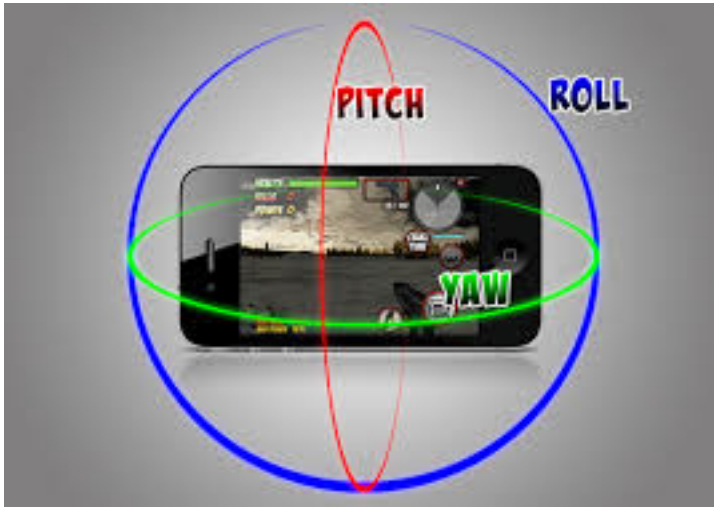
ITP 342
Mobile App Dev



Accelerometer
Gyroscope

Motion Events

- Users generate motion events when they move, shake, or tilt the device
- These motion events are detected by the device hardware, specifically, the **accelerometer** and the **gyroscope**
- https://developer.apple.com/library/ios/documentation/uikit/reference/UIAccelerometer_Class/Reference/UIAccelerometer.html
- https://developer.apple.com/library/ios/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/motion_event_basics/motion_event_basics.html
- https://developer.apple.com/library/ios/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/Introduction/Introduction.html

Accelerometer

- One of the coolest features of the iOS devices is the built-in **accelerometer**
- Tiny device that lets the iOS device (iPhone, iPad, iPod Touch) know how it's being held and if it's being moved
- The accelerometer is actually made up of three accelerometers, one for each axis – x, y, and z
  - Each one measures changes in velocity over time along a linear path
  - Combining all three accelerometers lets you detect device movement in any direction and get the device's current orientation
  - Although there are three accelerometers, the remainder of this document refers to them as a single entity

