University of Regina

CS 455/855
Mobile Computing

Sensors

Dr. Orland Hoeber
orland.hoeber@uregina.ca
http://www.cs.uregina.ca/~hoeber/cs455/2018F

Readings

- Location and Maps Programming Guide
- Online documentation for frameworks
  - Core Location
  - Map Kit
  - Core Motion

Ambient Intelligence

- An important elements that sets mobile computing apart from other kinds of computing is the access to sensor data
  - location awareness
  - motion sensors
  - camera/video/audio

- The use of these features within the context of some target app can lead to ambient intelligence
  - device awareness of the setting/context without explicit user input
  - use of this setting/context makes the app appear to be intelligent and aware of its surroundings

Ambient Intelligence

- Examples of possible ambient intelligence within mobile devices
  - reporting of local weather wherever you are
  - providing localized advertising (e.g., walking past a store having a sale)
  - setting a mobile phone on silent mode while moving fast (in an automobile)
    - problem – detect whether you are the driver or passenger
  - performing facial recognition and providing personalized services (e.g., news)
  - learning from past behaviour (where, when) to provide future recommendations

Topics

- Location awareness
  - uses data on the current location, compass orientation, and how the location is changing over time
  - uses GPS, supported by WiFi and cellular network locations (assisted GPS)
  - Core Location

- Motion awareness
  - uses data regarding the instantaneous and changing speed and attitude
  - uses accelerometer and gyroscope
  - Core Motion

Hardware Requirements

- It is important to be aware of the hardware requirements when performing sensor programming
  - not all devices have all the hardware you might be expecting
  - you must be prepared to fail gracefully and provide a subset of functionality if possible
  - it is also important to be aware of the fact that some sensors take time to "warm up"
    - GPS is inaccurate to start, but gets better the longer you wait
    - converting latitude and longitude to an address is dependent on network resources
    - accessing the camera may take a moment to open the shutter

Hardware Requirements

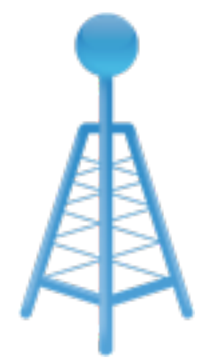
- The latest devices have much of the functionality we expect
  - iPhone & cellular iPad
    - GPS/Cellular/WiFi location data
    - 3-axis accelerometer & gyroscope
    - front- & back-facing cameras
  - iPod Touch & non-cellular iPad
    - WiFi location data
    - 3-axis accelerometer & gyroscope
    - front- & back-facing camera
- If you require specific hardware that is not available on older devices, you may want to impose restrictions on the which devices your app will run
  - UIRequiredDeviceCapabilities in Info.plist

Location Awareness

- Location awareness is provided by three types of sensors:
  - WiFi
    - scan for nearby devices/access points and compare these against an online database
  - Cellular network towers
    - compare accessible cellular towers against an online database
  - GPS
    - look for GPS satellites and get a position fix from them based on their known location and the time it takes to receive their signals
    - not effective when indoors

Why three

- Why use these three methods?
  - GPS is most accurate, but takes the longest time to get a fix
  - bootstrap this process by getting data from other sources quicker, and then become more accurate over time
  - non-GPS methods can provide location services while indoors



Multi-Source Location Services

- Even though there are multiple sources of the location data, Core Location will automatically use whatever facilities the device has
  - all you have to do is ask for the device’s location
  - at the very least, you will get the WiFi location
  - if available, this will be supplanted by cellular and GPS locations
- Accuracy
  - Core Location allows you to specify the desired accuracy
  - the more accuracy required, the more time it takes to provide this information

Constraints on Location Services

- There are a few constraints on the use of the location services that you should be aware of:
  - accuracy depends on a number of factors that we have little control over
    - accuracy of WiFi access point and cellular tower locations
    - network and/or cellular connectivity
    - visibility of GPS satellites
  - GPS is most accurate, but takes a long time to return accurate results
  - sensors take power (may have battery life implications)
    - GPS is the most battery-intensive of all onboard sensors
  - users can deny your app from using Core Location

Map Kit & Core Location

- If all you want to do is display a user's location on a map, you can use Map Kit and take advantage of its automatic integration of Core Location
  - Map Kit View is available in Interface Builder
    - need to "import MapKit" in the view controller
    - need to create an @IBOutlet for the map
  - you can programmatically control what is shown on the map (location, zoom level), as well as add annotations
  - there are a multitude of tutorials about how to program around the Map Kit View
    - https://www.raywenderlich.com/90971/introduction-mapkit-swift-tutorial

- Rather than talking about Map Kit further, we'll focus on the lower-level functionality of Core Location

