CSC 491 / 391 Mobile Application Development for iOS II

Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
@DePaulSWEng

1

Motion Sensors

2

Outline

- Sensors & motion events
- Accelerometer
- Gyroscope
- Core Motion
- Sensor fusion

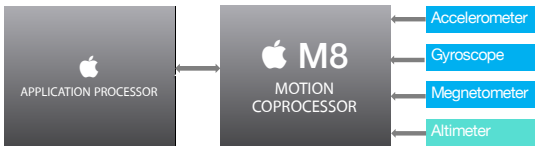
3

Sensors in Mobile Devices

- Convert a wide range of physical measurements into digital signals.
- Built-in sensors on iOS devices
  - Accelerometer – Linear acceleration on 3 axes
  - Gyroscope – Device orientation, rotational velocity on 3 axes
  - Magnetometer – Earth's magnetic field, digital compass (not available on iPods)
  - Barometer – Air pressure, altitude/elevation
  - GPS (Global Positioning System) – Location, i.e., latitude and longitude on Earth
  - Microphone – Ambient sound

4

iOS Device Motion Sensing



5

Why Motion Sensors?

- Your device is aware of its precise motion within its environment
- Controls are no longer limited to the UI widgets on the screen
- Opens the door to a new world of
  - gaming possibilities
  - motion based gestures
  - medical applications
  - other creative uses

6

Core Motion Framework

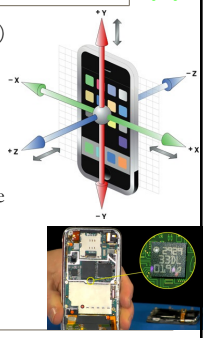
- Receive and handle motion events
  - Acceleration
  - Rotation rate
  - Orientation
  - Attitude
  - Altitude
  - Step count

DePaul University 7

7

The Accelerometer

- MEMS (micro-electromechanical system)
- Measure *accelerations* (in $g \approx 9.81 m/s^2$) in each of the three physical axes.
  - Acceleration is the *rate of change* of velocity.
  - *Acceleration of a device* =
    Earth Gravity + *User Acceleration*
- Available in all iOS devices, including the very first iPhone
- Low power consumption
- Responsive
- Noisy



DePaul University 8

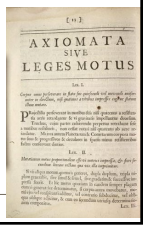
8

Acceleration Data

- The accelerometer measures the acceleration of the device (A_d)
  - The relation to the forces applied to the sensor (F_s)
    $A_d = - \sum F_s$ / mass
  - The forces include the force of *gravity*

Newton's *Second Law of Motion* (1687)
$\boldsymbol{F = m \bullet a}$
$1 N = 1 kg \cdot m/s^2$

Sir Isaac Newton
1642 – 1726

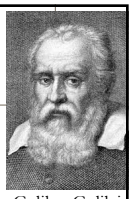


DePaul University 11

11

Gravity

- When the device is still, the accelerometer reading is
  $1 g \approx 9.81 m/s^2$
  Average Earth gravitational acceleration at sea level
- Separate the gravity
  $A_d = - g - \sum F$ / mass
  User acceleration
- Two software-based sensors
  - *Gravity* – Earth gravitational acceleration
  - *User Acceleration* – Acceleration sans gravity



Galileo Galilei
1564 – 1642

DePaul University 12

12

The Gyroscope

- A device for measuring and maintaining orientation
- Measure the rotation rate (in radian/s) around each of the three physical axes
  - Radian: standard unit of angular measure
    1 rad = 180° / π ≈ 57.2958°

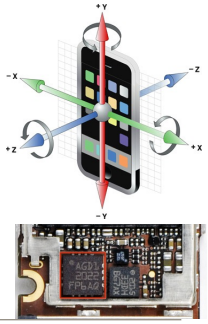
Gyroscope invented by Léon Foucault in 1852.

