Sensors

Mobile Application Development in iOS

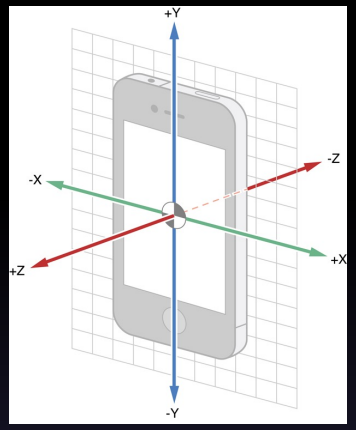
School of EECS

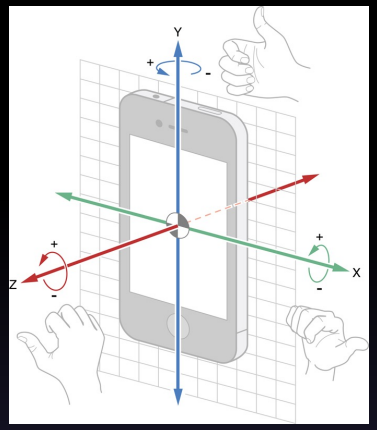
Washington State University

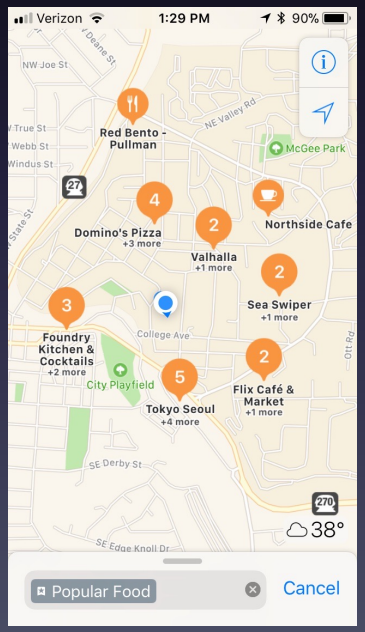
Instructor: Larry Holder

Outline

- Sensor types
- Sensor availability
- Accessing sensor data
  - Core Motion
  - Core Location
- MapKit

