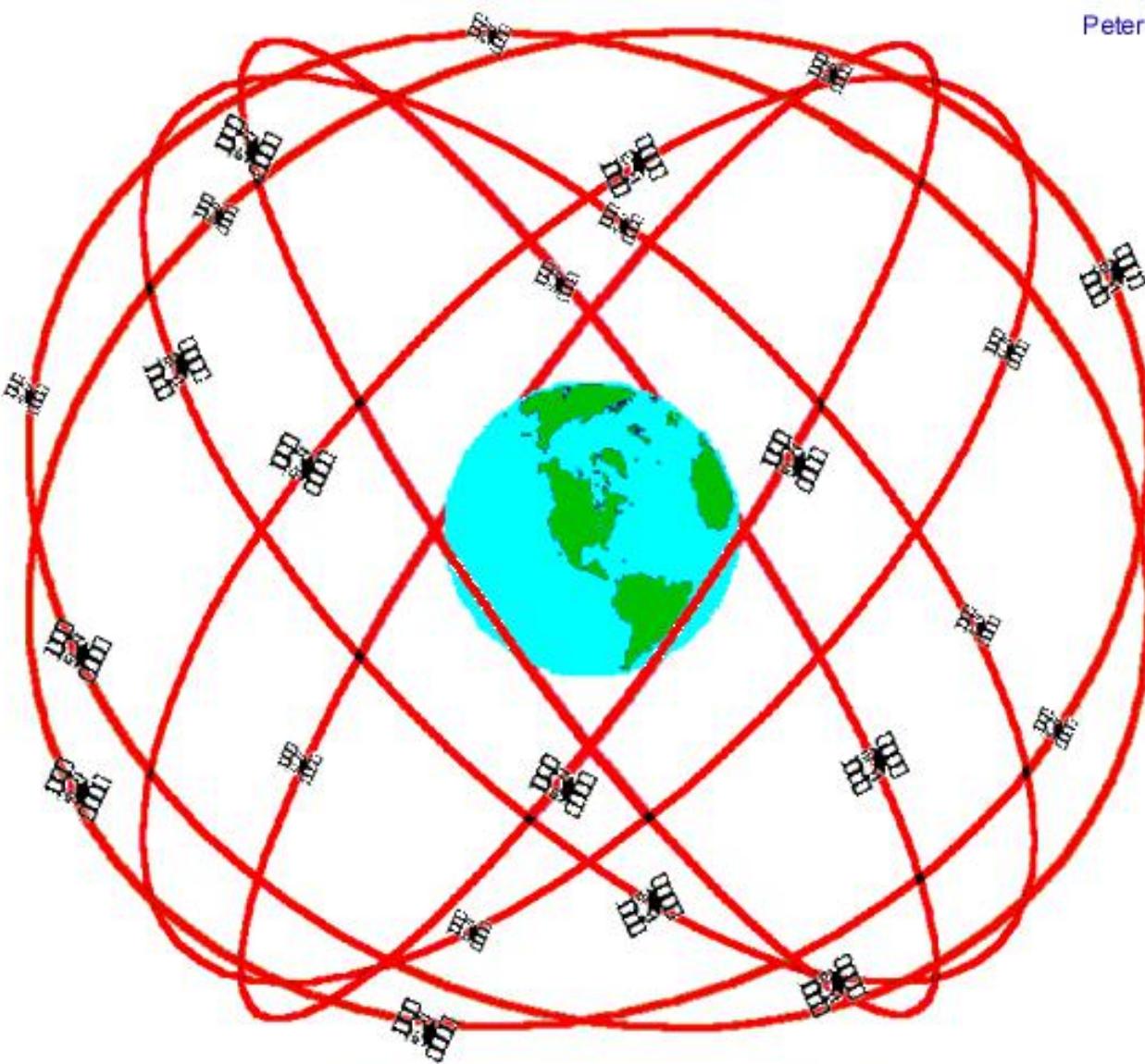
Two generations of GPS satellite vehicles (SVs)



GPS block I:
Experimental



GPS block II:
Full-scale operational



GPS Nominal Constellation
24 Satellites in 6 Orbital Planes
4 Satellites in each Plane
20,200 km Altitudes, 55 Degree Inclination

basic concept is that the GPS constellation replaces “stars” and gives us reference points for navigation

examples of some applications (users):

- navigation (very important for ocean travel)
- zero-visibility landing for aircraft
- collision avoidance
- surveying
- precision agriculture
- delivery vehicles
- emergency vehicles
- electronic maps
- Earth sciences (volcano monitoring; seismic hazard)
- tropospheric water vapor



examples of applications



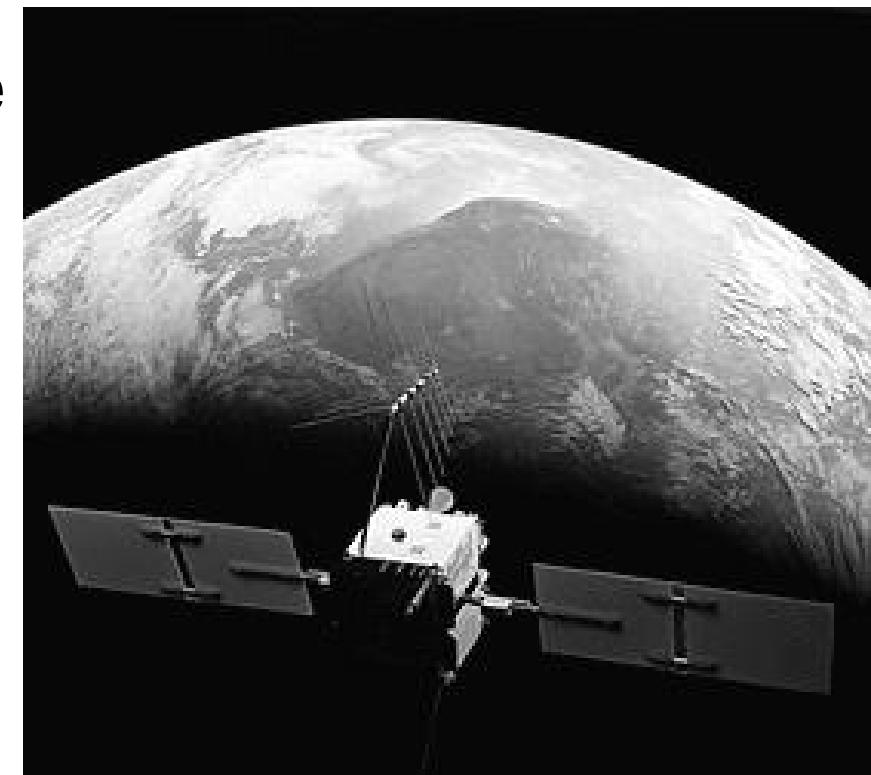
Four Basic Functions of GPS

- Position and coordinates.
- The distance and direction between any two waypoints, or a position and a waypoint.
- Travel progress reports.
- Accurate time measurement.

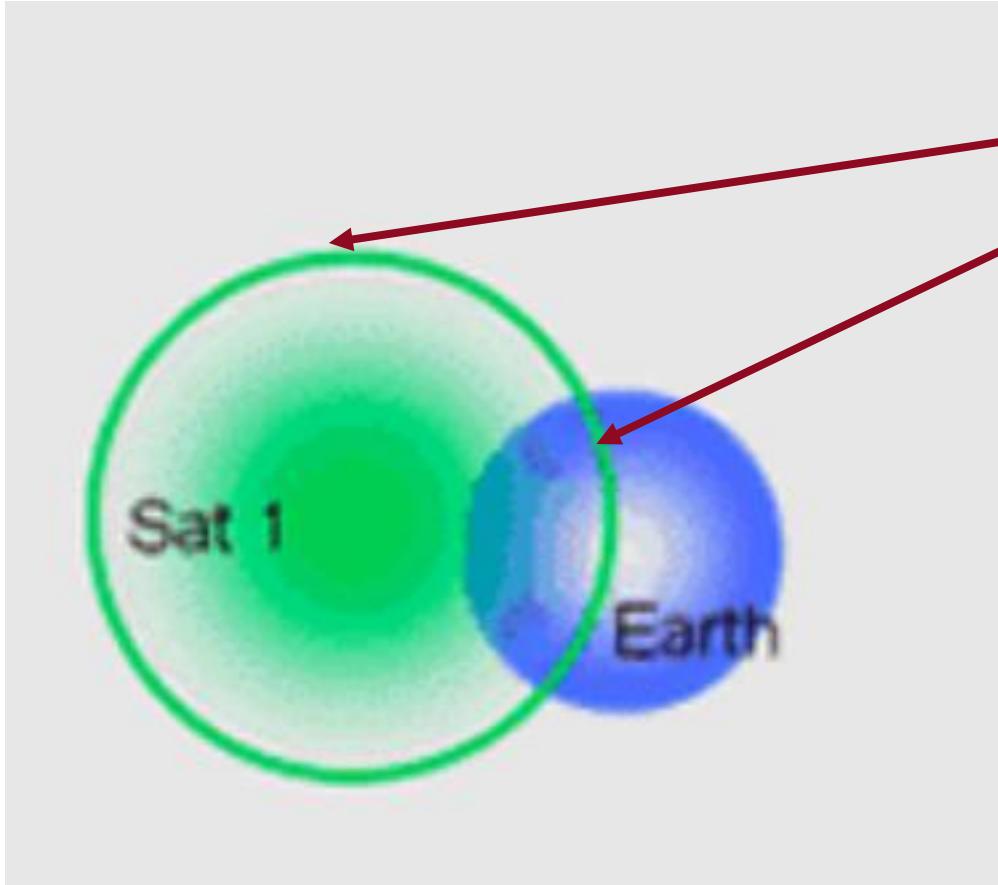


How GPS works

- step 1: using satellite ranging
- step 2: measuring distance from satellite
- step 3: getting perfect timing
- step 4: knowing where a satellite is in space
- step 5: identifying errors

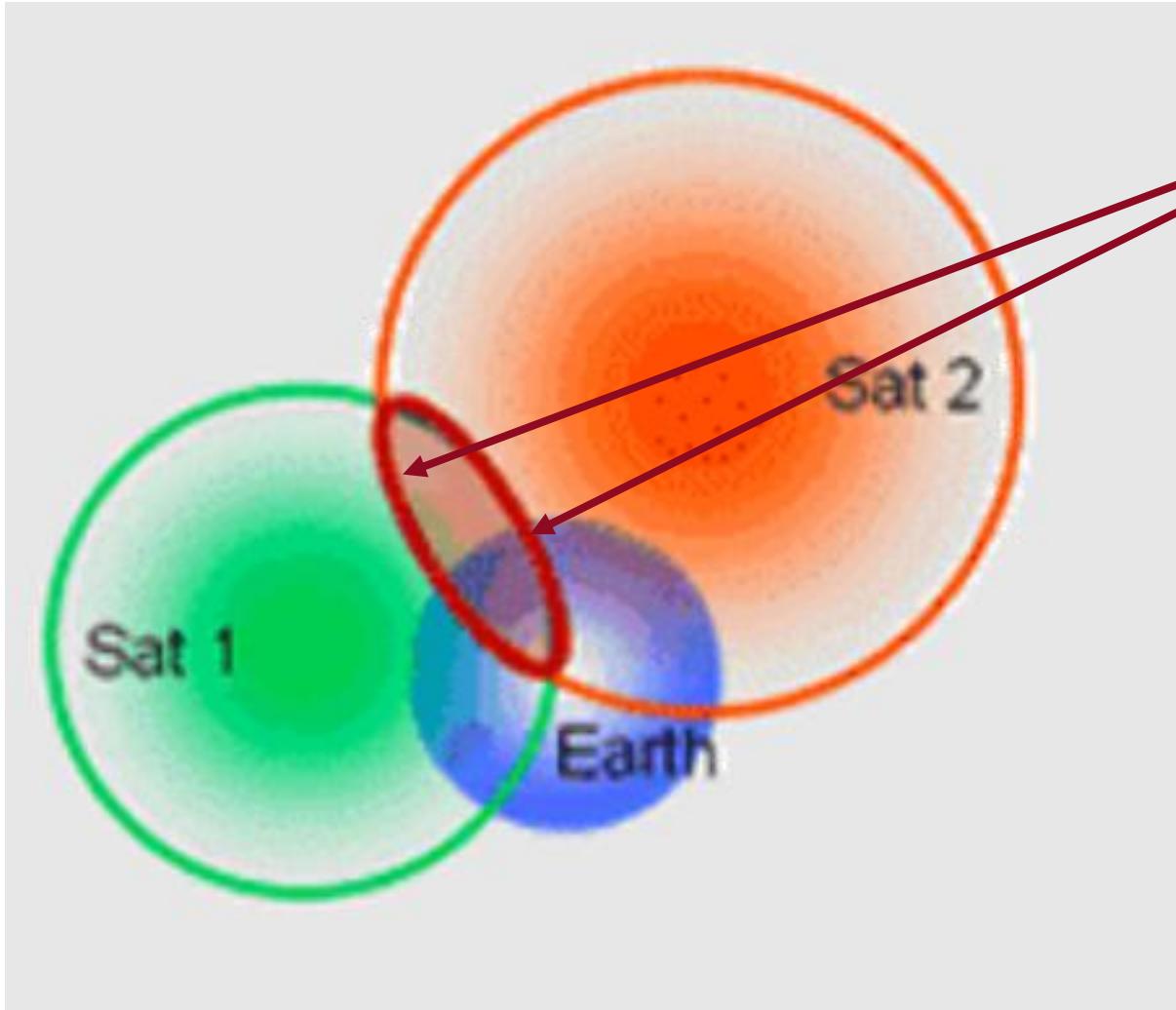


Signal From One Satellite



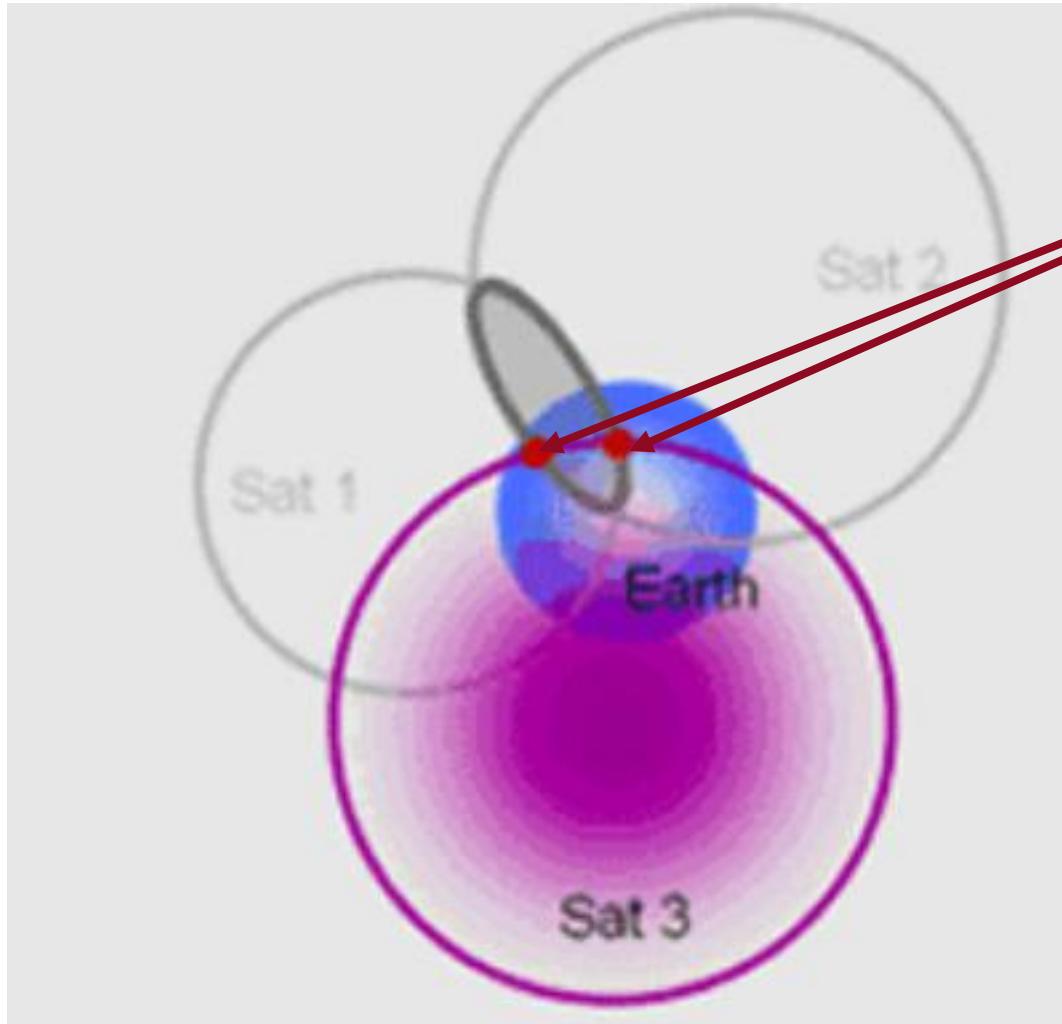
The receiver is
somewhere on
this sphere.

Signals From Two Satellites



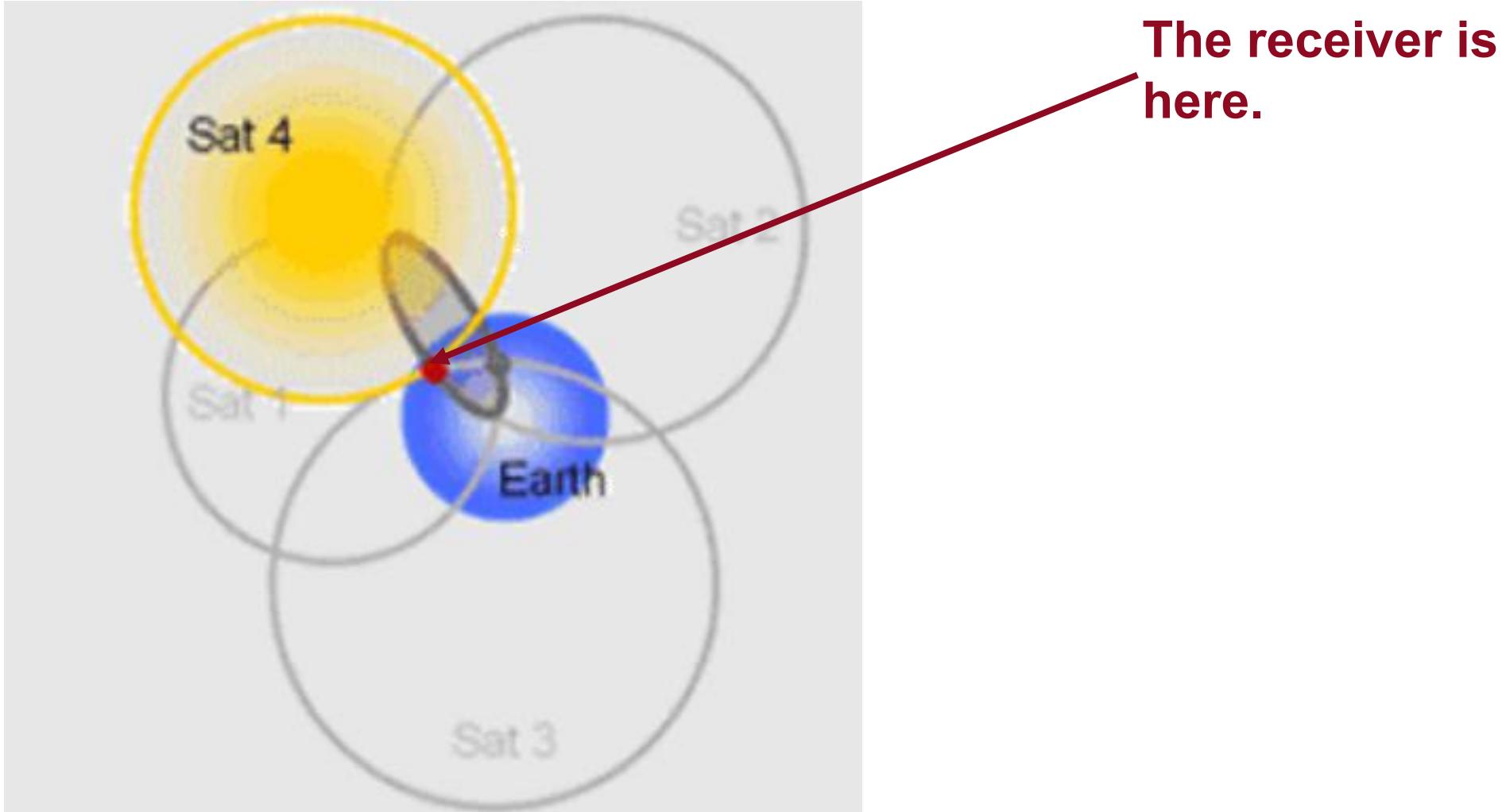
The receiver is
somewhere on
this circle.

Signals From Three Satellites



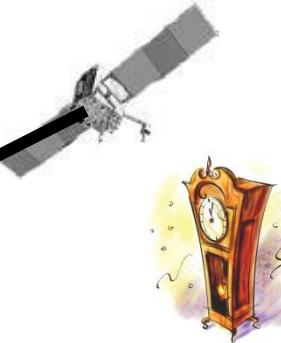
The receiver is
somewhere
between these
two points.

Signals From Four Satellites



Travel time

Signal leaves at 8:03:02.12



For example: 13,000 some miles

Radio waves travel about 186,000 miles (300,000 km) per second.



