Health and Fitness with Core Motion

Session 713

Bharath Rao Engineer
Paul Thompson Engineer

Agenda

Agenda

Historical Accelerometer

Agenda

Historical Accelerometer

Pedometer Events

Agenda

Historical Accelerometer

Pedometer Events

Device Motion on Apple Watch

Historical Accelerometer

Historical Accelerometer

CMSensorRecorder

Historical Accelerometer

CMSensorRecorder

Access to raw sensor samples over long durations

Historical Accelerometer

CMSensorRecorder

Access to raw sensor samples over long durations

Samples buffered while app is suspended

Historical Accelerometer

CMSensorRecorder

Historical Accelerometer

CMSensorRecorder

	watchOS 2	watchOS 3
Query Duration	12 hours	36 hours

Historical Accelerometer

CMSensorRecorder

	watchOS 2	watchOS 3
Query Duration	12 hours	36 hours
Sample Delay	~180 seconds	~3 seconds

Historical Accelerometer

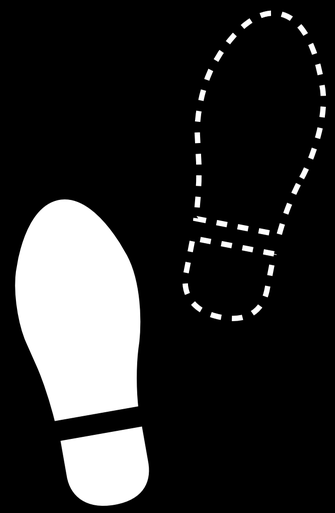
CMSensorRecorder

	watchOS 2	watchOS 3
Query Duration	12 hours	36 hours
Sample Delay	~180 seconds	~3 seconds
Use Cases	Sleep Tracking	Tremor Diagnosis

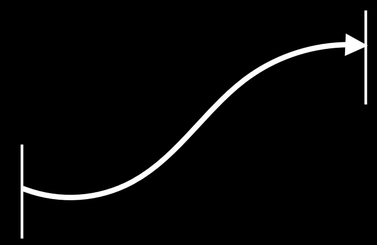
Pedometer

Pedometer Data

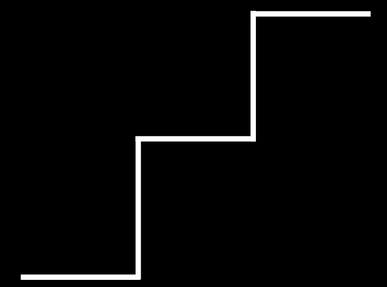
CMPedometer



Steps



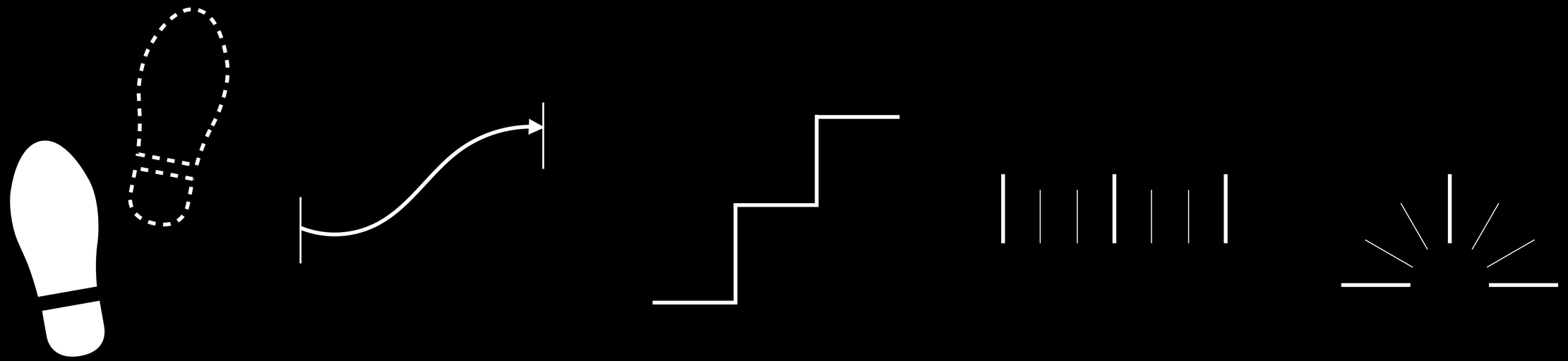
Distance



Floor Counting

Pedometer Data

CMPedometer



Steps Distance Floor Counting Pace Cadence

Pedometer Events

CMPedometer

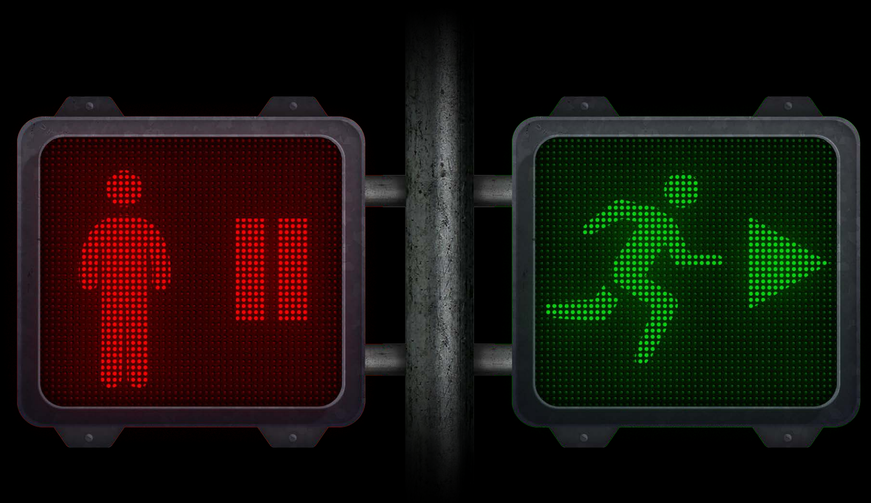
NEW

Pedometer Events

CMPedometer

NEW

Urban run



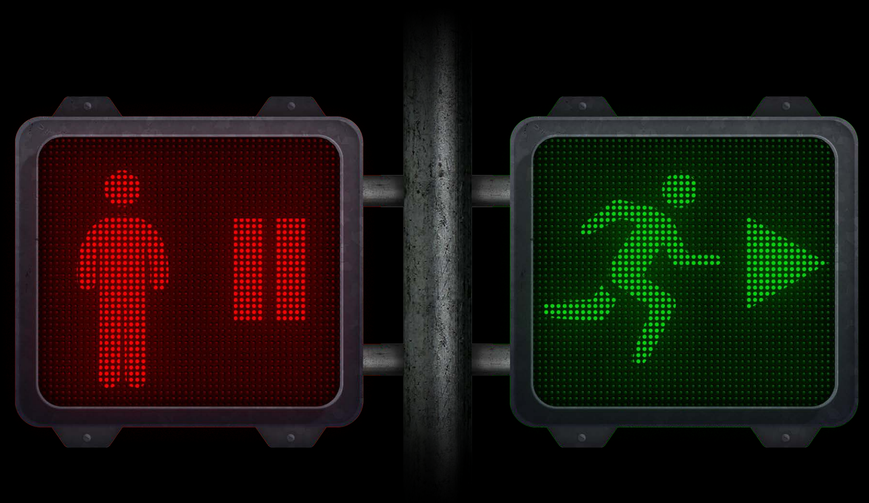
Pedometer Events

CMPedometer

NEW

Urban run

- Manual Pause and Resume



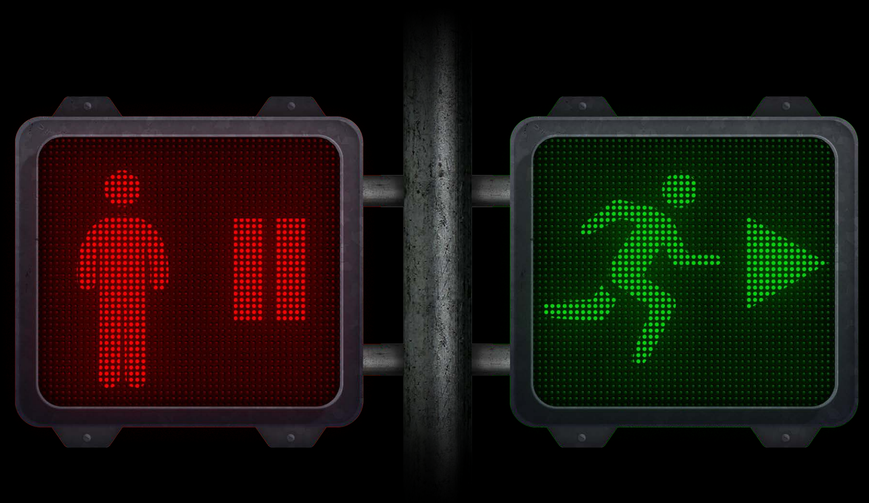
Pedometer Events

CMPedometer

NEW

Urban run

- Manual Pause and Resume
- Inaccurate distance and pace



Pedometer Events

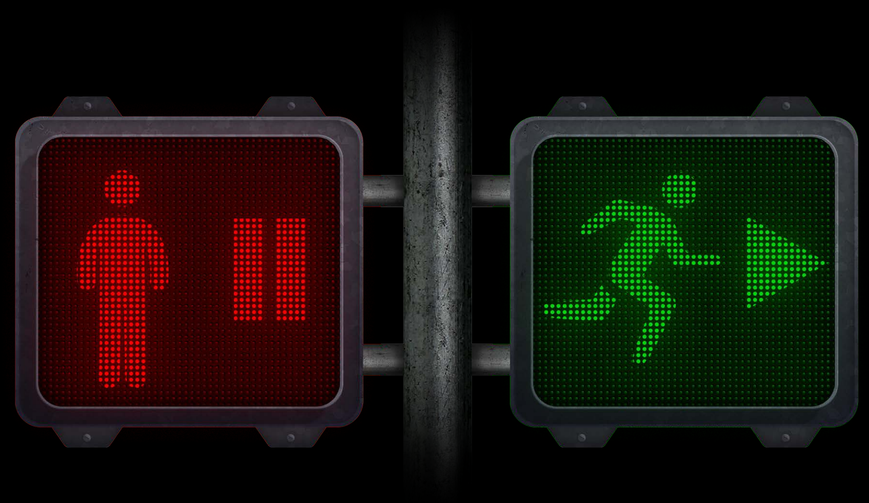
CMPedometer

NEW

Urban run

- Manual Pause and Resume
- Inaccurate distance and pace

Why not use GPS or steps to auto-pause and resume?



Pedometer Events

CMPedometer

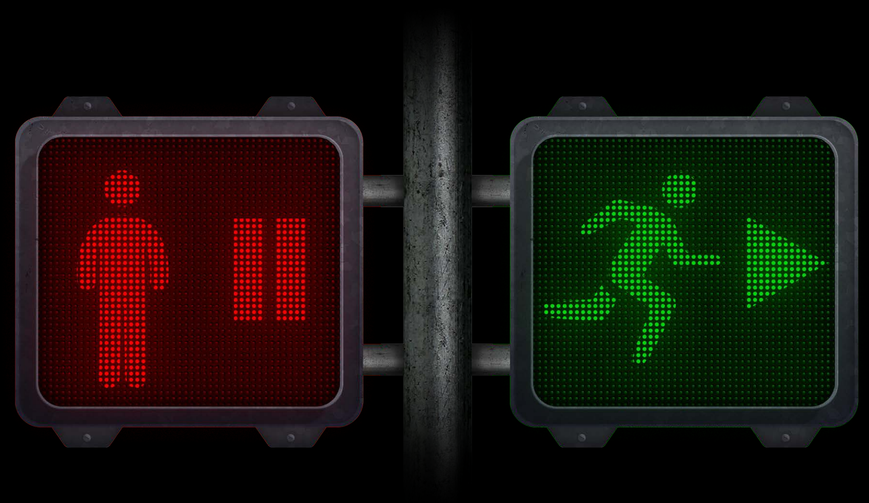
NEW

Urban run

- Manual Pause and Resume
- Inaccurate distance and pace

Why not use GPS or steps to auto-pause and resume?

