Advanced Mobile Computing

course

César Torres Huitzil

CINVESTAV-Tamaulipas

May-August, 2012
Outline

1. Inertial proprioceptive devices
2. Multi-touch and gesture interaction
3. Energy in mobile and embedding computing
4. Speech processing: recognition and interaction
5. Mobile application development: WEB vs. native
Introduction

- Smartphones embed some inertial sensors, these devices are capable of autonomously sensing their own motions and orientations and reacting according to this information.
- Sensor attaches directly to the moving body of interest and gives \textit{an output signal proportional to its own motion with respect to an inertial frame of reference}.
- Two types of sensors comprise inertial sensing: \textbf{Accelerometers} sense and respond to translational accelerations; \textbf{gyroscopes} sense and respond to rotational rates.
- Applications: image and video stabilization (gyroscope), drop protection (accelerometers), motion control.
Inertial frame of reference
Inertial sensors

- Inertial sensors are desirable for general motion sensing because they operate regardless of external references, friction, winds, directions, and dimensions. However, inertial systems are not well-suited for absolute position tracking.

- In such systems, positions are found by *integrating, over time, the signals of the sensors as well as any signal errors*, thus position errors accumulate.

- Inertial systems are most effective in motion-sensing applications involving relative motion or for short-duration position-tracking applications.
Recent advances in microelectromechanical system (MEMS) technologies have enabled inertial sensors to become available on the small size and price scales associated with such commonplace devices as consumer appliances.

In smartphones, these sensors enable a wide range of applications, many of which have already been implemented:

- Automatic portrait or landscape image selection based on the phone’s orientation
- Gesture-based dialing and text writing
- Interactive gaming where actual body motions are used
Inertial sensing

- Inertial sensing is accomplished with two types of sensors: *accelerometers and gyroscopes*.
- Typically, both of these sensors are sensitive to only one axis of motion.
- Inertial navigation systems (INSs) used in aircraft, spacecraft, and other vehicles are ordinarily based on an inertial measurement unit that consists of a set of *three orthogonal accelerometers and three mutually orthogonal gyroscopes*. 
These inertial devices sense and respond to translational accelerations.

Their outputs need to be integrated once with respect to time to get velocity and integrated twice to get position.

Numerous technologies are used to implement today’s accelerometer designs, including piezoelectric, piezoresistive, and capacitive technologies.

While there are a variety of types of accelerometers, the most common design for handheld consumer applications uses differential capacitance transducers.
Accelerometers fundamentals

- **Accelerometers sense linear motion** by measuring the shift in the proof mass when a force is applied. This output can be integrated once to provide velocity, and again for position.

- When the accelerometer is stationary and an axis of the accelerometer is vertical relative to the Earth, the output of that axis is not zero; instead, it will be one $g$, due to the displacement of the proof mass by gravity.

- When the accelerometer is rotated 90 degrees with the sensitive axis horizontal to the earth, the output will be zero.
Accelerometer proof mass
Accelerometer basic equation

- Accelerometers react to many types of movement, including linear and centripetal acceleration, gravity and vibration. Different algorithms can be used to extract measurements of tilt, position, vibration, shock and free-fall.

- Position is found with the translational kinematic equation
  \[ x = x_0 + v_0 t + \frac{1}{2} a t^2 \]
  where \( x \) equals position, \( v \) equals velocity, and \( a \) equals acceleration, the output of an accelerometer.
Processing and error in accelerometers

- When used as a linear acceleration sensor in navigation systems or user interfaces, the corruption of the signal due to gravity is important.
- When the output signal is integrated twice to provide linear position, the final error due to gravity will be amplified. This error can be corrected using gyroscopes and some digital processing algorithms.
- In applications where the accelerometer is used as a tilt sensor, other algorithms will attempt to isolate the signal due to gravity and use that as an orientation reference.
A common compromise is to low-pass filter the accelerometer to isolate tilt. This makes a correct assumption that, in human movement, linear accelerations will be transient, and can be filtered out.

A better solution is to combine the accelerometer with a gyroscope; this provides a rapid, accurate, high bandwidth tilt sensor. For applications in which dynamic tilt is desired, but the fixed reference of gravity is not necessary, gyroscopes can be used without accelerometers.
High-pass and low-pass filtering

- A high-pass filter is used for an application that is looking for rapid changes by weighting the current reading more than the previous reading. A good use case for a high-pass filter is detecting a shake motion.

- A low-pass filter is used for an application that is looking for fine-tuned, stable values. This type of filter would be used in a game of precision, such as carefully pushing an object toward a target.
Gyroscope

- Gyroscopes, also called angular rate sensors, measure how quickly an object rotates. This rate of rotation can be measured along any of the three axes: X (roll), Y (pitch) and Z (yaw).
- Gyroscopes were single axis and were specific for measuring rotation along an axis parallel to the circuit board, or an axis perpendicular to it.
- The design of a gyroscope is inherently more challenging than a MEMS accelerometer due to the need for generating oscillating or resonating mechanical motion with minimal off-axis motion.
Gyroscope reference
The orientation of an object, given a sensed rotational rate, $\omega$, during each time step, $t$, is given by:

$$\theta = \theta_0 + \omega t$$

where $\theta$ equals the orientation angle, and $t$ equals the time step. The output of a gyroscope is the rotational rate $\omega$.

Gyroscope errors increase linearly with time, via the above equation, and are therefore typically less “harmful” than accelerometer errors.
Gyroscope technologies

- Four of the so-called “six degrees of freedom” in inertial sensing are well served by commercially available 3-axis accelerometers and Z-axis gyroscopes. However, the last two degrees of freedom, X and Y rotation, have eluded a cost-effective solution until now.