Signal arrives at 8:03:02.19

High accuracy is needed

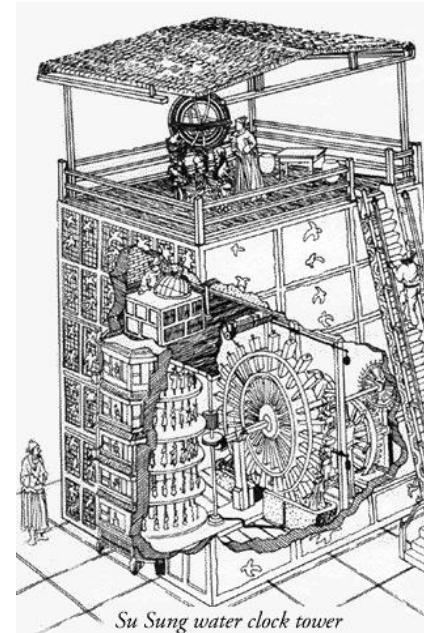
8:03:02.19

- 8:03:02.12

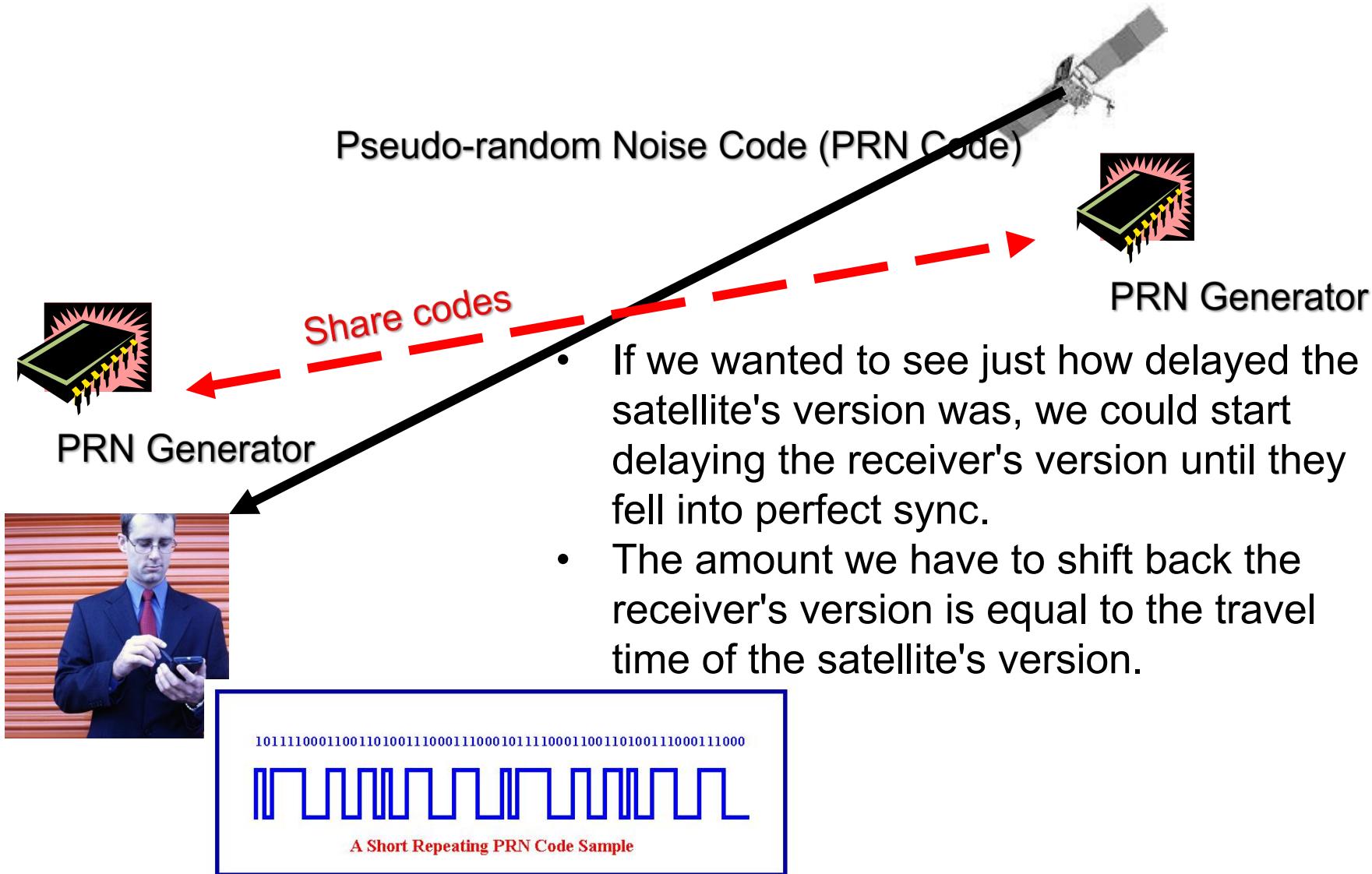
0:00:00.07

7 hundredths of a second
difference for the 13,000 mile
(i.e. 20,000 km) distance

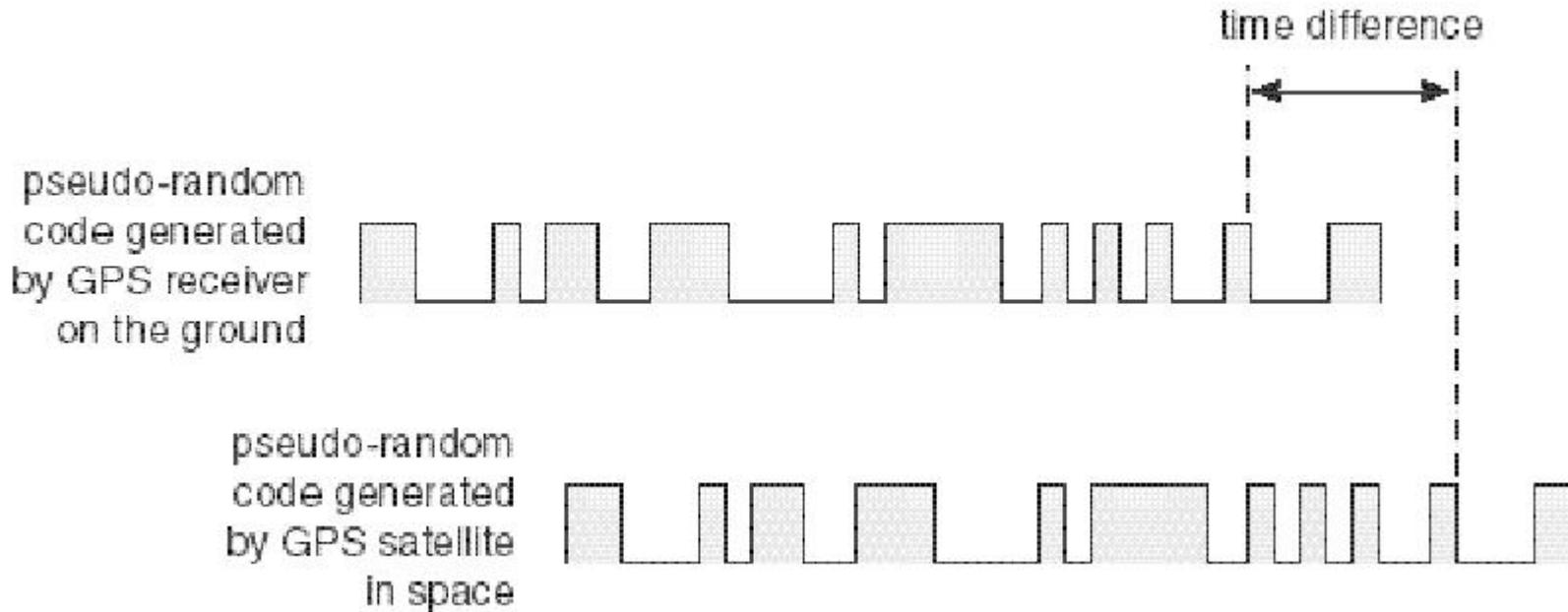
Takes some really good clocks



So how do you measure the time difference (ranging)?



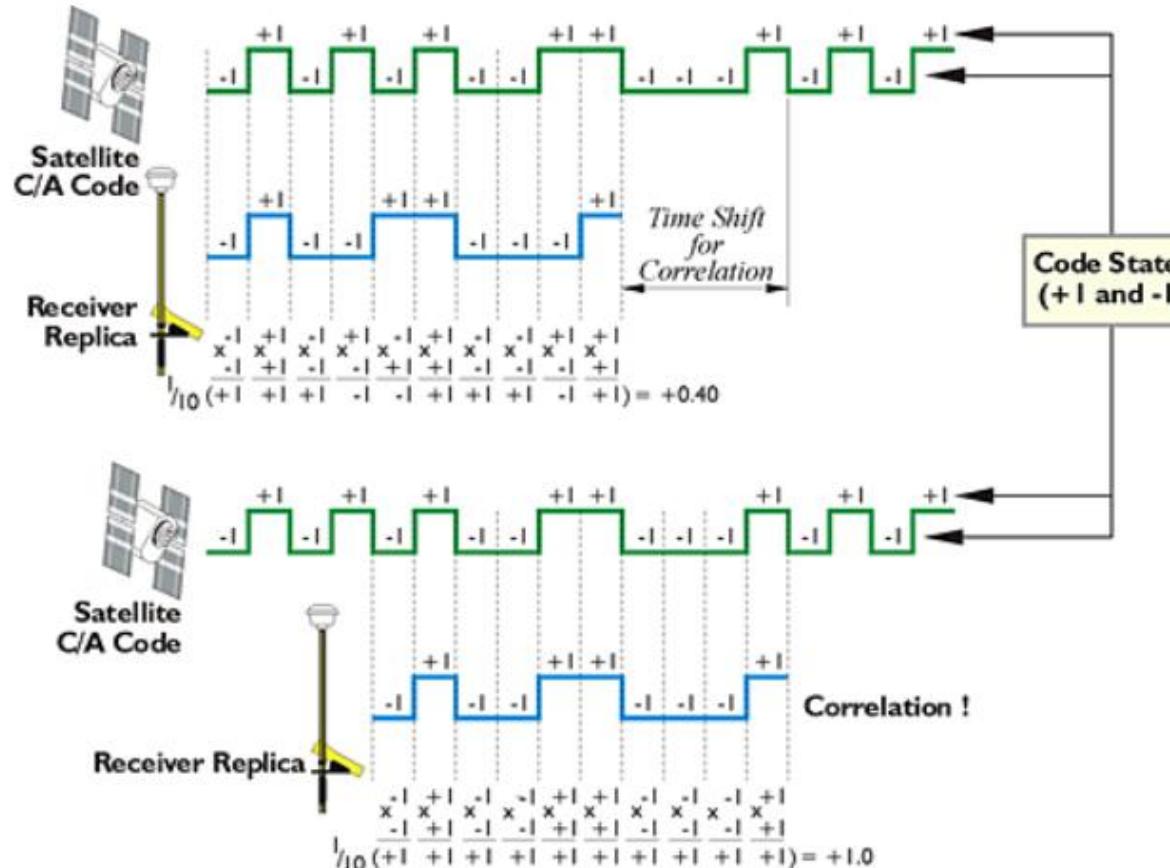
GPS Ranging



<http://maps.unomaha.edu/Peter%20son/gis/notes/GPS.pdf>

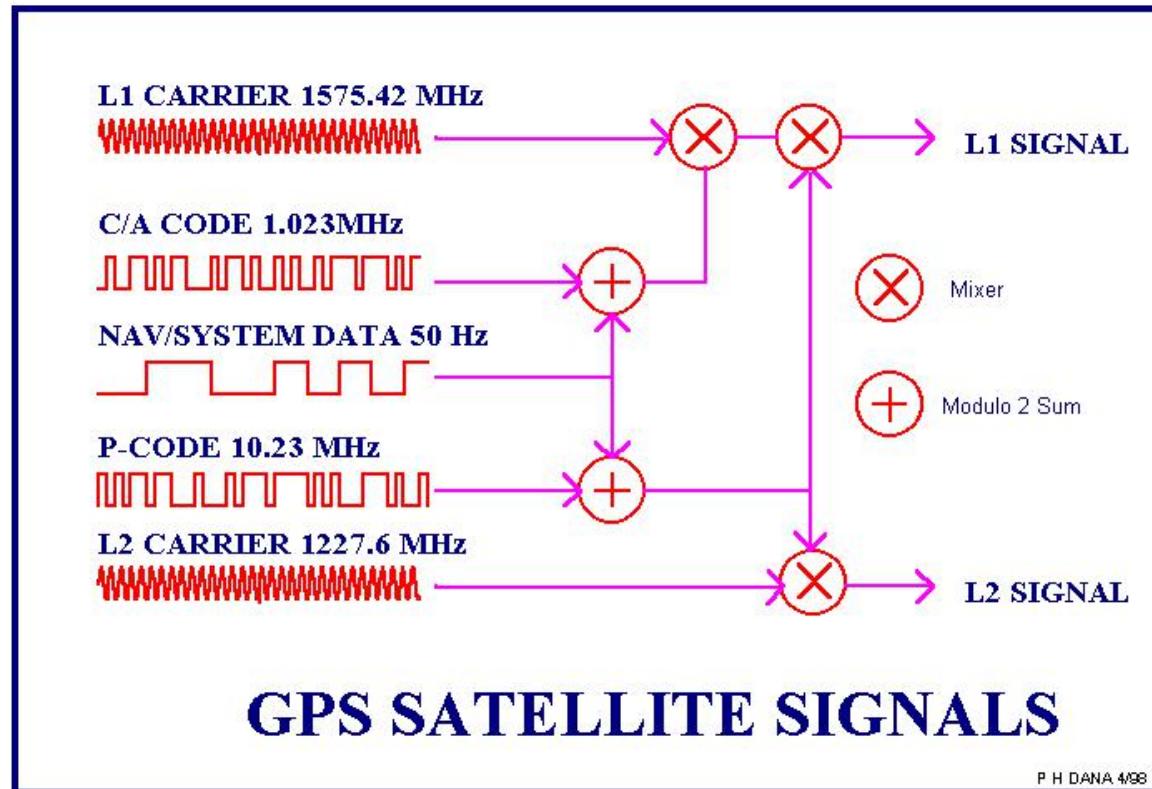
- The local receiver and the satellite generate the same pseudo-random code at the same time.
- Upon receiving the signal from the satellite, the difference of the two signals is compared.
- The time difference times speed of light determines the distance to satellite.

Algorithm Implementation



- The signals time slices are characterized as +1 and –1 states.
- We multiply each time slices and sum them up.
- We time shift the receiver replica of the signal to find the best correlation.
- When the sum of product of the two signal equals to 1, we have a perfect match

So what do the real signals look like?

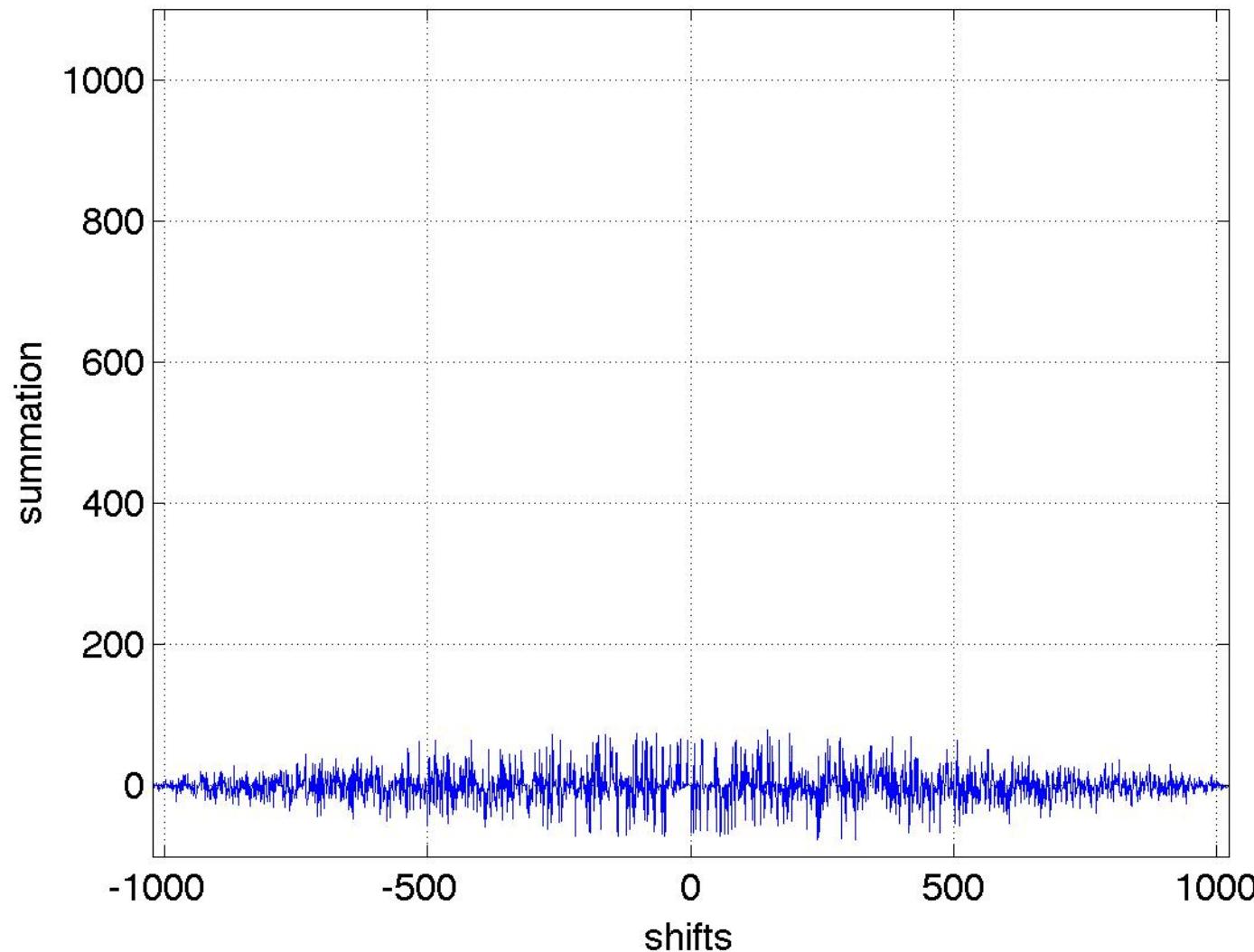


The information is sent either C/A (Course Acquisition Code) or P codes (Precision Code).

The C/A code is broadcast on L1 Carrier Frequency. 1-5 meter accuracy.
P Code – Precision Code is used by the military (L1 and L2).

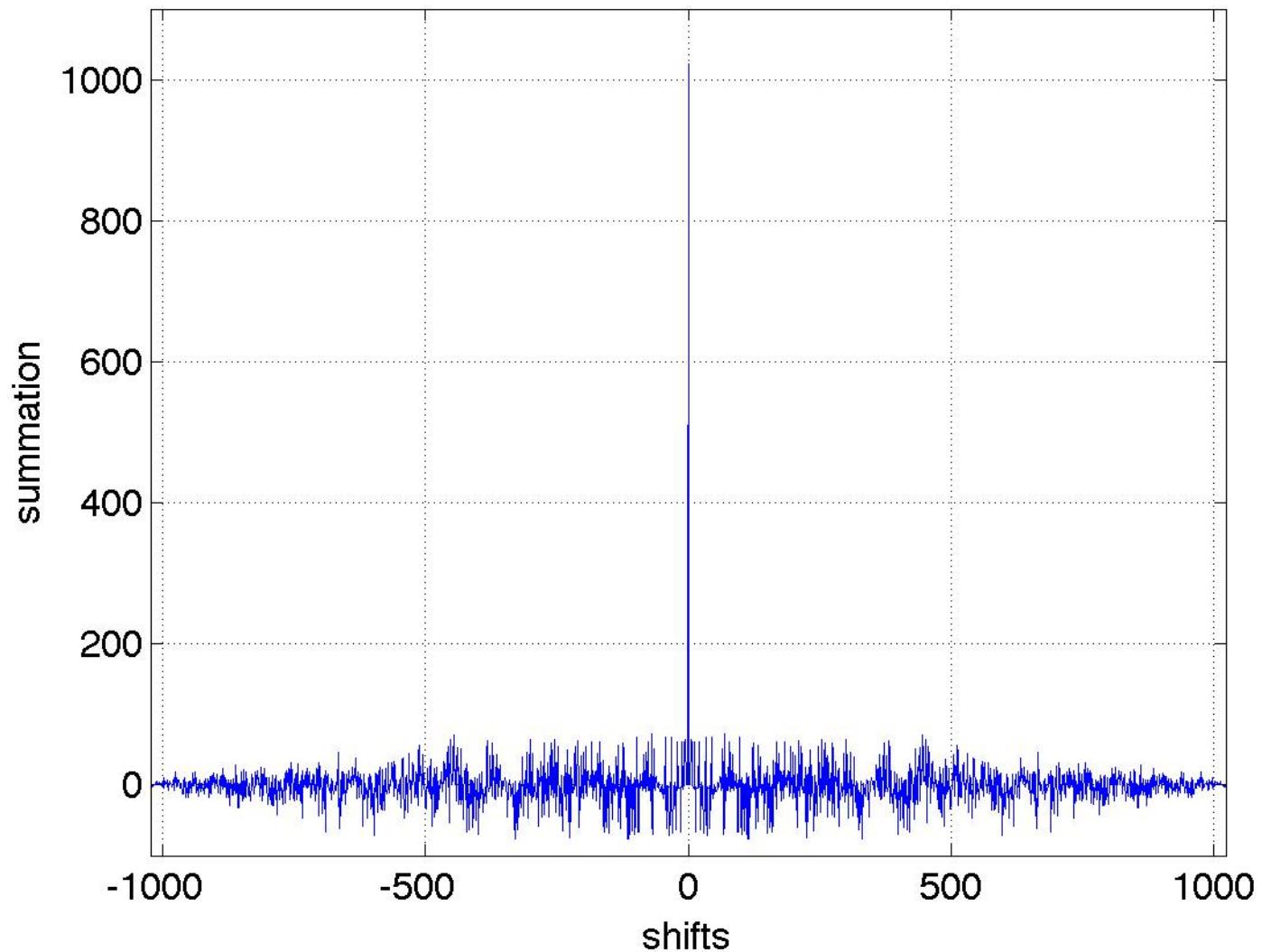
Satellite 9 compared to Satellite 10 code