Location Manager

- The primary object in Core Location for accessing the location services is CLLocationManager
  - because determining the location may take a non-trivial amount of time, there is also a delegate protocol for handling future events: CLLocationManagerDelegate
  - the key steps are:
    - confirm that the services are available
    - create an instance of the CLLocationManager and set the delegate
    - get permission from the user to track the location
    - configure the settings (e.g., accuracy)
    - tell the location manager to start finding the location
    - capture the updates in the delegate protocol methods
    - stop the location services when the location is found (with sufficient accuracy)

Permission

- Before you can use any of the location services, you app must ask the user if this is okay
  - create an instance of the CLLocaitonManager object
  - call the method .requestAlwaysAuthorization() or .requestWhenInUseAuthorization()
  - ensure that you have a reason entered in the Info.plist file
    - NSLocationAlwaysUsageDescription
    - NSLocationWhenInUseUsageDescription

- This wasn't always necessary. Why do you think the app should request access to the location services?

Accuracy

- The desired accuracy can be set by assigning the desiredAccuracy property of the CLLocationManager instance
  - a set of constants are available:
    - kCLLocationAccuracyBestForNavigation
    - kCLLocationAccuracyBest (default)
    - kCLLocationAccuracyNearestTenMeters
    - kCLLocationAccuracyNearestHundredMeters
    - kCLLocationAccuracyNearestKilometer
    - kCLLocationAccuracyNearestThreeKilometers
  - this setting does not act as a filter
    - the delegate method will be called each time it has new information
    - it is up to you to check the horizontalAccuracy property of the results

  - if you want to avoid getting bombarded with delegate messages, you should set the distanceFilter property
    - specifies the minimum distance change before the next update is provided

CLLocationManagerDelegate

- In order to start receiving location updates, call the startUpdatingLocation method of the CLLocationManager instance

- There are two core methods that must be implemented in the class that conforms to the location manager delegate protocol:
  - locationManager(_, didUpdateLocations)
  - locationManager(_, didFailWithError)

  - both provide an instance of the location manager (so that you may stop the location services or change the configuration)
  - the results in the didUpdateLocations method are in an array
    - CLLocation instances
    - most recent location at the end of the array

CLLocation

- The abstract information associated with a location is encapsulated in a CLLocation class
  - coordinate : CLLocationCoordinate2D
    - latitude and longitude coordinates
  - horizontalAccuracy : CLLocationAccuracy
    - accuracy of coordinates in meters
  - altitude : CLLocationDistance
    - distance in meters
  - verticalAccuracy : CLLocationAccuracy
    - accuracy of altitude in meters
  - timestamp : NSDate
    - time at which the location was measured

  - speed : CLLocationSpeed
    - speed in meters per second
    - only available with GPS
  - course: CLLocationDirection
    - degrees clockwise from north
    - only available with GPS

Location Services & Background Apps

- It is possible to keep the location services running while the app is not active
  - e.g., position monitor, exercise apps, interactive games, etc.
  - to avoid significant power drain, set a coarse filter and adjust the accuracy accordingly
  - different types of monitoring
    - significant location monitoring
      - device's location has changed significantly
    - region monitoring (geofencing)
      - device has entered or exited a specified region (location and radius)
  - must use .requestAlwaysAuthorization() and set the appropriate privacy policy in the Info.plist
  - make sure you set the pausesLocationUpdatesAutomatically setting to help conserve power

Heading

- Core Location also supports using the magnetometer (compass) to determine the way the device is facing
  - this can be used even if location services is turned off
    - in this case, it only represents *magnetic north*
    - in order to report *true north*, the device needs to know its precise location in order to compensate for the difference
  - to use this:
    - in the CLLocationManager instance, call the startUpdateHeading method
    - heading updates are captured in the delegate method locationManager(_, didUpdateHeading)
    - the headings are encapsulated in a CLHeading object
    - a headingFilter can be used to specify the minimum heading change before another update is sent

Geocoding

- Geocoding is the translation of an address to a set of GPS coordinates
  - reverse geocoding is the translation of a set of GPS coordinates to an address

- The iOS SDK has a specific class that supports geocoding: CLGeocoder
  - the process of geocoding (or reverse geocoding) takes time, and may not be successful at all
    - depends on network and server availability
    - the results may not be accurate

Forward Geocoding

- Forward geocoding starts with a user-readable address and tries to find the corresponding latitude and longitude
  - three methods are available
    - geocodeAddressDictionary:completionHandler:
    - geocodeAddressString:completionHandler:
    - geocodeAddressString:inRegion:completionHandler:
  - the difference is in the type of input
  - all results are handled by a code block
    - results are in an array of CLPlacemark objects
    - multiple object are returned because of the ambiguity in addresses; the best guess is in the first array element

- Forward geocoding is often used to take user input of an address, obtain the GPS coordinates, and mark this on a map