- There are two main branches of gyroscope design: mechanical gyroscopes that operate using the inertial properties of matter, and optical gyroscopes that operate using the inertial properties of light.
Inertial sensor fusion

Diagram showing the process of inertial sensor fusion. The diagram includes:

- Input signals from 3 gyros: $\omega_x$, $\omega_y$, $\omega_z$
- Compute attitude vector
- Output signals from 3 accelerometers: $a_x$, $a_y$, $a_z$
- Axis transform
- ‘Gyro torquing’ signals
- ‘Standard’ I.N. equations as per gimbaled system
- Output signals: Position, Velocity
Electronic compass

- A electronic device, constructed from magnetometers, that provides heading measurements relative to the Earth’s magnetic north by observing the direction of the Earth’s local magnetic field.

- To convert the compass into an actual north heading, the declination angle is needed, which is position dependent.

- The declination angle is the angle between the geographic and magnetic north.

- Knowledge of the compass position is necessary to calculate the heading relative to the geographic north.
Electronic compass

- The compass is constructed around three magnetoresistive or flux-gate magnetometer, together with pitch and roll sensors.
- Pitch and roll measurements are needed to determine the attitude between the coordinate system spanned by magnetic sensors sensitivity axes and the local horizontal plane so the horizontal component of the Earth’s magnetic field can be calculated.
Coordinate systems

Earth rotation

North

East

Geographic frame

Equator

Latitude

Longitude

Focus

Vernal equinox

Reference meridian

$z^i \equiv z^c$

$x^i$

$x^c$

$y^i$

$y^c$
Characteristics of inertial sensors

- **Accuracy**: the degree of conformity of information concerning position, velocity, etc., provided by the navigation system relative to actual values.

- **Integrity**: a measure of the trust that can be put in the information from the navigation system, i.e., the likelihood of undetected failures in the specified accuracy of the system.

- **Availability**: a measure of the percentage of the intended coverage area in which the navigation system works.

- **Continuity of service**: the system’s probability of continuously providing information without nonscheduled interruptions during the intended working period.
Inertial error sources

- There are several error sources associated with inertial sensors that must be considered. Some of the most significant inertial sensor errors can be categorized as follows:
  - *biases*: the bias error occurs as nonzero output from the sensor for a zero input.
  - *scale factors*: the uncertainty in linear scaling between the input and output.
  - *nonlinearities*: the uncertainty in nonlinear scaling between the input and output.
  - *noise*: any unwanted signal.
Outline

1. Inertial proprioceptive devices
2. Multi-touch and gesture interaction
3. Energy in mobile and embedding computing
4. Speech processing: recognition and interaction
5. Mobile application development: WEB vs. native
Multi-touch and gesture definition

- Multitouch is a technology that is capable of detecting one or more touches (typically via a finger) and movements on a surface simultaneously.
- A gesture is a series of touches recognized as a pattern and registered as a single event.
Multi-touch technology

- *Resistive technology* requires two flexible sheets, electrically conductive, with vertical and horizontal lines for precision location. The sheets are separated by an air gap or microdots.

- When pressed firmly, they make contact and a change in the electrical current is registered and sent to a controller. This technology is very accurate and fairly inexpensive, but it does not support multitouch and does not work well with gestures.
Multi-touch technology

- **Capacitive technology** uses a surface made of insulator-like glass, also called a ground plane, and coated with a transparent conductor. Voltage is applied to the surface. The finger works as the capacitor that modifies the surface’s electrostatic field—more specifically, the coupling between row and column electrodes.

- The controller can determine the location of the touch from the change in capacitance as measured from the four corners of the surface.
Other technologies, used in nonmobile devices or equipment, include infrared, acoustic digitizer, and in-cell.
Multi-touch technology
Multi-touch technology
Multi-touch technology
Models for gestures

- Start-up grounds for a gesture-as-command fundamentals for Human Computer Interaction.
- The amounts of static and dynamic information that are needed in order to sufficiently and completely describe a gesture.
- Multiple representations that gestures take at the various levels of processing for an interaction system.
Posture definition

- A posture can be defined as a set of measurements $p = (p_1, p_2, ... p_n)$ from given values in a domain, $p_i \in D_i$, that describe the pose of body or body parts at one instant of time.

- The features may be quantitative (continuous, discrete) or qualitative (nominal, ordinal).

- Interaction is performed using both motion trajectories and postures hence dynamic and static information.

- A classification of gesture commands with regards to their structural pattern, i.e. the amount of posture / motion considered for performing the command: simple static, complex static, simple dynamic, and complex dynamic gestures.
Gesture taxonomy

- **POSTURES**
  - Simple static
  - Complex static

- **GESTURES**
  - Simple dynamic
  - Complex dynamic

- **MOTION**
**Gesture taxonomy**

- **Simple static gestures** ($g_{ss}$) are gestures that convey the desired information only through the use of a single posture that is maintained for a certain amount of time.
  
  \[ g_{ss} = (\text{posture}, \text{time}) \in P \times [0, \infty) \]

- **Complex static gestures** ($g_{cs}$) are gestures that are represented by a series of consecutive postures which are maintained for certain amounts of time. Again, only posture information is sufficient for grasping the meaning of the gesture command.
  
  \[ g_{cs} = \{g_{ss}^1, g_{ss}^2, \ldots\} \]
Simple dynamic gestures \((g_{sd})\) are gestures for which the posture information is not important as all the meaning lies within the underlying motion trajectory.