- Step counter has built-in delay to avoid false positives



Pedometer Events

CMPedometer

NEW

Solution

- Predictive algorithm to recover latency

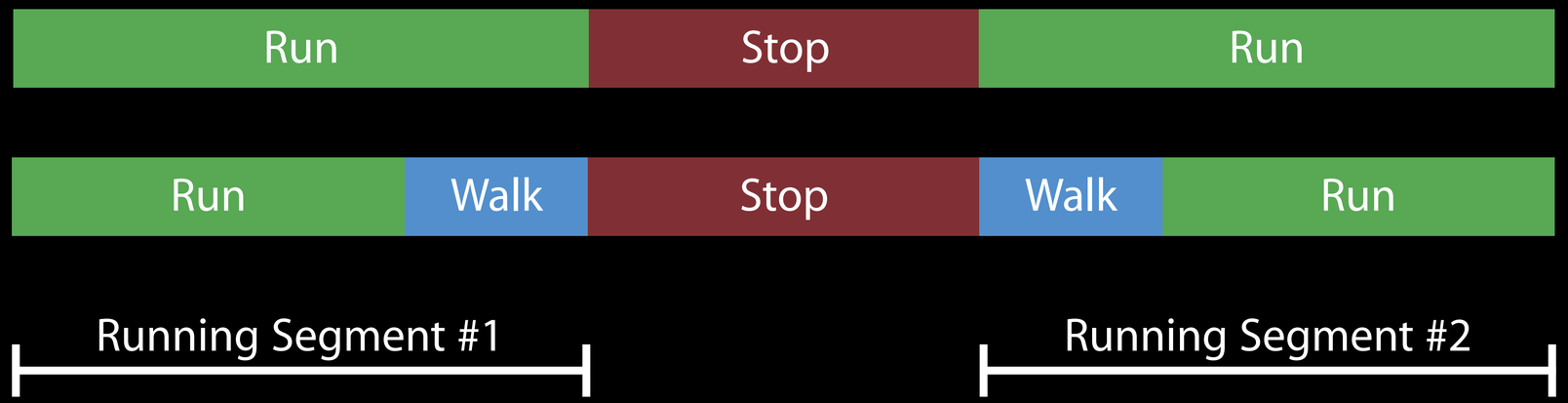
Pedometer Events

CMPedometer

NEW

Solution

- Predictive algorithm to recover latency
- Supports both running and walking pace



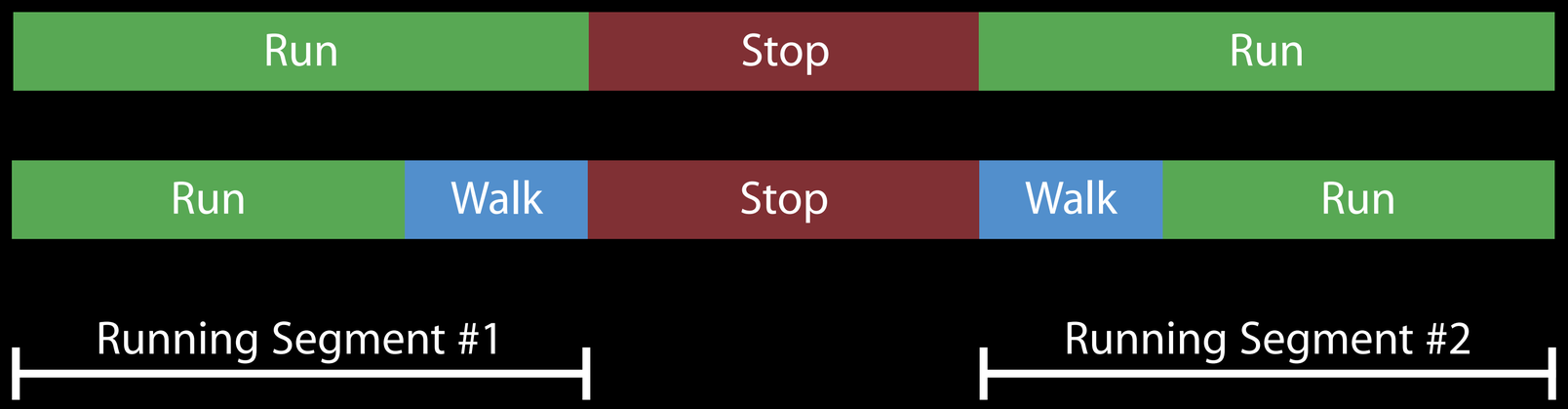
Pedometer Events

CMPedometer

NEW

Solution

- Predictive algorithm to recover latency
- Supports both running and walking pace
- Available in iOS 10 and watchOS 3



```
// Pedometer Events

public enum CMPedometerEventType : Int {
    case pause
    case resume
}

public class CMPedometerEvent : NSObject, NSSecureCoding, NSCopying {
    public var date: Date { get }
    public var type: CMPedometerEventType { get }
}

public class CMPedometer : NSObject {
    public func startEventUpdates(handler: CoreMotion.CMPedometerEventHandler)
    public func stopEventUpdates()
}
```

```
// Pedometer Events

public enum CMPedometerEventType : Int {
    case pause
    case resume
}

public class CMPedometerEvent : NSObject, NSSecureCoding, NSCopying {
    public var date: Date { get }
    public var type: CMPedometerEventType { get }
}

public class CMPedometer : NSObject {
    public func startEventUpdates(handler: CoreMotion.CMPedometerEventHandler)
    public func stopEventUpdates()
}
```

```
// Pedometer Events

public enum CMPedometerEventType : Int {
    case pause
    case resume
}

public class CMPedometerEvent : NSObject, NSSecureCoding, NSCopying {
    public var date: Date { get }
    public var type: CMPedometerEventType { get }
}

public class CMPedometer : NSObject {
    public func startEventUpdates(handler: CoreMotion.CMPedometerEventHandler)
    public func stopEventUpdates()
}
```

```
// Pedometer Events

public enum CMPedometerEventType : Int {
    case pause
    case resume
}

public class CMPedometerEvent : NSObject, NSSecureCoding, NSCopying {
    public var date: Date { get }
    public var type: CMPedometerEventType { get }
}

public class CMPedometer : NSObject {
    public func startEventUpdates(handler: CoreMotion.CMPedometerEventHandler)
    public func stopEventUpdates()
}
```

Trail Running with iPhone

CMPedometer

Trail Running with iPhone

CMPedometer

Contextual interactions

Trail Running with iPhone

CMPedometer

Contextual interactions

Elevation changes

```
// Trail Running with iPhone

class MyTrailRunningApp {

    let appQueue = OperationQueue()
    let pedometerEventManager = CMPedometer() // For Pedometer Events
    let elevationManager = CMAltimeter()      // For Relative Altitude Updates
    var relativeAltitudeNow = 0.0
    var relativeAltitudeAtResume = 0.0

    // ...
```

```
// Trail Running with iPhone

class MyTrailRunningApp {

    let appQueue = OperationQueue()
    let pedometerEventManager = CMPedometer() // For Pedometer Events
    let elevationManager = CMAltimeter()      // For Relative Altitude Updates
    var relativeAltitudeNow = 0.0
    var relativeAltitudeAtResume = 0.0

    // ...
```

```
// Trail Running with iPhone

func startMyTrailRun {

   // Developer Tip #1
   // Core Location Best Practices

   if CMAltimeter.isRelativeAltitudeAvailable() {
      elevationManager.startRelativeAltitudeUpdates(to: appQueue, withHandler: {
         (altitudeData: CMAltitudeData?, error: NSError?) in
         // Insert error handling here
         self.relativeAltitudeNow = try altitudeData?.relativeAltitude.doubleValue
   })
}
```

```
// Trail Running with iPhone

func startMyTrailRun {

    // Developer Tip #1
    // Core Location Best Practices

    if CMAltimeter.isRelativeAltitudeAvailable() {
        elevationManager.startRelativeAltitudeUpdates(to: appQueue, withHandler: {
            (altitudeData: CMAltitudeData?, error: NSError?) in
            // Insert error handling here
            self.relativeAltitudeNow = try altitudeData?.relativeAltitude.doubleValue
    })
}
```

```
// Trail Running with iPhone

func startMyTrailRun {

   // Developer Tip #1
   // Core Location Best Practices

   if CMAltimeter.isRelativeAltitudeAvailable() {
      elevationManager.startRelativeAltitudeUpdates(to: appQueue, withHandler: {
         (altitudeData: CMAltitudeData?, error: NSError?) in
         // Insert error handling here
         self.relativeAltitudeNow = try altitudeData?.relativeAltitude.doubleValue
   })
}
```

```
// Trail Running with iPhone

func startMyTrailRun {

   // Developer Tip #1
   // Core Location Best Practices

   if CMAltimeter.isRelativeAltitudeAvailable() {
      elevationManager.startRelativeAltitudeUpdates(to: appQueue, withHandler: {
         (altitudeData: CMAltitudeData?, error: NSError?) in
            // Insert error handling here
            self.relativeAltitudeNow = try altitudeData?.relativeAltitude.doubleValue
      })
   }
```

```swift
// Trail Running with iPhone

func startMyTrailRun {

   // Developer Tip #1
   // Core Location Best Practices

   if CMAltimeter.isRelativeAltitudeAvailable() {
      elevationManager.startRelativeAltitudeUpdates(to: appQueue, withHandler: {
         (altitudeData: CMAltitudeData?, error: NSError?) in
            // Insert error handling here
            self.relativeAltitudeNow = try altitudeData?.relativeAltitude.doubleValue
      })
   }
```

```
// Trail Running with iPhone

if CMPedometer.isPedometerEventTrackingAvailable() {
   pedometerEventManager.startEventUpdates { (event: CMPedometerEvent?, error: NSError?) in
      // Insert error handling here
      // Developer Tip #2
      self.appQueue.addOperation({
         if event?.type == CMPedometerEventType.resume {
            self.relativeAltitudeAtResume = self.relativeAltitudeNow
         } else if event?.type == CMPedometerEventType.pause {
            if self.relativeAltitudeNow - self.relativeAltitudeAtResume > 0.0 {
               // Take that, hill
            } else if self.relativeAltitudeNow - self.relativeAltitudeAtResume < 0.0 {
               // Whoa, watch those knees
            }
         }
      })
   }
}
```

```
// Trail Running with iPhone

if CMPedometer.isPedometerEventTrackingAvailable() {
    pedometerEventManager.startEventUpdates { (event: CMPedometerEvent?, error: NSError?) in
        // Insert error handling here
        // Developer Tip #2
        self.appQueue.addOperation({
            if event?.type == CMPedometerEventType.resume {
                self.relativeAltitudeAtResume = self.relativeAltitudeNow
            } else if event?.type == CMPedometerEventType.pause {
                if self.relativeAltitudeNow - self.relativeAltitudeAtResume > 0.0 {
                    // Take that, hill
                } else if self.relativeAltitudeNow - self.relativeAltitudeAtResume < 0.0 {
                    // Whoa, watch those knees
                }
            }
        })
    }
}
```

```
// Trail Running with iPhone

if CMPedometer.isPedometerEventTrackingAvailable() {
    pedometerEventManager.startEventUpdates { (event: CMPedometerEvent?, error: NSError?) in
        // Insert error handling here
        // Developer Tip #2
        self.appQueue.addOperation({
            if event?.type == CMPedometerEventType.resume {
                self.relativeAltitudeAtResume = self.relativeAltitudeNow
            } else if event?.type == CMPedometerEventType.pause {
                if self.relativeAltitudeNow - self.relativeAltitudeAtResume > 0.0 {
                    // Take that, hill
                } else if self.relativeAltitudeNow - self.relativeAltitudeAtResume < 0.0 {
                    // Whoa, watch those knees
                }
            }
        })
    }
}
```

```
// Trail Running with iPhone

if CMPedometer.isPedometerEventTrackingAvailable() {
   pedometerEventManager.startEventUpdates { (event: CMPedometerEvent?, error: NSError?) in
      // Insert error handling here
      // Developer Tip #2
      self.appQueue.addOperation({
         if event?.type == CMPedometerEventType.resume {
            self.relativeAltitudeAtResume = self.relativeAltitudeNow
         } else if event?.type == CMPedometerEventType.pause {
            if self.relativeAltitudeNow - self.relativeAltitudeAtResume > 0.0 {
               // Take that, hill
            } else if self.relativeAltitudeNow - self.relativeAltitudeAtResume < 0.0 {
               // Whoa, watch those knees
            }
         }
      })
   }
}
```

```swift
// Trail Running with iPhone

if CMPedometer.isPedometerEventTrackingAvailable() {
   pedometerEventManager.startEventUpdates { (event: CMPedometerEvent?, error: NSError?) in
      // Insert error handling here
      // Developer Tip #2
      self.appQueue.addOperation({
         if event?.type == CMPedometerEventType.resume {
            self.relativeAltitudeAtResume = self.relativeAltitudeNow
         } else if event?.type == CMPedometerEventType.pause {
            if self.relativeAltitudeNow - self.relativeAltitudeAtResume > 0.0 {
               // Take that, hill
            } else if self.relativeAltitudeNow - self.relativeAltitudeAtResume < 0.0 {
               // Whoa, watch those knees
            }
         }
      })
   }
}
```

```
// Trail Running with iPhone

if CMPedometer.isPedometerEventTrackingAvailable() {
   pedometerEventManager.startEventUpdates { (event: CMPedometerEvent?, error: NSError?) in
      // Insert error handling here
      // Developer Tip #2
      self.appQueue.addOperation({
         if event?.type == CMPedometerEventType.resume {
            self.relativeAltitudeAtResume = self.relativeAltitudeNow
         } else if event?.type == CMPedometerEventType.pause {
            if self.relativeAltitudeNow - self.relativeAltitudeAtResume > 0.0 {
               // Take that, hill
            } else if self.relativeAltitudeNow - self.relativeAltitudeAtResume < 0.0 {
               // Whoa, watch those knees
            }
         }
      })
   }
}
```

```
// Trail Running with iPhone

func stopMyTrailRun {

   pedometerEventManager.stopEventUpdates()
   elevationManager.stopRelativeAltitudeUpdates()

}
```

```
// Trail Running with iPhone

func stopMyTrailRun {

    pedometerEventManager.stopEventUpdates()
    elevationManager.stopRelativeAltitudeUpdates()

}
```

Pedometer

Availability

	iPhone 6/6+	iPhone 6s/6s+	iPhone SE	Apple Watch
Steps	✓	✓	✓	✓
Distance	✓	✓	✓	✓
Floor Counting	✓	✓	✓	✓
Pace	✓	✓	✓	✓
Cadence	✓	✓	✓	✓
Pedometer Events		✓	✓	✓

Device Motion on Apple Watch

Device Motion on Apple Watch

NEW

CMDeviceMotion

Device Motion on Apple Watch

NEW

CMDeviceMotion

How to observe user interaction with environment?