Satellite 9 compared with Satellite 10

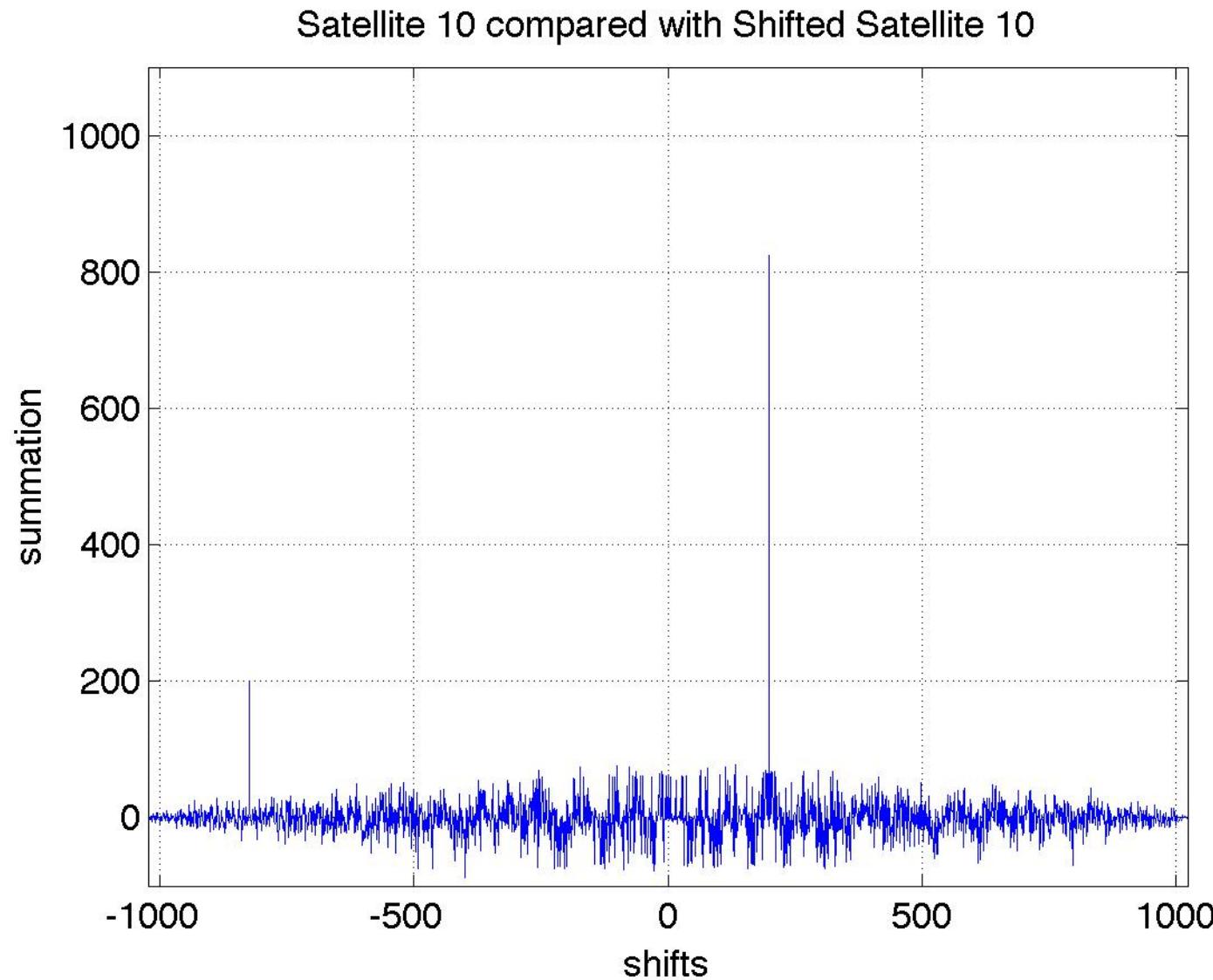


Satellite 10 compared to Satellite 10 code

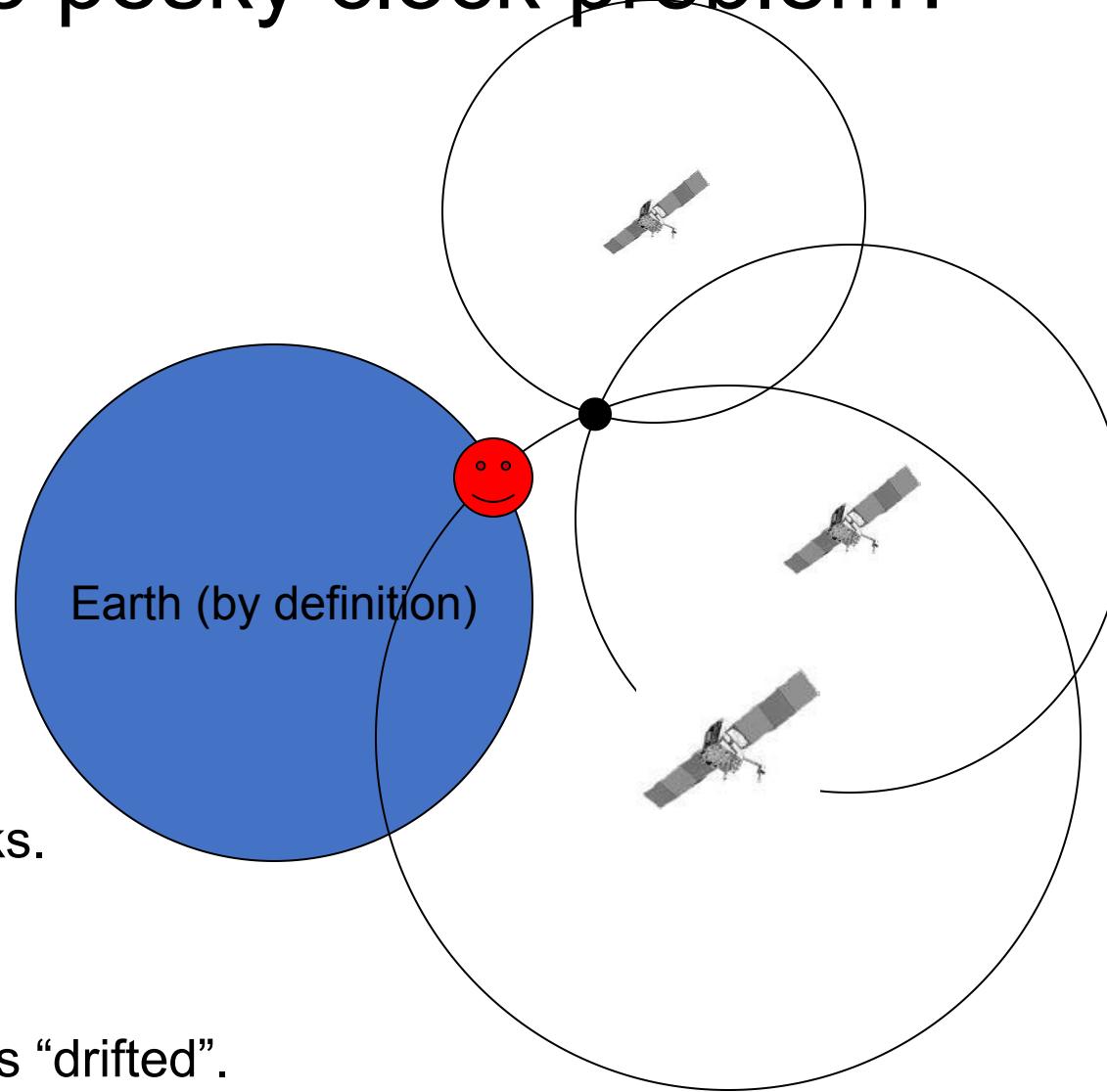
Satellite 10 compared with Satellite 10



Satellite 10 compared to Satellite 10 code that has been shifted by 200.



Remember the pesky clock problem?

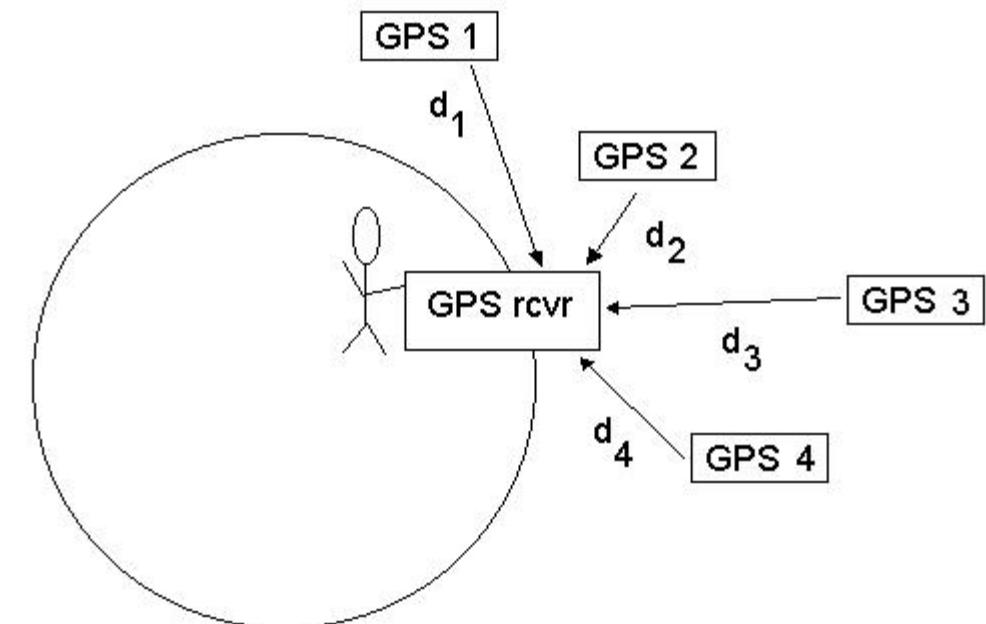


Satellites have expensive clocks.
Our receiver doesn't!

Key idea: user clock is “drifted”.
So our distance is off – but by a constant amount!
Treat it as another unknown variable

Details of how GPS works

- The GPS calculates four unknowns $\mathbf{x}, \mathbf{y}, \mathbf{z}, t_B$, where $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are the receiver's coordinates, and t_B is the time correction for the GPS receiver's clock.
- For satellite i , the position x_i, y_i, z_i is known



Solving for receiver position

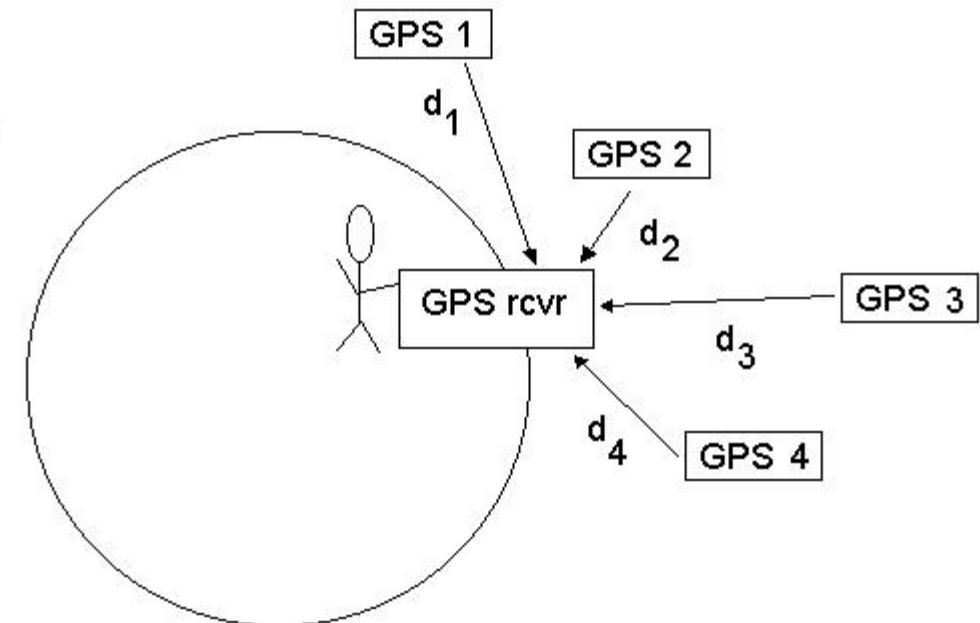
$$\sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2} + ct_B = d_1$$

$$\sqrt{(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2} + ct_B = d_2$$

$$\sqrt{(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2} + ct_B = d_3$$

$$\sqrt{(x - x_4)^2 + (y - y_4)^2 + (z - z_4)^2} + ct_B = d_4$$

- c : speed of light
 - x_i : the satellite positions, known to the receivers
- A quadratic programming problem. Solved by mature solvers



Satellite position information

- GPS satellites transmit information about their location (current and predicted), timing and "health" via what is known as **ephemeris** data.
- This data is used by the GPS receivers to estimate location relative to the satellites and thus position on earth.

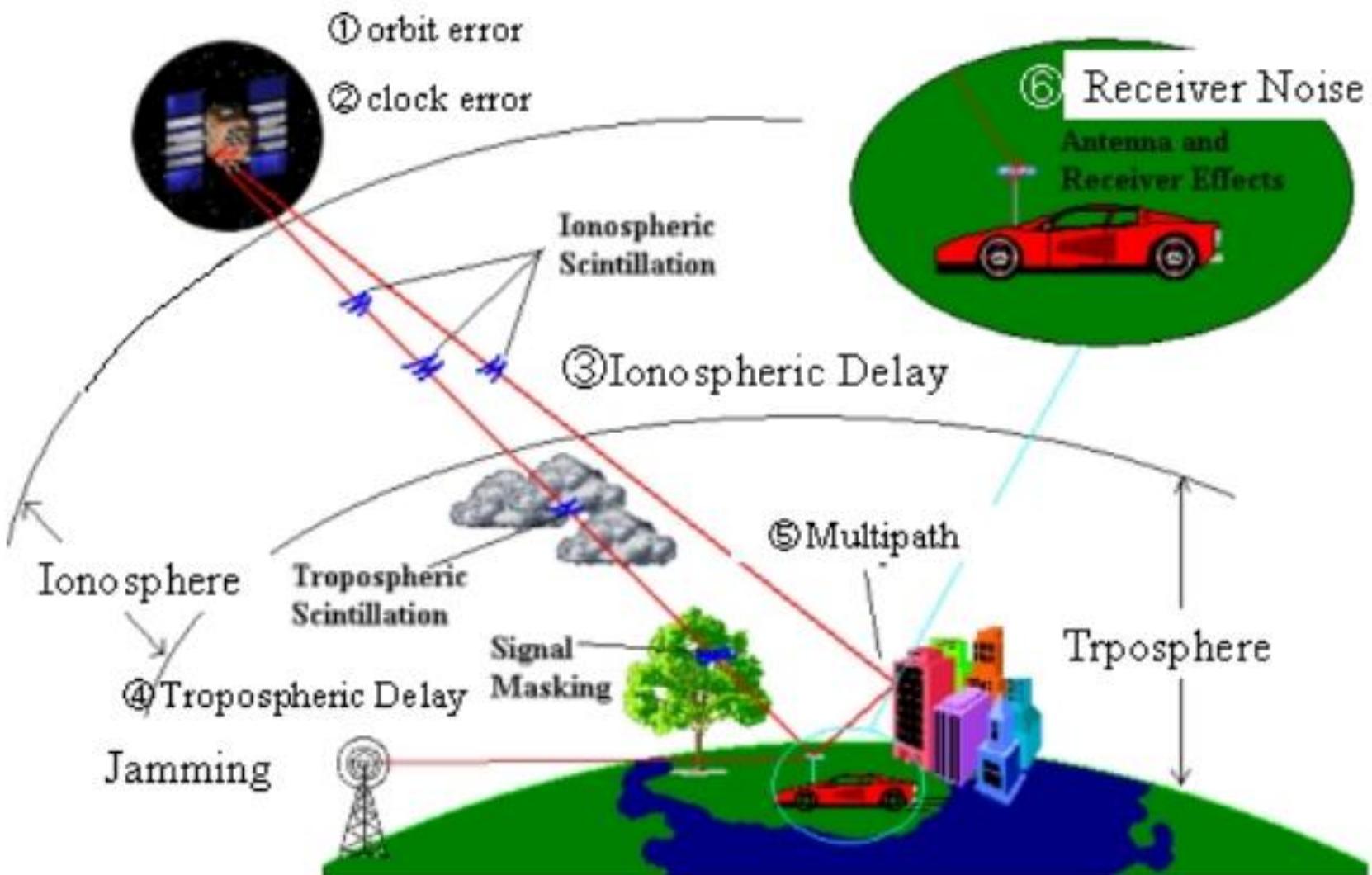
Adafruit Ultimate GPS Module

- -165 dBm sensitivity, 10 Hz updates, 66 channels
- 5V friendly design and only 20mA current draw
- Breadboard friendly + two mounting holes
- RTC battery-compatible
- Built-in datalogging
- PPS output on fix
- Internal patch antenna
- u.FL connector for external active antenna



<https://www.adafruit.com/product/746>

Errors on GPS Signal



GPS error: Satellite clocks

- Satellites use atomic clocks, which are very accurate but can drift up to a millisecond. Errors: 1.5-3.6 meters
- Mitigation: minimized by calculating clock corrections (at monitoring stations) and transmitting the corrections along with the GPS signal to appropriately outfitted GPS receivers.

GPS error: Orbital Errors

- GPS receivers calculate coordinates relative to the known locations of satellites in space. Errors<1 meter
- Mitigation: The GPS Control Segment monitors satellite locations at all times, calculates orbit eccentricities, and compiles these deviations in documents called ephemerides.
- GPS receivers that are able to process ephemerides can compensate for some orbital errors.

GPS error: Upper Atmosphere (Ionosphere)

- As GPS signals pass through the upper atmosphere (the ionosphere 50-1000km above the surface), signals are delayed and deflected. Errors: 5-7 meters
- Mitigation: By modeling ionosphere characteristics, GPS monitoring stations can calculate and transmit corrections to the satellites, which in turn pass these corrections along to receivers.

GPS error: Lower Atmosphere (Troposphere)

- The lower atmosphere delays GPS signals, adding slightly to the calculated distances between satellites and receivers.
Errors: < 1 meter
- Note: weather conditions, such as clouds, storms, and rains, have limited impacts on accuracy

GPS error: Multipath Effects

- Ideally, GPS signals travel from satellites through the atmosphere directly to GPS receivers.
- In reality, GPS receivers must discriminate between signals received directly from satellites and other signals that have been reflected from surrounding objects, such as buildings, trees, and even the ground. Errors: up to 1.2 meters
- Mitigation: use antenna technique to track signals that are at least 15 degrees above horizon.

GPS error: Wireless Interference

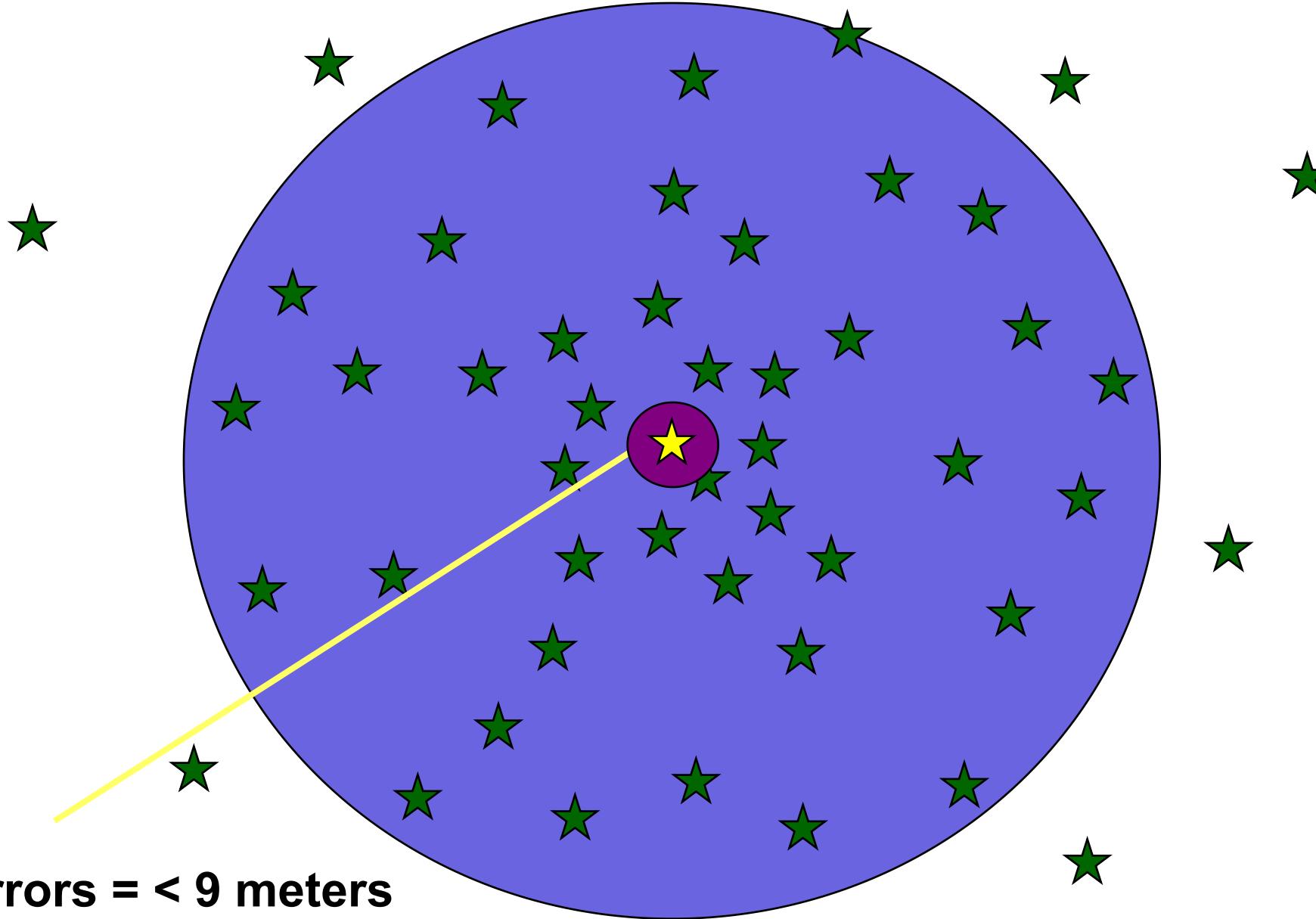
- Like any other wireless systems, GPS signals are susceptible to interference. Error: from small to unbounded.

Sources of GPS Error

<u>Source</u>	<u>Amount of Error</u>
Ø Satellite clocks:	1.5 to 3.6 meters
Ø Orbital errors:	< 1 meter
Ø Ionosphere:	5.0 to 7.0 meters
Ø Troposphere:	0.5 to 0.7 meters
Ø Receiver noise:	0.3 to 1.5 meters
Ø Multipath:	0.6 to 1.2 meters

Errors are cumulative and increased by PDOP.

Receiver Errors are Cumulative!



System errors = < 9 meters

Which sources of errors?