A simple dynamic gesture may be defined as a function of time (either continuous or discrete) having as values the coordinates in \(R^d\) of the motion trajectory:

\[
g_{sd}(t) : R \rightarrow R^d\]

where \(sd\) stands for dynamic simple and \(d\) is the dimension of motion coordinates space.
Complex dynamic gestures \((g_{cd})\) are gestures that are represented by a series of consecutive motions which may be separated by periods of pause. Both the motion trajectory and posture are equally important for grasping the meaning of the meaning of the gesture command.

\[ g_{cd} = \{ g^1_{sd}, g^2_{sd}, \ldots \} \]
A gesture $g$ is a sequence of functions of time with values into the Cartesian product of the coordinate’s space $R^d$ and the set of all postures $P$:

$$g = \{g_1(t) : R \rightarrow R^d \times P, g_2(t) : R \rightarrow R^d \times P, \ldots \}$$

The cardinality of a gesture $g$ is given by the number of functions in the gesture set and is denoted by $|g|$.
Gesture architecture

- Gesture toolkit
  - Touch input
  - Gesture overlay
    - Alphabet gestures
      - Letter recognizer
      - Applications semantics
    - User-defined gestures
      - Customizable recognizer
      - Graphical user interfaces
Gesture examples

- The zoom gesture is also referred to as pinching. With this gesture, the user places two fingers on the object, increasing and decreasing the distance between the fingers to scale the object up and down in size.

- Swipe gesture
Gesture examples: pan, press and tap, and rotate

- Pan gesture

- Press and tap gesture

- Rotate gesture
Accelerometer-mobile device interaction

MOBILE DEVICE

GPIO 1 ... GPIO n

ax & ay in g, and t (floating point numbers)

Firmware

G command

ax & ay in duty cycle (4 bytes)

Control Event

Signal Processing 1

Signal Processing n

ax & ay in g, and t after calibration (floating point numbers)

I/O to External World

SENSOR NODE

Microcontroller

ax & ay in PWM

Accelerometer
Accelerometer signal processing
Accelerometer signal profiles
Feature extraction
Defining gestures flow

Sending system command

Idle

[Device button pressed]

[Home-button pressed]

Recording acceleration data

[Button released]

[Option selected]

Display function list for device

[Match positive]

[Match negative]

[Matching done]

[Gesture assigned for the selected device: yes]

Running gesture recognition algorithm

[Gesture assigned for the selected device: no]
Gesture-action binding
Outline

1. Inertial proprioceptive devices
2. Multi-touch and gesture interaction
3. Energy in mobile and embedding computing
4. Speech processing: recognition and interaction
5. Mobile application development: WEB vs. native
Introduction

- Mobile systems, such as smartphones, have become the primary computing platform for a large number of users.
- Several studies identify *longer battery lifetime* as one the most desired features of such systems in conjunction with performance.
- Storage capabilities are becoming less important.
- Many applications are highly computational intensive to be performed on a mobile embedded system: *the computation must be performed externally, for instance in the Cloud.*
- Some other applications might run on the device at the cost of significant amounts of energy.
Energy savings approaches

- Adopt new generations of semiconductor technology: the smaller the transistors, the less power consumption.

- Avoid wasting energy: whole systems or individual components may enter standby or sleep modes to save energy.

- Reduce clock frequency: roughly, if the clock speed is reduced by half, the execution time doubles, but only one quarter of the energy is consumed.

- Reduce on-chip computations: computation is not performed in the device but somewhere else taking advantage of communication facilities.
Offloading computations to save energy

- Sending data and computation to another machine is not a new idea: client-server computing model enable mobile user to launch Web browsers and search Internet, for instance.

- *Cloud computing* is a new paradigm in which computing resources such as processing, memory, and storage are not physically present at the users location.

- *Cloud computing* distinguishing feature is *virtualization*: instead of service providers managing programs running on servers, virtualization allow cloud vendors to run arbitrary applications from different costumers on virtual machines.

- *Cloud computing* might save energy for mobile users through *computation offloading*. 
Suppose the computation (task) requires $C$ instructions, and let $S$ and $M$ be the speed, in instructions per second, of the cloud server and the mobile system, respectively. Consider that $B$ is the network bandwidth, and $D$ as number of bytes interchanged. The mobile system consumes (watts), $P_c$ for computing, $P_i$ while being iddle, and $P_{tr}$ for sending and receiving data.

- The task takes $C/S$ seconds on the server and $C/M$ seconds on the mobile system
- It takes $D/B$ seconds to transmit data.
Energy analysis for computation offloading

- If the mobile system performs the computation, the energy consumption is $P_c \times (C/M)$. If the server performs the computation, the energy consumption is $P_i \times (C/S) + P_{tr} \times (D/B)$. The amount of energy saved is:

$$P_c \times \frac{C}{M} - P_i \times \frac{C}{S} - P_{tr} \times \frac{D}{B}$$

- If the server is $F$ times faster than the mobile system, $S = F \times M$:

$$\frac{C}{M} \times \left\{P_c - \frac{P_i}{F}\right\} - P_{tr} \times \frac{D}{B}$$
Offloading is beneficial when large amounts of computation $C$ are needed with relatively small amounts of communication $D$. 
Computation offloading and the cloud

- Shifting all data in a traditional client-server model: the server does not already contain data, all data must be sent to the service provider. The client must offload the program and data to the server.

- *Cloud computing* changes that assumption. The cloud stores data and performs computation on it.