Device Motion on Apple Watch

NEW

CMDeviceMotion

How to observe user interaction with environment?

Device motion at wrist using sensor fusion

Device Motion on Apple Watch

NEW

CMDeviceMotion

How to observe user interaction with environment?

Device motion at wrist using sensor fusion

- Attitude
- Gravity
- Rotation rate
- User acceleration

Related Sessions

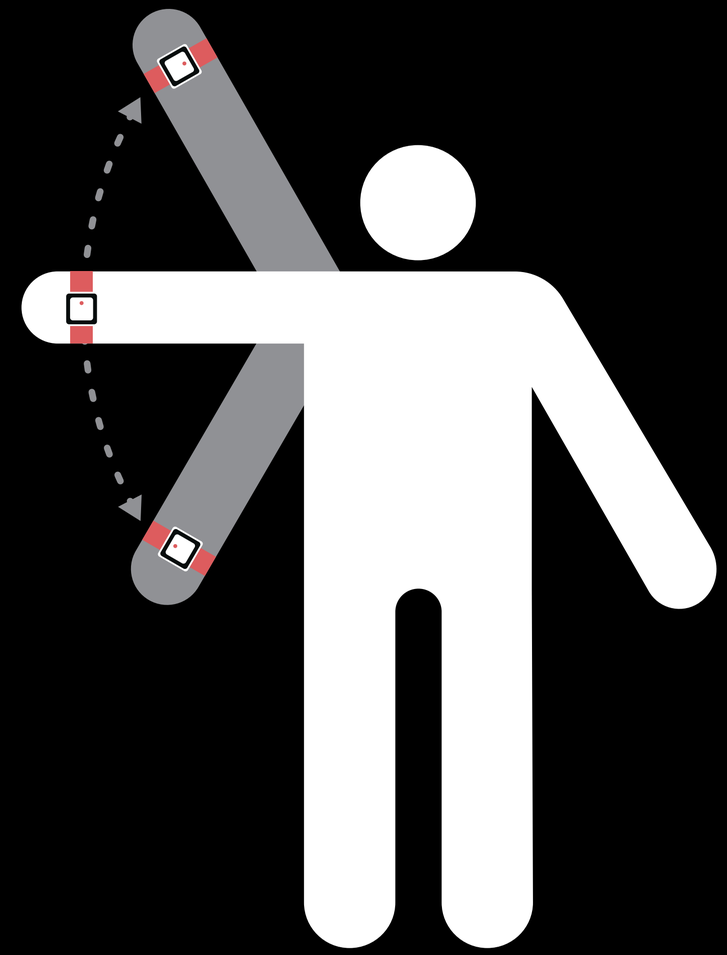
CMDeviceMotion

Understanding Core Motion	WWDC 2012
What's new in Core Motion	WWDC 2011

Attitude

CMDeviceMotion

NEW



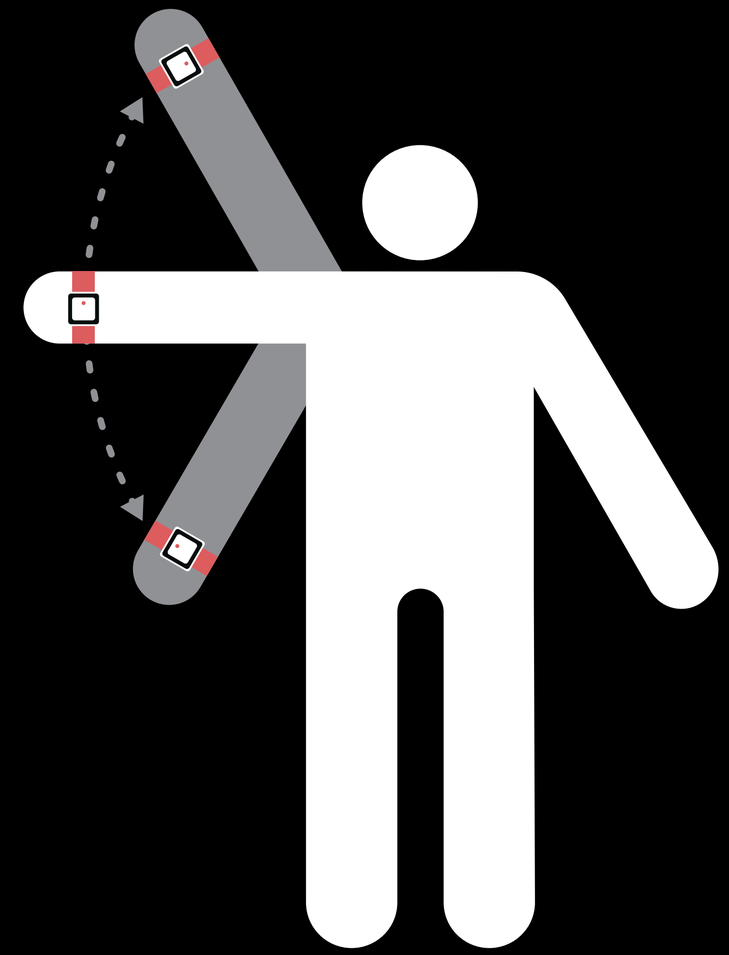
Attitude

CMDeviceMotion

NEW

Device orientation represented as

- Quaternion
- Rotation matrix
- Euler angle (roll, pitch, yaw)



Attitude

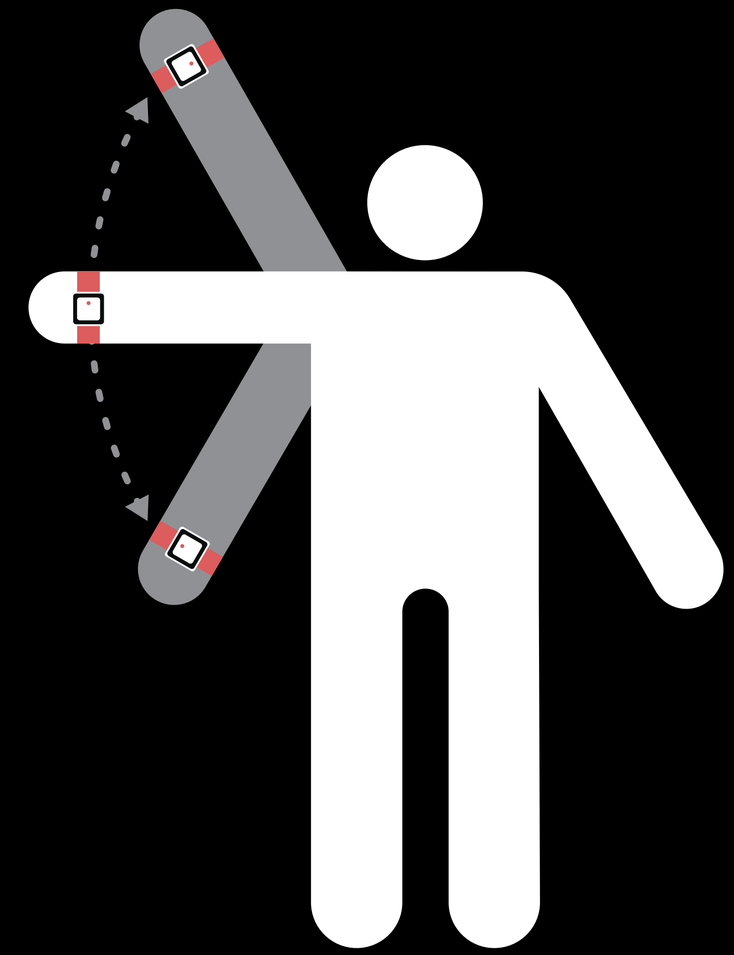
CMDeviceMotion

NEW

Device orientation represented as

- Quaternion
- Rotation matrix
- Euler angle (roll, pitch, yaw)

Relative to reference frame



Attitude

CMDeviceMotion

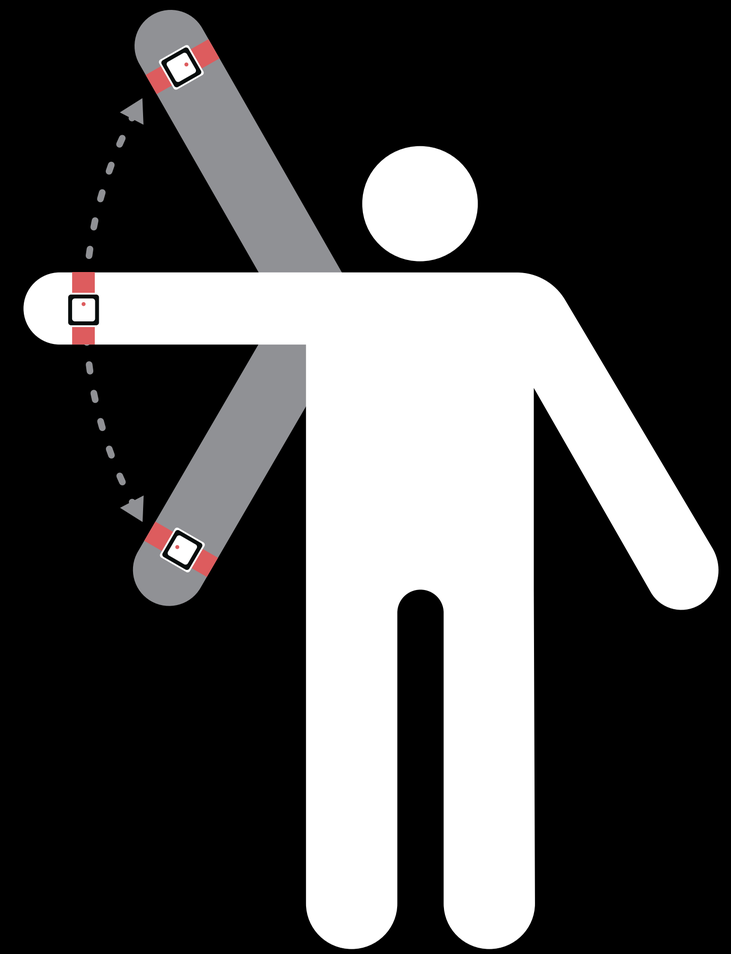
NEW

Device orientation represented as

- Quaternion
- Rotation matrix
- Euler angle (roll, pitch, yaw)