Accelerometer

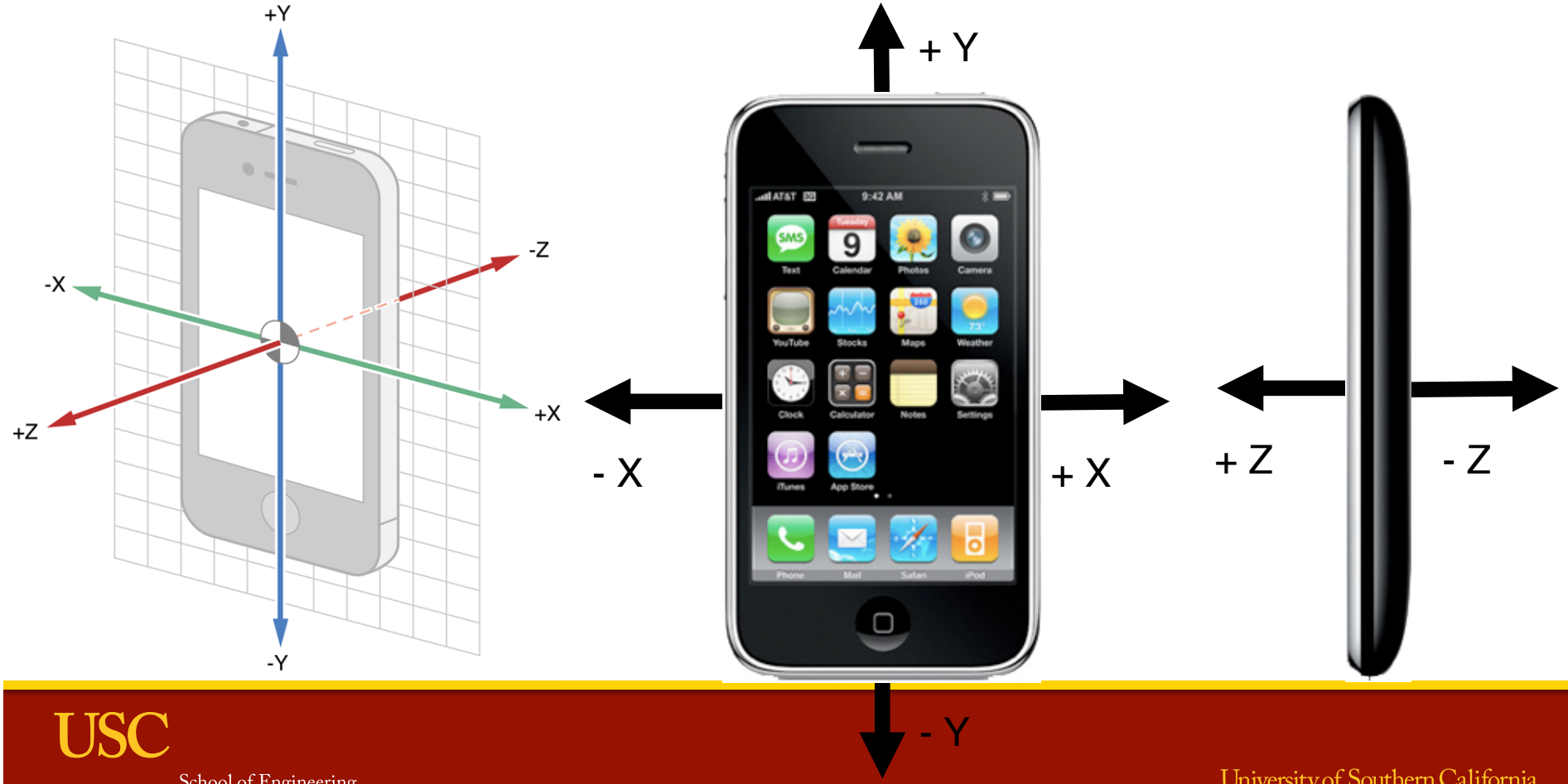
- The accelerometer inside an iOS device uses three elements:
  - a silicon mass
  - a set of silicon springs
  - an electrical current
- How it works
  - The silicon springs measure the position of the silicon mass using the electrical current
  - Rotating iOS device causes a fluctuation in the electrical current passing through the silicon springs
  - The accelerometer registers these fluctuations and tells iOS device to adjust the display accordingly

Accelerometer Physics

- An accelerometer measures both acceleration & gravity by sensing the amount of inertial force in a given direction
- Three-axis = movement in 3D space
- Tell how the iOS device is being held (rotation), if it's laying on a table and if it's face down or face up
- Give measurements in g-forces
  - Value of 1.0 means that 1 g is sensed in a particular direction
  - If the iPhone is being held with no movement, there will be approximately 1 g of force exerted on it by the pull of the earth

Accelerometer's Axes

- Graphic representation of the three axes
  - Increases in y indicate upward force



Access Motion Data

- If you want your app to respond when a user shakes the device, you can use the UIKit motion-event handling methods to get information from the passed-in **UIEvent** object
- If you need to detect the general orientation of a device, but you don't need to know the orientation vector, use the **UIDevice** class
- If neither the UIDevice nor the UIEvent classes are sufficient, it's likely you'll want to use the **Core Motion** framework to access the accelerometer, gyroscope, and device motion classes

Detecting Shake-Motion Events

- When users shake a device, iOS evaluates the accelerometer data.
  - If the data meets certain criteria, iOS interprets the shaking gesture and creates a **UIEvent** object to represent it.
  - Then, it sends the event object to the currently active app for processing.
  - Note that your app can respond to shake-motion events and device orientation changes at the same time.
- Motion events are simpler than touch events.
  - The system tells an app when a motion starts and stops, but not when each individual motion occurs.
  - And, motion events include only an event type (UIEventTypeMotion), event subtype (UIEventSubtypeMotionShake), and timestamp.