- As a result a significant change in the value of D for most applications. There is no longer a need to send data over the wireless network, it suffices to send a pointer to the data.
The value of $F$ becomes elastic since a larger number of processors can be obtained on the Cloud. In Cloud computing, Web applications are seen as traditional native stand-alone programs. Computing resources to the cloud can have implications for privacy and security (cryptography, watermarking, steganography).

$$\frac{C}{M} \times \left\{ P_c - \frac{P_i}{F} \right\} - P_{tr} \times \frac{D}{B} - P_c \times \frac{C_p}{M}$$

where $P_c \times \frac{C_p}{M}$ is the additional energy required to protect privacy and security.
Device power management

- The number of functions that an average embedded mobile device executes is increasing rapidly, and the number of I/O devices that an embedded system should control increases accordingly.

- One of the main challenges lies in how to manage power consumption, because mobile devices should operate with limited battery charge.

- Sometimes sacrifice delay or area to reduce power consumption.
Circuit delay and switching power

- Circuit delay depends on voltage $V_{dd}$ and threshold voltage $V_t$:

$$delay = k \times \frac{V_{dd}}{(V_{dd} - V_t)^\alpha}$$

where $k$ and $\alpha$ are constants.

- Switching power consumption, CMOS technology, is given by:

$$P_{switch} = N C_L (V_{dd})^2 f$$

$N$ is the switching activity, $C_L$ the load capacitance and $f$ the frequency.
System-level power techniques

- With increasing demands for low-power techniques, research topics on how to reduce power consumption broadly cover from the circuit/logic level to architecture, software, and system level techniques.
- Among them, system-level power management techniques have been actively studied because, to reduce power consumption, management techniques are often more important than low power design techniques themselves.
- Two of the most commonly applied techniques are *Dynamic Power Management (DPM)* and *Dynamic Voltage and Frequency Scaling (DVFS).*
DVFS power technique

- To reduce power consumption of embedded processors, hardware-based DVFS techniques are widely accepted.
- In DVFS, different computation and communication tasks are run at different voltages and clock frequencies in order to fill up the idle periods in the schedule, while still providing an adequate level of performance.
DPM technique

- To reduce power consumption of I/O devices, a Power Management Unit (PMU) with DPM capability is often employed.
- DPM is a well-known technique that tries to shut down unused devices to reduce power consumption. One of the major problems of the DPM is that waking up a sleeping device may take a long time.
- If the system cannot tolerate such a long wake-up delay, the system should not shut the device down.
- Maintaining a device at an idle state may cause significantly more power consumption than shutting the device down.
Software-based power management

- Managing DVFS and DPM relies on system software such as operating systems. Software-based power-aware management has the merit of flexible control, but it has a potential problem of suffering from significant runtime overhead to decide how to manage effectively.

- A significant fraction of the software and resource usage of a handheld computer system is devoted to its graphical user interface (GUI).

- Many modern applications are interactive in nature, necessitating new DVS/DPM techniques that take into account the user’s perspective.
Experimental setup for power measurement
Smartphone power consumption profile

![Graph showing power consumption over time during different stages: Power button pressed, Phone vibrates, Booting, Idle with dim backlight (~0.34W), Standby (~0.01W), Off. The graph includes time in seconds on the x-axis and power in Watts on the y-axis.]
Smartphone power consumption profile: idle state
Smartphone power consumption profile: wifi connection
Smartphone power consumption profile: message size
## Smartphone power consumption: sensors

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Sensor State</th>
<th>HTC Touch Pro State (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Active (mW)</td>
<td>Sleep (µW)</td>
</tr>
<tr>
<td>Accelerometer</td>
<td>0.6</td>
<td>3</td>
</tr>
<tr>
<td>Temperature</td>
<td>0.225</td>
<td>3</td>
</tr>
<tr>
<td>Pressure</td>
<td>1.8</td>
<td>0.3</td>
</tr>
<tr>
<td>Compass</td>
<td>2.7</td>
<td>7.5</td>
</tr>
<tr>
<td>Gyro</td>
<td>19.5</td>
<td>15</td>
</tr>
<tr>
<td>GPS chip (1MIPS CPU required)</td>
<td>214 (acq. state)</td>
<td>5</td>
</tr>
<tr>
<td>Total (with GPS)</td>
<td>238.825</td>
<td>34.1</td>
</tr>
<tr>
<td>Total (without GPS)</td>
<td>24.825</td>
<td>29.1</td>
</tr>
</tbody>
</table>
Low-power interactive systems

- Power consumption is a major concern in mobile computing: *a significant fraction of the software is interactive, not compute-intensive*.
- Over 90 per cent of the time and energy in such systems may be spent waiting for the user input.
- Since displays tend to consume a significant fraction of the total system power in such systems and GUIs mediate the user and the system, GUI is also an important new concern.
Energy-efficient GUI design

- Traditional software power optimization techniques usually reduce system energy during its busy time, but interactive systems waits for the user input most of the time.
- A effective way to reduce system energy is to improve user productivity.
- Studies form psychology have shown that reading speed depends on the GUI layout in addition to visibility.
- Energy efficiency of interactive systems is really the product of power efficiency and user productivity.
Characterizing user’s impact

According to the *model human processor*, three basic processes are involved in the user response time for applications based on GUIs:

- Perceptual capacity
- Cognitive speed
- Motor speed
Perceptual capacity

- Humans read through discrete eye movements, which consist of a series of fixations which are separated by saccades.
- During fixations, the eye stares at a particular point of interest until the next saccade, which is a quick movement to the next fixation.
  - People frequently read text titles before view images related to the text.
  - Proper selection of content placement can improve GUI interaction speed.
  - GUIs with better color schemes and contrast ratios are easier to read: visibility depends on the color scheme, contrast ratios and luminance.
Cognitive speed

- The Hick-Hyman law states that the reaction time, $RT$, required to make a decision based on $N$ distinct and equally possible choices is given by:

\[ RT = a + b \times \log_2 N \]

where $a$ and $b$ are constants

- A GUI should present as few choices as possible: split menus achieve this goal by separating out most common functionality into smaller menus
Motor speed

- The motor speed of human users are governed by the Fitts law presented as the following equation:

\[
T = c_1 + c_2 \times \log_2\left(\frac{D}{W} + 1\right)
\]

where \(T\) is the time required to complete a task (moving to point A to point B), \(D\) is the distance between A and B, \(W\) is the width of the target and \(c_1\) and \(c_2\) are experimentally determined constants.