DePaul University 11

14

The Gyroscope

- MEMS sensor
- First appeared in iPhone 4 (2010), also in newer iPod and iPad
  - 3 additional axes of motion sensitivity
  - Awareness of rotational motion
- Responsive
- Precise
- Inherent bias drift



DePaul University 12

15

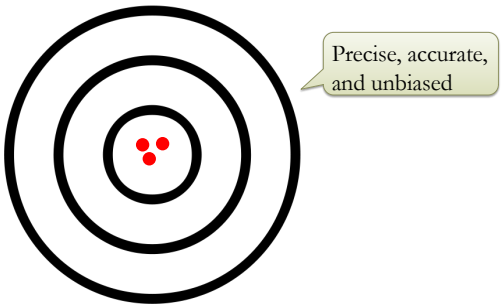
Precision, Accuracy, and Bias

- Precision: the closeness of multiple measurements
  - The variability of the measurements
- Bias: the difference between the estimation based the measurements and the true value
  - The closeness to the true value, i.e., the *accuracy*
- Unbiased = Accurate
- Precise ≠ Unbiased

DePaul University 13

16

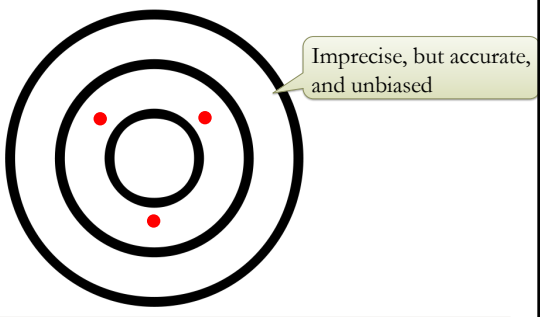
Precision, Accuracy, and Bias – Illustrations



DePaul University 14

17

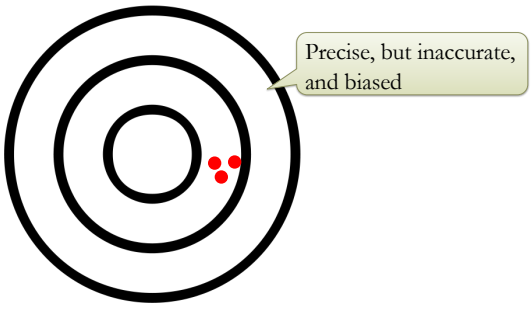
Precision, Accuracy, and Bias – Illustrations



DePaul University 15

18

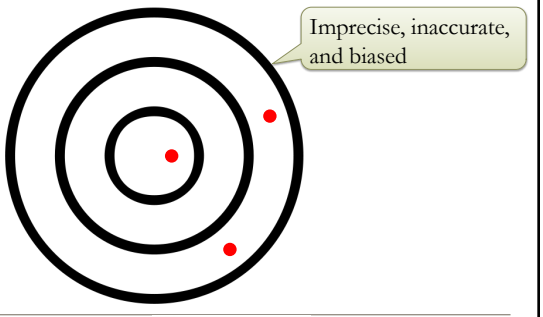
Precision, Accuracy, and Bias – Illustrations



DePaul University 16

19

Precision, Accuracy, and Bias – Illustrations



DePaul University 17

20

Handling Motion Events

21

21

Motion Events

- High level (coarse-grained) events originating from the accelerometers and gyroscopes
  - Shaking event
  - Device orientation and tilting
- High rate motion events
  - Raw sensor data
- Core Motion Framework
  - All sensors
  - Sensor fusion

22

Shake Gesture

- When users shake a device
  - The system determines whether it is a shaking gesture using the accelerometer data
  - If yes, the systems send a shake motion event to the first responder, when the shaking starts and stops
- Handling the shake gesture
  - Become the first responder
    - Motion events are sent to the first responder
    - Unlike touch events, the first responder must be set explicitly
  - Implement the methods responding to motion events

23

Become the First Responder

- In the *View Controller* class

Override a read-only computed property

```
override var canBecomeFirstResponder: Bool {
    return true
}

override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
    becomeFirstResponder()
}
```


24

Shake Motion Methods

- Implement the motion event handling methods in the first responder – the *View Controller*

```
override func motionBegan(_ motion: UIEventSubtype,
                          with event: UIEvent?) {
    …
}
override func motionEnded(_ motion: UIEventSubtype,
                          with event: UIEvent?) {
    … handle shake …
}
override func motionCancelled(_ motion: UIEventSubtype,
                          with event: UIEvent?) {
    …
}
```


