Sensors

Mobile Application Development in iOS
School of EECS
Washington State University
Instructor: Larry Holder
Outline

• Sensor types
• Sensor availability
• Accessing sensor data
  – Core Motion
  – Core Location
• MapKit
Sensor Types

• **Accelerometer**
  – Movement

• **Gyroscope**
  – Rotation

• **GPS**
  – Location, course
Sensor Types (cont.)

- iPhone/iPad only
  - Magnetometer
  - Barometer

- Watch only
  - Heart rate
  - Green and infrared LEDs
  - Photodiodes
Sensor Types (cont.)

- Device orientation
- Shake motion
- Proximity (to user’s face)
- Battery level
- Microphone & cameras
- Bluetooth (proximity to beacon)
- Wifi & cellular radios (IPs, carrier)
Aggregated Sensors

• Location services
  – Maps, regions (beacon, circular)
  – Geocoders, placemarks
  – Altitude, speed, heading, floor

• Motion services
  – User acceleration (minus gravity)
  – Pedometer, step counter
  – Activity: Stationary, walking, running, cycling, driving
Sensor Availability

• Required device capabilities
  – App Info plist
  – App won’t install on real devices without these capabilities
Sensor Availability

- Programmatically check device availability
- CMMotionManager (create instance)
  - isAccelerometerAvailable
  - isGyroAvailable
  - isMagnetometerAvailable
  - isDeviceMotionAvailable
- CMMotionActivityManager (use singleton)
  - isActivityAvailable
- CLLocationManager (use singleton)
  - locationServicesEnabled
import CoreMotion
import CoreLocation

class ViewController: UIViewController {

    var motionManager = CMMotionManager()

    func checkSensorAvailability() {
        print("accelerometer: " +
            (motionManager.isAccelerometerAvailable ? "yes" : "no"))
        print("magnetometer: " +
            (motionManager.isMagnetometerAvailable ? "yes" : "no"))
        print("gyroscope: " +
            (motionManager.isGyroAvailable ? "yes" : "no"))
        print("device motion: " +
            (motionManager.isDeviceMotionAvailable ? "yes" : "no"))
        print("activity: " +
            (CMMotionActivityManager.isActivityAvailable() ? "yes" : "no"))
        print("location services: " +
            (CLLocationManager.locationServicesEnabled() ? "yes" : "no"))
    }
}
Sensor Authorization

- App must provide reasons for using motion (activity) and location
  - To protect user privacy
- App Info.plist
  - Privacy – Motion Usage Description
  - Privacy – Location When In Use Usage Description
  - Privacy – Location Always and When In Use Usage Description
Sensor Authorization

• Motion Activity
  – Permission requested at first call to `startActivityUpdates()`
  – Check using `CMMotionActivityManager.authorizationStatus()`

• Location
  – `requestWhenInUseAuthorization`
  – `requestAlwaysAuthorization`
  – `didChangeAuthorization`
Core Motion

- Create Core Motion manager
- Set update interval
- Start updates with reference frame, queue, and handler
  - Handler gets CMDeviceMotion structure
    - Attitude, rotation rate, acceleration, heading
- Stop updates
- See developer.apple.com/documentation/coremotion
import CoreMotion

class ViewController: UIViewController {

    var motionManager = CMMotionManager()

    func initializeMotion() { // called from viewDidLoad
        motionManager.deviceMotionUpdateInterval = 1.0 // secs
    }

    func startMotion() {
        motionManager.startDeviceMotionUpdates(  
            using: CMAttitudeReferenceFrame.xTrueNorthZVertical,  
            to: OperationQueue.current!, withHandler: motionHandler)
    }

    func stopMotion() {
        motionManager.stopDeviceMotionUpdates()
    }
}
```swift
func motionHandler (deviceMotion: CMDeviceMotion?, error: Error?) {
    if let err = error {
        print("motionHandler error: ",(err.localizedDescription))
    } else {
        if let dm = deviceMotion {
            print("Attitude: yaw = \(dm.attitude.yaw), " +
                  "pitch = \(dm.attitude.pitch), " +
                  "roll = \(dm.attitude.roll)"")
            print("Acceleration: x = \(dm.userAcceleration.x), " +
                  "y = \(dm.userAcceleration.y), " +
                  "z = \(dm.userAcceleration.z)")
        } else {
            print("motionHandler: deviceMotion = nil")
        }
    }
}
```
import CoreMotion

class ViewController: UIViewController {

    var activityManager = CMMotionActivityManager()

    func startActivity() {
        activityManager.startActivityUpdates(
            to: OperationQueue.current!, withHandler: activityHandler)
    }

    func stopActivity() {
        activityManager.stopActivityUpdates()
    }
}
func activityHandler (motionActivity: CMMotionActivity?) {
    if let ma = motionActivity {
        print("stationary: " + (ma.stationary ? "yes" : "no"))
        print("walking: " + (ma.walking ? "yes" : "no"))
    }
}
Core Motion: Testing