- GUI should utilize as much screen area as possible for widgets to hit. Widgets that are supposed to be hit sequentially should be placed close to each other.
Mobile GUI design techniques for energy consumption

- Most mobile devices utilize relatively small displays in order to achieve higher portability: GUI design requires a different approach from traditional GUI design.
- Mobile computing device energy efficiency can be improved by better utilizing the limited screen real state.
- The first step is categorize GUIs based on the primary interaction between a system and its user.
- Optimization techniques can be divided into: *power reduction techniques, performance enhancement techniques and facilitators.*
Mobile GUI taxonomy

- **Input-centric**: the primary action performed is obtaining user input and GUls can be subdivided based on the complexity of the input mechanisms.

- **Content-centric**: mostly used for viewing stored data and examples include map software, text viewers and Web browsers.

- **Hybrid**: require important user input while dedicating a significant portion of the screen to displaying data.

The former GUI type should be used for ease of input and the latter for ease of browsing.
## Mobile GUI taxonomy

<table>
<thead>
<tr>
<th>Input-centric</th>
<th>Content-centric</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Simple</strong></td>
<td><strong>Web browsing</strong></td>
<td><strong>Text editor</strong></td>
</tr>
<tr>
<td><strong>Calculator</strong></td>
<td><strong>Movie viewing</strong></td>
<td><strong>Email</strong></td>
</tr>
<tr>
<td><strong>Complex</strong></td>
<td><strong>Map software</strong></td>
<td><strong>Address book</strong></td>
</tr>
<tr>
<td><strong>Text messages</strong></td>
<td><strong>Text viewing</strong></td>
<td><strong>Configuration</strong></td>
</tr>
</tbody>
</table>
Mobile GUI power techniques

- **Power reduction**: reduce the power consumed by the display subsystem.
- **Performance enhancement**: reduce energy by providing user interaction speed.
- **Facilitators**: do not reduce energy directly, instead they enable other techniques to be used more effectively.
# Mobile GUI optimization techniques

<table>
<thead>
<tr>
<th>Technique</th>
<th>Input</th>
<th>Content</th>
<th>Hybrid</th>
<th>Power</th>
<th>Performance</th>
<th>Facilitator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low-energy color scheme</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reduced screen changes</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hot keys</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>User input cache</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Content placement</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Paged display</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Quick buttons</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
Power reduction

- *Low energy color schemes*: reduce energy by using colors and color patterns that consume less power. Fine patterns and textures should be avoided, since they increase power consumption by increasing switching capacity.

- *Reduced screen changes*: reduce energy by reducing switching activity as well as computation required for screen data generation.

- *Scrollbars* are very energy-inefficient since they cause frequent screen updates leading to much more power consumption.
Performance enhancement

- *Hot keys*: enable traditional user inputs via multikey gestures. Quick buttons facilitators enable the operation to be performade in a single gesture.

- *User input caches*: store the most recent or most frequent inputs by users. User input caches are most effective for user input intensive programs and can also benefit from quick buttons.

- *Content placement*: reduces the user interaction time for frequent inputs by strategically laying out the GUI content.
Facilitators

- The facilitator are a pair of techniques that enable or enhance the effects of other technique.
- *Paged display*: enables increased user interface functionality by increasing the effective display size.
- *Quick buttons*: use available hardware buttons to increase user interaction capabilities for the current application.
Power saving color transformations

**Structured transformations**: GUIs are composed of multiple objects, each with properties (size, color, location and color) specified in software. Additionally most platforms support color themes.

- Background/foreground transformation: employ a low-power color for background but colors of high contrast for border and foreground
- Theme-based transformations: platforms support color themes that apply consistent colors to platform specific features.
Power saving color transformations

- **Unstructured transformation**: usually focuses in raw pixels without taking into account the context.
  - **Color counting**: to get the power consumption of the whole display, how many pixels for each color in the GUI must be determined.
  - **Color mapping**: reduce the power consumption by replacing each color of the histogram with a new color.
Formulate the process as an optimization problem. A GUI employs multiple colors $c_1, c_2, \ldots, c_n$ specified by a three component vector $c_i = (R_i, G_i, B_i)$ where $i = 1, 2, \ldots, n$. Let $num_i$ be the number of times that color $i$ appears. The power consumption of the GUI can be calculated as:

$$P_{\text{GUI}} = \sum_{i=1}^{n} num_i \cdot P_{\text{pixel}}(R_i, G_i, B_i)$$

where $P_{\text{pixel}}(R_i, G_i, B_i)$ is the power consumption of a given color. The objective is to find $n$ colors such that power consumption is minimized, while satisfying certain human perception constraints.
Outline

1. Inertial proprioceptive devices
2. Multi-touch and gesture interaction
3. Energy in mobile and embedding computing
4. Speech processing: recognition and interaction
5. Mobile application development: WEB vs. native
Speaker identification and verification system