Relative to reference frame

Orientation from a point not fixed to device



Attitude

CMDeviceMotion

NEW

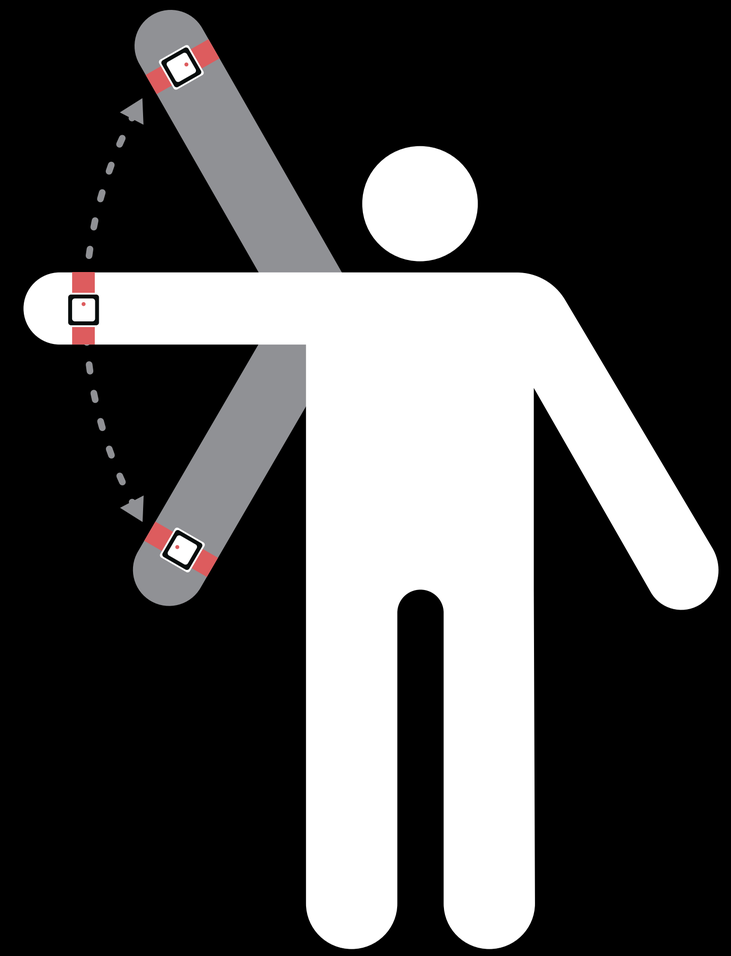
Device orientation represented as

- Quaternion
- Rotation matrix
- Euler angle (roll, pitch, yaw)