- GPS navigation errors in a city with tall buildings
- Drone GPS spoofing

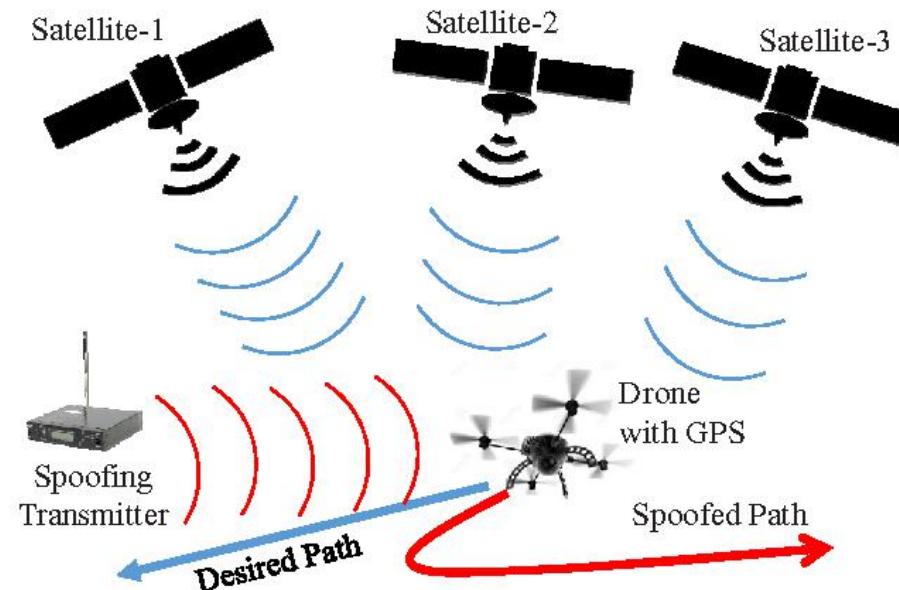
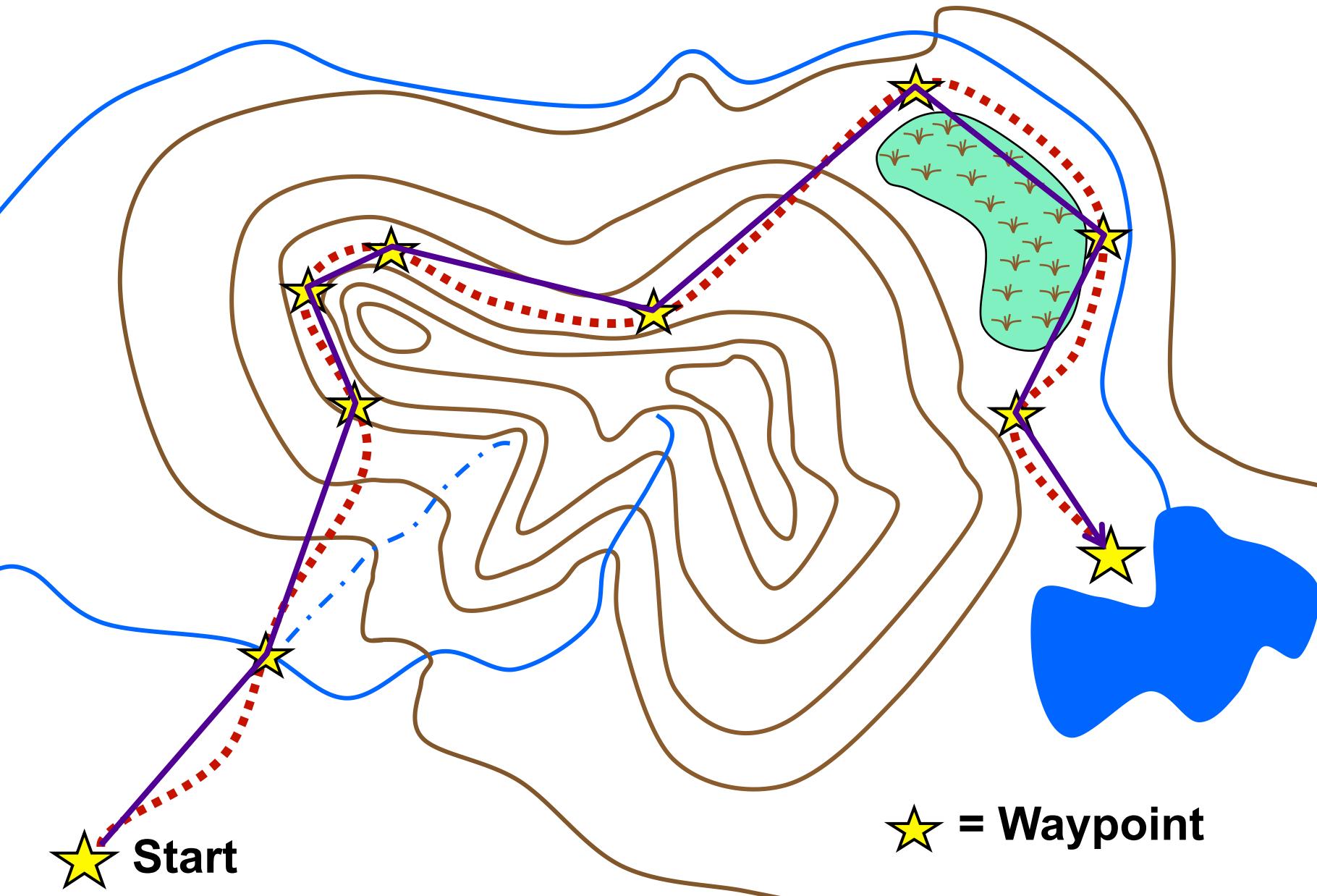


Fig. 4: GPS spoofing attack Scenario, which changes the actual

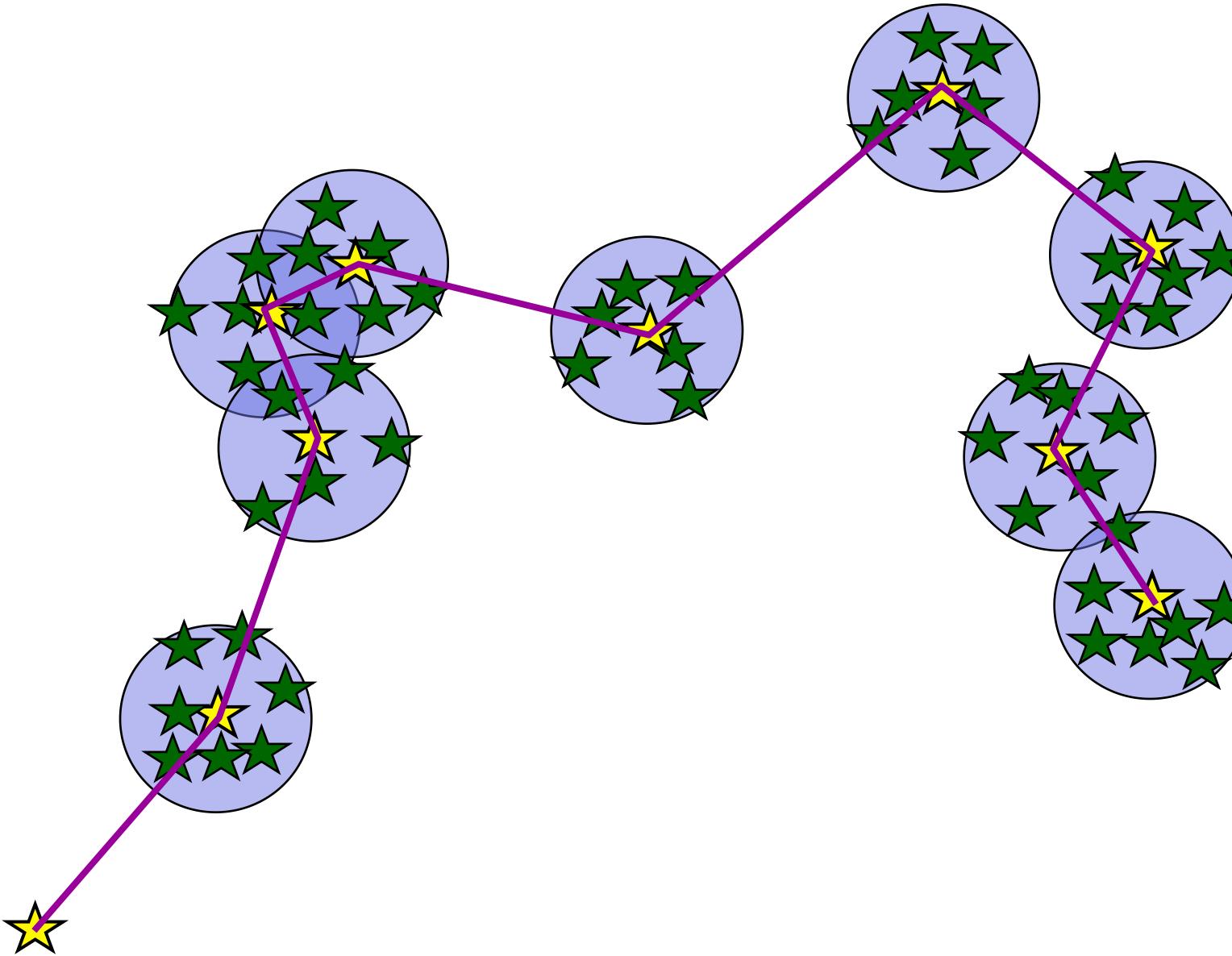
Waypoint

- A waypoint is based on coordinates entered into a GPS receiver's memory.
- It can be created for any remote point on earth.
- It must have a receiver designated code or number, or a user supplied name.
- Once entered and saved, a waypoint remains unchanged in the receiver's memory until edited or deleted.

Planning a Navigation Route



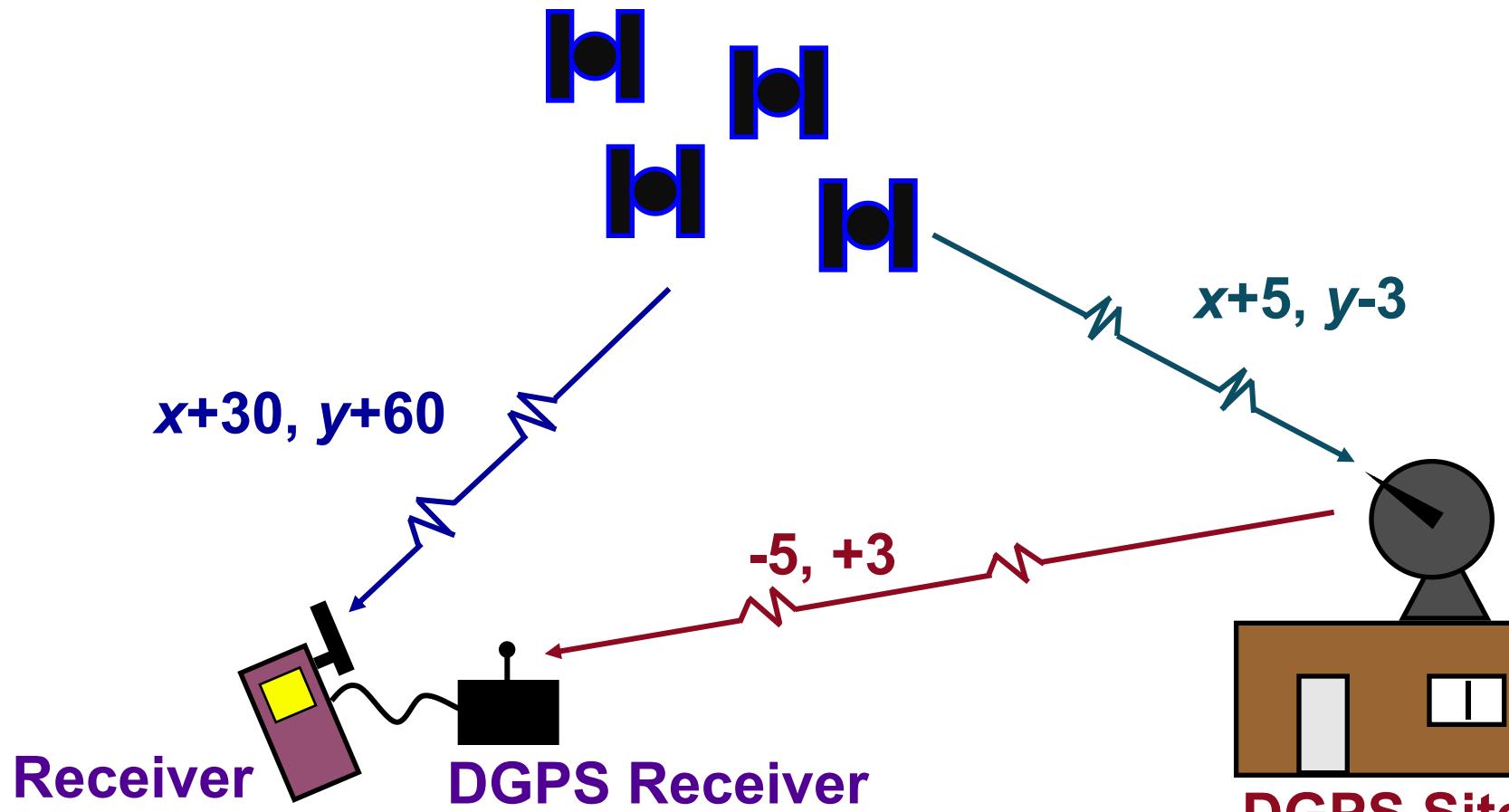
How A Receiver Sees Your Route



Differential GPS

- Improves nominal GPS accuracy of 15m to up to 3cm
- Basic idea:
 - Use a network of fixed reference ground-based stations to broadcast the difference between GPS result and the ground-truth position
 - The GPS receiver use this difference to correct its own errors.

Real Time Differential GPS



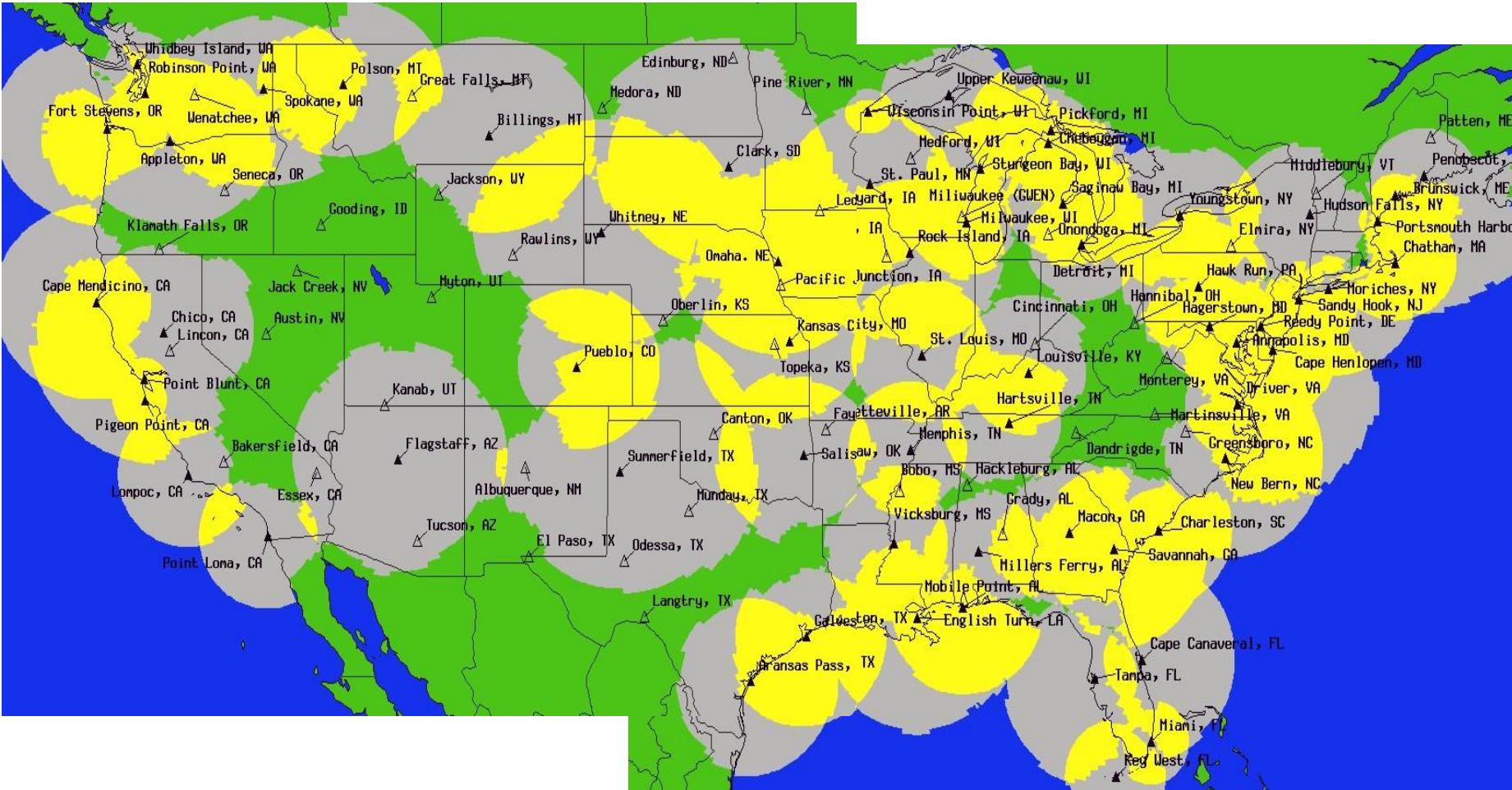
DGPS correction = $x+(30-5)$ and $y+(60+3)$

Corrected coordinates = $x+25, y+63$

True coordinates = x, y

Correction = $-5, +3$

National Differential Global Positioning System Coverage



Yellow areas show overlap between NDGPS stations. Green areas are little to no coverage.

Exercise

Find the ground truth coordinate for receiver 1

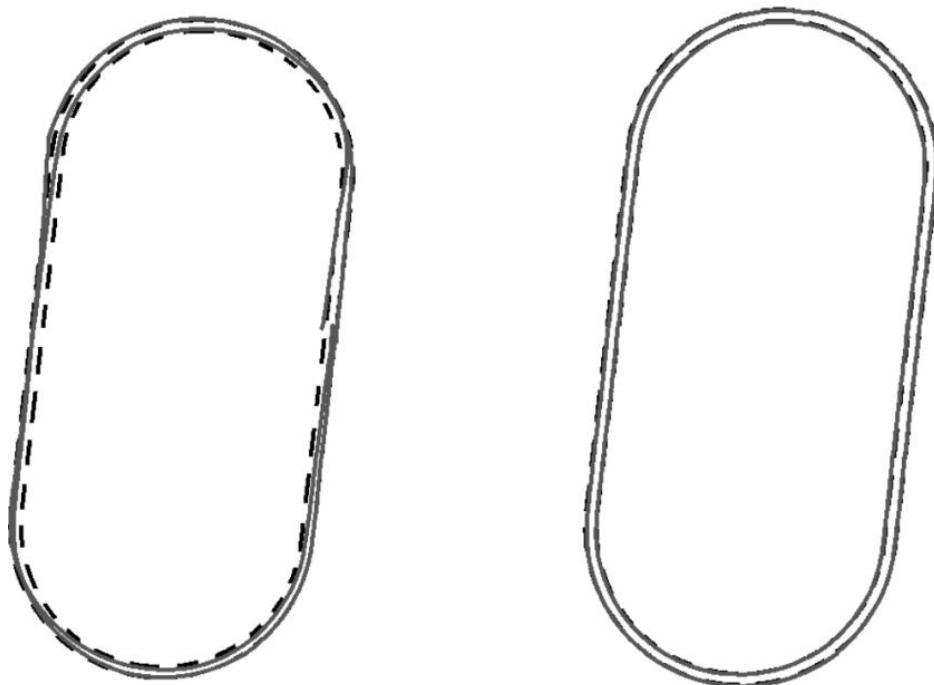
	Lat_meas	Long_meas	alt_meas	Lat_ground	Long_ground	Alt_ground
DGPS	37.0366833	35.3744433	60	37.0367146	35.3744596	64.5
Receiver 1	37.03671	35.3744467	65			

Question

- What types of errors can be mitigated by DGPS
 - ∅ Satellite clocks
 - ∅ Orbital errors
 - ∅ Ionosphere
 - ∅ Troposphere
 - ∅ Receiver noise
 - ∅ Multipath

Low-cost Differential GPS*

- Use multiple GPS receivers with known relative positions
- Sub-meter accuracy



CSE 162 Mobile Computing

Lecture 15: Location Programming and Processing

Hua Huang

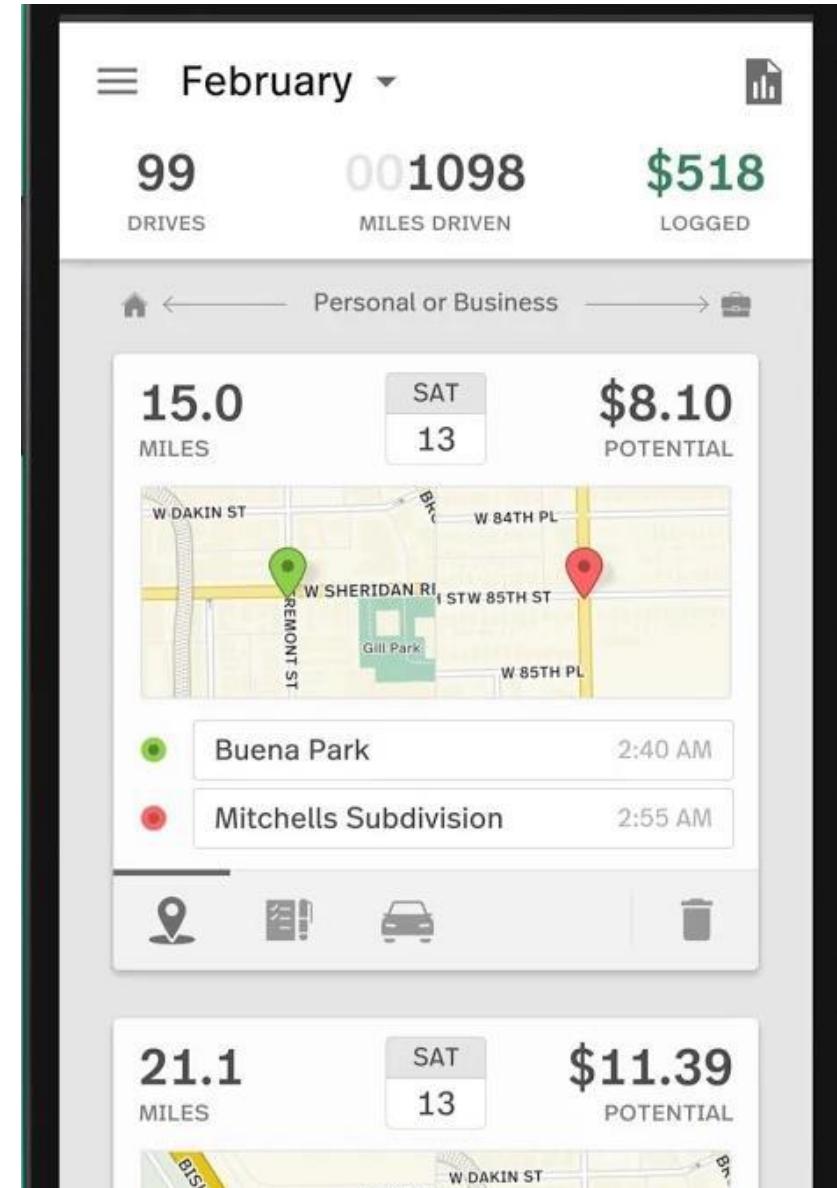
Department of Computer Science and Engineering

University of California, Merced

Some Interesting Location-Aware Apps

MileIQ

- **The Problem:** Mileage tracking is useful but a burden.
 - IRS deductions on taxes
 - Some companies reimburse employees for mileage,
- Passively, automatically tracks business mileage, IRS compliant
- Swipe right after drive to indicate it was a business trip