Motion Events

- To receive motion events, designate a responder object as the first responder.
- Motion events use the responder chain to find an object that can handle the event.
- If a shaking-motion event travels up the responder chain to the window without being handled and the `applicationSupportsShakeToEdit` property of **UIApplication** is set to YES (the default), iOS displays a sheet with Undo and Redo commands.

Motion-Handling Methods

- There are three motion-handling methods:
  - `motionBegan:withEvent:`
  - `motionEnded:withEvent:`
  - `motionCancelled:withEvent:`
- To handle motion events, you must implement either the `motionBegan:withEvent:` method or the `motionEnded:withEvent:` method, and sometimes both.
- A responder should also implement the `motionCancelled:withEvent:` method to respond when iOS cancels a motion event.
  - An event is canceled if the shake motion is interrupted or if iOS determines that the motion is not valid after all – for example, if the shaking lasts too long.

Handling a Motion Event

```
// XYZViewController.m

- (BOOL) canBecomeFirstResponder {
  return YES;
}

- (void) viewDidAppear: (BOOL) animated {
  [super viewDidAppear:animated];
  [self becomeFirstResponder];
}

- (void) motionEnded: (UIEventSubtype) motion
          withEvent: (UIEvent *) event {
  if (motion == UIEventSubtypeMotionShake)
  {
     NSLog (@"You shook me!");
  }
}
```

UIEventSubtype

- `UIEventSubtypeNone`
- **`UIEventSubtypeMotionShake`**
- `UIEventSubtypeRemoteControlPlay`
- `UIEventSubtypeRemoteControlPause`
- `UIEventSubtypeRemoteControlStop`
- `UIEventSubtypeRemoteControlTogglePlayPause`
- `UIEventSubtypeRemoteControlNextTrack`
- `UIEventSubtypeRemoteControlPreviousTrack`
- `UIEventSubtypeRemoteControlBeginSeekingBackward`
- `UIEventSubtypeRemoteControlEndSeekingBackward`
- `UIEventSubtypeRemoteControlBeginSeekingForward`
- `UIEventSubtypeRemoteControlEndSeekingForward`

Device Orientations

x:0.0 y:-1.0 z:0.0

x:1.0 y:0.0 z:0.0

x:0.0 y:1.0 z:0.0

x:-1.0 y:0.0 z:0.0

x:0.0 y:0.0 z:-1.0

x:0.0 y:0.0 z:1.0

Current Device Orientation

- Use the methods of the **UIDevice** class when you need to know only the general orientation of the device and not the exact vector of orientation.
  - Using UIDevice is simple and doesn't require you to calculate the orientation vector yourself.
- Before you can get the current orientation, you need to tell the UIDevice class to begin generating device orientation notifications by calling the `beginGeneratingDeviceOrientationNotifications` method.
  - This turns on the accelerometer hardware, which may be off to conserve battery power.

Device Orientation

- After enabling orientation notifications, get the current orientation from the orientation property of the **UIDevice** object.
- If you want to be notified when the device orientation changes, register to receive `UIDeviceOrientationDidChangeNotification` notifications.
- The device orientation is reported using `UIDeviceOrientation` constants, indicating whether the device is in landscape mode, portrait mode, screen-side up, screen-side down, and so on.
- When you no longer need to know the orientation of the device, always disable orientation notifications by calling the UIDevice method, `endGeneratingDeviceOrientationNotifications`.
  - This gives the system the opportunity to disable the accelerometer hardware if it's not being used elsewhere, which preserves battery power.