Relative to reference frame

Orientation from a point not fixed to device

Reference frame set at start of updates



Gravity

CMDeviceMotion

NEW

Gravity

CMDeviceMotion

NEW

Unit vector in device frame

Gravity

CMDeviceMotion

NEW

Unit vector in device frame

Tip and Tilt of the device

Gravity

CMDeviceMotion

NEW

Unit vector in device frame

Tip and Tilt of the device

Set using accelerometer while static

Gravity

CMDeviceMotion

NEW

Unit vector in device frame

Tip and Tilt of the device

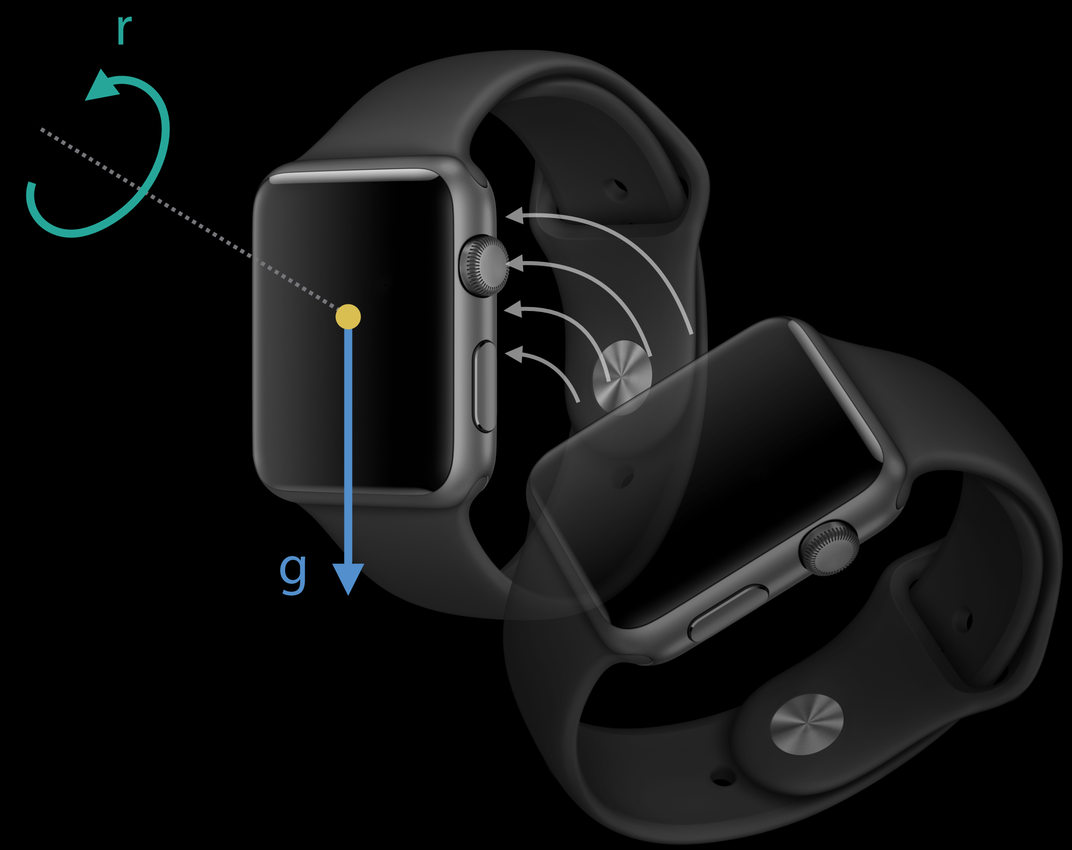
Set using accelerometer while static

Tracked using gyroscope while moving

Rotation Rate

CMDeviceMotion

NEW

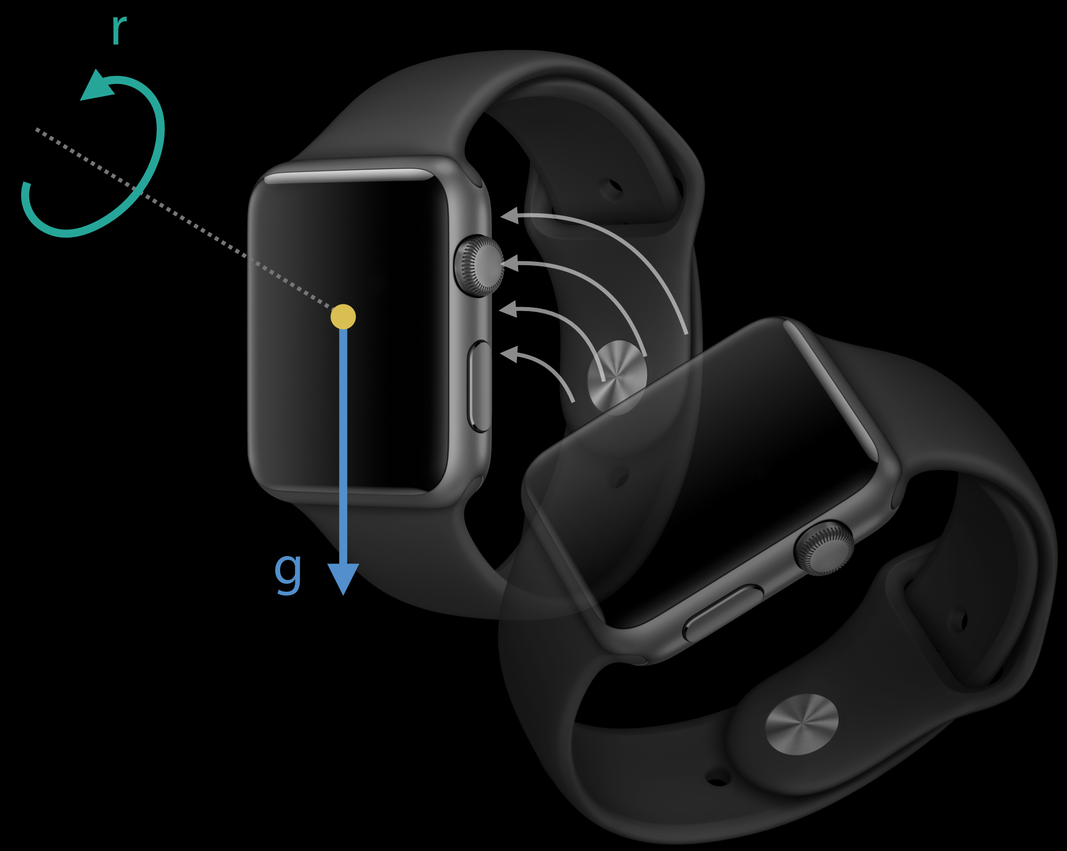


Rotation Rate

CMDeviceMotion

NEW

Change in angular motion in device frame



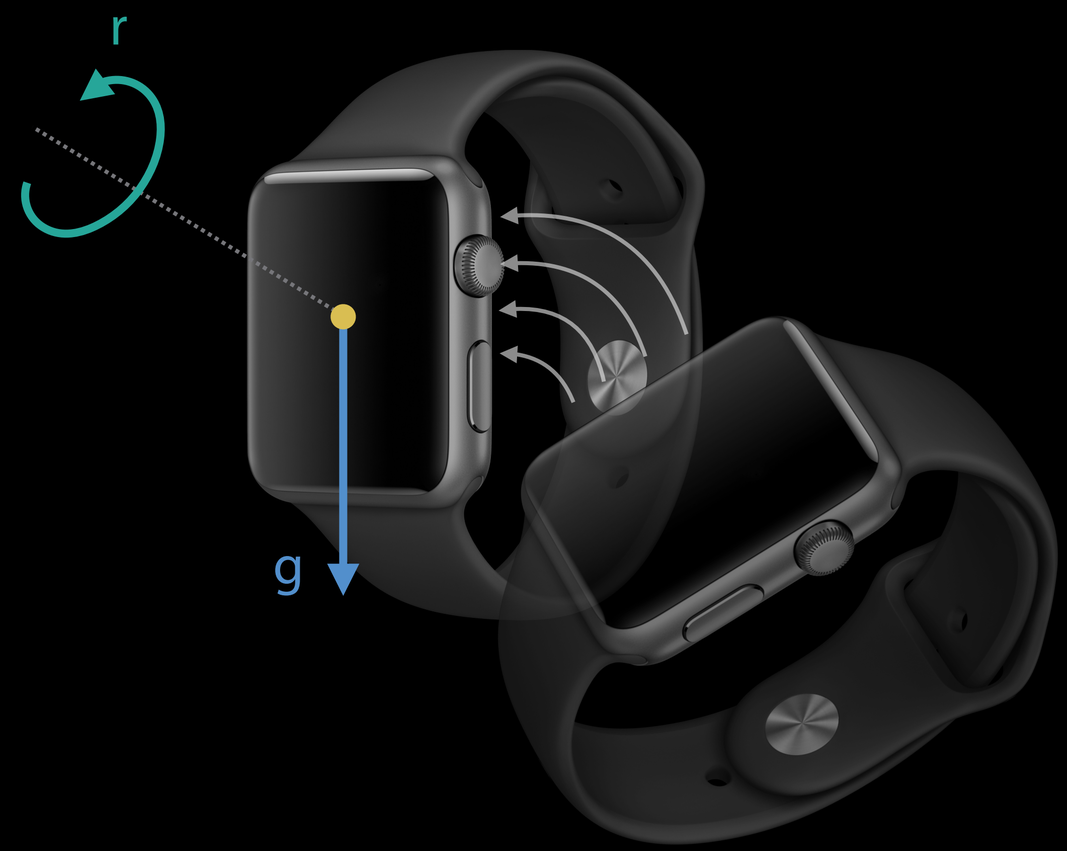
Rotation Rate

CMDeviceMotion

NEW

Change in angular motion in device frame

Wrist rotation



Rotation Rate

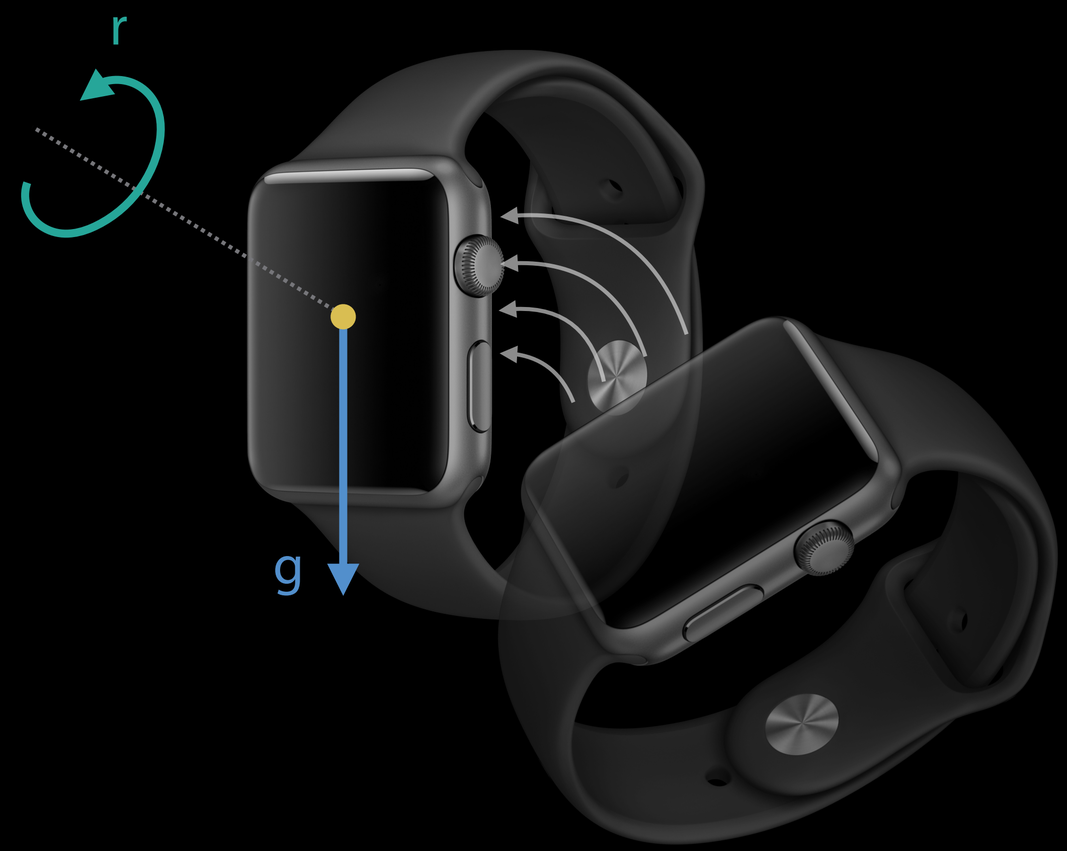
CMDeviceMotion

NEW

Change in angular motion in device frame

Wrist rotation

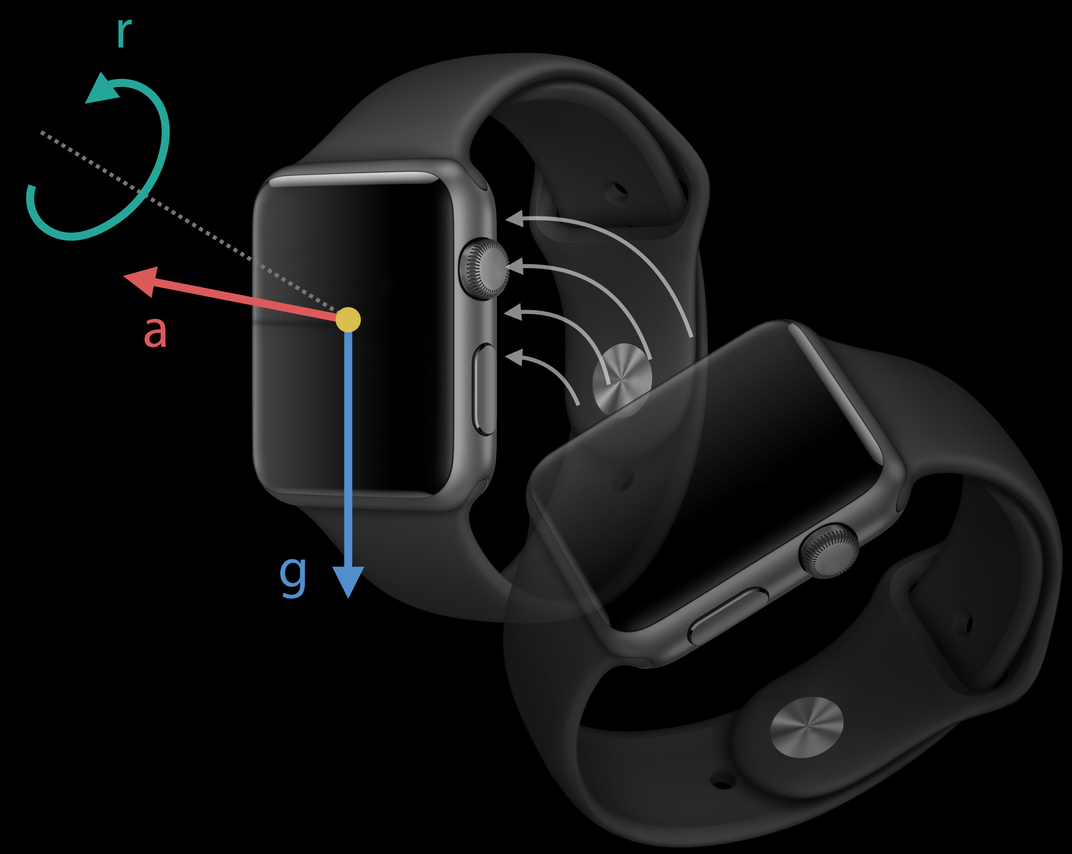
Rotation around body



User Acceleration

CMDeviceMotion

NEW

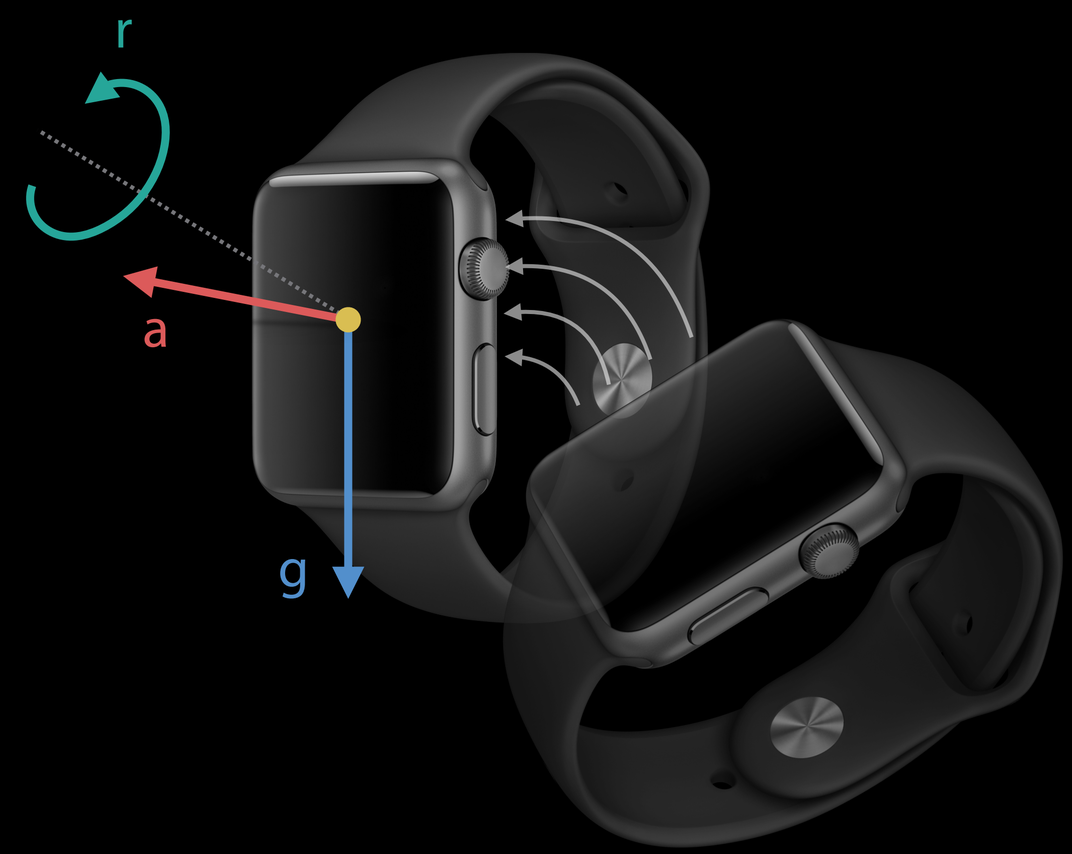


User Acceleration

CMDeviceMotion

NEW

Change in motion in device frame



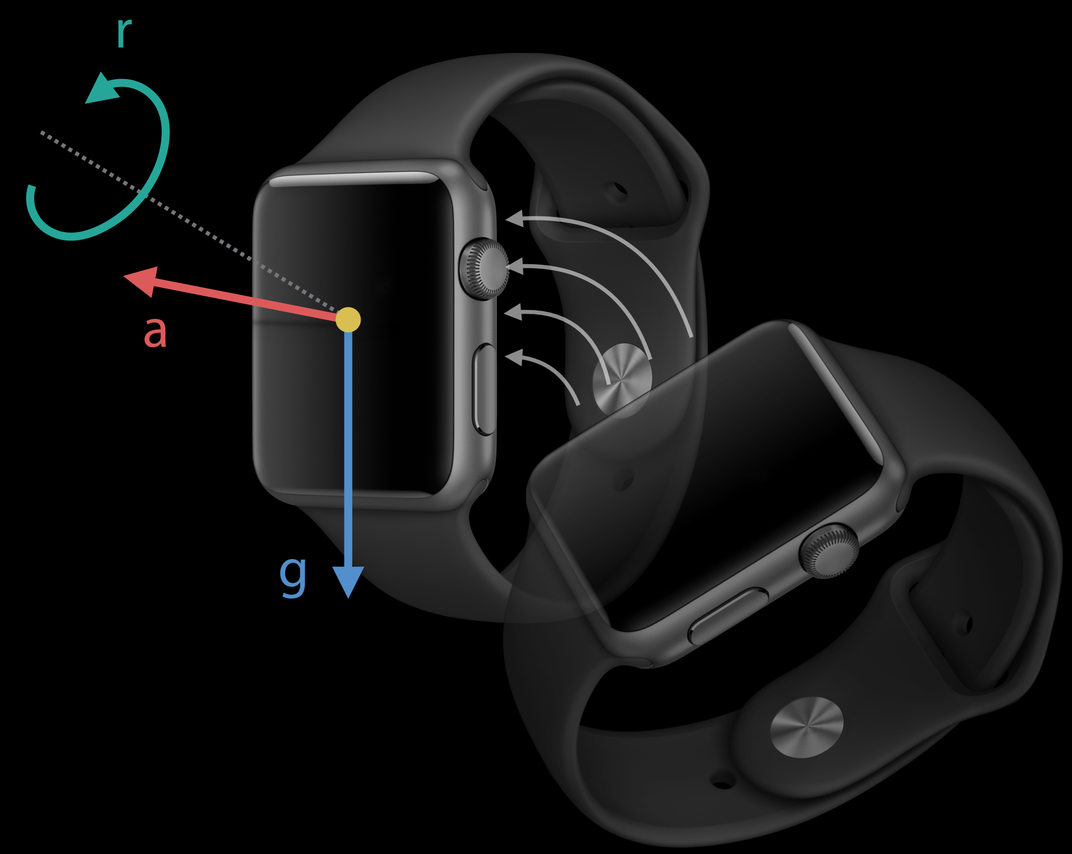
User Acceleration

CMDeviceMotion

NEW

Change in motion in device frame

Compensated for gravity



Device Motion

CMDeviceMotion

NEW

Device Motion

CMDeviceMotion

NEW

Property	Example Measurement	Example Use Case
Attitude	Rep counting	Weight training

Device Motion

CMDeviceMotion

NEW

Property	Example Measurement	Example Use Case
Attitude	Rep counting	Weight training
Gravity	Tracking poses	Yoga

Device Motion

CMDeviceMotion

NEW

Property	Example Measurement	Example Use Case
Attitude	Rep counting	Weight training
Gravity	Tracking poses	Yoga
Rotation Rate	Speed of circular motion	Bat speed