Trigger

- Use geofences, NFC, bluetooth, WiFi connections, etc to set auto-behaviors
 - Battery low -> turn off bluetooth + auto sync
 - Silence phone every morning when you get to work
 - Turn off mobile data when you connect to your home WiFi
 - Silence phone and set alarm once I get into bed
 - Use geofence for automatic foursquare checkin
 - Launch maps when you connect to your car's bluetooth network

The screenshot shows the 'Suggested Tasks' screen from the IFTTT mobile application. At the top, there is a header with a menu icon and the title 'Suggested Tasks'. Below the header, a blue banner contains the text: 'Use your phone's sensors to automatically change settings, launch apps or send messages.' and 'Get started with our examples below or create your own tasks.' There are two buttons at the bottom of the banner: 'Create your own' and 'OK, got it'. Below the banner, there are three cards, each representing a suggested task:

- Save time when driving**
turns off wifi and opens Google
- PRO**
Silence my phone while I sleep
silences incoming notifications while you
- PRO**
Help me save battery when my battery gets low
turns off wifi and turns down screen brightness

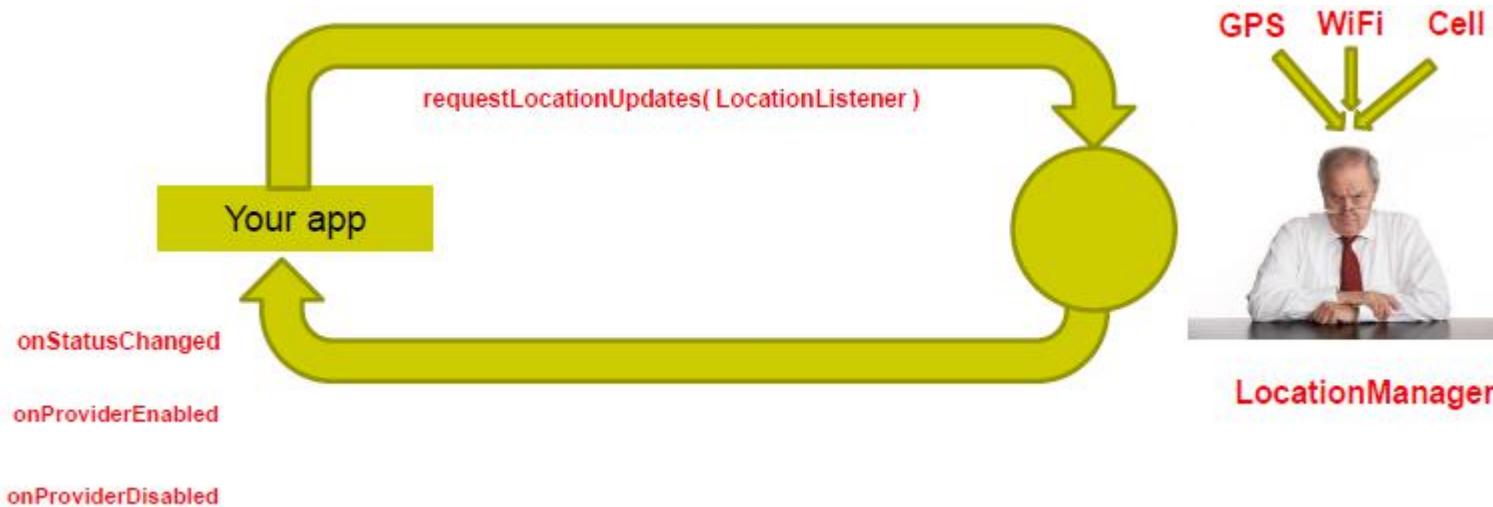
A green circular button with a '+' sign is located in the bottom right corner of the screen.

Location Sensing in Android Apps

The Basic Location APIs

- **LocationManager:**

- Android module receives location updates from GPS, WiFi, etc
- App registers/requests location updates from LocationManager



```
// Acquire a reference to the system Location Manager
LocationManager locationManager = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);

// Define a listener that responds to location updates
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
        // Called when a new location is found by the network location provider.
        makeUseOfNewLocation(location);
    }

    public void onStatusChanged(String provider, int status, Bundle extras) {}

    public void onProviderEnabled(String provider) {}

    public void onProviderDisabled(String provider) {}
};

// Register the listener with the Location Manager to receive location updates
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, locationListener)
```

Create listener for location info

Callback methods called by Location manager (e.g. when location changes))

Requesting User Permissions

- Need smartphone owner's permission to use their GPS

```
<manifest ... >
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    ...
    <!-- Needed only if your app targets Android 5.0 (API level 21) or higher. -->
    <uses-feature android:name="android.hardware.location.gps" />
    ...
</manifest>
```

- **ACCESS_FINE_LOCATION:** GPS
- **ACCESS_COARSE_LOCATION:** WiFi or cell towers

Getting Cached Copy of Location (Fast)

- Getting current location may take a while
- Can choose to use location cached (possibly stale) from Location Manager

```
String locationProvider = LocationManager.NETWORK_PROVIDER;  
// Or use LocationManager.GPS_PROVIDER
```

```
Location lastKnownLocation = locationManager.getLastKnownLocation(locationProvider);
```

Stopping Listening for Location Updates

- Location updates consume battery power
- Stop listening for location updates whenever you no longer need

```
// Remove the listener you previously added  
locationManager.removeUpdates(locationListener);
```

Location Representation in Android

Semantic Location

- GPS represents location as <longitude,latitude>
- **Semantic location** is better for reasoning about locations
- **E.g.** Street address (140 Park Avenue, Worcester, MA) or (building, floor, room)
- **Android supports:**
 - **Geocoding:** Convert addresses into longitude/latitude coordinates
 - **Reverse geocoding:** convert longitude/latitude coordinates into human readable address
- **Android Geocoding API:** access to **geocoding** and **reverse geocoding** services using HTTP requests

Latitude: 37.422005 Longitude: -122.084095

Address:
1600 Amphitheatre Pkwy
Mountain View, CA 94043
Mountain View
94043
United States

Google Places API Overview

- Access **high-quality photos** of a place
- Users can also add place information to the database
 - E.g. business owners can add their business as a place in Places database
 - Other apps can then retrieve info after moderation
- **On-device caching:** Can cache places data locally on device to avoid roundtrip delays on future requests



Google Places

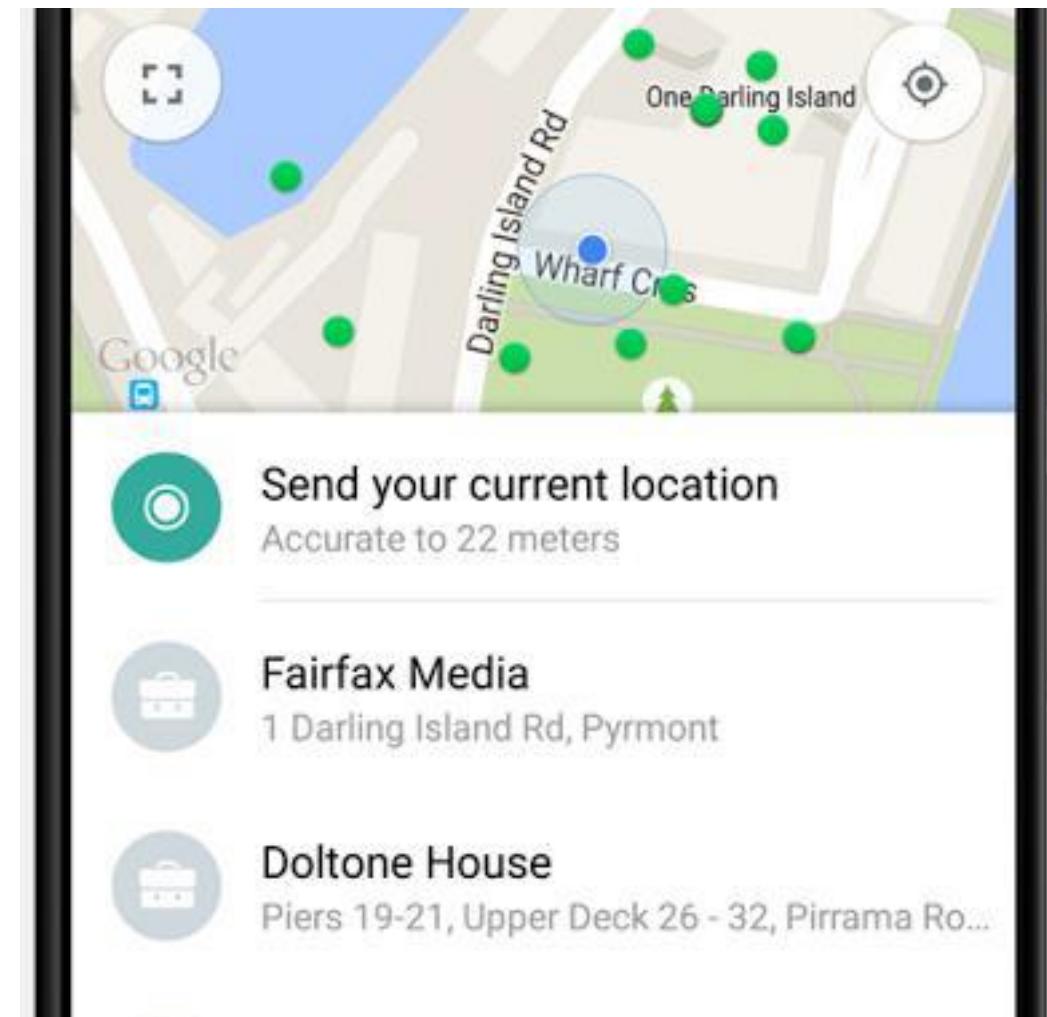
- **Place:** physical space that has a name (e.g. local businesses, points of interest, geographic locations)
 - E.g Logan airport, place type is **airport**
- **API:** Provides Contextual information about places near device.
 - **E.g:** name of place, address, geographical location, place ID, phone number, place type, website URL, etc.
- Compliments geographic-based services offered by Android location services

Sample Place Types

accounting	hospital	city_hall	physiotherapist
airport	insurance_agency	clothing_store	place_of_worship (deprecated)
amusement_park	jewelry_store	convenience_store	plumber
aquarium	laundry	courthouse	police
art_gallery	lawyer	dentist	post_office
atm	library	department_store	real_estate_agency
bakery	liquor_store	doctor	restaurant
bank	local_government_office	electrician	roofing_contractor
bar	locksmith	electronics_store	rv_park
beauty_salon	lodging	embassy	school
bicycle_store	meal_delivery	establishment (deprecated)	shoe_store
book_store	meal_takeaway	finance (deprecated)	shopping_mall
bowling_alley	mosque	fire_station	spa
bus_station	movie_rental	florist	stadium
cafe	movie_theater	food (deprecated)	storage
campground	moving_company	funeral_home	store
car_dealer	museum	furniture_store	subway_station
car_rental	night_club	gas_station	synagogue
car_repair	painter	general_contractor (deprecated)	taxi_stand
car_wash	park	grocery_or_supermarket	train_station
		gym	transit_station
		hair_care	travel_agency
		hardware_store	university
		health (deprecated)	veterinary_care
		hindu_temple	zoo
		home_goods_store	

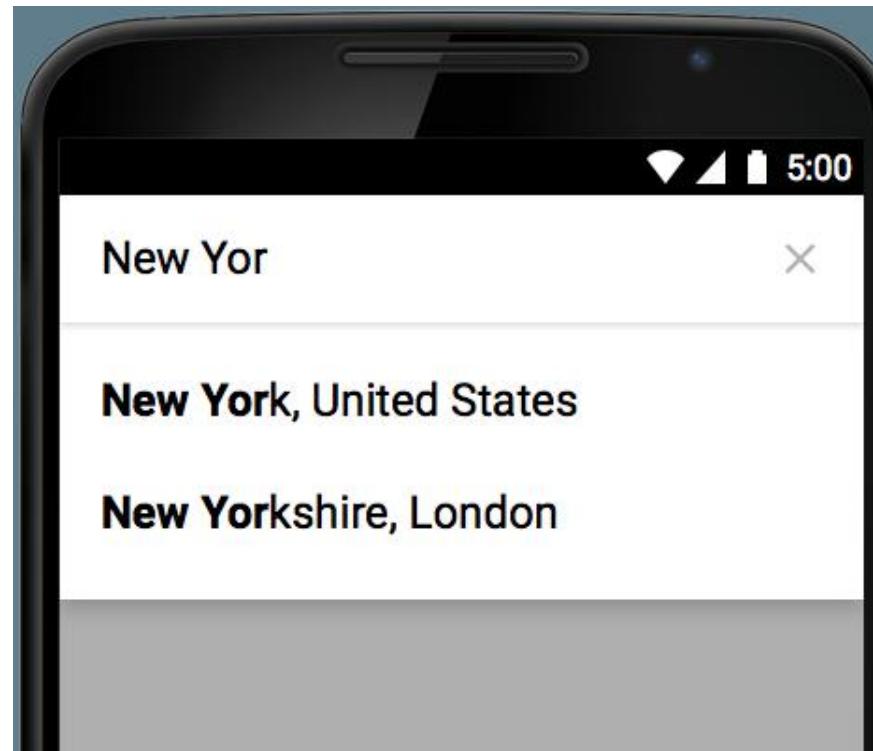
Google Places API Overview

- **Use Place picker UI:** allows users select place from “possible place” on a map
- **Get current place:** place where device is last known to be located
 - Returns **list** of likely places + likelihood device is in that place



Google Places API Overview

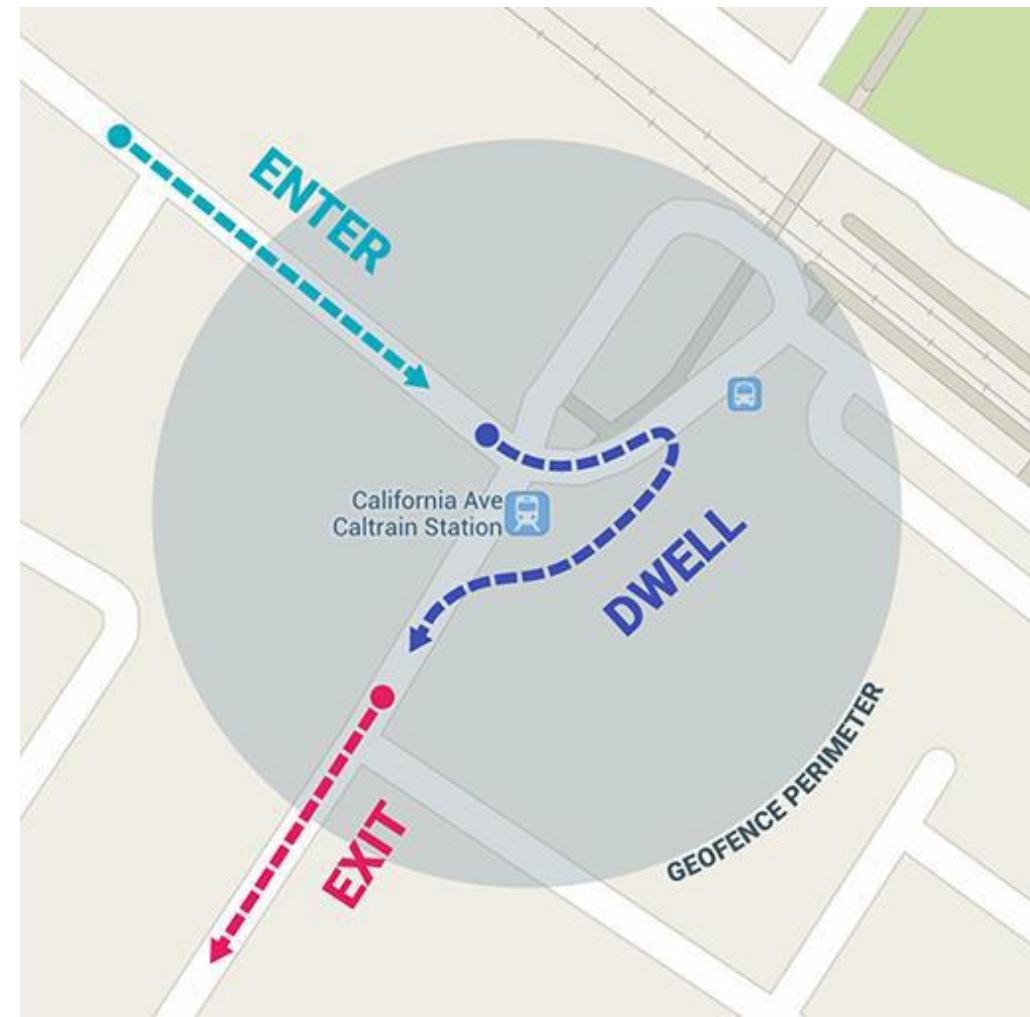
- **Autocomplete:** queries the location database as users type, suggests nearby places matching letters typed in



Other Useful Google Maps/Location APIs

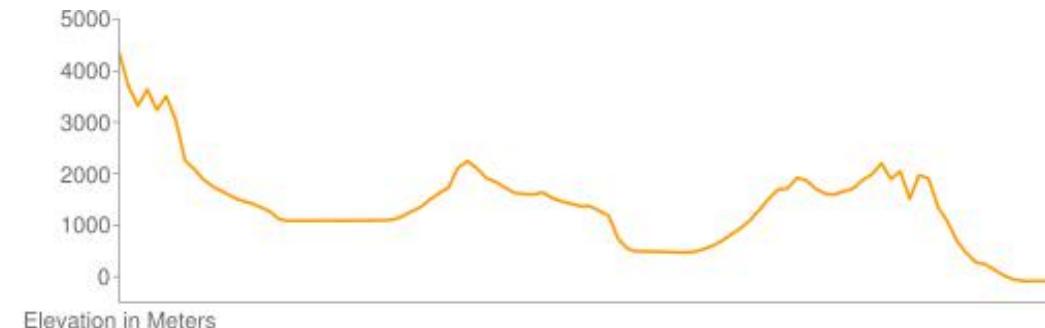
GeoFencing

- **Geofence:** Sends alerts when user is within a certain radius to a location of interest
- Can be configured to send:
 - **ENTER** event when user enters circle
 - **EXIT** event when user exits circle
- Can also specify a duration or **DWELL** user must be in circle before triggering event



Other Maps/Useful Location APIs

- **Maps Directions API:** calculates directions between locations (walking, driving) as well as public transport directions
- **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



Other Useful Maps/Location APIs

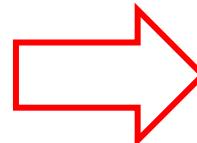
- **Roads API:**
 - sends set of GPS coordinates to road user was likely travelling on (best fit)
 - Returns posted speed limits for any road segment (premium plan)
- **Time Zone API:** request time zone for location on earth

GPS Clustering & Analytics

Determining Points of Interest from GPS Location Sequences

- **Points of Interest:** Places where a person spends lots of time (e.g. home, work, café, etc)
- **Given a sequence GPS <longitude, latitude> points,** how to infer points of interest
- **General steps:**
 - Pre-process sequence of GPS points (remove outliers, etc)
 - Cluster points
 - Convert to semantic location

LATITUDE	LONGITUDE
35.33032098	80.42152478
35.29244028	80.42382271
35.33021993	80.45339956
35.35529007	80.45222096



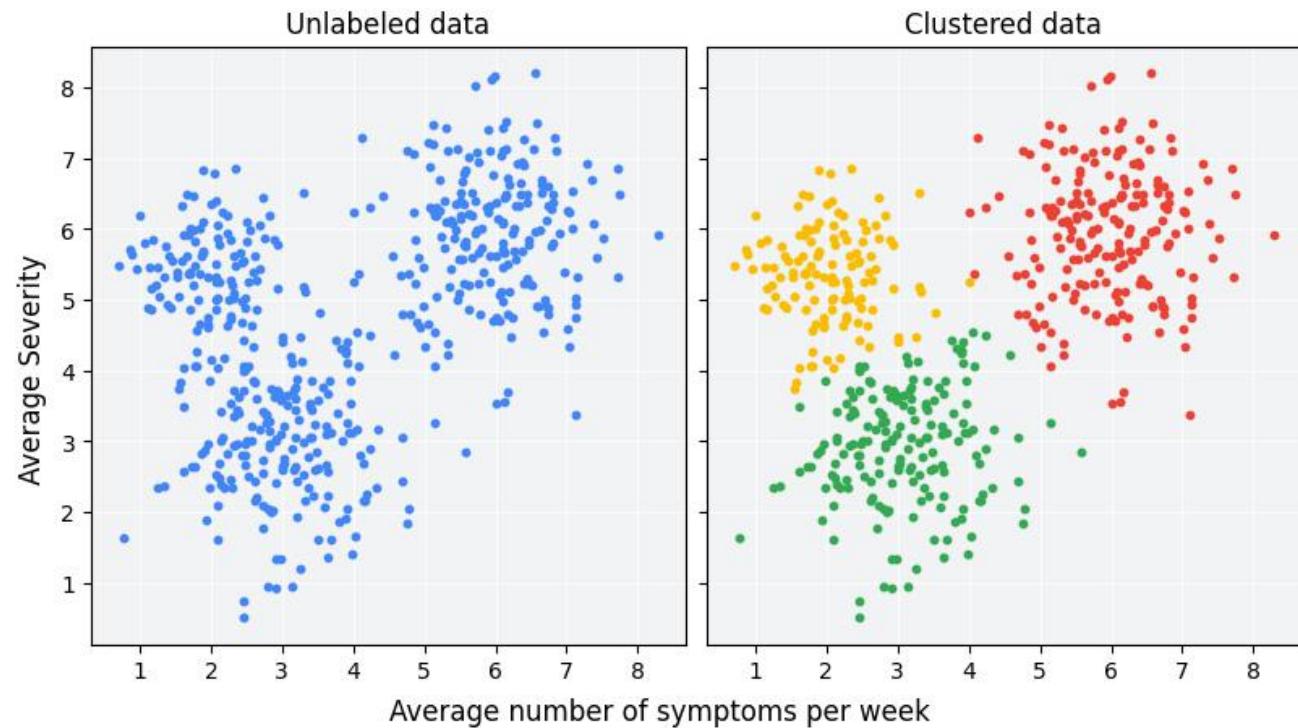
Step 1: Pre-Processing GPS Points (Remove Noise and Outliers)

- **Remove low density points (few neighbors):**
 - i.e. places where little time was spent
 - E.g. radius of 20 meters, keep only clusters with at least 50 points
 - If GPS coordinates retrieved every minute, only considering places where you spent at least 50 minutes
- **Remove points with movement:**
 - GPS returns speed as well as <longitude, latitude> coordinates
 - If speed user is moving, discard that GPS point
- **Reduce data for stationary locations:**
 - When user is stationary at same location for long time, too many points generated (e.g. sitting at a chair)
 - Remove some points to speed up processing