• iOS simulator does not simulate motion sensors
  – Check using motionManager.isDeviceMotionAvailable

• Core motion handler not called by iOS simulator
  – But can call yourself (e.g., Timer)
Core Location

- Conform to `CLLocationManagerDelegate`
- Create instance of `CLLocationManager`
- Check `CLLocationManager.authorizationStatus()`
  - Request if needed
- Set `distanceFilter` and `desiredAccuracy`
- Start/stop location updates as needed
- Changes sent to `didUpdateLocations` delegate method
- Most recent retrieved location: `CLLocationManager.location`
- See developer.apple.com/documentation/corelocation

Mobile Application Development in iOS
import CoreLocation

class ViewController: UIViewController, CLLocationManagerDelegate {

    var locationManager = CLLocationManager()

    func initializeLocation() { // called from start up method
        locationManager.delegate = self
        let status = CLLocationManager.authorizationStatus()
        switch status {
            case .authorizedAlways, .authorizedWhenInUse:
                startLocation()
            case .denied, .restricted:
                print("location not authorized")
            case .notDetermined:
                locationManager.requestWhenInUseAuthorization()
        }
    }
}
// Delegate method called whenever location authorization status changes
func locationManager(_ manager: CLLocationManager,
    didChangeAuthorization status: CLAuthorizationStatus) {
    if ((status == .authorizedAlways) || (status == .authorizedWhenInUse)) {
        self.startLocation()
    } else {
        self.stopLocation()
    }
}

func startLocation () {
    locationManager.distanceFilter = kCLDistanceFilterNone
    locationManager.desiredAccuracy = kCLLocationAccuracyBest
    locationManager.startUpdatingLocation()
}

func stopLocation () {
    locationManager.stopUpdatingLocation()
}
Core Location

// Delegate method called when location changes
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    let location = locations.last
    if let latitude = location?.coordinate.latitude {
        print("Latitude: \(latitude)")
    }
    if let longitude = location?.coordinate.longitude {
        print("Longitude: \(longitude)")
    }
}

// Delegate method called if location unavailable (recommended)
func locationManager(_ manager: CLLocationManager, didFailWithError error: Error) {
    print("locationManager error: \(error.localizedDescription)")
}
Core Location: Testing

- iOS simulator does simulated GPS
Reverse Geocoding

• Lookup information about a location
  – Create instance of CLGeocoder
  – Use reverseGeoCodeLocation method
  – Handler receives array of CLPlacemark’s
  – developer.apple.com/documentation/corelocation/clplacemark
import CoreLocation

var geoCoder = CLGeocoder()

func lookupLocation() {
    if let location = locationManager.location {
        geoCoder.reverseGeocodeLocation(location,
            completionHandler: geoCodeHandler)
    }
}

func geoCodeHandler (placemarks: [CLPlacemark]?, error: Error?) {
    if let placemark = placemarks?.first {
        if let name = placemark.name {
            print("place name = \$(name)")
        }
    }
}
MapKit

- Import MapKit
- Add Map Kit View in Storyboard
- Add IBOutlet
- Enable User Location
  - showUserLocation = true
- Enable user tracking
  - userTrackingMode = .follow
- MKMapViewDelegate
MapKit Annotations

• Create MapKit search request
  – Current region
  – Natural language search query
• Start search
• Results to completion handler
• Add/remove annotations in MapKit View
func findPizza() {
    let request = MKLocalSearch.Request()
    request.naturalLanguageQuery = "pizza"
    request.region = mapView.region
    let search = MKLocalSearch(request: request)
    search.start(completionHandler: searchHandler)
}

func searchHandler (response: MKLocalSearch.Response?, error: Error?) {
    if let err = error {
        print("Error occured in search: \(err.localizedDescription)")
    } else if let resp = response {
        print("\(resp.mapItems.count) matches found")
        self.mapView.removeAnnotations(self.mapView.annotations)
        for item in resp.mapItems {
            let annotation = MKPointAnnotation()
            annotation.coordinate = item.placemark.coordinate
            annotation.title = item.name
            self.mapView.addAnnotation(annotation)
        }
    }
}
Resources

• Core Motion Reference
  – developer.apple.com/documentation/coremotion

• Core Location Reference
  – developer.apple.com/documentation/corelocation

• Map Kit
  – developer.apple.com/documentation/mapkit