Device Motion

CMDeviceMotion

NEW

Property	Example Measurement	Example Use Case
Attitude	Rep counting	Weight training
Gravity	Tracking poses	Yoga
Rotation Rate	Speed of circular motion	Bat speed
User Acceleration	Change in linear motion	Punch / Recoil

```
// Device Motion on Apple Watch

public class CMDeviceMotion : CMLogItem {
   public var attitude: CMAttitude { get }
   public var gravity: CMAcceleration { get }
   public var rotationRate: CMRotationRate { get } // Units are in rad/sec
   public var userAcceleration: CMAcceleration { get } // Units are in G’s
}
```

```
public class CMMotionManager : NSObject {
   public var deviceMotionUpdateInterval: TimeInterval
   public var deviceMotion: CMDeviceMotion? { get }
   public func startDeviceMotionUpdates()
   public func startDeviceMotionUpdates(to queue: OperationQueue, withHandler handler:
      CoreMotion.CMDeviceMotionHandler)
   public func stopDeviceMotionUpdates()
}
```

```
// Device Motion on Apple Watch

public class CMDeviceMotion : CMLogItem {
   public var attitude: CMAttitude { get }
   public var gravity: CMAcceleration { get }
   public var rotationRate: CMRotationRate { get } // Units are in rad/sec
   public var userAcceleration: CMAcceleration { get } // Units are in G’s
}
```

```
public class CMMotionManager : NSObject {
   public var deviceMotionUpdateInterval: TimeInterval
   public var deviceMotion: CMDeviceMotion? { get }
   public func startDeviceMotionUpdates()
   public func startDeviceMotionUpdates(to queue: OperationQueue, withHandler handler:
      CoreMotion.CMDeviceMotionHandler)
   public func stopDeviceMotionUpdates()
}
```

```
// Device Motion on Apple Watch

public class CMDeviceMotion : CMLogItem {
   public var attitude: CMAttitude { get }
   public var gravity: CMAcceleration { get }
   public var rotationRate: CMRotationRate { get } // Units are in rad/sec
   public var userAcceleration: CMAcceleration { get } // Units are in G’s
}
```

```
public class CMMotionManager : NSObject {
   public var deviceMotionUpdateInterval: TimeInterval
   public var deviceMotion: CMDeviceMotion? { get }
   public func startDeviceMotionUpdates()
   public func startDeviceMotionUpdates(to queue: OperationQueue, withHandler handler:
      CoreMotion.CMDeviceMotionHandler)
   public func stopDeviceMotionUpdates()
}
```

```
// Device Motion on Apple Watch

public class CMDeviceMotion : CMLogItem {
   public var attitude: CMAttitude { get }
   public var gravity: CMAcceleration { get }
   public var rotationRate: CMRotationRate { get } // Units are in rad/sec
   public var userAcceleration: CMAcceleration { get } // Units are in G’s
}
```

```
public class CMMotionManager : NSObject {
   public var deviceMotionUpdateInterval: TimeInterval
   public var deviceMotion: CMDeviceMotion? { get }
   public func startDeviceMotionUpdates()
   public func startDeviceMotionUpdates(to queue: OperationQueue, withHandler handler:
      CoreMotion.CMDeviceMotionHandler)
   public func stopDeviceMotionUpdates()
}
```

```
// Device Motion on Apple Watch

public class CMDeviceMotion : CMLogItem {
   public var attitude: CMAttitude { get }
   public var gravity: CMAcceleration { get }
   public var rotationRate: CMRotationRate { get } // Units are in rad/sec
   public var userAcceleration: CMAcceleration { get } // Units are in G’s
}
```

```
public class CMMotionManager : NSObject {
   public var deviceMotionUpdateInterval: TimeInterval
   public var deviceMotion: CMDeviceMotion? { get }
   public func startDeviceMotionUpdates()
   public func startDeviceMotionUpdates(to queue: OperationQueue, withHandler handler:
      CoreMotion.CMDeviceMotionHandler)
   public func stopDeviceMotionUpdates()
}
```

```
// Device Motion on Apple Watch

public class CMDeviceMotion : CMLogItem {
   public var attitude: CMAttitude { get }
   public var gravity: CMAcceleration { get }
   public var rotationRate: CMRotationRate { get } // Units are in rad/sec
   public var userAcceleration: CMAcceleration { get } // Units are in G’s
}
```

```
public class CMMotionManager : NSObject {
   public var deviceMotionUpdateInterval: TimeInterval
   public var deviceMotion: CMDeviceMotion? { get }
   public func startDeviceMotionUpdates()
   public func startDeviceMotionUpdates(to queue: OperationQueue, withHandler handler:
      CoreMotion.CMDeviceMotionHandler)
   public func stopDeviceMotionUpdates()
}
```

Swing Watch

Code walkthrough

Sample App Concept

Sample App Concept

Start a Workout Session

Sample App Concept

Start a Workout Session

Register for Device Motion

Sample App Concept

Start a Workout Session

Register for Device Motion

Consume data in detection algorithm

Sample App Concept

Start a Workout Session

Register for Device Motion

Consume data in detection algorithm

`CMMotionManager`

Sample App Concept

Start a Workout Session

Register for Device Motion

Consume data in detection algorithm

CMMotionManager

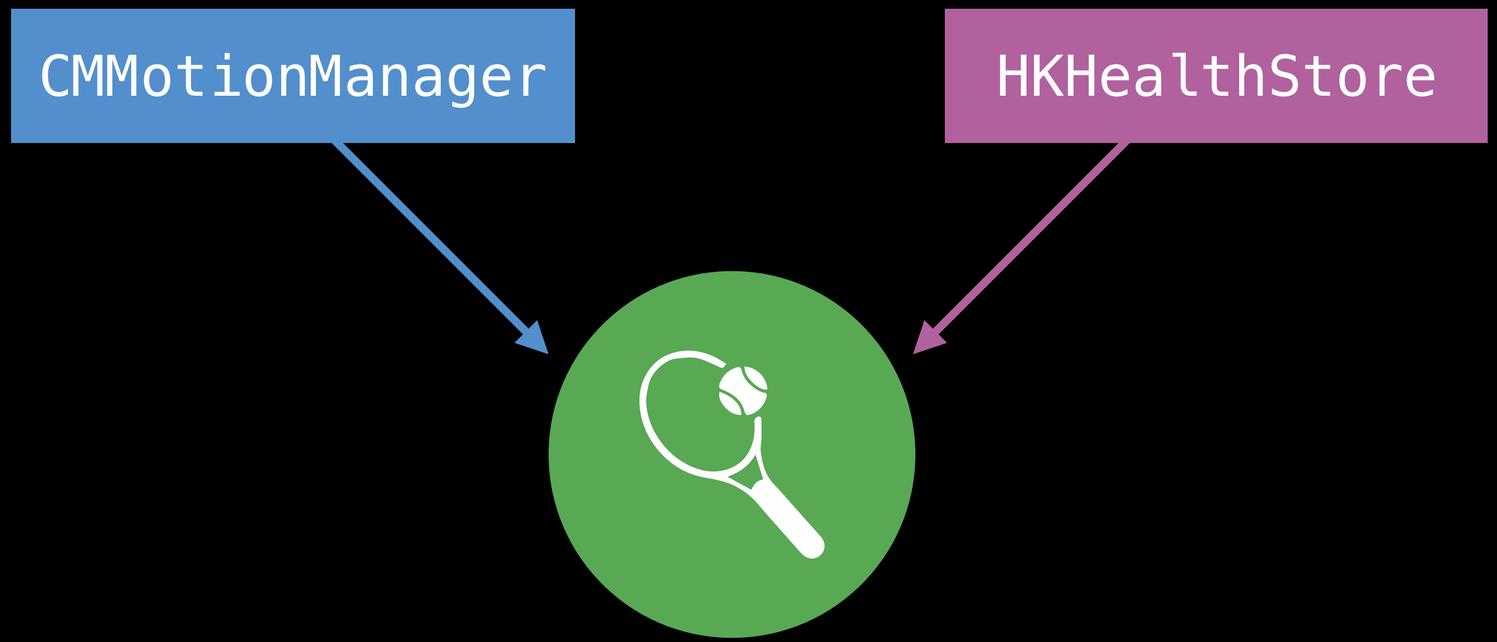
HKHealthStore

Sample App Concept

Start a Workout Session

Register for Device Motion

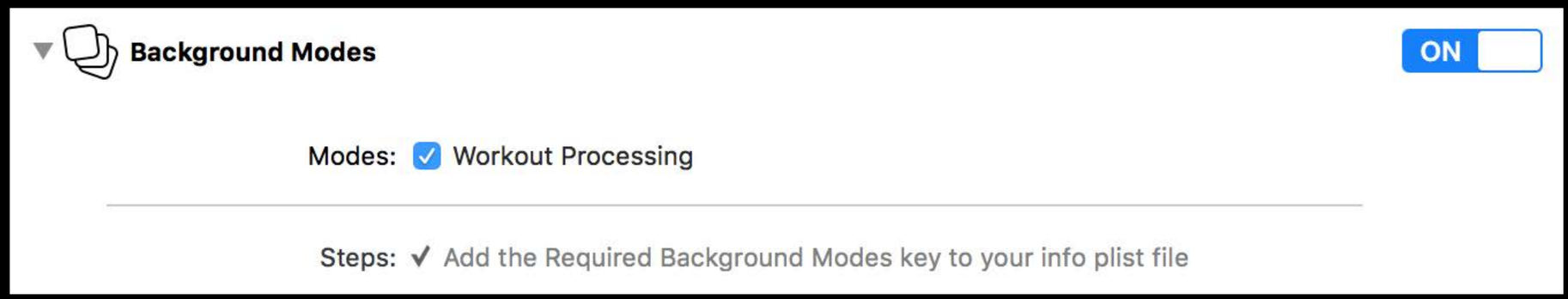
Consume data in detection algorithm



Configuration

Configuration

Enable Background Modes: Workout processing

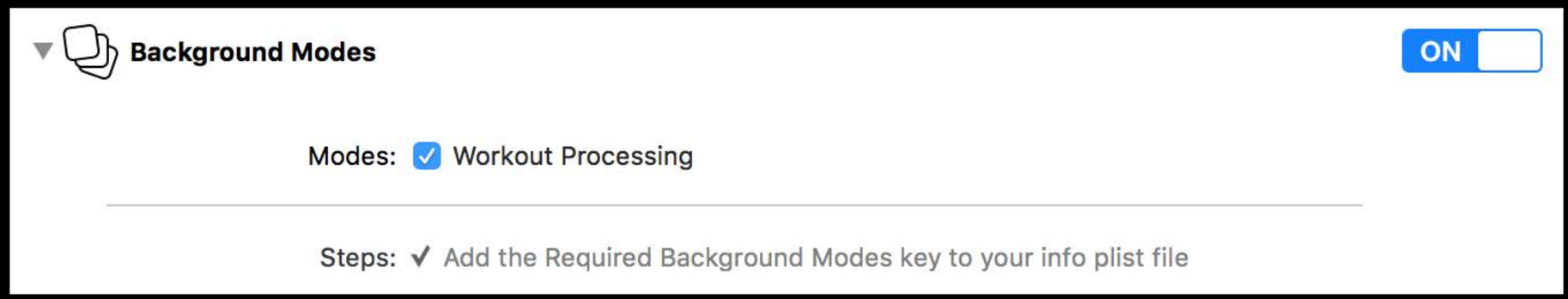



```
// WatchKit Extension/Info.plist
<key>WKBackgroundModes</key>
<array>
    <string>workout-processing</string>
</array>
```

Configuration

Enable Background Modes: Workout processing

Only during an active workout session

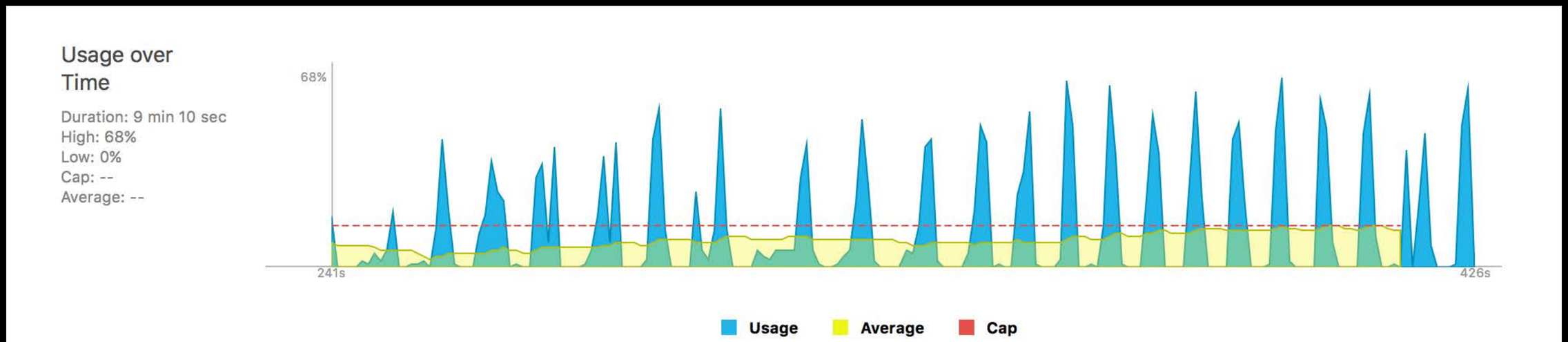



```
// WatchKit Extension/Info.plist
<key>WKBackgroundModes</key>
<array>
    <string>workout-processing</string>
</array>
```