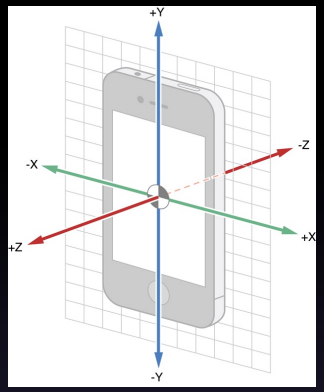


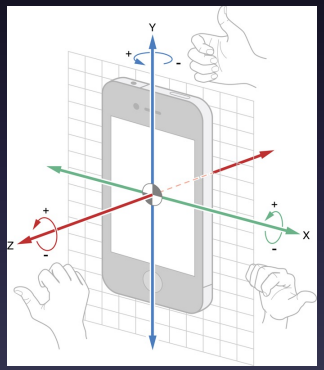




Sensor Types

- Accelerometer
  - Movement
- Gyroscope
  - Rotation
- GPS
  - Location, course



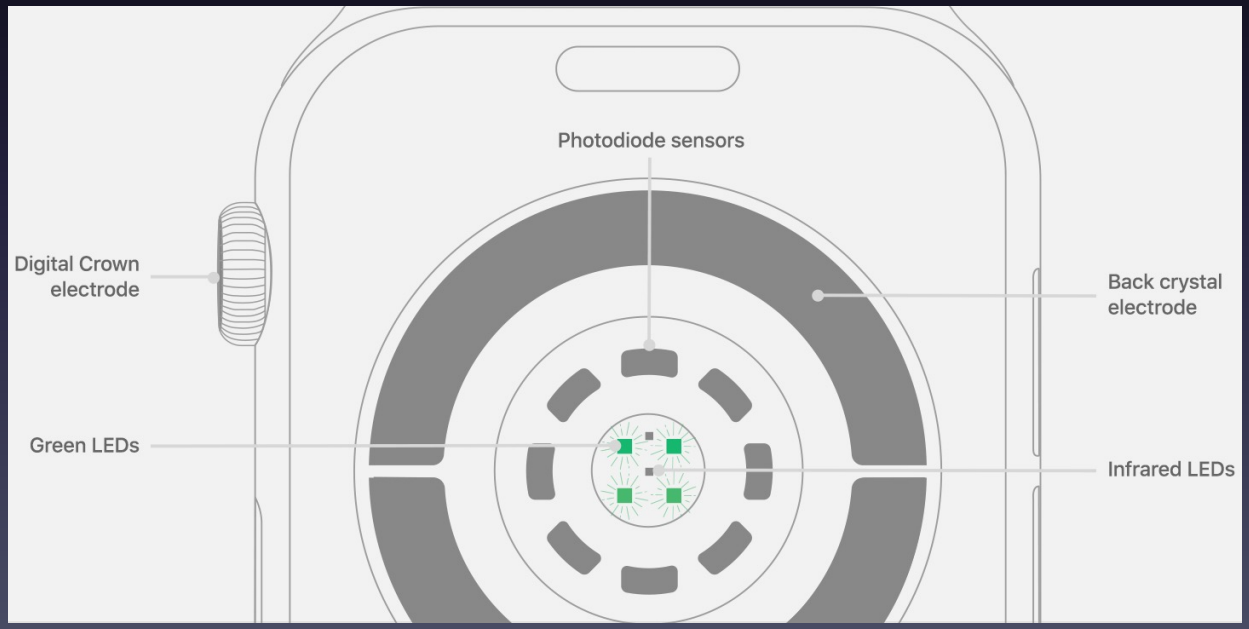


Sensor Types (cont.)

- Barometer
  - Altimeter
- Magnetometer
  - Compass

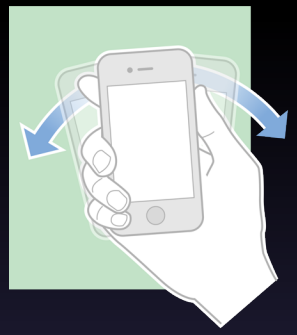
Sensor Types: Watch Only

- Heart rate, ECG



Sensor Types: UIDevice

- Device orientation
- Shake motion
- Proximity (to user's face)
- Battery level
- Microphone & cameras
- Bluetooth (proximity to beacon)
- Wifi & cellular radios (IPs, carrier)
- developer.apple.com/documentation/uikit/uidevice



UIDevice

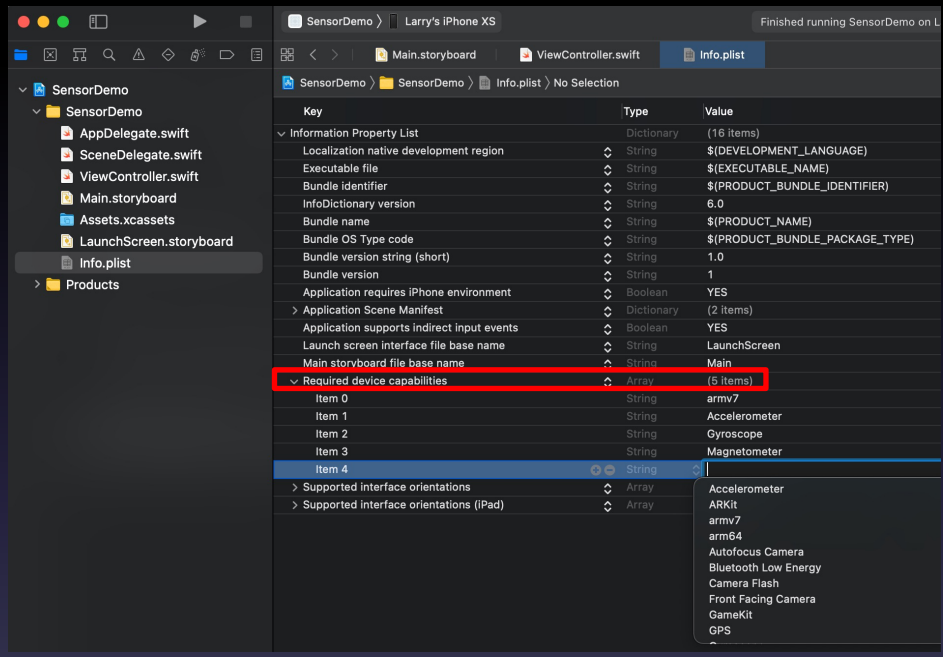
```
override func viewWillTransition(to size: CGSize,
             with coordinator: UIViewControllerTransitionCoordinator) {
    switch UIDevice.current.orientation {
    case .portrait, .portraitUpsideDown: print("orientation: portrait")
    case .landscapeLeft, .landscapeRight: print("orientation: landscape")
    default: print("orientation: unknown")
    }
}

override func motionEnded(_ motion: UIEvent.EventSubtype,
             with event: UIEvent?) {
    if motion == .motionShake {
        print("shake detected")
    }
}
```