Step 2: Cluster GPS Points

- **Cluster Analysis:** Group points

- Two main clustering approaches
 - K-means clustering
 - DBSCAN



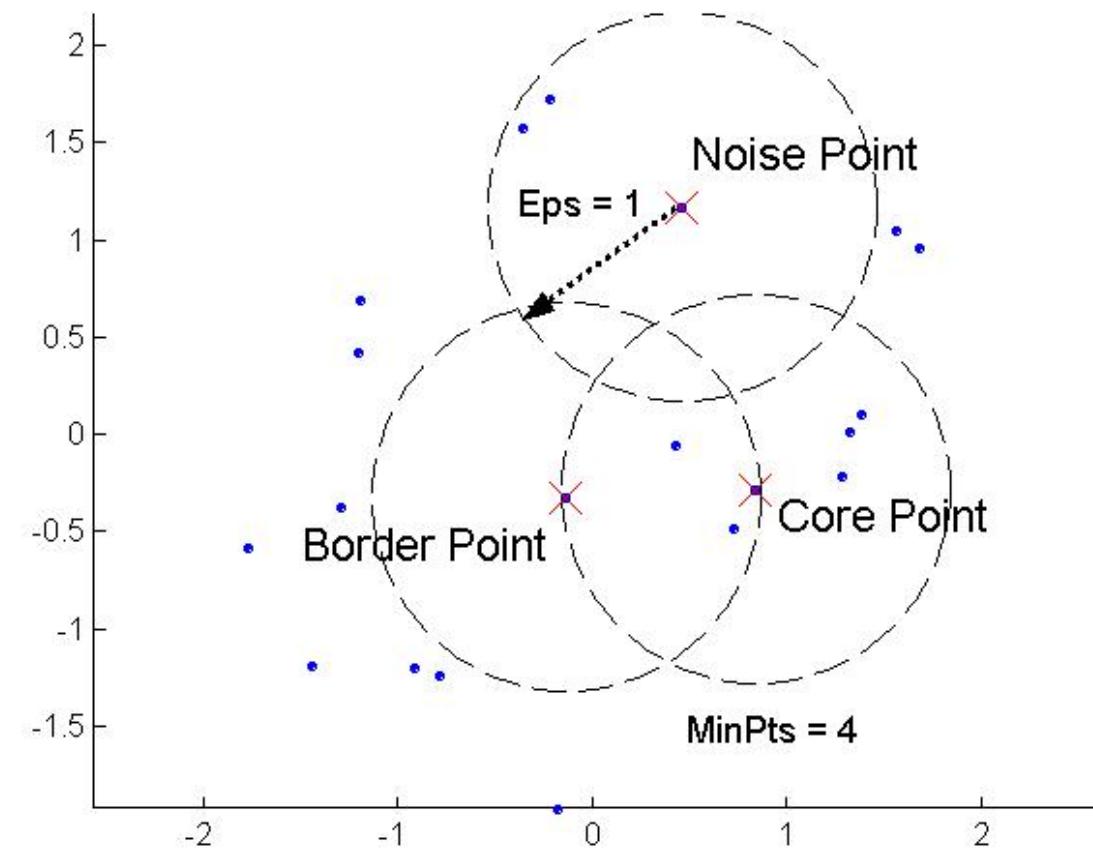
K-Means Clustering

- Each cluster has a center point (centroid)
- Each point associated to cluster with closest centroid
- Number of clusters, K , must be specified

-
- 1: Select K points as the initial centroids.
 - 2: **repeat**
 - 3: Form K clusters by assigning all points to the closest centroid.
 - 4: Recompute the centroid of each cluster.
 - 5: **until** The centroids don't change

DBSCAN Clustering

- Density-based clustering
- **Density:** Number of points within specified radius (Eps)
- **Core points:** has $> minPoints$ density
- **Border point:** has $< minPoints$ density but within neighborhood of core point
- **Noise point:** not core point or border point



DBSCAN Algorithm

- Eliminate noise points
- **Cluster remaining points**

```
current_cluster_label ← 1
for all core points do
    if the core point has no cluster label then
        current_cluster_label ← current_cluster_label + 1
        Label the current core point with cluster label current_cluster_label
    end if
    for all points in the  $Eps$ -neighborhood, except  $i^{th}$  the point itself do
        if the point does not have a cluster label then
            Label the point with cluster label current_cluster_label
        end if
    end for
end for
```

Converting Clusters to Semantic Locations

- Can simply call reverse geocoding or Google Places on the centroid of the clusters
- Determining work? Cluster where user spends longest time most time (9-5pm)
- Determining home? Cluster where user spends most time 6pm –6am

CSE 162 Mobile Computing

Lecture 16: Context-Aware Computing and SQL

Hua Huang

Context Awareness in Mobile Systems

Ubicomp - Physical World Computing



„In the 21st century the technology revolution will move into the everyday, the small and the invisible...“

Mark Weiser (1952 – 1999), XEROX PARC

“Ubiquitous Computing enhances computer use by making computers **available throughout** the physical environment, while making them **effectively invisible** to the user”

- Ubicomp: a field on a physical world richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in everyday objects of lives and connected through a continuous network.
 - Mark Weiser in his last article in IBM Sys. Journal, 1999

What are contexts

- A context represents the state or situation in the environment of a system that affects that system's (application specific) behaviour

Types of Contexts, a systematic view

- Physical Contexts
 - What
 - Where
 - When
- Computing System Contexts
 - How
- User Context
 - Who

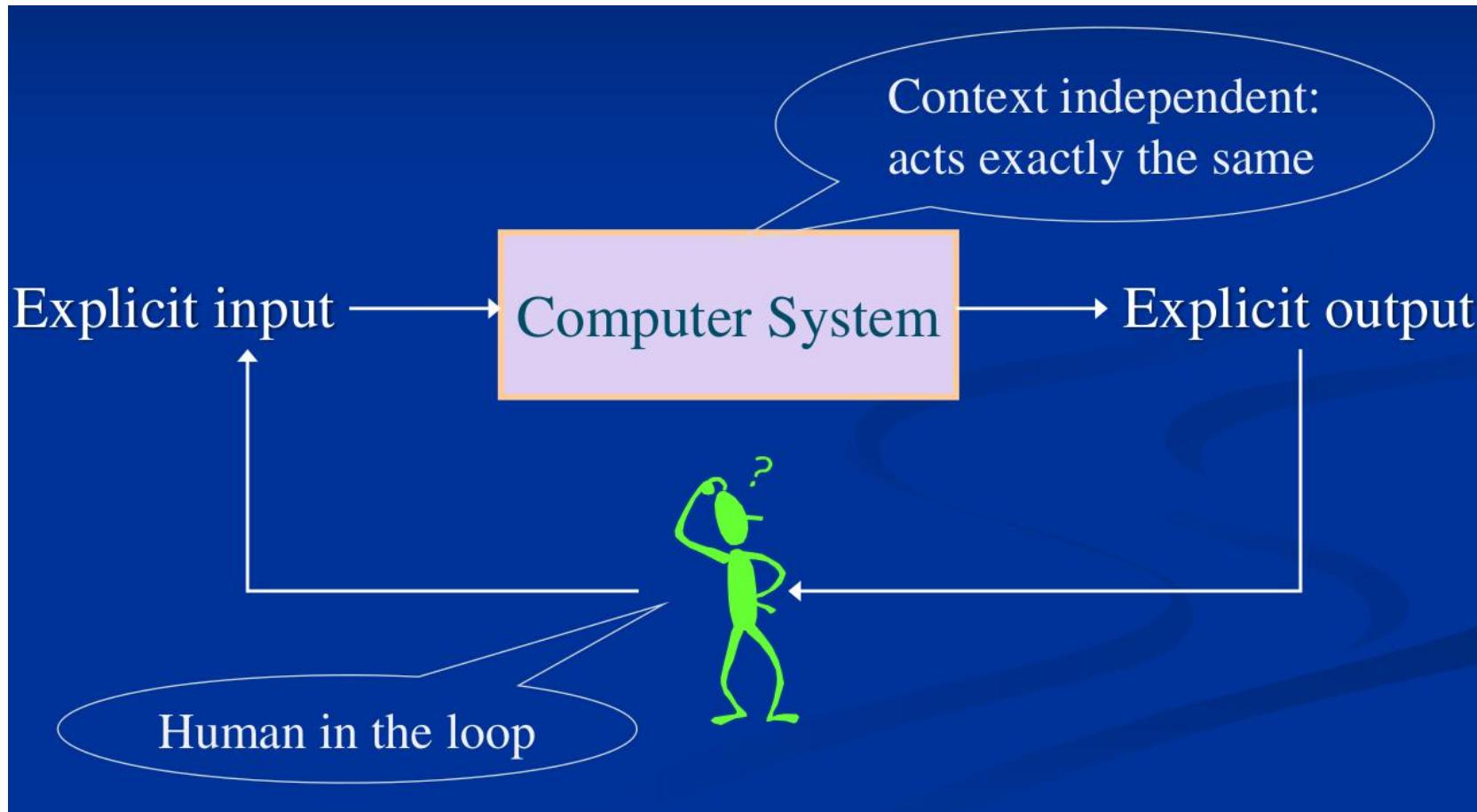
Physical Contexts: What, Where, and When

- Physical environment or phenomena
 - Temperature, light intensity, or chemical
- Location
 - Absolute or logical locations
- Time
 - Absolute or logical time

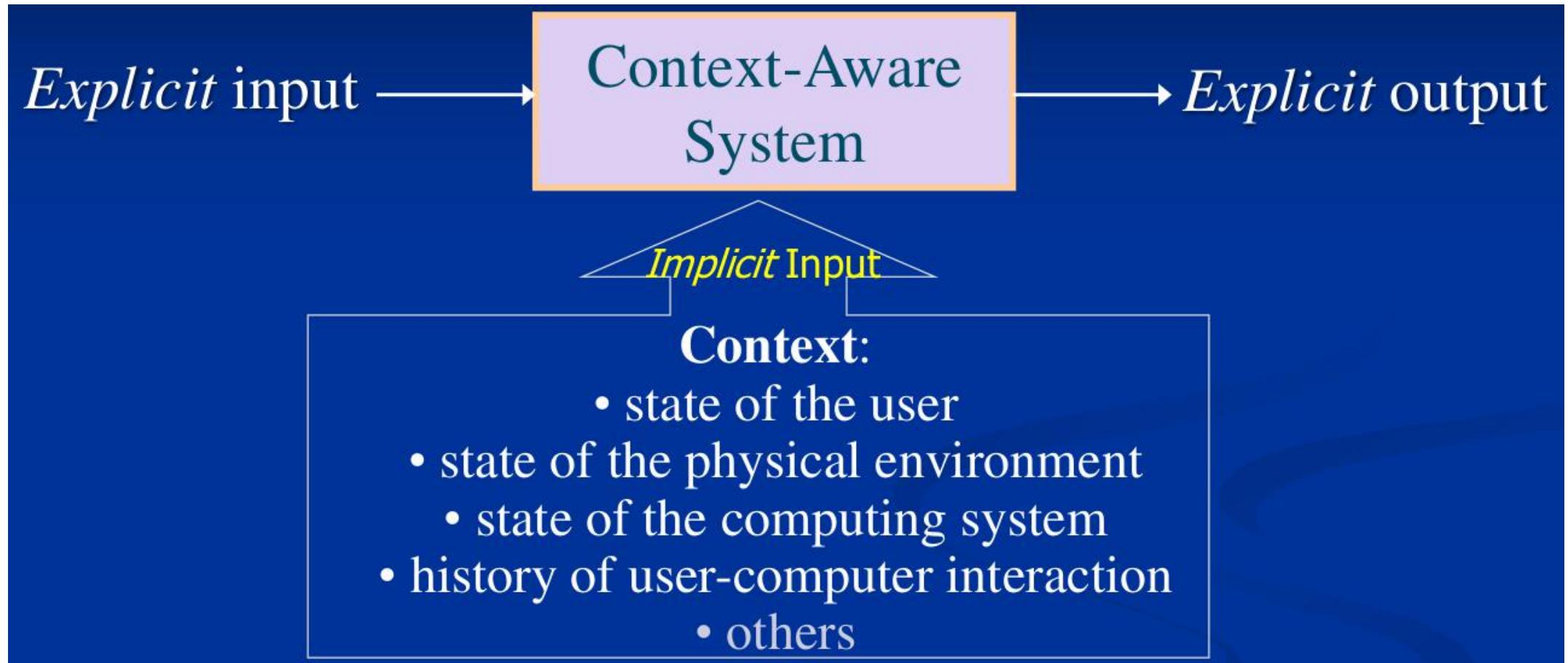
What is Context Aware Computing

- General awareness of the surrounding environment in which the user is operating, located, or situated
 - Context examples: user's preferences, likings, dislikes, location, etc
- Context-awareness is considered to be one of the fundamental properties of ubiquitous computing systems and is a key property of smart environments.

Traditional View of Computer Systems



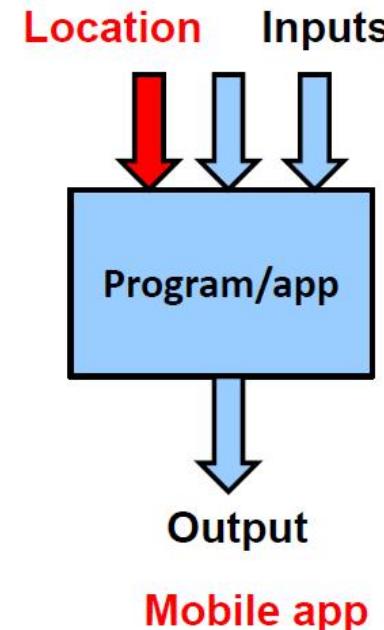
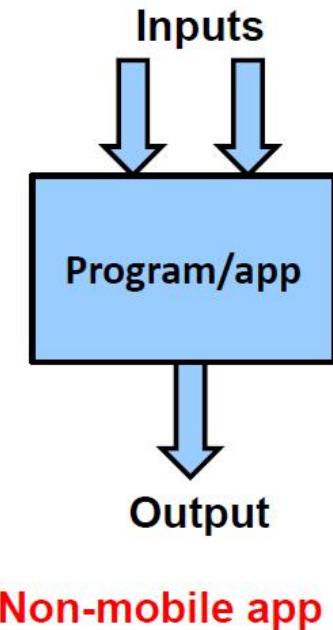
Context as Implicit Input/Output



Context-aware computing examples

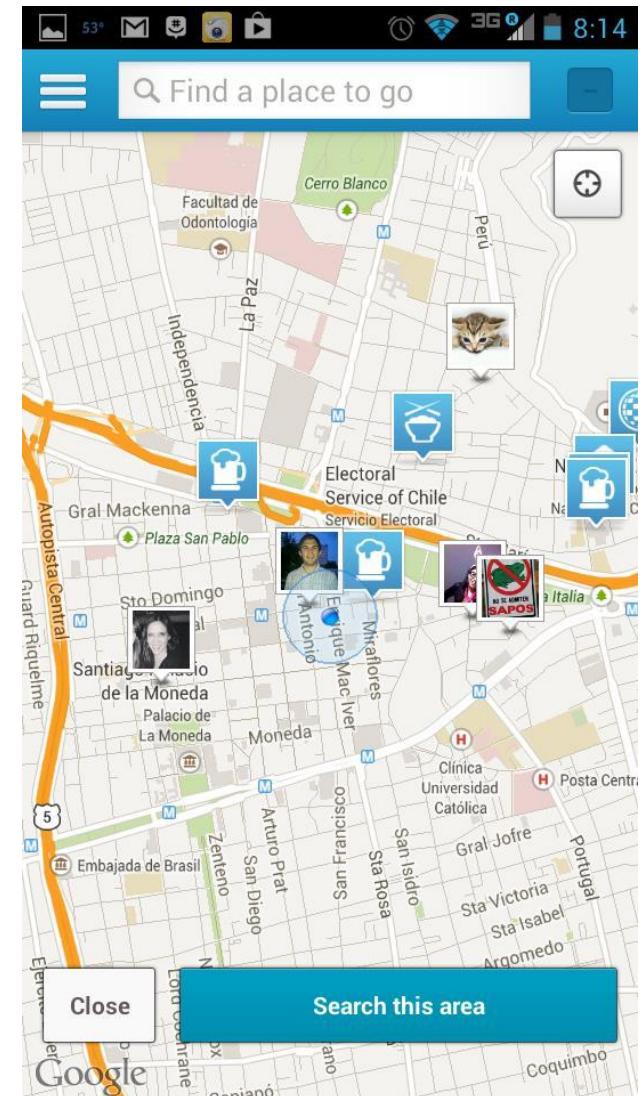
Example: Location-aware computing

- **Location-aware:** Location must be one of app/program's inputs
- Different user location = different output (e.g. maps)



Location-Aware Computing

- Examples:
 - Map of user's "current location"
 - Print to "closest" printer
 - Apps that find user's friends "closeby"
 - Reviews of "closeby" restaurants



Reactive vs. Proactive LBS

- LBS can be either Reactive (“pull”) or Proactive (“push”)
- A Reactive LBS application is triggered by the user who queries the system in search of information based on the current location
- **Reactive LBS examples**
 - Finding restaurants or places of interest
 - Obtaining directions
 - Obtaining weather information
 - Sending emergency notifications to police, insurance companies, roadside assistance companies, etc.

Reactive vs. Proactive LBS

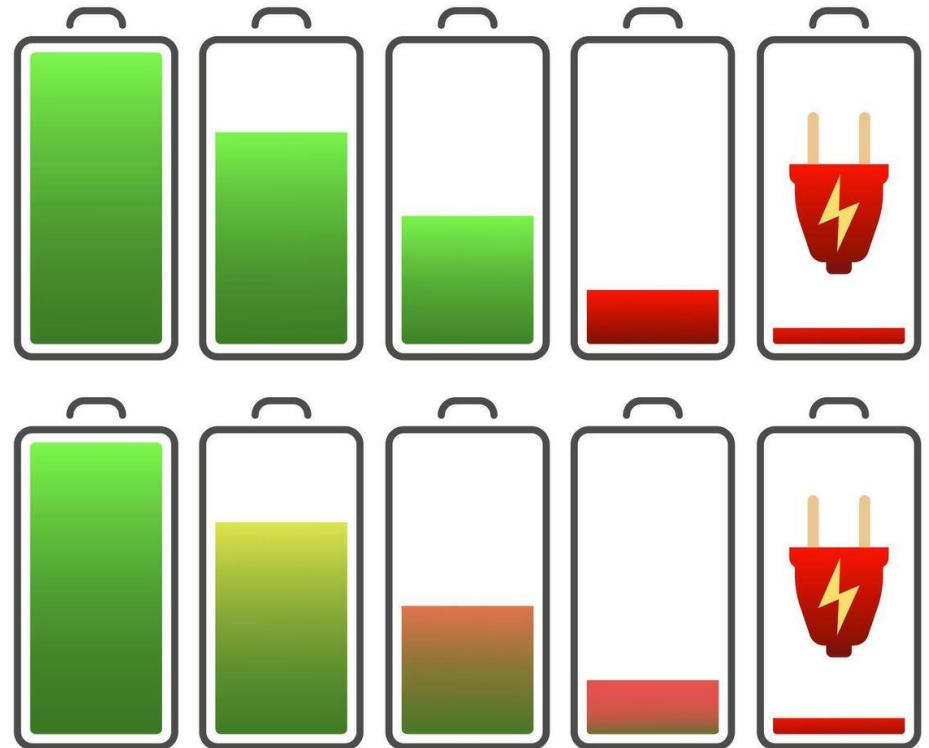
- In Proactive LBS applications, the system needs to continuously know where you are.
- Localization queries or actions are automatically triggered once a predefined set of conditions are met
- **Proactive LBS** examples
 - Geofencing, e.g., children outside predefined boundary
 - Fleet management
 - Real-time traffic congestion notifications
 - Real-time friend finding
 - Proximity-based actuation
 - Travel assistant device for riding public transportation, tourism, museum guided visits, etc

Questions: Proactive or Reactive?

- Location-based advertisement
- Payment based on proximity (EZ pass, toll watch)
- Play a radio station in local area
- App shows grocery list when near Walmart

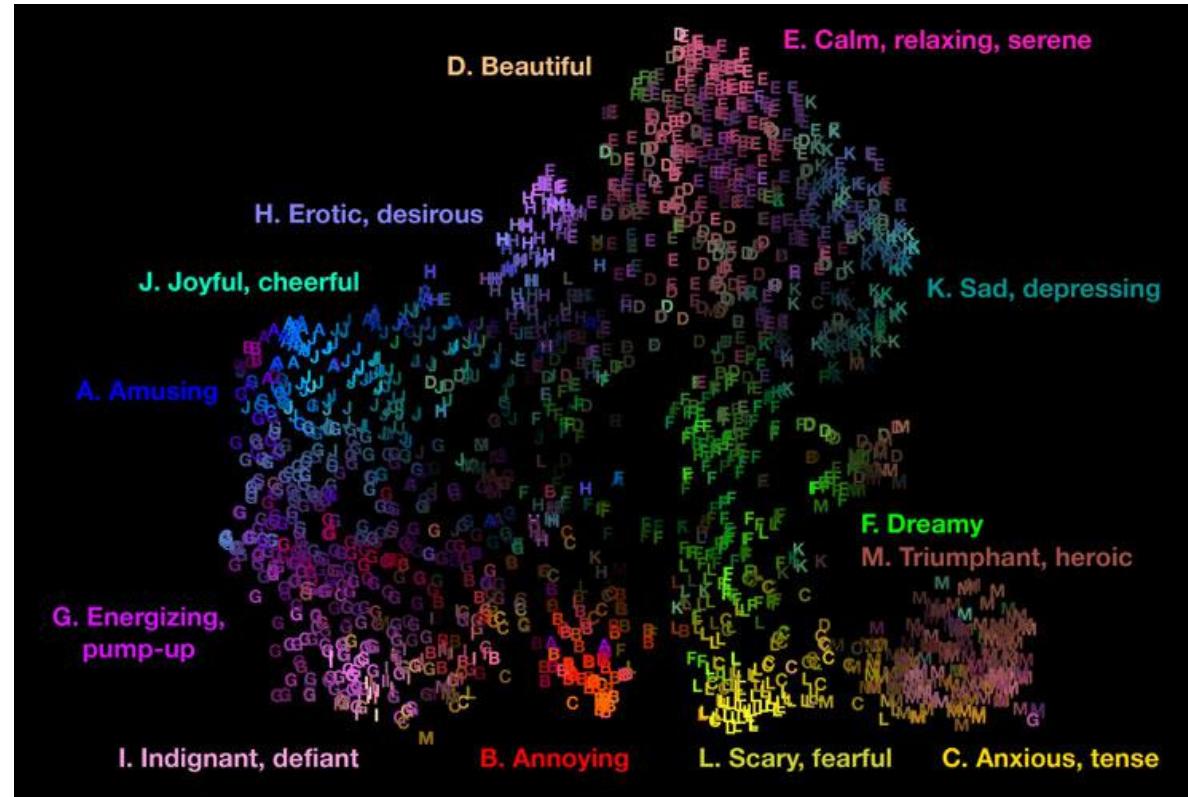
System Context

- System Awareness
 - how any context is created and adapted over an information and computing infrastructure
 - E.g., Wireless connectivity, Charging status



User context

- Consider a smart music player that adapts to user's mood



User Context

- Personal Context
 - **Preference.** E.g., a referee at a sports activity may prefer to blow the whistle for minor versus major sports offences.
 - **Identity.** E.g., owner vs guest
 - **Activity and Task.** E.g., running vs standing
- Social Context
 - how the actions of someone may affect others
 - E.g., who blows the whistle? The referee, policeman, or spectator?