Responding to Changes in Orientation

```
// XYZViewController.m

- (void) viewDidLoad {
   // Request to turn on accelerometer and begin receiving accelerometer events
   [[UIDevice currentDevice] beginGeneratingDeviceOrientationNotifications];
   [[NSNotificationCenter defaultCenter] addObserver:self
      selector:@selector(orientationChanged:)
      name:UIDeviceOrientationDidChangeNotification object:nil];
}

- (void) orientationChanged: (NSNotification *) notification {
   // Respond to changes in device orientation
   UIDeviceOrientation currentOrientation = [[UIDevice currentDevice]
      orientation];
}

- (void) viewDidDisappear {
   // Request to stop receiving accelerometer events and turn off accelerometer
   [[NSNotificationCenter defaultCenter] removeObserver:self];
   [[UIDevice currentDevice] endGeneratingDeviceOrientationNotifications];
}
```

UIDeviceOrientation

- Constants
  - `UIDeviceOrientationUnknown`
    - The orientation of the device cannot be determined.
  - `UIDeviceOrientationPortrait`
    - The device is in portrait mode, with the device held upright and the home button at the bottom.
  - `UIDeviceOrientationPortraitUpsideDown`
    - The device is in portrait mode but upside down, with the device held upright and the home button at the top.
  - `UIDeviceOrientationLandscapeLeft`
    - The device is in landscape mode, with the device held upright and the home button on the right side.
  - `UIDeviceOrientationLandscapeRight`
    - The device is in landscape mode, with the device held upright and the home button on the left side.
  - `UIDeviceOrientationFaceUp`
    - The device is held parallel to the ground with the screen facing upwards.
  - `UIDeviceOrientationFaceDown`
    - The device is held parallel to the ground with the screen facing downwards.

Core Motion

- Setting and checking required hardware capabilities for motion events
  - Add two `UIRequiredDeviceCapabilities` keys to your app's Info.plist file.
    - accelerometer
    - gyroscope
  - At runtime, iOS launches your app only if the device has the required capabilities.
- The App Store also uses this list to inform users so that they can avoid downloading apps that they can't run.
- You don't need to include the accelerometer key if your app detects only device orientation changes.

Core Motion

- The Core Motion framework is primarily responsible for accessing raw accelerometer and gyroscope data and passing that data to an app for handling.
- Core Motion uses unique algorithms to process the raw data it collects, so that it can present more refined information.
- Core Motion is distinct from UIKit.
  - It is not connected with the UIEvent model and does not use the responder chain.
- Instead, Core Motion simply delivers motion events directly to apps that request them.

Core Motion Events

- Core Motion events are represented by three data objects, each encapsulating one or more measurements:
  - A **CMAccelerometerData** object captures the acceleration along each of the spatial axes.
  - A **CMGyroData** object captures the rate of rotation around each of the three spatial axes.
  - A **CMDeviceMotion** object encapsulates several different measurements, including attitude and more useful measurements of rotation rate and acceleration.

Core Motion Manager

- The **CMMotionManager** class is the central access point for Core Motion.
  - You create an instance of the class, specify an update interval, request that updates start, and handle motion events as they are delivered.
- An app should create only a single instance of the CMMotionManager class.
  - Multiple instances of this class can affect the rate at which an app receives data from the accelerometer and gyroscope.
- All of the data-encapsulating classes of Core Motion are subclasses of **CMLogItem**, which defines a timestamp so that motion data can be tagged with a time and logged to a file.
  - An app can compare the timestamp of motion events with earlier motion events to determine the true update interval between events.

Obtaining Motion Data

- For each of the data-motion types described, the CMMotionManager class offers two approaches for obtaining motion data:
  - **Pull**: An app requests that updates start and then periodically samples the most recent measurement of motion data.
  - **Push**: An app specifies an update interval and implements a block for handling the data. Then, it requests that updates start, and passes Core Motion an operation queue and the block. Core Motion delivers each update to the block, which executes as a task in the operation queue.
- **Pull** is the recommended approach for most apps, especially games. It is generally more efficient and requires less code.
- **Push** is appropriate for data-collection apps and similar apps that cannot miss a single sample measurement.
- Both approaches have benign thread-safety effects; with push, your block executes on the operation-queue's thread whereas with pull, Core Motion never interrupts your threads.

Update Interval

- When you request motion data with Core Motion, you specify an update interval.
- You should choose the largest interval that meets your app's needs.
  - The larger the interval, the fewer events are delivered to your app, which improves battery life.
- You can set the reporting interval to be as small as 10 milliseconds (ms), which corresponds to a 100 Hz update rate, but most app operate sufficiently with a larger interval.