Reverse Geocoding

- Reverse geocoding starts with a latitude and longitude and tries to find the corresponding user-readable address
  - only one method:
    - reverseGeocodeLocation:completionHandler:
  - the latitude and longitude of the location must be encapsulated in a CLLocation object
  - the results are handled by a code block
    - again, you get an array of CLLocation objects
    - the ability to turn a GPS location into an address is more precise, so the number of matches will be very low

  - reverse geocoding is often used to provide users with an address for their current location (or some other location)

Recap

- Location services are provided by Core Location
  - key classes and delegates:
    - CLLocationManager
      - configure the Core Location services
    - CLLocationManagerDelegate
      - protocol for handling asynchronous updates of the location information
    - CLLocation
      - coordinates and altitude
      - accuracy information (horizontal and vertical)
      - timestamp
      - speed and course (when getting data from GPS)
    - CLGeocoder
      - forward and reverse geocoding (convert between addresses and latitude/longitude, and vice versa)

Motion Awareness

- There are numerous cases where we may want to be aware of the motion of the device:
  - motion-based games
  - awareness of device context (stationary or moving)
  - measurement and logging motion data (e.g., g-forces)
- The general process for measuring motion is very similar to that for measuring location
  - Core Motion works very similar to Core Location

Measuring Raw Motion

- There are some complexities to measuring raw motion
  - even though the device is stationary, there is always a force on being applied to the device (gravity)
  - the raw data reported is a combination of the actual forces of motion, plus gravity
  - it is possible to separate the device motion from gravity using a high-pass filter, however, this is not perfect
    - the measurements in the direction of gravity (down) will be less accurate than measurements in directions orthogonal to gravity
    - the accuracy of the motion sensor is not perfect

Gyroscope

- The inclusion of the gyroscope on the iOS devices has resulted in a huge improvement in the accuracy and speed of gravity and attitude reporting
  - the attitude of the gyroscope remains fixed, so it can measure the attitude of the device that contains it
  - this allows the accelerometer to quickly detect the difference between gravity and device movement
  - the gyroscope can also detect pure rotation (where there is little or no force in any particular direction, just rotation around some axis)

Types of Motion Data

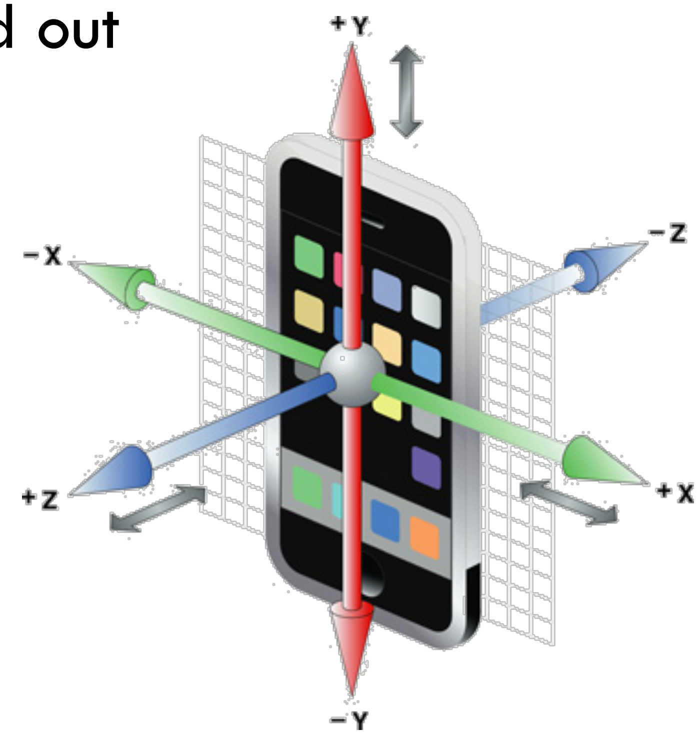
- The motion manager returns a number of different types of data:
  - accelerometerData : CMAccelerometerData
    - raw accelerometer data
    - must separate motion from gravity manually
  - gyroData : CMGyroData
    - raw rotation rate around three axes
  - magnetometerData : CMMagnetometerData
    - raw magnetic field data measured by the device
  - deviceMotion : CMDeviceMotion
    - each of the above raw data elements has some bias to it
    - used together, these biases can be removed and real device motion (and magnetic field data) can be reported

Motion Manager

- The primary object in Core Motion for determining the motion of the device is CMMotionManager
  - because determining the motion of the device is simply a matter of asking for its instantaneous measurements, there is no delegate protocol for handling future events
  - the key steps are:
    - create an instance of the CMMotionManager and retain the instance
    - confirm using the instance properties that the desired hardware is available (deviceMotionAvailable)
    - set the interval for which you wish the motion manager to update itself with new data (deviceMotionUpdatesInterval)
    - call the appropriate start method for the type of updates desired (startDeviceMotionUpdates)
    - whenever motion data is required, ask the instance of the motion manager class for the data (deviceMotion)
    - call the corresponding stop method when motion data is no longer needed (stopDeviceMotionUpdates)