Aggregated Sensors

- Location services
  - Maps, regions (beacon, circular)
  - Geocoders, placemarks
  - Altitude, speed, heading, floor
- Motion services
  - User acceleration (minus gravity)
  - Pedometer, step counter
  - Movement disorder: tremor
  - Activity: Stationary, walking, running, cycling, driving

Sensor Availability

- Required device capabilities
  - App Info plist
  - App won't install on real devices without these capabilities



  - https://developer.apple.com/documentation/bundleresources/information_property_list/uirequireddevicecapabilities

Sensor Availability

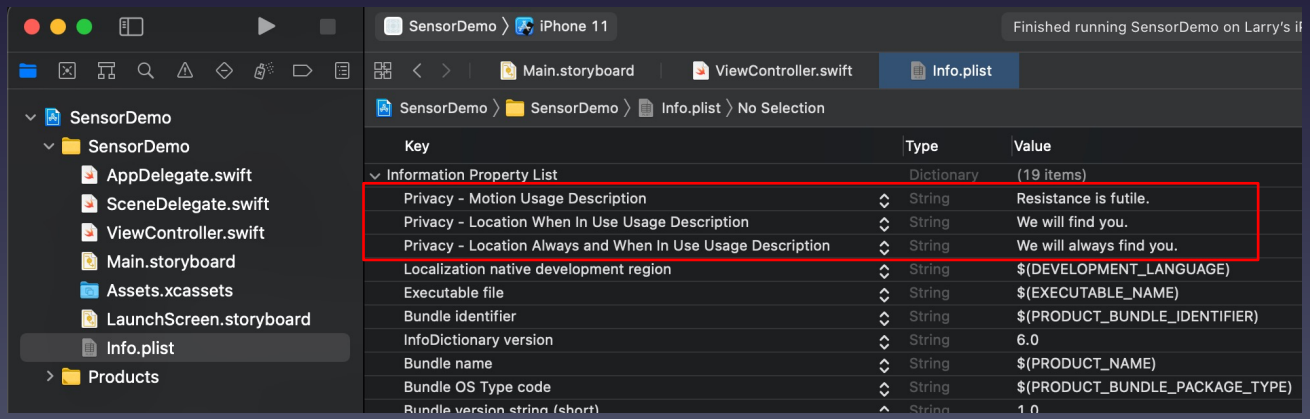
- Programmatically check device availability
- CMMotionManager (create instance)
  - isAccelerometerAvailable
  - isGyroAvailable
  - isMagnetometerAvailable
  - isDeviceMotionAvailable
- CMMotionActivityManager (singleton)
  - isActivityAvailable
- CLLocationManager (singleton)
  - locationServicesEnabled

Sensor Availability

```
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"))
    }
```