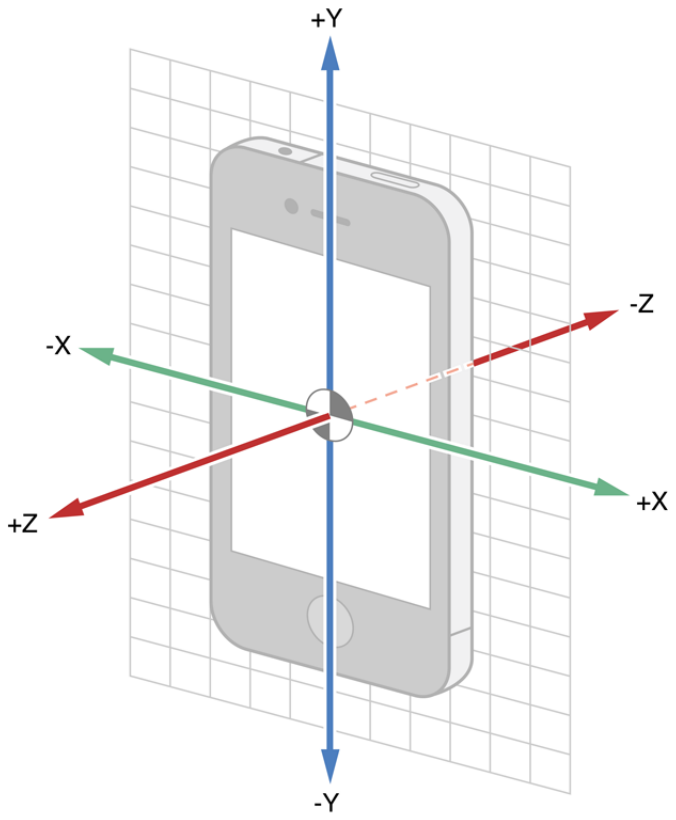
Update Interval

- Common update intervals for acceleration events

Event frequency (Hz)	Usage
`10 – 20`	Suitable for determining a device's current orientation vector.
`30 – 60`	Suitable for games and other apps that use the accelerometer for real-time user input.
`70 – 100`	Suitable for apps that need to detect high-frequency motion. For example, you might use this interval to detect the user hitting the device or shaking it very quickly.

Accelerometer Events with Core Motion

- The accelerometer measures velocity over time along three axes.
- With Core Motion, each movement is captured in a **CMAccelerometerData** object, which encapsulates a structure of type **CMAcceleration**.