25

Shake Motion Methods

- The `motionBagan` method:
  - When shaking started
- The `motionEnded` method:
  - When shaking stopped, in less than (≈) 1 second
- The `motionCancelled` method
  - When shaking lasted more than (≈) 1 second

26

Shake Event Type and Subtype

- The `UIEvent` class
  - Properties

```
var type: UIEventType { get }
var subtype: UIEventSubtype { get }
```

  - Values for the shake event

```
type:    UIEventtype.motion
subtype: UIEventSubtype.motionShake
```


27

Handle Shake Events

- Responding to shake gesture

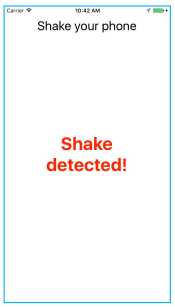
```
override func motionEnded(_ motion: UIEventSubtype,
                          with event: UIEvent?) {
    if motion == .motionShake {
        label.text = "Shake\ndetected!"
        DispatchQueue.main.asyncAfter(deadline:
                .now() + .seconds(3)) {
            self.label.text = ""
        }
    }
}
```

DePaul University 25

28

Handle Shake Events

- Simulator ▶
  Hardware ▶
  Shake Gesture



DePaul University 26

29

Current Device Orientation

- The `UIDevice` class
  - A singleton: `UIDevice.current`
- Start monitoring device orientation

```
UIDevice.current.
    beginGeneratingDeviceOrientationNotifications()
```

- Get orientation
  - Notification for `UIDeviceOrientationDidChange`
  - Use the `UIDevice.current.orientation` property
- Stop monitoring device orientation

```
UIDevice.current.
    endGeneratingDeviceOrientationNotifications()
```

DePaul University 27

30

Device Orientation

```
enum UIDeviceOrientation : Int {
    case unknown
    case portrait
    case portraitUpsideDown
    case landscapeLeft
    case landscapeRight
    case faceUp
    case faceDown

    public var isLandscape: Bool { get }
    public var isPortrait: Bool { get }
    public var isFlat: Bool { get }
}
```

DePaul University 28

31

Device Orientation Demo

```
override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
    UIDevice.current.beginGeneratingDeviceOrientationNotifications()
    self.observer = NotificationCenter.default.addObserver(
        forName: .UIDeviceOrientationDidChange,
        object: nil, queue: nil) { notification in
        let orientation = UIDevice.current.orientation
        Use orientation …
    }
}

override func viewDidDisappear(_ animated: Bool) {
    super.viewDidDisappear(animated)
    UIDevice.current.endGeneratingDeviceOrientationNotifications()
    NotificationCenter.default.removeObserver(self.observer!)
}
```

DePaul University 29

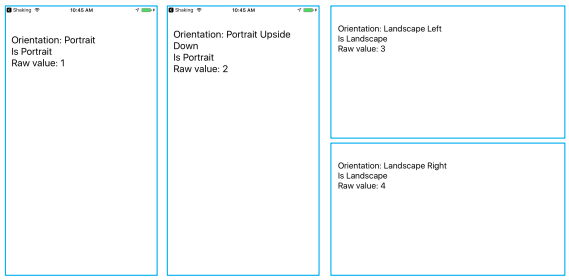
32

Device Orientation Demo

```
self.observer = NotificationCenter.default.addObserver(…) { _ in
    let orientation = UIDevice.current.orientation
    var message = "Orientation: "
    switch orientation {
    case .unknown: message += "Unknown\n"
    case .portrait: message += "Portrait\n"
    case .portraitUpsideDown: message += "Portrait Upside Down\n"
    case .landscapeLeft: message += "Landscape Left\n"
    case .landscapeRight: message += "Landscape Right\n"
    case .faceUp: message += "Face Up\n"
    case .faceDown: message += "Face Down\n"
    }
    if orientation.isPortrait { message += "Is Portrait\n"
    } else if orientation.isLandscape { message += "Is Landscape\n"
    } else if orientation.isFlat { message += "Is Flat\n"
    }
    message += "Raw value: \(orientation.rawValue)" …
}
```