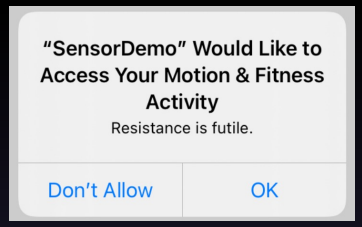
Sensor Authorization

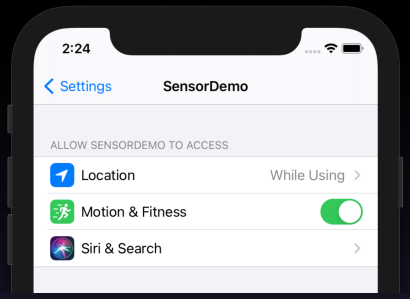
- App must provide reasons for using motion (activity) and location
  - To protect user privacy
- App Info.plist
  - Privacy – Motion Usage Description
  - Privacy – Location When In Use Usage Description
  - Privacy – Location Always and When In Use Usage Description

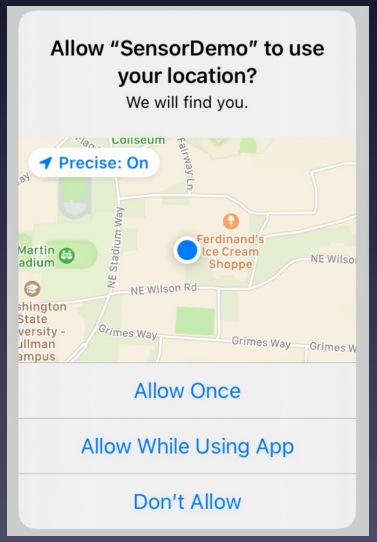


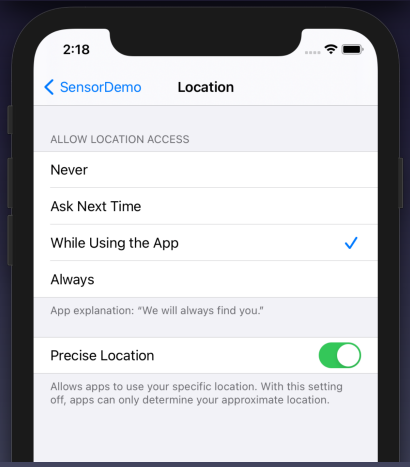
Sensor Authorization

- Motion Activity
  - Permission requested at first call to startActivityUpdates()
  - Check using CMMotionActivityManager.authorizationStatus()
- Location
  - requestWhenInUseAuthorization
  - requestAlwaysAuthorization
  - didChangeAuthorization



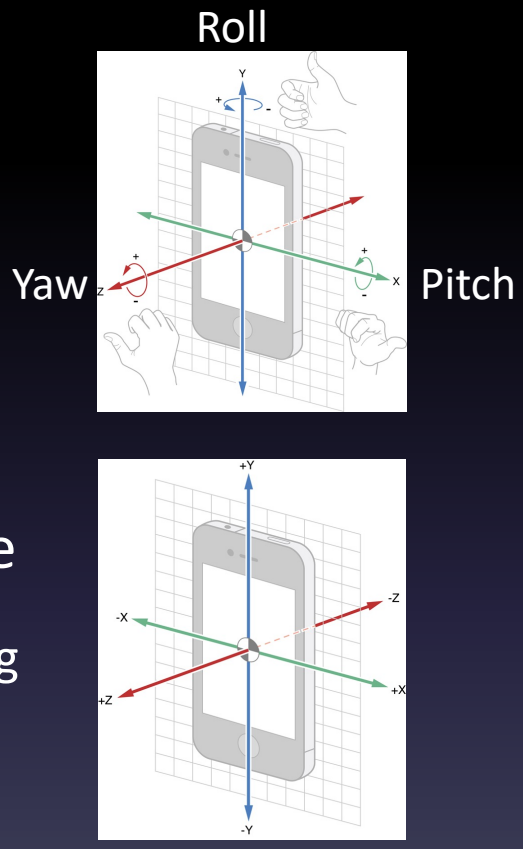






Core Motion

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



Core Motion

```
import CoreMotion

class ViewController: UIViewController {

    var motionManager = CMMotionManager()

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

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

    func stopMotion() {
        motionManager.stopDeviceMotionUpdates()
    }
}
```