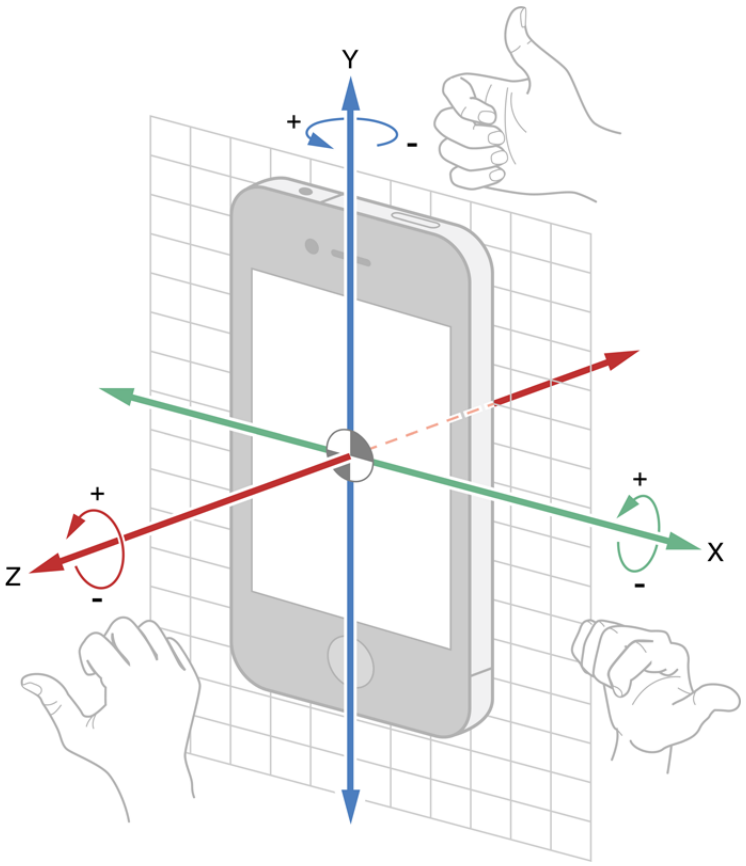


Accelerometer Events with Core Motion

- To start receiving and handling accelerometer data, create an instance of the **CMMotionManager** class and call one of the following methods:
  - `startAccelerometerUpdates` – the pull approach
    - After you call this method, Core Motion continually updates the `accelerometerData` property of CMMotionManager with the latest measurement of accelerometer activity.
  - `startAccelerometerUpdatesToQueue: withHandler:` – the push approach
    - Before you call this method, assign an update interval to the accelerometerUpdateInterval property, create an instance of NSOperationQueue and implement a block of type CMAccelerometerHandler that handles the accelerometer updates.
    - Then, call the startAccelerometerUpdatesToQueue:withHandler: method on the motion-manager object, passing in the operation queue and the block.

Gyroscope – Rotation Rate Data

- A gyroscope measures the rate at which a device rotates around each of the three spatial axes.
- Each time you request a gyroscope update, Core Motion takes a biased estimate of the rate of rotation and returns this information in a **CMGyroData** object.



Right-hand Rule

- When analyzing rotation-rate data – specifically, when analyzing the fields of the CMRotationMatrix structure – follow the "right-hand rule" to determine the direction of rotation.
  - For example, if you wrap your right hand around the x-axis such that the tip of the thumb points toward positive x, a positive rotation is one toward the tips of the other four fingers.
  - A negative rotation goes away from the tips of those fingers.

Rotation Rate Data

- To start receiving and handling rotation-rate data, create an instance of the **CMMotionManager** class and call one of the following methods:
  - `startGyroUpdates` – the pull approach
    - After you call this method, Core Motion continually updates the `gyroData` property of CMMotionManager with the latest measurement of gyroscope activity.
  - `startGyroUpdatesToQueue:withHandler:` – the push approach
    - Before you call this method, assign an update interval to the `gyroUpdateInterval` property, create an instance of `NSOperationQueue`, and implement a block of type `CMGyroHandler` that handles the gyroscope updates.
    - Then, call the `startGyroUpdatesToQueue:withHandler:` method on the motion-manager object, passing in the operation queue and the block.

Device Motion Data

- If a device has both an accelerometer and a gyroscope, Core Motion offers a device-motion service that processes raw motion data from both sensors.
- Device motion uses sensor fusion algorithms to refine the raw data and generate information for a device's attitude, its unbiased rotation rate, the direction of gravity on a device, and the user-generated acceleration.
- An instance of the **CMDeviceMotion** class encapsulates all of this data.

CMAttitude

- You can access attitude data through a **CMDeviceMotion** object's attitude property, which encapsulates a **CMAttitude** object.
- Each instance of the CMAttitude class encapsulates three mathematical representations of attitude:
  - a quaternion
  - a rotation matrix
  - the three Euler angles (roll, pitch, and yaw)

Device Motion Data

- To start receiving and handling device-motion updates, create an instance of the **CMMotionManager** class and call one of the following two methods on it:
  - `startDeviceMotionUpdates` – the pull approach
    - After you call this method, Core Motion continuously updates the deviceMotion property of CMMotionManager with the latest refined measurements of accelerometer and gyroscope activity, as encapsulated in a CMDeviceMotion object.
  - `startDeviceMotionUpdatesToQueue:withHandler:` – the push approach
    - Before you call this method, assign an update interval to the `deviceMotionUpdateInterval` property, create an instance of `NSOperationQueue`, and implement a block of the `CMDeviceMotionHandler` type that handles the accelerometer updates.
    - Then, call the `startDeviceMotionUpdatesToQueue: withHandler:` method on the motion-manager object, passing in the operation queue and the block.