Resource Limits

Average CPU Usage must be limited

Violating this will suspend your app



Related Session

Getting the Most Out of HealthKit	Nob Hill	Wednesday 9:00AM

Sample App Concept

Sample App Concept

Sample App Concept

Sample App Concept

User interface

Sample App Concept

User interface

Workout Manager

Sample App Concept

User interface

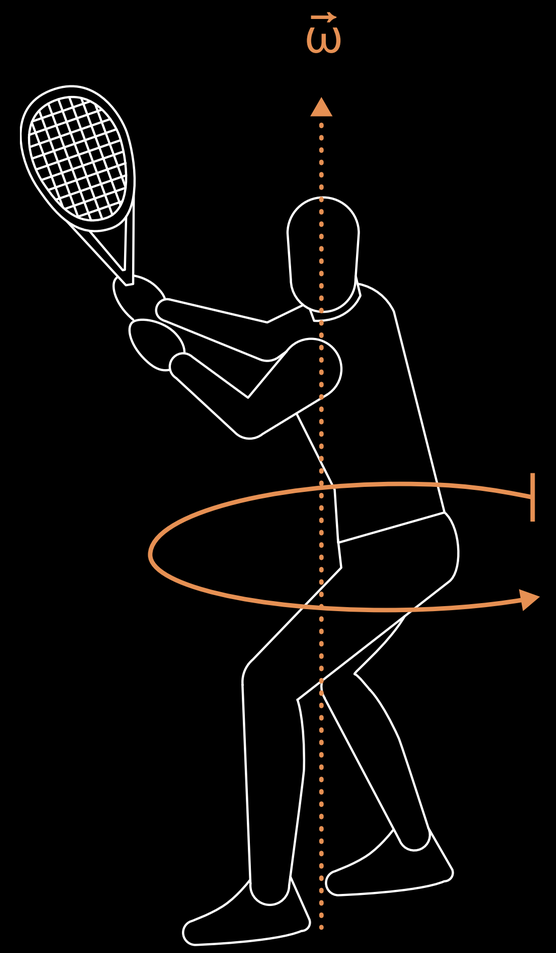
Workout Manager

Motion Manager

Swing Algorithm

Swing Algorithm

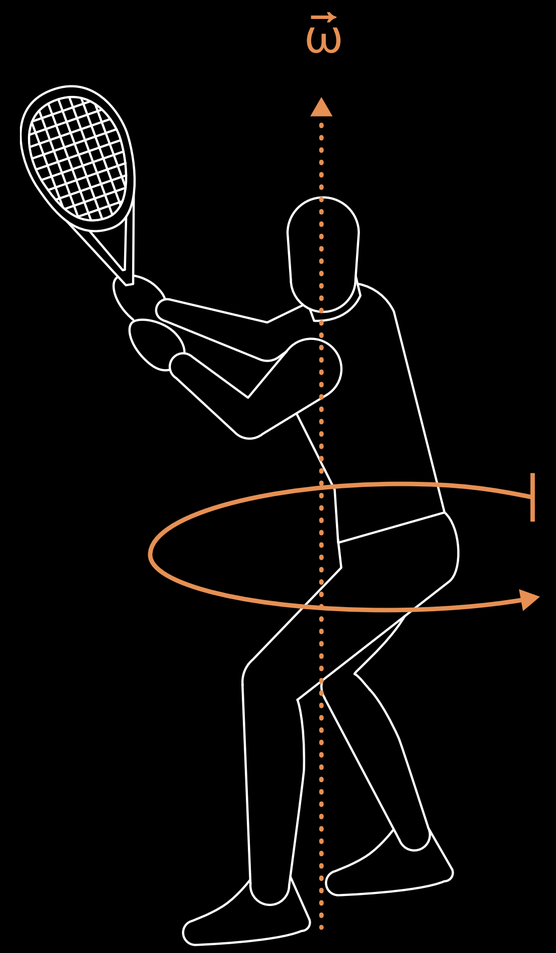
Model swing as a rotation about the user



Swing Algorithm

Model swing as a rotation about the user

Extract component parallel with ground

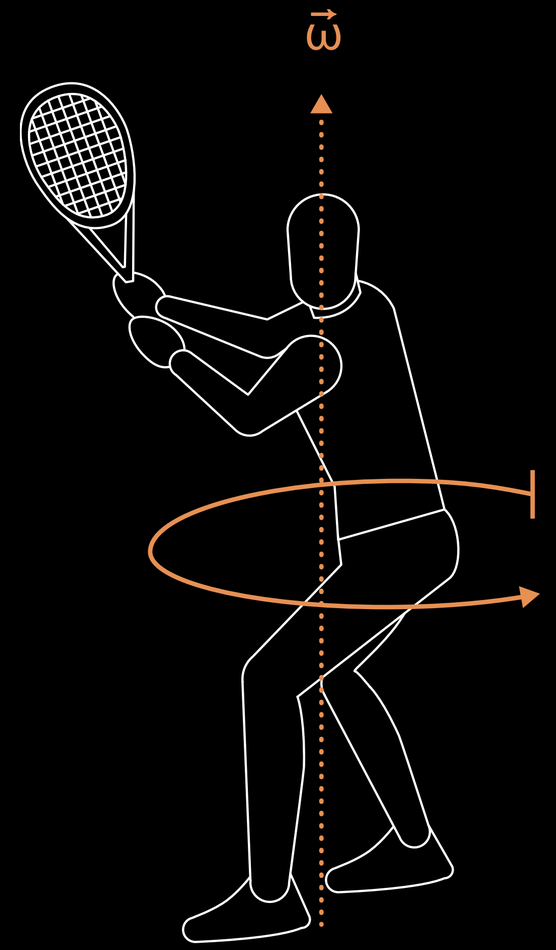


Swing Algorithm

Model swing as a rotation about the user

Extract component parallel with ground

Gate detections on sufficient angle and speed



Code!

User Interface

User Interface

User Interface

User Interface

User Interface

User Interface

User Interface

User Interface

Workout Manager

Workout Manager

Create Workout Sessions

Workout Manager

Create Workout Sessions

Direct Motion Manager

```
// class WorkoutManager {
// ...
    func startWorkout() {
        if (session != nil) {
            return
        }
        let workoutConfiguration = HKWorkoutConfiguration()
        workoutConfiguration.activityType = .tennis
        workoutConfiguration.locationType = .outdoor
        do {
            self.session = try HKWorkoutSession(configuration: workoutConfiguration)
        } catch {
            fatalError("Unable to create the workout session!")
        }

        healthStore.start(session!)
        motionManager.startUpdates()
    }
// ...
```

```
// class WorkoutManager {
// ...
    func startWorkout() {
        if (session != nil) {
            return
        }
        let workoutConfiguration = HKWorkoutConfiguration()
        workoutConfiguration.activityType = .tennis
        workoutConfiguration.locationType = .outdoor
        do {
            self.session = try HKWorkoutSession(configuration: workoutConfiguration)
        } catch {
            fatalError("Unable to create the workout session!")
        }

        healthStore.start(session!)
        motionManager.startUpdates()
    }
// ...
```

```
// class WorkoutManager {
// ...
    func startWorkout() {
        if (session != nil) {
            return
        }
        let workoutConfiguration = HKWorkoutConfiguration()
        workoutConfiguration.activityType = .tennis
        workoutConfiguration.locationType = .outdoor
        do {
            self.session = try HKWorkoutSession(configuration: workoutConfiguration)
        } catch {
            fatalError("Unable to create the workout session!")
        }

        healthStore.start(session!)
        motionManager.startUpdates()
    }
// ...
```

```
// class WorkoutManager {
// ...
    func stopWorkout() {
        if (session == nil) {
            return
        }
        motionManager.stopUpdates()
        healthStore.end(session!)
        // Store session data with HealthKit
        session = nil
    }
}
```

```
// class WorkoutManager {
// ...
    func stopWorkout() {
        if (session == nil) {
            return
        }
        motionManager.stopUpdates()
        healthStore.end(session!)
        // Store session data with HealthKit
        session = nil
    }
}
```