Core Motion

```
func motionHandler (deviceMotion: CMDeviceMotion?, error: Error?) {
    if let err = error {
        print("motionHandler error: \(err.localizedDescription)")
    } else {
        if let dm = deviceMotion {
            let attitudeStr = "Attitude (y,p,r): \(dm.attitude.yaw),
\(dm.attitude.pitch), \(dm.attitude.roll)"
            let accelerationStr = "Acceleration (x,y,z):
\(dm.userAcceleration.x), \(dm.userAcceleration.y), \(dm.userAcceleration.z)"
            print(attitudeStr)
            print(accelerationStr)
        } else {
            print("motionHandler: deviceMotion = nil")
        }
    }
}
```

Better string formatting: String(format: "%.3f", x)

Core Motion Activity

- Create Core Motion Activity Manager
- Check that activities authorized
  - CMMotionActivityManager.authorizationStatus()
- Start updates
- Stop updates
- See developer.apple.com/documentation/coremotion/cmmotionactivitymanager

Core Motion Activity

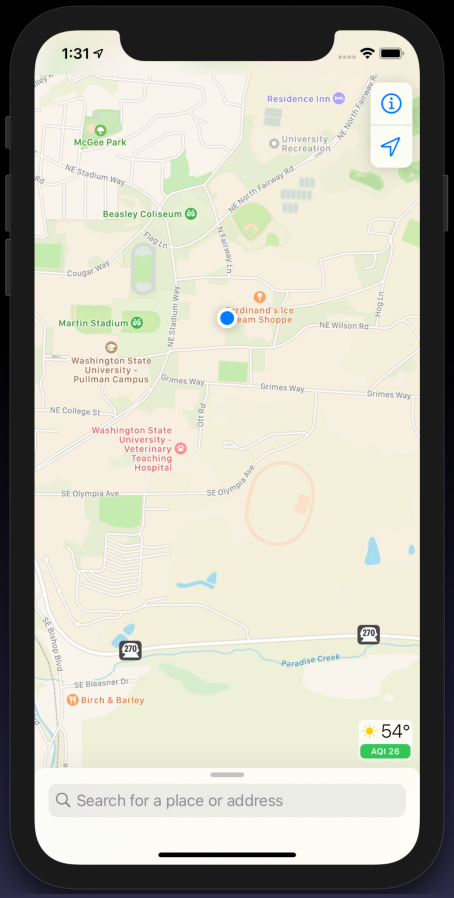
```
// In ViewController

var activityManager = CMMotionActivityManager()

func startActivity() {
    if CMMotionActivityManager.authorizationStatus() != .denied {
        activityManager.startActivityUpdates(
            to: OperationQueue.current!, withHandler: activityHandler)
    } else {
        print("activity not authorized")
    }
}

func stopActivity() {
    activityManager.stopActivityUpdates()
}
```

Core Motion Activity

```
// In ViewController

func activityHandler (motionActivity: CMMotionActivity?) {
    if let ma = motionActivity {
        var activityStr = "Activity ="
        if ma.automotive {activityStr += " automotive"}
        if ma.cycling {activityStr += " cycling"}
        if ma.running {activityStr += " running"}
        if ma.stationary {activityStr += " stationary"}
        if ma.walking {activityStr += " walking"}
        if ma.unknown {activityStr += " unknown"}
        print(activityStr)
    }
}
```

Remember: No motion or activity capabilities in iOS simulator.

