Core Motion
Core Motion is a framework that allows your application to receive motion data from device hardware.

For an iOS developer, this means you can create applications that can observe and respond to the motion and orientation of an iOS device.

Important note:
You can only test or use the functionality of Core Motion on an actual device. The simulator does not have any facilities for reproducing physical motion for your app.
Hardware Elements of Core Motion

Accelerometer
• Measures acceleration in all three dimensions

Gyroscope
• Calculates orientation and rotation in all three dimensions

Magnetometer
• Measures magnetic forces
Coordinate System

+x is towards the right of the screen

+y is towards the top of the screen

+z is towards the user when the screen is faceup
A `CMDeviceMotion` object contains the following objects as properties:

- **attitude**: `CMAttitude`
  - Returns the orientation of the device
  - Can access the data in any of 3 representations

- **rotationRate**: `CMRotationRate`
  - Returns the rotation rate of the device for devices with a gyro
  - x, y, z values in radians per second

- **gravity**: `CMAcceleration`
  - Returns the gravity vector expressed in the device's reference frame
  - x, y, z values in g’s (gravitational force)

- **userAcceleration**: `CMAcceleration`
  - Returns the acceleration that the user is giving to the device
  - x, y, z values in g’s (gravitational force)

- **magneticField**: `CMCalibratedMagneticField`
  - Returns the magnetic field vector with respect to the device for devices with a magnetometer
CMAttitude contains three different representations of the device’s orientation:

- Euler angles
- Quaternions
- Rotation matrices

Each of these is in relation to a given reference frame.
Euler Angles

Euler Angles are the most readily understood of the 3 representations, as they simply describe rotation around each of the axes.

- **Pitch** - is rotation around the X-axis, increasing as the device tilts toward you, decreasing as it tilts away.
- **Roll** - is rotation around the Y-axis, decreasing as the device rotates to the left, increasing to the right.
- **Yaw** - is rotation around the Z-axis, decreasing clockwise, increasing counter-clockwise.
CMMotionManager provides a consistent interface for each of the four motion data types:

- Attitude (rotation)
- Acceleration
- Gravity
- Magnetic Field

Although you can access data for each of these motion types individually, it’s simplest to create a CMMotionManager instance to access all of the above.
If `deviceMotion` is a `CMDeviceMotion` object:

```javascript
deviceMotion.gravity.x
deviceMotion.gravity.y
deviceMotion.gravity.z

deviceMotion.userAcceleration.x
deviceMotion.userAcceleration.y
deviceMotion.userAcceleration.z

deviceMotion.attitude.roll
deviceMotion.attitude.pitch
deviceMotion.attitude.yaw

deviceMotion.magneticField.field.x
deviceMotion.magneticField.field.y
deviceMotion.magneticField.field.z
```
1. Create a motion manager:

In your ViewController class:

```swift
let motionManager = CMMotionManager()
```
2. Start receiving updates at the desired frequency:

```swift
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.

    motionManager.deviceMotionUpdateInterval = 0.1

    motionManager.startDeviceMotionUpdates(
        to: OperationQueue.current!,
        withHandler: {
            (deviceMotion, error) -> Void in

            if (error == nil) {
                self.handleDeviceMotionUpdate(deviceMotion: deviceMotion!)
            } else {
                // handle the error
            }
        }
    )
}
```
3. Write code specifying what you want to happen at each update

```swift
func handleDeviceMotionUpdate(deviceMotion: CMMotionUpdate) {
    let gravity = deviceMotion.gravity
    let rotation = atan2(gravity.x, gravity.y) - Double.pi
    self.cooperImageView.transform = CGAffineTransform(rotationAngle: 
        CGFloat(rotation))
}
```