Motion Manager

Motion Manager

Interact with Core Motion

Motion Manager

Interact with Core Motion

Implement detection algorithm

```swift
import CoreMotion
class MotionManager {
    let motionManager = CMMotionManager()
    let queue = OperationQueue()
    let wristLocationIsLeft = WKInterfaceDevice.current().wristLocation == .left

    var forehandCount = 0
    var backhandCount = 0
    var recentDetection = false

    let sampleInterval = 1.0 / 50
    let rateAlongGravityBuffer = RunningBuffer(size: 50)

    let yawThreshold = 1.95 // Radians
    let rateThreshold = 5.5 // Radians / sec
    let resetThreshold = 5.5 * 0.05 // 5% of rate threshold

    weak var delegate: MotionManagerDelegate?
// ...
```

```
import CoreMotion
class MotionManager {
    let motionManager = CMMotionManager()
    let queue = OperationQueue()
    let wristLocationIsLeft = WKInterfaceDevice.current().wristLocation == .left

    var forehandCount = 0
    var backhandCount = 0
    var recentDetection = false

    let sampleInterval = 1.0 / 50
    let rateAlongGravityBuffer = RunningBuffer(size: 50)

    let yawThreshold = 1.95 // Radians
    let rateThreshold = 5.5 // Radians / sec
    let resetThreshold = 5.5 * 0.05 // 5% of rate threshold

    weak var delegate: MotionManagerDelegate?
// ...
```

```swift
import CoreMotion
class MotionManager {
    let motionManager = CMMotionManager()
    let queue = OperationQueue()
    let wristLocationIsLeft = WKInterfaceDevice.current().wristLocation == .left

    var forehandCount = 0
    var backhandCount = 0
    var recentDetection = false

    let sampleInterval = 1.0 / 50
    let rateAlongGravityBuffer = RunningBuffer(size: 50)

    let yawThreshold = 1.95 // Radians
    let rateThreshold = 5.5 // Radians / sec
    let resetThreshold = 5.5 * 0.05 // 5% of rate threshold

    weak var delegate: MotionManagerDelegate?
// ...
```

```
import CoreMotion
class MotionManager {
    let motionManager = CMMotionManager()
    let queue = OperationQueue()
    let wristLocationIsLeft = WKInterfaceDevice.current().wristLocation == .left

    var forehandCount = 0
    var backhandCount = 0
    var recentDetection = false

    let sampleInterval = 1.0 / 50
    let rateAlongGravityBuffer = RunningBuffer(size: 50)

    let yawThreshold = 1.95 // Radians
    let rateThreshold = 5.5 // Radians / sec
    let resetThreshold = 5.5 * 0.05 // 5% of rate threshold

    weak var delegate: MotionManagerDelegate?
// ...
```

```
import CoreMotion
class MotionManager {
    let motionManager = CMMotionManager()
    let queue = OperationQueue()
    let wristLocationIsLeft = WKInterfaceDevice.current().wristLocation == .left

    var forehandCount = 0
    var backhandCount = 0
    var recentDetection = false

    let sampleInterval = 1.0 / 50
    let rateAlongGravityBuffer = RunningBuffer(size: 50)

    let yawThreshold = 1.95 // Radians
    let rateThreshold = 5.5 // Radians / sec
    let resetThreshold = 5.5 * 0.05 // 5% of rate threshold

    weak var delegate: MotionManagerDelegate?
// ...
```

```swift
import CoreMotion
class MotionManager {
    let motionManager = CMMotionManager()
    let queue = OperationQueue()
    let wristLocationIsLeft = WKInterfaceDevice.current().wristLocation == .left

    var forehandCount = 0
    var backhandCount = 0
    var recentDetection = false

    let sampleInterval = 1.0 / 50
    let rateAlongGravityBuffer = RunningBuffer(size: 50)

    let yawThreshold = 1.95 // Radians
    let rateThreshold = 5.5 // Radians / sec
    let resetThreshold = 5.5 * 0.05 // 5% of rate threshold

    weak var delegate: MotionManagerDelegate?
// ...
```

```swift
import CoreMotion
class MotionManager {
    let motionManager = CMMotionManager()
    let queue = OperationQueue()
    let wristLocationIsLeft = WKInterfaceDevice.current().wristLocation == .left

    var forehandCount = 0
    var backhandCount = 0
    var recentDetection = false

    let sampleInterval = 1.0 / 50
    let rateAlongGravityBuffer = RunningBuffer(size: 50)

    let yawThreshold = 1.95 // Radians
    let rateThreshold = 5.5 // Radians / sec
    let resetThreshold = 5.5 * 0.05 // 5% of rate threshold

    weak var delegate: MotionManagerDelegate?
// ...
```

```
// class MotionManager {
// ...
    init() {
        queue.maxConcurrentOperationCount = 1
        queue.name = "MotionManagerQueue"
    }

    func resetAllState() {
        forehandCount = 0
        backhandCount = 0
        recentDetection = false
        rateAlongGravityBuffer.reset()

        delegate?.didUpdateForehandSwingCount(self, forehandCount: 0)
        delegate?.didUpdateBackhandSwingCount(self, backhandCount: 0)
    }
// ...
```

```
// class MotionManager {
// ...
    init() {
        queue.maxConcurrentOperationCount = 1
        queue.name = "MotionManagerQueue"
    }

    func resetAllState() {
        forehandCount = 0
        backhandCount = 0
        recentDetection = false
        rateAlongGravityBuffer.reset()

        delegate?.didUpdateForehandSwingCount(self, forehandCount: 0)
        delegate?.didUpdateBackhandSwingCount(self, backhandCount: 0)
    }
// ...
```

```
// class MotionManager {
// ...
    init() {
        queue.maxConcurrentOperationCount = 1
        queue.name = "MotionManagerQueue"
    }

    func resetAllState() {
        forehandCount = 0
        backhandCount = 0
        recentDetection = false
        rateAlongGravityBuffer.reset()

        delegate?.didUpdateForehandSwingCount(self, forehandCount: 0)
        delegate?.didUpdateBackhandSwingCount(self, backhandCount: 0)
    }
// ...
```

```
// class MotionManager {
// ...
    func incrementForehandCountAndUpdateDelegate() {
        if (!recentDetection) {
            forehandCount += 1
            recentDetection = true
            delegate?.didUpdateForehandSwingCount(self, forehandCount: forehandCount)
        }
    }

    func incrementBackhandCountAndUpdateDelegate() {
        if (!recentDetection) {
            backhandCount += 1
            recentDetection = true
            delegate?.didUpdateBackhandSwingCount(self, backhandCount: backhandCount)
        }
    }
// ...
```

```
// class MotionManager {
// ...
    func incrementForehandCountAndUpdateDelegate() {
        if (!recentDetection) {
            forehandCount += 1
            recentDetection = true
            delegate?.didUpdateForehandSwingCount(self, forehandCount: forehandCount)
        }
    }

    func incrementBackhandCountAndUpdateDelegate() {
        if (!recentDetection) {
            backhandCount += 1
            recentDetection = true
            delegate?.didUpdateBackhandSwingCount(self, backhandCount: backhandCount)
        }
    }
// ...
```

```
// class MotionManager {
    func startUpdates() {
        if !motionManager.isDeviceMotionAvailable {
            print("Device Motion is not available.")
            return
        }
        resetAllState()
        motionManager.deviceMotionUpdateInterval = sampleInterval // 0.02 seconds
        motionManager.startDeviceMotionUpdates(to: queue) {
            (deviceMotion: CMDeviceMotion?, error:NSError?) in
            if error != nil {
                // Handle Error
            }
            if deviceMotion != nil {
                self.processDeviceMotion(deviceMotion!)
            }
        }
    }
// ...
```

```
// class MotionManager {
    func startUpdates() {
        if !motionManager.isDeviceMotionAvailable {
            print("Device Motion is not available.")
            return
        }
        resetAllState()
        motionManager.deviceMotionUpdateInterval = sampleInterval // 0.02 seconds
        motionManager.startDeviceMotionUpdates(to: queue) {
            (deviceMotion: CMDeviceMotion?, error:NSError?) in
            if error != nil {
                // Handle Error
            }
            if deviceMotion != nil {
                self.processDeviceMotion(deviceMotion!)
            }
        }
    }
// ...
```

```
// class MotionManager {
    func startUpdates() {
        if !motionManager.isDeviceMotionAvailable {
            print("Device Motion is not available.")
            return
        }
        resetAllState()
        motionManager.deviceMotionUpdateInterval = sampleInterval // 0.02 seconds
        motionManager.startDeviceMotionUpdates(to: queue) {
            (deviceMotion: CMDeviceMotion?, error:NSError?) in
            if error != nil {
                // Handle Error
            }
            if deviceMotion != nil {
                self.processDeviceMotion(deviceMotion!)
            }
        }
    }
// ...
```

```
// class MotionManager {
    func startUpdates() {
        if !motionManager.isDeviceMotionAvailable {
            print("Device Motion is not available.")
            return
        }
        resetAllState()
        motionManager.deviceMotionUpdateInterval = sampleInterval // 0.02 seconds
        motionManager.startDeviceMotionUpdates(to: queue) {
            (deviceMotion: CMDeviceMotion?, error:NSError?) in
            if error != nil {
                // Handle Error
            }
            if deviceMotion != nil {
                self.processDeviceMotion(deviceMotion!)
            }
        }
    }
// ...
```

```swift
// class MotionManager {
    func processDeviceMotion(_ deviceMotion: CMDeviceMotion) {
        let gravity = deviceMotion.gravity
        let rotationRate = deviceMotion.rotationRate

        let rateAlongGravity = rotationRate.x * gravity.x // ω⃗ · ĝ
                             + rotationRate.y * gravity.y
                             + rotationRate.z * gravity.z
        rateAlongGravityBuffer.addSample(rateAlongGravity)

        if !rateAlongGravityBuffer.isFull() {
            return
        }

        let accumulatedYawRot = rateAlongGravityBuffer.sum() * sampleInterval
        let peakRate = accumulatedYawRot > 0 ?
            rateAlongGravityBuffer.max() : rateAlongGravityBuffer.min()
//      ...
```

```swift
// class MotionManager {
    func processDeviceMotion(_ deviceMotion: CMDeviceMotion) {
        let gravity = deviceMotion.gravity
        let rotationRate = deviceMotion.rotationRate

        let rateAlongGravity = rotationRate.x * gravity.x // ω⃗ · ĝ
                             + rotationRate.y * gravity.y
                             + rotationRate.z * gravity.z
        rateAlongGravityBuffer.addSample(rateAlongGravity)

        if !rateAlongGravityBuffer.isFull() {
            return
        }

        let accumulatedYawRot = rateAlongGravityBuffer.sum() * sampleInterval
        let peakRate = accumulatedYawRot > 0 ?
            rateAlongGravityBuffer.max() : rateAlongGravityBuffer.min()
//      ...
```

```swift
// class MotionManager {
    func processDeviceMotion(_ deviceMotion: CMDeviceMotion) {
        let gravity = deviceMotion.gravity
        let rotationRate = deviceMotion.rotationRate

        let rateAlongGravity = rotationRate.x * gravity.x // ω⃗ · ĝ
                             + rotationRate.y * gravity.y
                             + rotationRate.z * gravity.z
        rateAlongGravityBuffer.addSample(rateAlongGravity)

        if !rateAlongGravityBuffer.isFull() {
            return
        }

        let accumulatedYawRot = rateAlongGravityBuffer.sum() * sampleInterval
        let peakRate = accumulatedYawRot > 0 ?
            rateAlongGravityBuffer.max() : rateAlongGravityBuffer.min()
//      ...
```

```swift
// class MotionManager {
    func processDeviceMotion(_ deviceMotion: CMDeviceMotion) {
        let gravity = deviceMotion.gravity
        let rotationRate = deviceMotion.rotationRate

        let rateAlongGravity = rotationRate.x * gravity.x // ω⃗ · ĝ
                             + rotationRate.y * gravity.y
                             + rotationRate.z * gravity.z
        rateAlongGravityBuffer.addSample(rateAlongGravity)

        if !rateAlongGravityBuffer.isFull() {
            return
        }

        let accumulatedYawRot = rateAlongGravityBuffer.sum() * sampleInterval
        let peakRate = accumulatedYawRot > 0 ?
            rateAlongGravityBuffer.max() : rateAlongGravityBuffer.min()
//      ...
```

```
// class MotionManager {
    func processDeviceMotion(_ deviceMotion: CMDeviceMotion) {
        let gravity = deviceMotion.gravity
        let rotationRate = deviceMotion.rotationRate

        let rateAlongGravity = rotationRate.x * gravity.x // ω⃗ · ĝ
                             + rotationRate.y * gravity.y
                             + rotationRate.z * gravity.z
        rateAlongGravityBuffer.addSample(rateAlongGravity)

        if !rateAlongGravityBuffer.isFull() {
            return
        }

        let accumulatedYawRot = rateAlongGravityBuffer.sum() * sampleInterval
        let peakRate = accumulatedYawRot > 0 ?
            rateAlongGravityBuffer.max() : rateAlongGravityBuffer.min()
//      ...
```

```
// class MotionManager {
//    func processDeviceMotion(_ deviceMotion: CMDeviceMotion) {
//      ...
        if (accumulatedYawRot < -yawThreshold && peakRate < -rateThreshold) {
            // Counter clockwise swing.
            if (wristLocationIsLeft) {
                incrementBackhandCountAndUpdateDelegate()
            } else {
                incrementForehandCountAndUpdateDelegate()
            }
        } else if (accumulatedYawRot > yawThreshold && peakRate > rateThreshold) {
            // Clockwise swing.
            if (wristLocationIsLeft) {
                incrementForehandCountAndUpdateDelegate()
            } else {
                incrementBackhandCountAndUpdateDelegate()
            }
        }
//      ...
```

```
// class MotionManager {
//     func processDeviceMotion(_ deviceMotion: CMDeviceMotion) {
//       ...
            if (accumulatedYawRot < -yawThreshold && peakRate < -rateThreshold) {
                // Counter clockwise swing.
                if (wristLocationIsLeft) {
                    incrementBackhandCountAndUpdateDelegate()
                } else {
                    incrementForehandCountAndUpdateDelegate()
                }
            } else if (accumulatedYawRot > yawThreshold && peakRate > rateThreshold) {
                // Clockwise swing.
                if (wristLocationIsLeft) {
                    incrementForehandCountAndUpdateDelegate()
                } else {
                    incrementBackhandCountAndUpdateDelegate()
                }
            }
//       ...
```

```
// class MotionManager {
//     func processDeviceMotion(_ deviceMotion: CMDeviceMotion) {
//       ...
          if (accumulatedYawRot < -yawThreshold && peakRate < -rateThreshold) {
              // Counter clockwise swing.
              if (wristLocationIsLeft) {
                  incrementBackhandCountAndUpdateDelegate()
              } else {
                  incrementForehandCountAndUpdateDelegate()
              }
          } else if (accumulatedYawRot > yawThreshold && peakRate > rateThreshold) {
              // Clockwise swing.
              if (wristLocationIsLeft) {
                  incrementForehandCountAndUpdateDelegate()
              } else {
                  incrementBackhandCountAndUpdateDelegate()
              }
          }
//       ...
```

```
// class MotionManager {
//    func processDeviceMotion(_ deviceMotion: CMDeviceMotion) {
//      ...
        if (recentDetection && abs(rateAlongGravityBuffer.recentMean()) < resetThreshold) {
            recentDetection = false
            rateAlongGravityBuffer.reset()
        }
    }

    func stopUpdates() {
        if motionManager.isDeviceMotionAvailable {
            motionManager.stopDeviceMotionUpdates()
        }
    }
}
```

```
// class MotionManager {
//     func processDeviceMotion(_ deviceMotion: CMDeviceMotion) {
//       ...
          if (recentDetection && abs(rateAlongGravityBuffer.recentMean()) < resetThreshold) {
              recentDetection = false
              rateAlongGravityBuffer.reset()
          }
      }

      func stopUpdates() {
          if motionManager.isDeviceMotionAvailable {
              motionManager.stopDeviceMotionUpdates()
          }
      }
  }
```

```
// class MotionManager {
//     func processDeviceMotion(_ deviceMotion: CMDeviceMotion) {
//       ...
        if (recentDetection && abs(rateAlongGravityBuffer.recentMean()) < resetThreshold) {
            recentDetection = false
            rateAlongGravityBuffer.reset()
        }
    }

    func stopUpdates() {
        if motionManager.isDeviceMotionAvailable {
            motionManager.stopDeviceMotionUpdates()
        }
    }
}
```

Sample App Concept

Sample App Concept

User interface

Sample App Concept

User interface

Workout Manager

Sample App Concept

User interface

Workout Manager

Motion Manager

Device Motion

NEW

Availability

	iPhone 6/6+	iPhone 6s/6s+	iPhone SE	Apple Watch
Gravity	✓	✓	✓	✓
User Acceleration	✓	✓	✓	✓
Rotation Rate	✓	✓	✓	✓
Attitude	✓	✓	✓	✓
Magnetic Field	✓	✓	✓	

Device Motion

Points to ponder

NEW

Device Motion

Points to ponder

NEW

Wrist placement

Device Motion

Points to ponder

NEW

Wrist placement

Reference frame

Device Motion

Points to ponder

NEW

Wrist placement

Reference frame

Sample rate

Summary

Improvements to Historical Accelerometer

Pedometer Events

Device Motion on Apple Watch

More Information

https://developer.apple.com/wwdc16/713

Related Sessions

Getting the Most Out of HealthKit	Nob Hill	Wednesday 9:00AM
Core Location Best Practices	Pacific Heights	Thursday 4:00PM
What's New in watchOS 3	Presidio	Tuesday 5:00 PM

Labs

Core Motion Lab	Frameworks Lab D	Friday 12:30 PM

WWDC16