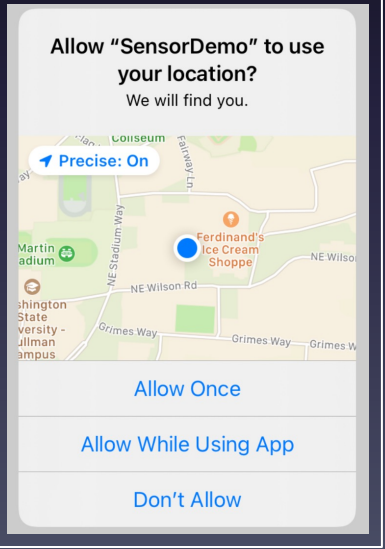
Core Location



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

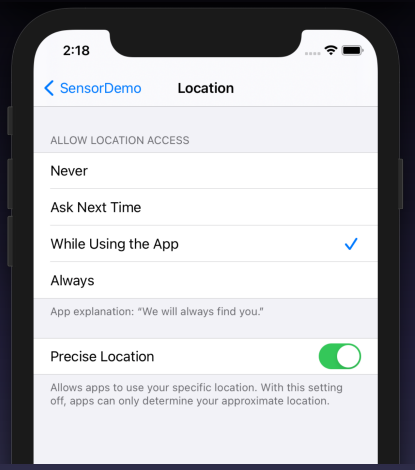
Core Location

```
import CoreLocation

class ViewController: UIViewController, CLLocationManagerDelegate {

    var locationManager = CLLocationManager()

    func initializeLocation() { // called from start up method
        locationManager.delegate = self
        locationManager.distanceFilter = kCLDistanceFilterNone
        locationManager.desiredAccuracy = kCLLocationAccuracyBest
        let status = locationManager.authorizationStatus
        switch status {
        case .authorizedAlways, .authorizedWhenInUse:
            print("location authorized")
        case .denied, .restricted:
            print("location not authorized")
        case .notDetermined:
            locationManager.requestWhenInUseAuthorization()
        @unknown default:
            print("unknown location authorization")
        }
    }
}
```