- Language in both its written and spoken form strongly influences all aspects of human interactions.
- The acoustic signal of human speech characterizes not only what is being said but also embodies individual characteristics of the speaker: pitch and vocal tract resonances as well as speaking styles and durations.
- Speaker recognition is one of the key research areas in signal processing and pattern recognition.
Speech recognition analogy

**STEP 1**
TRANSUDER

- **Ear**

**STEP 2**
SIGNAL PROCESSOR

- **Cochlea**

**STEP 3**
FEATURE EXTRACTOR

- **Auditory Nerve Cochlea and Cerebral Cortex**

**STEP 4**
UTTERANCE CLASSIFIER

- **Left Cerebral Cortex**

- **Computer**
  (Comparison of Reference Messages with Unknown)

**Microphone** ➔ **Spectrum Analyzer** ➔ **Formant Tracker, Phoneme Recognizer (Data Compressor)** ➔ **Computer**
In speaker identification, human speech from an individual, is used to identify who that individual is.

There are two distinct operational phases:

- **Training**: the speech that need to be identified is acquired to build the model for that speaker (acquired before the system is deployed)

- **Testing**: the true operation of the system is carried out where the speech from an unknown utterance is compared against each of the trained speaker models.

The main performance measure of such systems is the identification rate.
Speaker identification

- Although human speech is used to identify or verify an individual it should be remembered that human speech is primarily used to convey meaning by words, text-dependent speaker recognition.

![Diagram of Speaker Identification System]

*Figure 1. Block Diagram of a Speaker Identification System.*
Speaker verification

- In speaker verification human speech from an individual is used to verify the claimed identity of that individual.
  - Initial configuration of the system is carried out during training.
  - In testing the verification takes place when the individual has to make a claim as to who he/she is.
  - The speech of an individual person is compared against both the claimed identity and against all other speakers. The ratio of the two measures is then taken and compared to a threshold.

- *False rejection rate* (FRR), the number of times the true speaker is incorrectly rejected, and *False acceptance rate* (FAR), the number of times an imposter speaker is incorrectly accepted.
Speaker verification

Figure 2. Block diagram of a speaker verification system.
Speech feature extractions

- The most fundamental process common to all forms of speaker and speech recognition systems is that of extracting vectors of features from the acoustic form.
- Vectors of features are *uniformly spaced across* time from the time-domain sampled acoustic waveform (*framing*).
Framing of acoustic waveform

- The initial framing of the acoustic waveform proceeds as follows:
  - Pre-emphasis
  - Framing
  - Windowing

**Figure 3.** Analysis block diagram for framing.
Pre-emphasis

- A high pass filter is applied to the waveform to emphasizes higher frequencies and compensates for the human speech production process that attenuate high frequencies.
- A first order high pass filter is used, with a typical coefficient value of 0.97:

\[ y(t) = x(t) - 0.97x(t - 1) \]

where \( x(t) \) is the input speech data and \( y(t) \) is the output.
Framing

- The time-domain waveform of the utterance under consideration is divided into overlapping fixed duration *uniformly spaced across* segments called *frames*.
- Typical duration values for frames are anywhere from 20 ms to 30 ms (usually 25 ms) and a frame is generated every 10 ms.
- Consecutive 25 ms frames generated every 10 ms will overlap by 15 ms.
Windowing

- Each frame is multiplied by a window function. The window function is needed to smooth the effect of using a finite-sized segment for the subsequent feature extraction by tapering each frame at the beginning and end edges.

- A tapered window function creates a smoother and less distorted (by artefacts) spectrum.

- Any of the window functions used in FIR digital filter design can be deployed, with the Hamming window function being the most popular.
Speech recognition in mobile phones

- Several manufacturers offer today mobile phones with voice interface.
- The growth in this segment is determined by several factors: functionality offered by speech recognition based features, robustness under typical application conditions, implementation costs, and end users’ acceptance.
- Added values: voice control together with hands free operation seems to be very attractive.
- Speech recognition is expected to become one of the keys for mobile Internet access.
To call a person whose name is stored in a phone book, speaker-dependent (SD) name dialing based on dynamic time warping (DTW) algorithm is well suited. DTW based SD technology is frequently used in current products since it is of low complexity and language-independent.

Speaker independent (SI) HMM-based (Hidden Markov Model) technology offers command-and-control and digit dialing, i.e., recognition of commands and phone numbers without requiring a training phase.
Web-based multimodal interaction

- **Applications**
  - Information Search and Mining
  - Customer Care/Help Desk
  - Language Translation
  - Mobile Internet
  - Voice Print Security

- **Content Processing**
  - Speech Processing
  - Web Processing
  - Language Processing

- **Web Data**
  - Music, Video
  - Web Pages
  - Speech, Text

- **Multimedia Outputs** (Voice, Video, and Text)

- **Multimodal Inputs** (Voice, Text, and Gesture)
Web-based voice verification

"Please say your credit card."