CMDeviceMotion

- The deviceMotion property returns an instance of a CMDeviceMotion class
  - because all of the Core Motion data is collected, it can all be used to increase the accuracy of the data
  - internal senor bias is also factored out
  - unless a frame of reference is provided explicitly when the updates are started, all data is in reference to the device's natural frame of reference
  - other frames of reference include magnetic north and true north

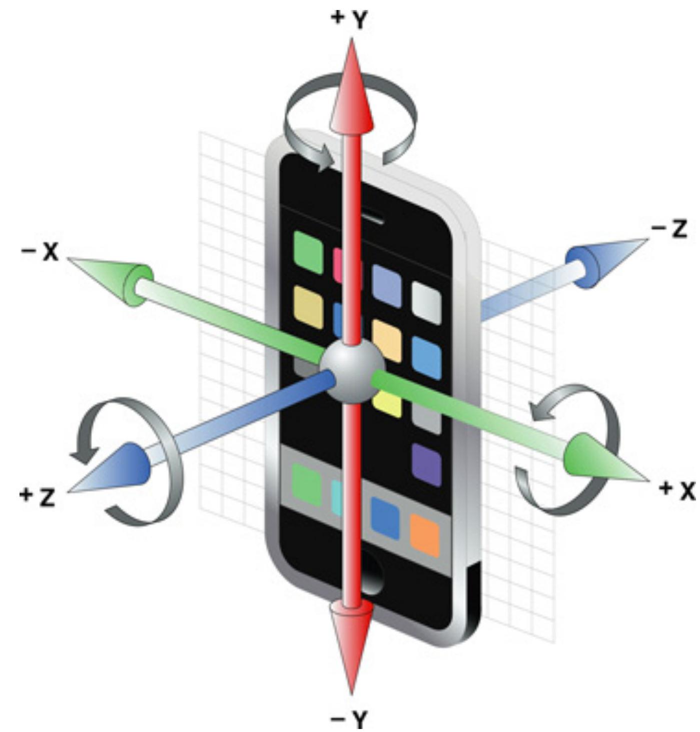


CMDeviceMotion

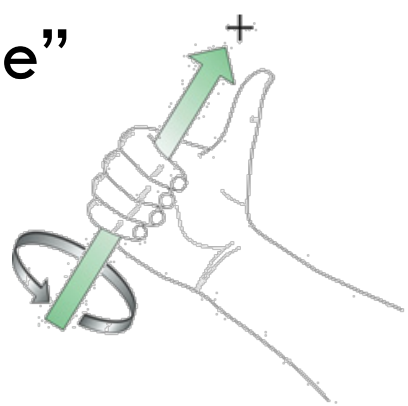
- The properties of the CMDeviceMotion class are:
  - gravity : CMAcceleration
    - coordinates (x, y, z) of the direction of gravity (down)
    - measured in G's
  - userAcceleration : CMAcceleration
    - coordinates (x, y, z) of the force of motion applied to the device
    - measured in G's
  - attitude : CMAttitude
    - orientation of the device with respect to the three axes of the frame of reference
    - measured in radians
  - rotationRate : CMRotationRate
    - rotation rate around the three axes of the frame of reference
    - measured in radians per second
  - magneticField : CMCalibratedMagneticField
    - coordinates (x, y, z) of the magnetic field
    - measured in microteslas

Pitch, Roll, and Yaw

- The attitude are often described as pitch, roll, and yaw
  - pitch = device's natural x axis
  - roll = device's natural y-axis
  - yaw = device's natural z-axis



- Direction of rotation is given by the "right hand rule"



CMAttitude

- The attitude instance variable of the CMDeviceMotion class is of type CMAttitude
  - provides not only roll, pitch, and yaw, but also some helper instances and methods:
    - rotationMatrix
      - matrix representation of the rotation of the device
    - quaternion
      - data structure used by OpenGL
    - multiplyByInverseOfAttitude:
      - when providing a previous attitude value, this calculates the change in attitude between the current instance and the input instance
      - the calculated change is written over the input attitude value

Recap

- Motion services are provided by Core Motion
  - key classes:
    - CMMotionManager
      - configure the Core Motion services
    - CMDeviceMotion
      - gravity
      - user acceleration
      - attitude
      - rotation rate
      - magnetic field
  - the data is in 3-d vectors representing the direction and value
  - the meaning of the value is dependent on the the motion being measured (force, angle, angular change, etc.)

Homework

- Next topic: Network Programming

- Assignment #2
  - due Oct 26
- Project Milestone 2: Project Update
  - due Nov 16
- Short Paper #2 (CS 855)
  - due Nov 23