Core Location

```
// Delegate method called whenever location authorization status changes
func locationManager(_ manager: CLLocationManager,
        didChangeAuthorization status: CLAuthorizationStatus) {
    if ((status == .authorizedAlways) || (status == .authorizedWhenInUse)) {
        print("location changed to authorized")
    } else {
        print("location changed to not authorized")
        self.stopLocation()
    }
}

func startLocation () {
    let status = locationManager.authorizationStatus
    if (status == .authorizedAlways) ||
        (status == .authorizedWhenInUse) {
        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
    var locationStr = "Location (lat,long): "
    if let latitude = location?.coordinate.latitude {
        locationStr += String(format: "%.6f", latitude)
    } else {locationStr += "?"}
    if let longitude = location?.coordinate.longitude {
        locationStr += String(format: ", %.6f", longitude)
    } else {locationStr += ", ?"}
    print(locationStr)
}

// 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

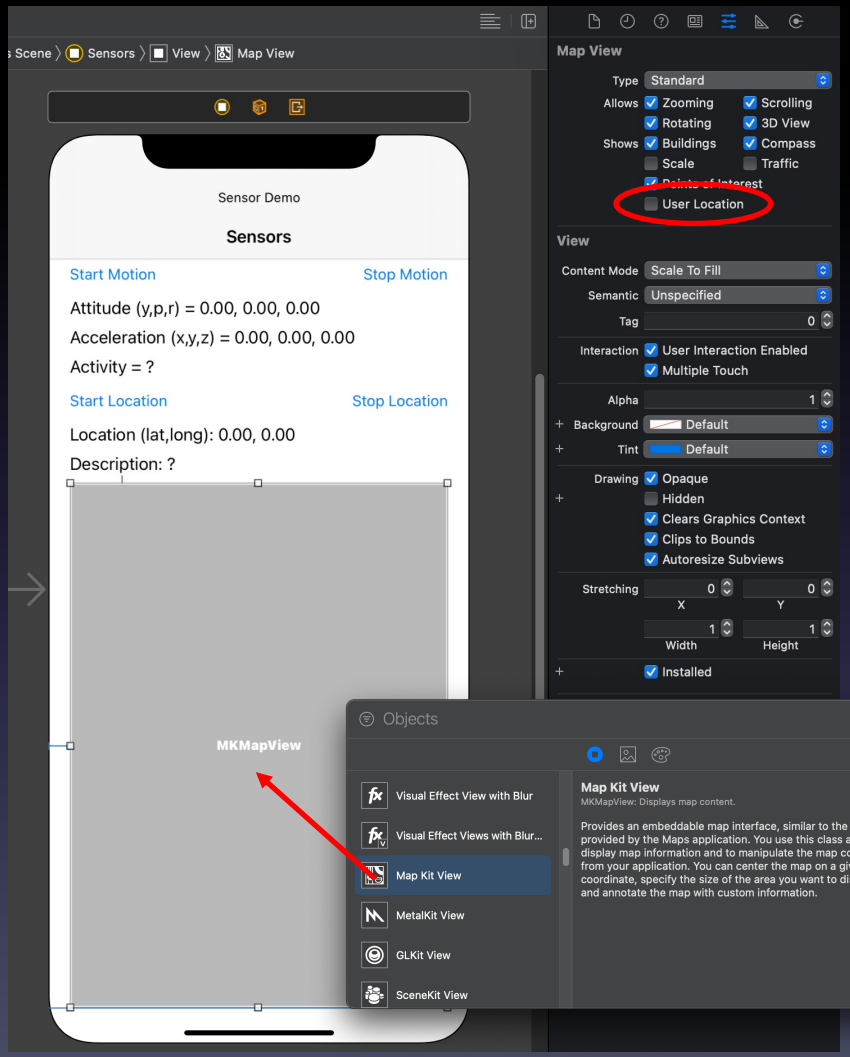
- Lookup information about a location
  - Create instance of CLGeocoder
  - Use reverseGeoCodeLocation method
  - Handler receives array of CLPlacemark's
  - developer.apple.com/documentation/corelocation/clplacemark

Reverse Geocoding

```
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 {
        print("placemark= \(placemark)")
    }
}
```

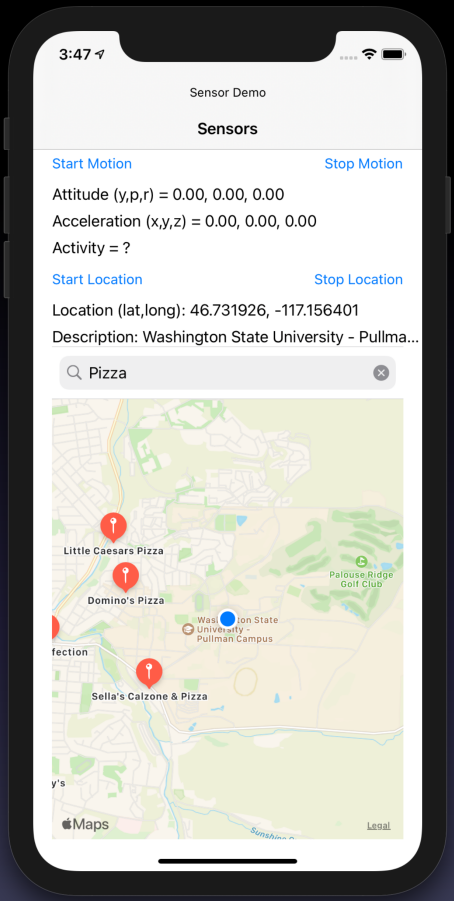
MapKit

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



MapKit Annotations

```
func searchMap(_ query: String) {
    let request = MKLocalSearch.Request()
    request.naturalLanguageQuery = query
    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
  - developer.apple.com/documentation/coremotion
- Core Location
  - developer.apple.com/documentation/corelocation
- Map Kit
  - developer.apple.com/documentation/mapkit