Static versus Dynamic CA

- Static context
 - describes those aspects that are invariant
 - E.g., date of birth, User preference, home location, etc
- Dynamic context:
 - Information that changes change frequently
 - E.g., user activities, current locations, etc

Three Levels of Interactions for Context-Aware Infrastructure

1. Personalization: users recognizes the context and decide how the system should behave.
2. Passive context awareness: the system provides the user with context information; however the system does not automatically change behavior based on this.
 - Instead, the user decides on the course of action based on the updated context information.
3. Active context awareness: the system recognizes the context and takes required actions.
 - It offloads the work from the user by taking active decisions.

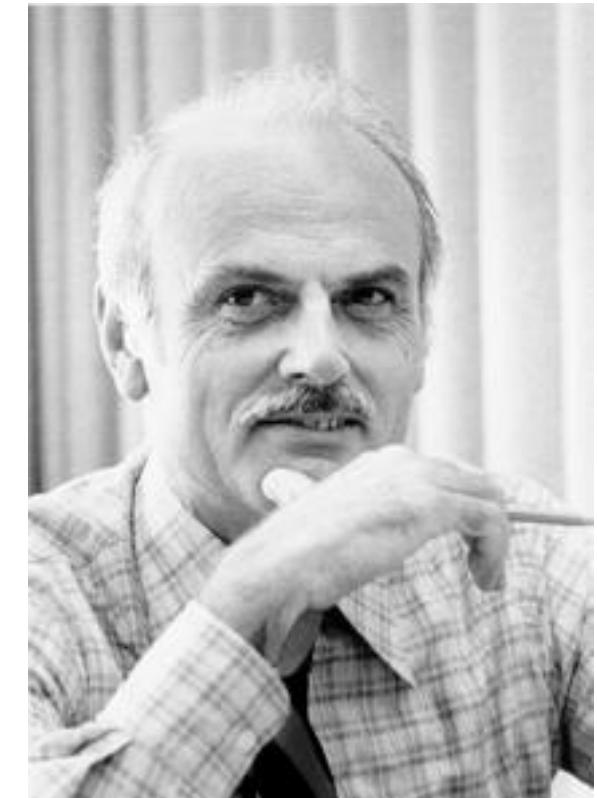
Discussion: Which CA Type?

1. Personalization
 - A smart device uses location-based services to suggest dining locations, entertainment centers in the area, or even emergency services like hospitals and urgent care centers.
 - A thermostat that changes temperature based on schedule
 - The phone automatically disables notification when driving
 - A smart watch could automatically adjust daylight savings or time zone based on the location context.
2. Passive Context Awareness
3. Active Context Awareness

Mobile Database

Databases

- RDBMS
 - relational data base management system
- Relational databases introduced by
 - E. F. Codd
 - Turing Award Winner
- Relational Database
 - data stored in tables
 - relationships among data stored in tables
 - data can be accessed and viewed in different ways



Example Database

- Wine and region tables

Winery Table

Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

Region Table

Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

Relational Data

- Data in different tables can be related

Winery Table

Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

Region Table

Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

Keys

- Each table has a key
- Column used to uniquely identify each row

KEYS

Winery Table			
Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

Region Table		
Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

SQL and Databases

- SQL is the language used to manipulate and
- manage information in a relational database
- management system (RDBMS)
- SQL Commands:
 - **CREATE TABLE** - creates new database table
 - **ALTER TABLE** - alters a database table
 - **DROP TABLE** - deletes a database table
 - **CREATE INDEX** - creates an index (search key)
 - **DROP INDEX** - deletes an index

SQL Commands

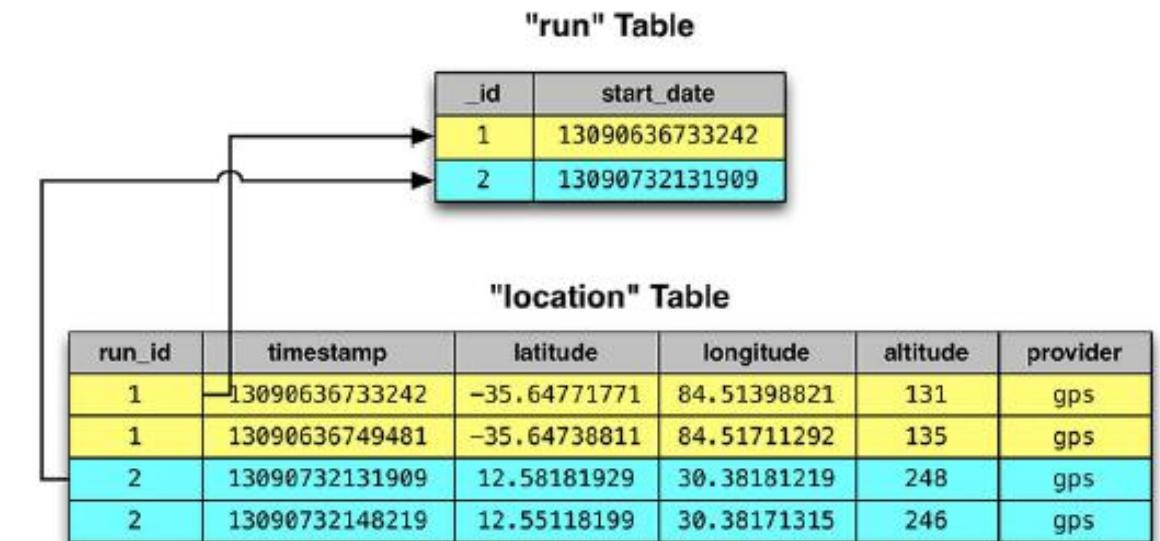
- **SELECT** - get data from a database table
- **UPDATE** - change data in a database table
- **DELETE** - remove data from a database table
- **INSERT INTO** - insert new data in a database table
- SQLite implements most, but not all of SQL
 - <http://www.sqlite.org/>

Why Local Databases?

- Data can get large
- Example: a running tracking app
 - What would such an app be like?
- User may track their runs forever
- **Solution:** Store runTracker runs and locations in SQLite database
 - **SQLite** is open source relational database
 - **SQLiteOpenHelper** encapsulates database creation, opening and updating
 - In `runTracker`, create subclass of **SQLiteOpenHelper** called `RunDatabaseHelper`

Use Local Databases in runTracker

- Create 1 table for each type of data
- Thus, we create 2 tables
 - **Run** table
 - **Location** table
- **Idea:** A run can have many locations visited



Create RunDatabaseHelper

- **onCreate:** establish schema for newly created database
- **onUpgrade():** execute code to migrate to new version of schema
- Implement **insertRun(Run)** to write to database

```
public class RunDatabaseHelper extends SQLiteOpenHelper {  
    private static final String DB_NAME = "runs.sqlite";  
    private static final int VERSION = 1;  
  
    private static final String TABLE_RUN = "run";  
    private static final String COLUMN_RUN_START_DATE = "start_date";  
  
    public RunDatabaseHelper(Context context) {  
        super(context, DB_NAME, null, VERSION);  
    }  
  
    @Override  
    public void onCreate(SQLiteDatabase db) {  
        // Create the "run" table  
        db.execSQL("create table run (" +  
            "_id integer primary key autoincrement, start_date integer");  
        // Create the "location" table  
        db.execSQL("create table location (" +  
            " timestamp integer, latitude real, longitude real, altitude real," +  
            " provider varchar(100), run_id integer references run(_id));  
    }  
  
    @Override  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
        // Implement schema changes and data massage here when upgrading  
    }  
  
    public long insertRun(Run run) {  
        ContentValues cv = new ContentValues();  
        cv.put(COLUMN_RUN_START_DATE, run.getStartDate().getTime());  
        return getWritableDatabase().insert(TABLE_RUN, null, cv);  
    }  
}
```

Add ID to Run Class

- Add ID property to **Runs** class
- ID required to
 - Distinguish runs
 - Support querying runs

```
public class Run {  
    private long mId;  
    private Date mStartDate;  
  
    public Run() {  
        mId = -1;  
        mStartDate = new Date();  
    }  
  
    public long getId() {  
        return mId;  
    }  
  
    public void setId(long id) {  
        mId = id;  
    }  
  
    public Date getStartDate() {  
        return mStartDate;  
    }  
}
```

Use RunManager to use the database

Use this method
when a new run
Is STARTED (When
Start button
is Pressed)

Use this method
when a new run
Is RESTARTED

Use this method when
Run Is STOPPED
(When STOP
button is Pressed)

```
private void broadcastLocation(Location location) {  
    Intent broadcast = new Intent(ACTION_LOCATION);  
    broadcast.putExtra(LocationManager.KEY_LOCATION_CHANGED, location);  
    mAppContext.sendBroadcast(broadcast);  
}  
  
public Run startNewRun() {  
    // Insert a run into the db  
    Run run = insertRun();  
    // Start tracking the run  
    startTrackingRun(run);  
    return run;  
}  
  
public void startTrackingRun(Run run) {  
    // Keep the ID  
    mCurrentRunId = run.getId();  
    // Store it in shared preferences  
    mPrefs.edit().putLong(PREF_CURRENT_RUN_ID, mCurrentRunId).commit();  
    // Start location updates  
    startLocationUpdates();  
}  
  
public void stopRun() {  
    stopLocationUpdates();  
    mCurrentRunId = -1;  
    mPrefs.edit().remove(PREF_CURRENT_RUN_ID).commit();  
}  
  
private Run insertRun() {  
    Run run = new Run();  
    run.setId(mHelper.insertRun(run));  
    return run;  
}
```

- Example Usage

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_run, container, false);

    ...

    mStartButton = (Button)view.findViewById(R.id.run_startButton);
    mStartButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.startLocationUpdates();
            mRun = new Run();
            mRun = mRunManager.startNewRun();
            updateUI();
        }
    });

    mStopButton = (Button)view.findViewById(R.id.run_stopButton);
    mStopButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.stopLocationUpdates();
            mRunManager.stopRun();
            updateUI();
        }
    });

    updateUI();

    return view;
}
```

Inserting Locations into Database

- Similar to inserting runs, need to insert locations when **LocationManager** gives updates
- Add **insertLocation** method

```
public class RunDatabaseHelper extends SQLiteOpenHelper {  
    private static final String DB_NAME = "runs.sqlite";  
    private static final int VERSION = 1;  
  
    private static final String TABLE_RUN = "run";  
    private static final String COLUMN_RUN_START_DATE = "start_date";  
  
    private static final String TABLE_LOCATION = "location";  
    private static final String COLUMN_LOCATION_LATITUDE = "latitude";  
    private static final String COLUMN_LOCATION_LONGITUDE = "longitude";  
    private static final String COLUMN_LOCATION_ALTITUDE = "altitude";  
    private static final String COLUMN_LOCATION_TIMESTAMP = "timestamp";  
    private static final String COLUMN_LOCATION_PROVIDER = "provider";  
    private static final String COLUMN_LOCATION_RUN_ID = "run_id";  
  
    ...  
  
    public long insertLocation(long runId, Location location) {  
        ContentValues cv = new ContentValues();  
        cv.put(COLUMN_LOCATION_LATITUDE, location.getLatitude());  
        cv.put(COLUMN_LOCATION_LONGITUDE, location.getLongitude());  
        cv.put(COLUMN_LOCATION_ALTITUDE, location.getAltitude());  
        cv.put(COLUMN_LOCATION_TIMESTAMP, location.getTime());  
        cv.put(COLUMN_LOCATION_PROVIDER, location.getProvider());  
        cv.put(COLUMN_LOCATION_RUN_ID, runId);  
        return getWritableDatabase().insert(TABLE_LOCATION, null, cv);  
    }  
}
```

Continuously Handle Location Updates

- System will continuously give updates
- Need to receive location intents whether app is visible or not
- Implement **dedicated Location Receiver** to insert location
- Inserts location into run whenever new location is received