Speech Recognition → Speaker Verification → Decision Threshold

Digits Acoustic Model

Kevin Anne Smith

Speakers

1234 4567 3939

Is it Anne?
Mobile application development

- Building a different application for each platform is very expensive if written in each native language.
- The performance argument that native applications are faster may apply to 3D games or image-processing applications, but there is a negligible or unnoticeable performance penalty in a well-built business application using Web technology.
- The use of Web technology to solve the platform fragmentation problem.
## Native code skill sets

<table>
<thead>
<tr>
<th>Mobile OS Type</th>
<th>Skill Set Require</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple iOS</td>
<td>C, Objective C</td>
</tr>
<tr>
<td>Google Android</td>
<td>Java (Harmony flavored, Dalvik VM)</td>
</tr>
<tr>
<td>RIM BlackBerry</td>
<td>Java (J2ME flavored)</td>
</tr>
<tr>
<td>Symbian</td>
<td>C, C++, Python, HTML/CSS/JS</td>
</tr>
<tr>
<td>Windows Mobile</td>
<td>.NET</td>
</tr>
<tr>
<td>Window 7 Phone</td>
<td>.NET</td>
</tr>
<tr>
<td>HP Palm webOS</td>
<td>HTML/CSS/JS</td>
</tr>
<tr>
<td>MeeGo</td>
<td>C, C++, HTML/CSS/JS</td>
</tr>
<tr>
<td>Samsung bada</td>
<td>C++</td>
</tr>
</tbody>
</table>
Native application development

- There are different tools, build systems, APIs, and devices with different capabilities for each platform.
- Native code usually is *compiled*, which is faster than *interpreted* languages such as *JavaScript*.
- In fact, the only thing that operating systems have in common is that they all ship with a mobile browser that is accessible programmatically from the native code.
- Each platform allows to instantiate a *browser instance*, chromeless, and interact with its JavaScript interface from native code. From within that Webview native code from JavaScript can be called.
Web application frameworks

- There are some open source frameworks that provide developers with environments where they can create applications in HTML, CSS, and JavaScript and still call native device features and sensors via a common JS API.

- Some frameworks contain the native-code pieces to interact with the underlying operating system and pass information back to the JavaScript application running in the Webview container.

- Webviews and browsers use HTML and CSS to create user interfaces with varying degrees of capability and success.
Web technology stack

- The Web technology stack (HTML/CSS/JS) is itself implemented in native code. The distance between the native layer and the browser is just one compile away.
- If you want to add a native capability to a browser, then you can either bridge it or recompile the browser to achieve that capability.
- *If a browser does not support a native capability, it is not because it cannot or that it won’t; it just means it hasn’t been done yet.*
Mobile web aspects

- As a consequence of wireless bandwidth and multimedia evolution, the mobile Web no longer has to differ from standard desktop environments.
- In the past, Wireless Applications Protocol (WAP) standards and Wireless Markup Language (WML) content were the choice.
- Mobile content is continuously evolving to a richer experience in which standard HTML pages are delivered and formatted to fit a mobile device screen.
- Web applications, on the other hand, are increasingly popular as interactive HTML experiences.
Mobile web paradigms

**Server transcoding:**
- A mobile client requesting a URL is sent to a proxy server on the Internet. The proxy server, on behalf of the client, makes the request to the encoded server on the URL.
- HTML content retrieved is converted into a special or proprietary format and rendered by the requesting mobile browser. Their rendering engines use server-based computing power to generate compressed image files displayed by the mobile browser.
- Because these images are preprocessed at the server, content is irrelevant to the mobile browser.
- In general, these rendering engines are proprietary and reside on the server. The mobile browser might also include additional compression to support low-bandwidth scenarios.
Mobile web paradigms

- **Direct delivery:**
  - In direct delivery, the mobile device renders HTML directly from a Web server or a local file.
  - A rendering engine—for example, WebKit (www.webkit.org), is embedded as part of the device firmware and in many cases these HTML rendering engines are open source.
Mobile web rendering engines

- Mobile browsers use rendering engines with specialized frameworks that parse HTML tags and draw images, text, and lines on a canvas area allocated for Web content.

- In addition to retrieving content from the Web or local repositories, these engines have become increasingly popular for managing state transitions, providing APIs to control and access XML and HTML content, and properly displaying Web applications containing embedded JavaScript.
WebKit

WebKit has emerged as the engine of choice for the mobile Web. It provides extensive support to HTML 2.0, including features such as Ajax, CSS, JavaScript, and a plug-in framework.

WebKit has been enhanced with touch-based interfaces and zooming capabilities to facilitate viewing and further interface small handsets with full-fledged HTML Web applications.

Additionally, the same Ajax-based support, or Web 2.0 applications, can now asynchronously update personal mobile Web content.
# Mobile web browsers

<table>
<thead>
<tr>
<th>Name</th>
<th>Engine</th>
<th>Zoom in/Out</th>
<th>Touch</th>
<th>Multimedia</th>
<th>Widgets</th>
<th>Server/Cost</th>
<th>Platforms</th>
<th>Narrowband/Broadband</th>
</tr>
</thead>
<tbody>
<tr>
<td>Openwave/Openweb</td>
<td>WAP2-Openwave</td>
<td>No</td>
<td>No</td>
<td>Limited to plug-in support</td>
<td>No</td>
<td>Yes/Low</td>
<td>Windows, Linux, BREW, Real-time Operating System (ROS)</td>
<td>Yes/No</td>
</tr>
<tr>
<td>Opera Mini 4/Skyfire</td>
<td>Presto/Proprietary</td>
<td>Yes</td>
<td>Depends on Java</td>
<td>Depends on Java</td>
<td>Yes, Opera widgets</td>
<td>Yes/Medium</td>
<td>J2ME MIDP</td>
<td>Yes/Yes</td>
</tr>
<tr>
<td>Opera Mobile</td>
<td>Presto/Proprietary</td>
<td>Yes</td>
<td>Planned</td>
<td>Several media supported</td>
<td>Yes, Opera widgets</td>
<td>No/Medium</td>
<td>Windows, Linux</td>
<td>No/Yes</td>
</tr>
<tr>
<td>Openwave full browser</td>
<td>Mercury</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, plug-in framework</td>
<td>Yes, but limited</td>
<td>No/No</td>
<td>Windows, Linux, BREW</td>
<td>No/Yes</td>
</tr>
<tr>
<td>Netfront</td>
<td>Proprietary</td>
<td>Yes</td>
<td>TAP</td>
<td>Yes, multiple formats</td>
<td>Yes, at client</td>
<td>No/No</td>
<td>Windows, Linux, Real-time Executable (REX)/BREW</td>
<td>Yes/Yes</td>
</tr>
<tr>
<td>Google Android</td>
<td>WebKit</td>
<td>Yes</td>
<td>Yes, driver-based</td>
<td>Plug-ins, Android and Linux-based</td>
<td>Yes, at client</td>
<td>No/No</td>
<td>Linux/Android</td>
<td>May not/Yes</td>
</tr>
<tr>
<td>Safari (iPhone)</td>
<td>WebKit</td>
<td>Yes</td>
<td>Multi</td>
<td>RTSP, No Windows formats</td>
<td>Yes, JavaScript</td>
<td>No/No</td>
<td>Mac OS</td>
<td>GPRS/WiFi Yes/Yes</td>
</tr>
</tbody>
</table>
WebViews and the web engines