```
public class TrackingLocationReceiver extends LocationReceiver {  
    @Override  
    protected void onLocationReceived(Context c, Location loc) {  
        RunManager.get(c).insertLocation(loc);  
    }  
}
```

Alternatives to sqlite

- SQLite is low level ("Down in the weeds")
- Various alternatives to work higher up the food chain
- Object Relational Mappers - ORM
- Higher level wrappers for dealing with SQL commands and sqlite databases
- Many ORMs exist

Mobile Crowd Sensing

CSE 162 – Mobile Computing

Hua Huang

Department of Computer Science and Engineering

University of California, Merced

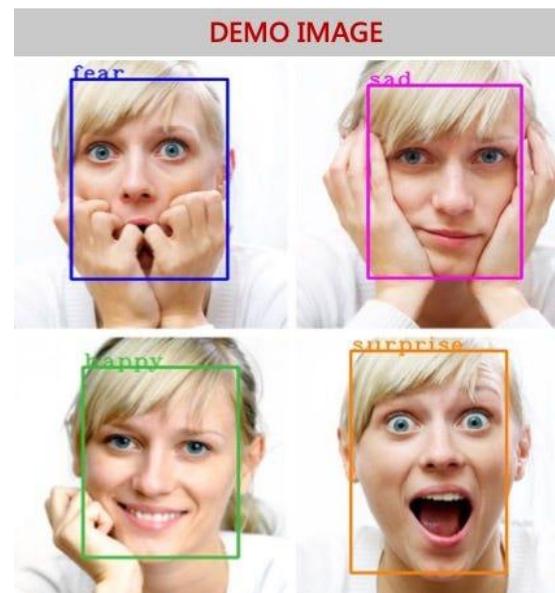
Recap: mobile personal sensing

- **Personal sensing:** phenomena for an individual.
 - Emotions from images or voices
 - Hand movement tracking
 - Activity recognition (smoking)
 - Localization
 - etc



Sensors on iPhone 4:

- Camera
- Audio
- Accelerometer
- GPS
- Gyroscope
- Compass
- Proximity
- Ambient light



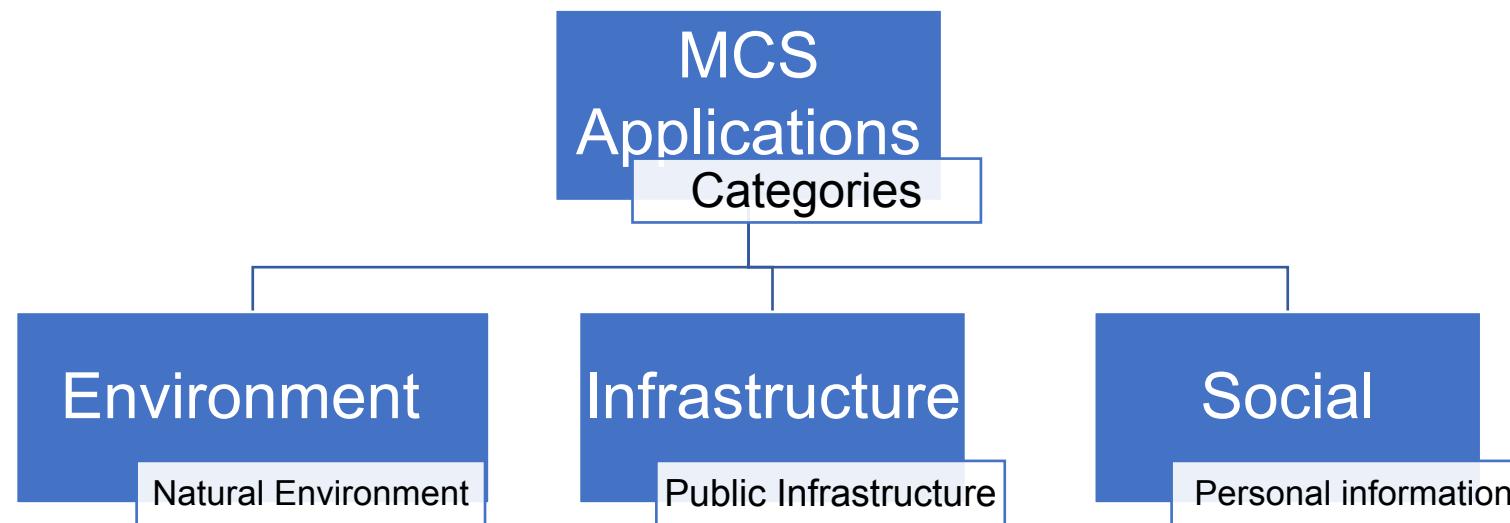
Introduction

- Leveraging on crowd
- Data, services, ideas, contents, skills, money, ... coming from crowds
- Crowdsourcing = Crowd + outsourcing

Introduction

- Pertains to the monitoring of large-scale phenomena that cannot be easily measured by a single individual.
- For example, intelligent transportation systems may require traffic congestion monitoring and air pollution level monitoring.
 - These phenomena can be measured accurately only when many individuals provide speed and air quality information from their daily commutes

Mobile Crowd Sensing Applications



Environmental MCS applications

- Environmental MCS applications,
 - pollution levels in a city
 - water levels in creeks
 - monitoring wildlife habitats

- Example: The **CommonSense** system uses specialized handheld air quality sensing devices that communicate with mobile phones (using Bluetooth) to measure various air pollutants (e.g. CO₂, NO_x).
 - These devices when deployed across a large population, collectively measure the air quality of a community or a large area.

Infrastructure Applications

- Infrastructure Applications involve the measurement of large scale phenomena related to public infrastructure.
 - Traffic congestion
 - road conditions
 - parking availability
 - outages of public works (e.g. malfunctioning fire hydrants, broken traffic lights)
 - real-time transit tracking.

Infrastructure MCS Example

- **CarTel** utilizes specialized devices installed in cars to measure the location and speed of cars and transmit the measured values using public WiFi hotspots to a central server
 - This central server can then be queried to provide information such as least delay routes or traffic hotspots
- **Nericell** utilizes individuals' mobile phones to not only determine average speed or traffic delays, but also detect honking levels (especially in countries like India where honking is common) and potholes on roads.

Social applications

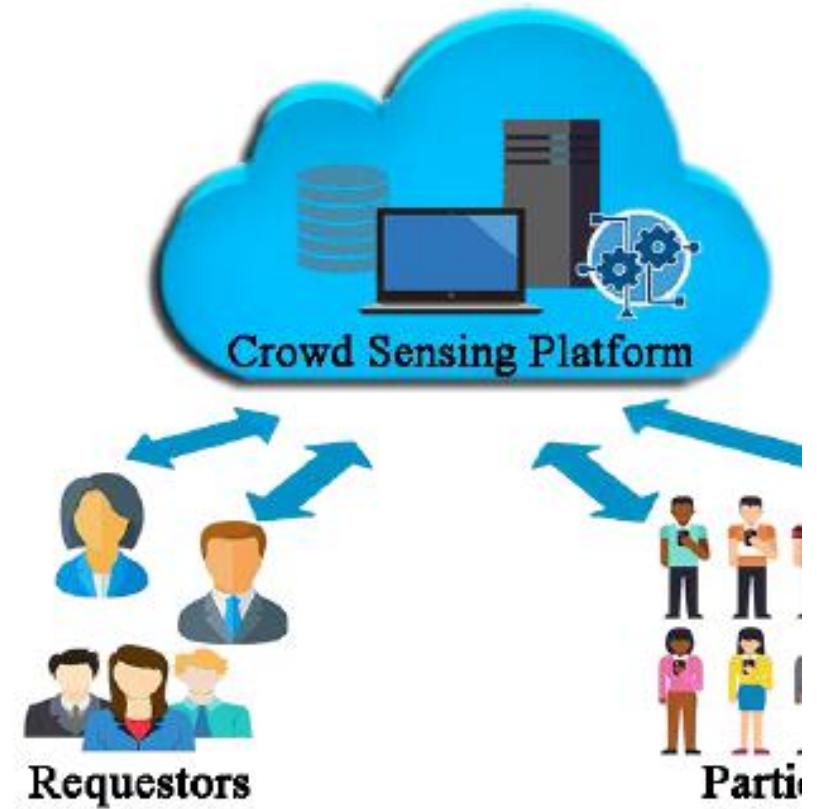
- Social applications involves individuals sharing sensed information amongst themselves.
 - Individuals can share their exercise data (e.g. how much time one exercises in a day) and compare their exercise levels with the rest of the community.
 - **BikeNet**, individuals measure location and bike route quality (e.g. CO₂ content on route, bumpiness of ride) and aggregate the data to obtain “most” bikeable routes
 - **DietSense**, individuals take pictures of what they eat and share it within a community to compare their eating habits.

MCS: Unique Characteristics

- Compared to traditional mote-class sensor networks, mobile crowdsensing has a number of unique characteristics that bring both new opportunities and problems.
 1. today's mobile devices have significantly more computing, communication and storage resources than mote-class sensors, and they are usually equipped with multimodality sensing capabilities.
 2. millions of mobile devices are already “deployed in the field”: people carry these devices wherever they go and whatever they do. By leveraging these devices, we could potentially build large scale sensing applications efficiently
 - For example, instead of installing road-side cameras and loop detectors, we can collect traffic data and detect congestion levels using smartphones carried by drivers.

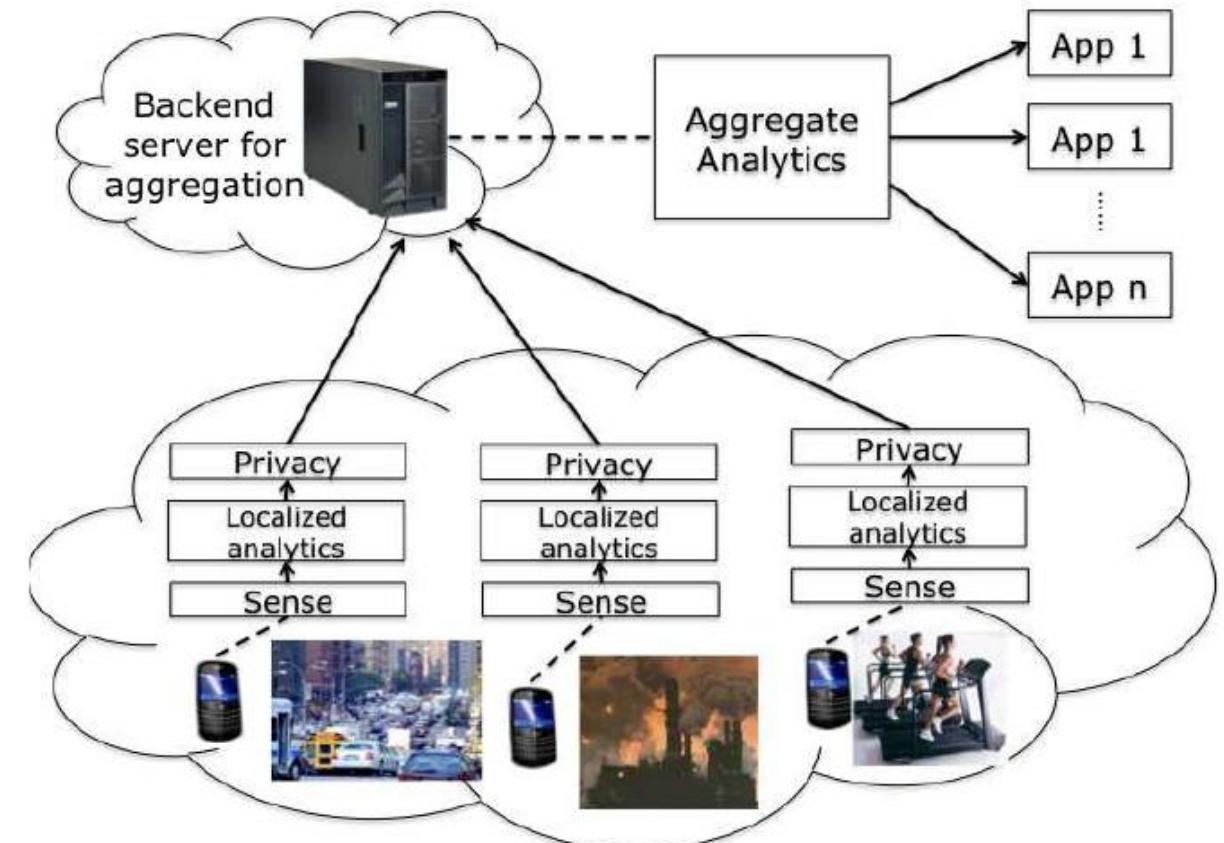
MCS Architecture

- The general architectural design of an MCS system comprises the following main entities:
 - **Requestors**, initiate targeted data collection process by publishing a task pertinent to their interests to the crowd sensing platform and analyze the collected data from the Workers.
 - **Workers** are assigned several tasks pertinent to data collection, thus are the main source of information and play a major role in data collection
 - **Crowd sensing Platform** is the main communication link between Requestors and Workers. The platform stores, processes and analyzes data provided by Workers and the Requestors.



MCS Workflow

- Raw sensing data is collected on devices.
- Local analytics process it to produce consumable data for applications.
- After privacy preservation, the data is sent to the backend.
- Aggregate analytics will further process it for different applications.



Participatory and opportunistic sensing

Participatory and opportunistic sensing

- Participatory sensing requires the **active involvement** of individuals to contribute sensor data (e.g. taking a picture, reporting a road closure) related to a large-scale phenomena.
- Opportunistic sensing is more autonomous and user involvement is minimal (e.g. continuous location sampling without the explicit action from the user).

Participatory Sensing

Opportunistic Sensing

Users actively engage in the data collection activity.

Users manually determine how, when, what, where to sample.

Can avoid phone context issues.

Higher burdens or costs.

Takes random sample which is application defined.

Easy to gather large amount data in small time.

Can't avoid phone context issues.

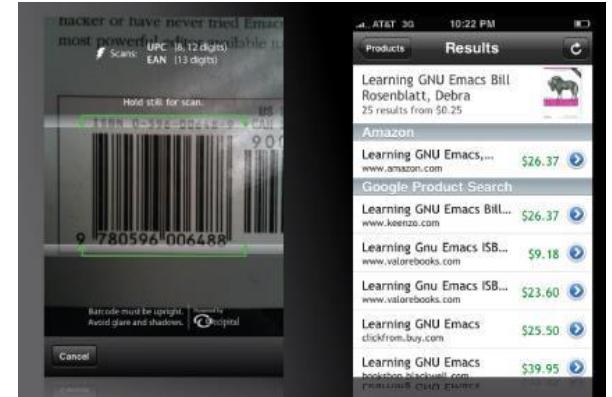
Lower burdens or costs if contextual problems are handled.

Filtering Data by Handling Privacy Issues & Localization.

Dataset is ready for research

Participatory crowdsensing examples

- **LiveCompare**
 - User-created database of UPCs and prices
 - GPS and cell tower info used to find nearby stores
- **PetroWatch**
 - Uses phone to photograph gasoline price
 - Uses GPS to know when gas station is near



Opportunistic crowdsensing examples



- **Pothole Monitor**
 - Combines GPS and accelerometer

The Challenges of MCS...

Incentivize Users

Resource Limitations

Privacy

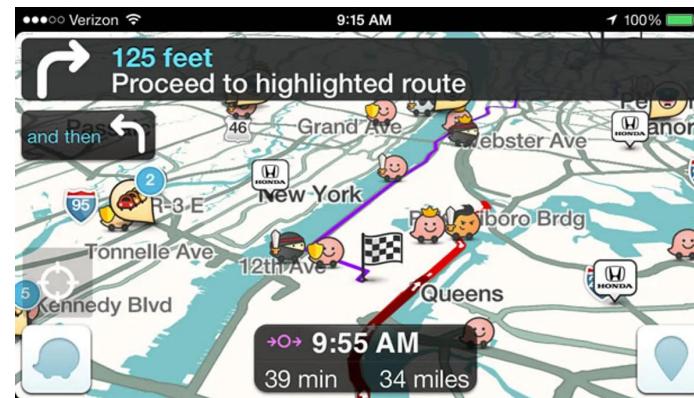
Aggregate Analytics

Challenge: Incentivize users

- Three basic approaches:
 - To make **entertainment an incentive**, crowd sensing tasks are turned into sensing games, such that users can contribute computation or sensing abilities of their mobile devices when they play these games. This paradigm makes users feel enjoyable when they perform tasks
 - The rationality of taking **service as an incentive** roots in the mutual-benefit principle. Service consumers are also service providers. In other words, if a user wants to benefit from the service provided by the system, she also has to contribute to the system.
 - The last category is based on **monetary incentives**. In this case, the system has to pay a certain amount of money to motivate potential participants, such that the participants can use their resources, usually smartphone sensors, to complete the distributed tasks.

examples: Incentivize users

- Entertainment as an incentive
 - Pokeman Go
- Service as incentive
 - Traffic monitoring
- Monetary incentive
 - measure web content usage by paying users money based on page visits to a site.



X Google Opinion Rewards

Question 1 of 3 or fewer:

What is your favorite type of dessert?

Cakes

Pastries

Ice creams

Candies & Chocolates

Cookies

None of the above

[NEXT](#)

Entertainment as an incentive

- Example: mobile game **Treasure** to build WiFi coverage maps of a given game area.
- Players carry mobile devices with GPS and WiFi. They need to pick up virtual coins scattered over the game area and then upload the coins to a server to gain game points. Better network connections give larger probabilities of uploading the collected coins successfully.
- Therefore players are motivated to find areas with stronger WiFi coverage.

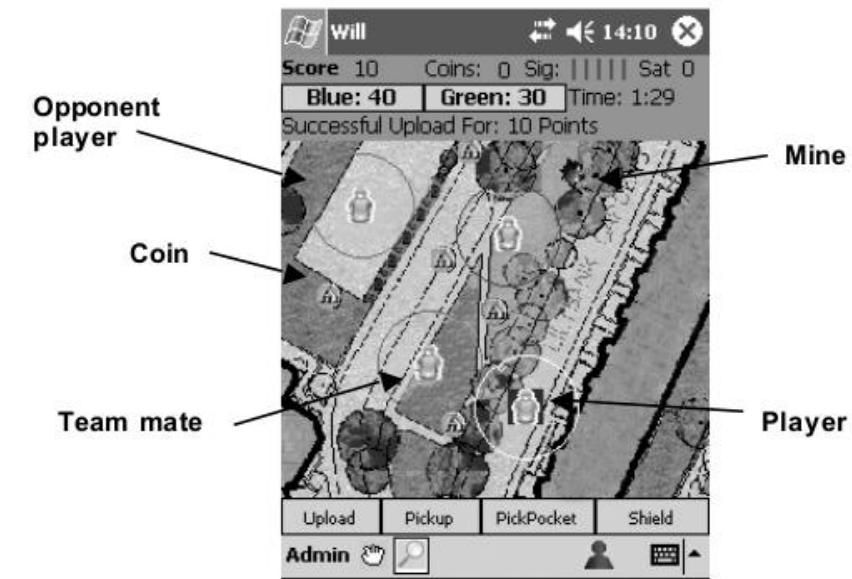
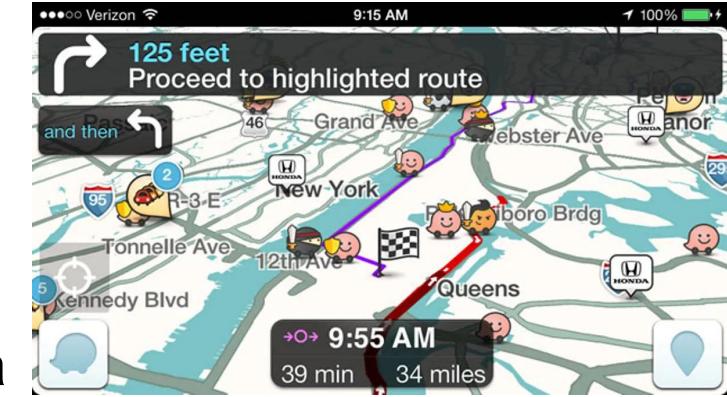


Fig. 1. The game interface. The map shows players' locations, along with coins that are often positioned outside the network. A semi-transparent map layer of green, yellow and red squares builds up as players move around, revealing network coverage, however for printing reasons these have not been included in this figure.

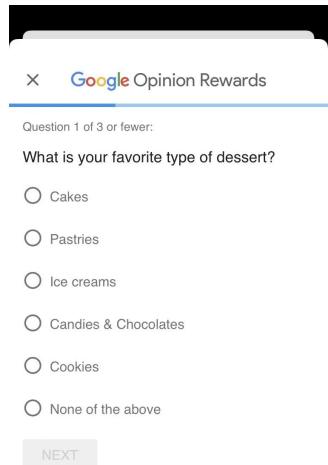
Service as incentive

- For some mobile crowd sensing systems, a participant (e.g., a smartphone user) may have two roles concurrently: a contributor and a consumer.
- Traffic monitoring is a typical example.
 - A participant acts as a contributor when she travels on a bus or car if she collects traffic data (e.g., GPS traces) to a service provider via networks such as WiFi, GPRS, and 3G.
 - The service provider then processes the data crowdsourced from a large amount of users and provides a real-time traffic information service, such as querying on traffic jams and bus crowdedness
 - In such sensing applications, to attract more users to contribute sensed data such that the system can provide services of good quality, the service provider will usually grant a participant some service quota, which determines how much service that user can receive.
 - In essence, this strategy is an exchange of contribution and consumption for each participant.



Monetary incentive

- Paying for sensed data in crowd sensing tasks is the most intuitive incentive, as it has made sensed data become goods in a free market. Any user who would like to make some money can sell her sensed data for crowd sensing tasks.



The image shows a screenshot of a Google Opinion Rewards survey. At the top right, there is a close button (X) and the text "Google Opinion Rewards". Below that, a question is displayed: "Question 1 of 3 or fewer: What is your favorite type of dessert?". A list of six options follows, each preceded by a radio button:

- Cakes
- Pastries
- Ice creams
- Candies & Chocolates
- Cookies
- None of the above

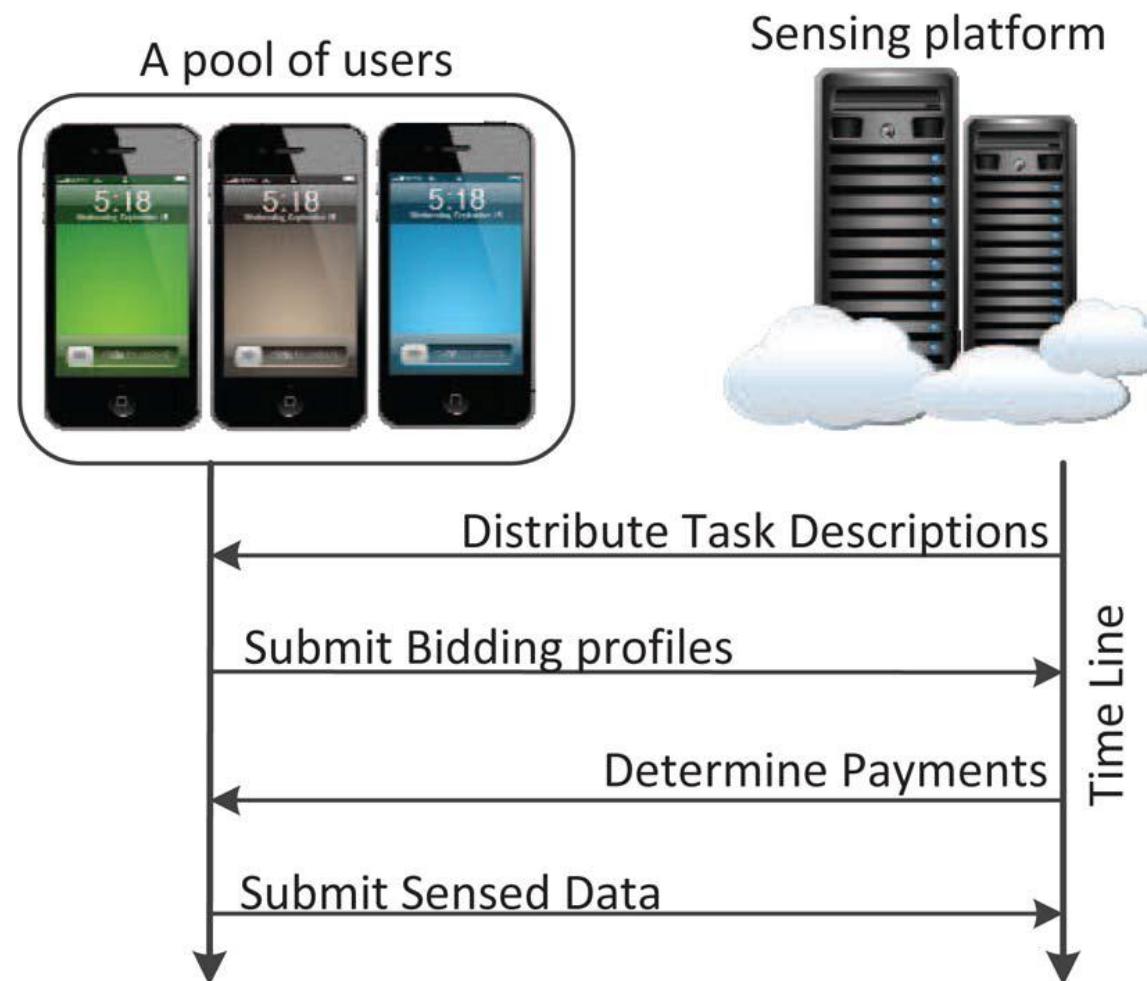
At the bottom right of the screenshot area, there is a "NEXT" button.

Reverse Auction Mechanism

- The system involves two participating roles: a platform that distributes sensing tasks and the mobile phone users who constitute potential labor force.
- The objective is to design a task assignment and payment negotiation scheme, which ensures that both the platform and users are satisfied

Reverse Auction Mechanism

- The platform initiates one round of task distribution by sending task descriptions.
- A set of n users are assumed to be interested in the sensing tasks after receiving the requests.
- If users participate in sensing tasks, they will consume multiple resources, including computation, communication, and energy. Thus it is rational for a user to expect certain profit based on her cost and sensing plan (e.g., sensing time).
- A participating user then submits a bidding profile (including a bidding price and a sensing plan) to the platform.
- After collecting all bidding profiles from the n users, the platform selects a subset of them and determines the payments for them.
- Finally, the selected users perform the assigned tasks and upload the sensed data to the platform.



The Challenges of MCS...

Incentivize Users

Resource Limitations

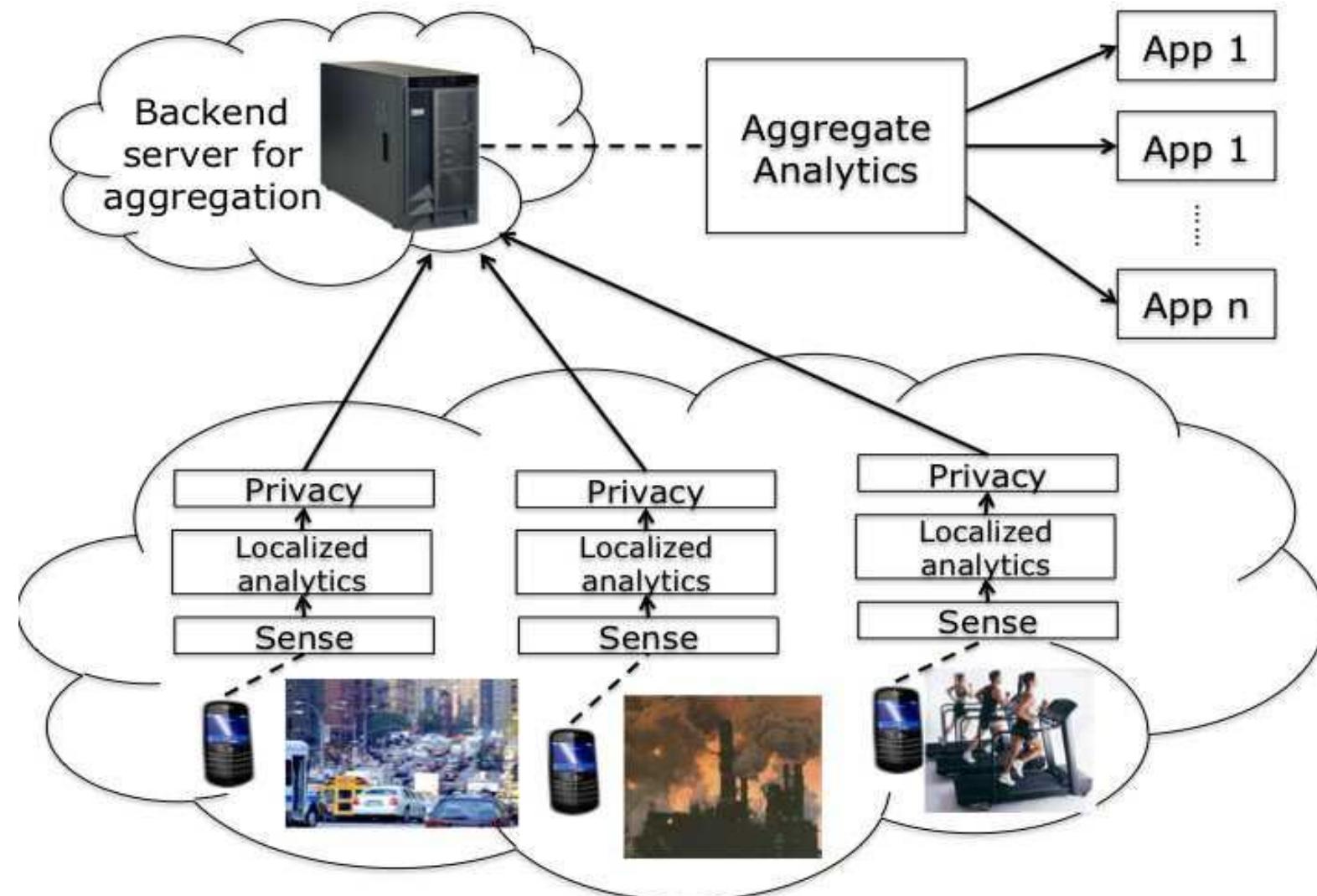
Privacy

Aggregate Analytics

Challenge: Resource limitations

- How to run the MCS application without significantly affecting the normal device operation?
- How to avoid overloading the server?
 - Single point failure problem

LOCALIZED ANALYTICS



LOCALIZED ANALYTICS

- Upload less amount and appropriately processed data
 - *Data mediation*, such as filtering of outliers, elimination of noise, or makeup for data gaps.
- Reduces the amount of processing that the backend has to perform

MCS: Localized Analytics

- Benefits
 - Transmitting data summary consumes lesser energy and bandwidth than transmitting the raw sensor readings.
 - Do the processing locally saves time.
 - Transmitting raw sensor data on intermittently connected channels can be time consuming.
 - *Example: Context inference*
 - *Transportation mode*
 - Kinetic modes of humans
 - Social settings

The Challenges of MCS...

Incentivize Users

Resource Limitations

Privacy

Aggregate Analytics

Privacy

- Potentially collect sensitive sensor data pertaining to individuals
 - the routes they take in daily commutes
 - their home and work locations

Privacy Preservation techniques

- Anonymization
- Cryptographic techniques
- Data Perturbation

Privacy Preserving Approach: Anonymization

- Remove any identifying information from the sensor data before sharing it with a third party.
 - Removes any identifying information from the sensor data before sharing it
 - Drawback: anonymized GPS (or location) sensor measurements can still be used to infer the frequently visited locations of an individual and derive their personal details.

Privacy Preserving Approach: cryptographic techniques

- Cryptographic techniques are used to transform the data to preserve the privacy of an individual.
 - Example: privacy leakage from text auto-completion
 - Example: suggested location of interest
 - are compute intensive and are not scalable because they require the generation and maintenance of multiple keys, which also leads to higher energy consumption.



Conformal encryption.
Talk about it.

Privacy Preserving Approach: Data Perturbation

- Add noise to sensor data before sharing it with the community to preserve privacy of an individual
 - Requirement on the noise: Preserve the privacy of an individual, while not influencing the statistics accuracy
- For example, in a weight watchers application, it is important to compute the average weight of the population.
 - Add a random number to the weight
 - When these values are averaged, the randomized component vanishes and the average weight of the community remain accurate

The Challenges of MCS...

Incentivize Users

Resource Limitations

Privacy

Aggregate Analytics

Challenge: Aggregate Analytics

- MCS applications rely on analyzing the data from a collection of mobile devices, identifying spatio-temporal patterns.
- There are two possible approaches for data mining.
 - Offline approach
 - Online approach
- Consider the restaurant recommendation app based on user ratings

Offline Aggregate Analytics

- One is a traditional approach where data is stored in a database first and then one can apply various mining algorithms against the database to detect patterns.
- However, if the application requires fast detection of patterns, this approach will not work.
- e.g., how to quickly recommend the best route considering traffic?

Online Aggregate Analytics

- Stream data mining algorithms take as input continuous data streams and identify patterns, without the need to first store the data.
- These algorithms are less accurate due to incomplete data.
- e.g., do we recommend a lunch restaurant based on the breakfast traffic data?