- These platforms let developers include Web-based interfaces as part of their applications. This is achieved by expanding the “view” concept and redefining it as a “WebView”.
- A WebView is an instrument available to developers which a user can instantiate from any application on the mobile phone, hence any HTML content can become part of any application.
- WebViews let mobile developers create any user interface design using standard Web-based technology.
Scripting languages

- Scripting languages are designed for different tasks than are system programming languages, and this leads to fundamental differences in the languages.
- **System programming languages** were designed for building data structures and algorithms from scratch, starting from the most primitive computer elements such as words of memory.
- **Scripting languages** are designed for gluing: they assume the existence of a set of powerful components and are intended primarily for connecting components.
Scripting languages

- **Typing** refers to the degree to which the meaning of information is specified in advance of its use.
- Scripting languages are typeless to simplify connections among components and provide rapid application development.
- Scripting languages assume that a collection of useful components already exist in other languages. They are intended not for writing applications from scratch but rather for combining components.
- Code and data are often interchangeable, so that a program can write another program and then execute it on the fly.
- Another key difference between scripting languages and system programming languages is that scripting languages are usually interpreted.
Scripting languages

- Is the application’s main task to connect preexisting components?
- Will the application manipulate a variety of different things?
- Does the application include a GUI?
- Does the application do a lot of string manipulation?
- Will the application’s functions evolve rapidly over time?
- Does the application need to be extensible?
Cross-platform development

Application Source Files
(HTML, CSS, Javascript, Ruby, Python)

Your Application
*UI API | *Desktop API | Optional Modules
Bridge to OS - JavaScript - Ruby - Python
OS - Windows / Mac / Linux

Your Application
*UI API | *Phone API | Optional Modules
Bridge - JavaScript - Java / JavaScript - Objective C
OS - Android / iPhone

Titanium Desktop
Native Android App
Native iPhone App

Titanium Mobile
## Cross-platform frameworks

<table>
<thead>
<tr>
<th></th>
<th>Appcelerator Titanium</th>
<th>PhoneGap</th>
<th>Rhodes</th>
<th>AIR for Mobile Devices</th>
<th>OpenPlug Elips Studio</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native UI support</td>
<td>✓</td>
<td>!</td>
<td>✓</td>
<td>!</td>
<td>!</td>
<td>PhoneGap, AIR: 3rd party libraries required</td>
</tr>
<tr>
<td>Native code support [1]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>!</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Development language</td>
<td>Javascript (HTML, CSS)</td>
<td>Javascript, HTML, CSS</td>
<td>Ruby (HTML, CSS, JavaScript)</td>
<td>ActionScript3 (MXML)</td>
<td>ActionScript3 (MXML)</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Cross-platform frameworks

<table>
<thead>
<tr>
<th>Interpreting</th>
<th>JavaScript mapped to native code</th>
<th>Rendered in WebView control</th>
<th>Runs a Ruby bytecode through bundled RubyVM interpreter</th>
<th>Adobe Air Runtime</th>
<th>AS3 compiled to C++, which compiled to standalone native application installer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>License</strong></td>
<td>Open Source: Apache 2.0</td>
<td>Open Source: MIT</td>
<td>Dual License: Open Source; MIT; Closed source for commercial projects</td>
<td>Closed source, Flex SDK mostly Open Source</td>
<td>Closed source, free for individuals (ad-supported)</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
# Cross-platform frameworks

<table>
<thead>
<tr>
<th></th>
<th>Appcelerator Titanium</th>
<th>PhoneGap</th>
<th>Rhodes</th>
<th>AIR for Mobile Devices</th>
<th>OpenPlug Elips Studio</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>iOS (iPhone, iPod, iPad)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Adobe: Not yet implemented in Flash Builder “Burrito”</td>
</tr>
<tr>
<td>BlackBerry Phone</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Titanium: support in closed beta</td>
</tr>
<tr>
<td>Symbian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Rhodes: Symbian not actively maintained</td>
</tr>
</tbody>
</table>

122 / 123
# Cross-platform frameworks

<table>
<thead>
<tr>
<th></th>
<th>Appcelerator Titanium</th>
<th>PhoneGap</th>
<th>Rhodes</th>
<th>AIR for Mobile Devices</th>
<th>OpenPlug Elips Studio</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Barcode</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>Titanium: available as an extension module (commercial)</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Calendar</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>Titanium: Android only, OpenPlug: not implemented on Android / iPhone</td>
</tr>
<tr>
<td>Camera</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Contacts</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Compass</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>Copy / Paste</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Email</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>File System</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>