33

Orientation



34

Core Motion Framework

35

Core Motion Framework

- The gateway class: `CMMotionManager`
- Raw data from individual sensors:
  - Accelerometer: `CMAccelerometerData`
  - Gyroscope: `CMGyroData`
  - Magnetometer: `CMMagnetometerData`
- Sensor fusion: `CMDeviceMotion`
  - Processed data from simultaneous tracking of multiple sensors. Sensor fusion algorithms.
  - Device attitude: `CMAttitude`
  - Gravity, user acceleration, and calibrated magnetic fields

36

Core Motion Framework

- Relative altitude
  - Altimeter: `CMAltimeter` `CMAltitudeData`
- Step count
  - Pedometer: `CMPedometer` `CMPedometerData`
- Exercise activities, running, walking, etc.
  - Activity manager: `CMMotionActivityManager`
- Historical motion data
  - Recorder: `CMSensorRecoder`

37

Receive Sensor Data

- Determine the availability of specific sensors
  - Not all sensors are available on all iOS devices
- Set an update interval
  - Hardware update frequency. Max frequency > 100Hz
- Start updates – sensor data becomes available
- Stop updates – when sensor data not needed
- Two approaches to process sensor data
  - Push: receive and process a steady stream of motion data
  - Pull: get and process motion data only when needed

38

Motion Data Update Intervals

- Configurable update frequency (approx. 10–100Hz)
  - Significant impact on resource consumption
- 10–20Hz
  - Suitable for determining the current orientation of the device
- 30–60Hz
  - Suitable for games and other applications that use the accelerometers for real-time user input
- 70–100Hz
  - Suitable for applications 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

Especially when pushing

39

Process Motion Data

- Push – receiving updates at specified intervals
  - Provide a handler (closure) to process the updates
  - Core Motion delivers updates to the handler at a constant rate
  - Suitable for data collection
- Pull – periodic sampling of motion data
  - Periodically sample the most recent measurement of motion data, when needed
  - Recommended approach
  - More resource efficient

40

Core Motion Gateway – Motion Manager

- Each app should have only a single instance of `CMMotionManager`
- Determine the availability of a hardware sensor

```
var isSensorAvailable: Bool { get }
```

where *Sensor* is one of:
`Accelerometer Gyro Magnetometer DeviceMotion`

- Start updates of a sensor

```
func startSensorUpdates()
func startSensorUpdates(to: queue, withHandler: code)
```

The main queue is not recommended

- Stop updates of a sensor

```
func stopSensorUpdates()
```


41

Acceleration Data

- Class `CMAccelerometerData`
  - Property

```
var acceleration: CMAcceleration { get }
```

- Struct `CMAcceleration`
  - Properties:

```
var x: Double
var y: Double
var z: Double
```

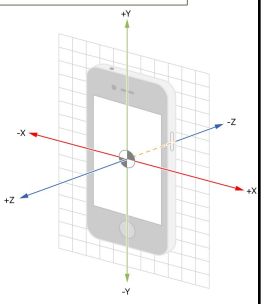
Three-axis accelerometer

- Measured in G's (after Galileo Galilei)
  - 1 g = 9.81 m/s² = 32.2 ft/s²

42

Acceleration Data – Direction of Axes

- Core Motion uses the standard 3-axis coordinate system
- Relative to the device's screen in the default orientation
- **Never changes when the device is rotated.**
  - The coordinate system for 2-D graphics rotates according to the device screen orientation.



43

Gyroscope Data

- Class `CMGyroData`
  - Property

```
var rotationRate: CMRotationRate { get }
```

- Struct `CMRotationRate`
  - Properties:

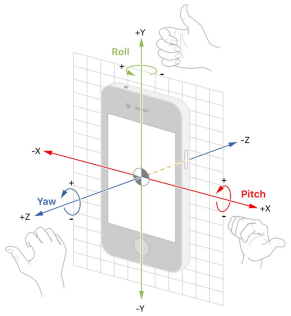
```
var x: Double
var y: Double
var z: Double
```

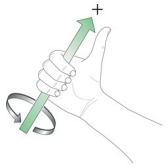
Three-axis gyroscope

- Measured in radians (≈ 57.2958°) per second
  - Direction of rotation: Right hand rule

44

Rotation Data – Direction of Rotation



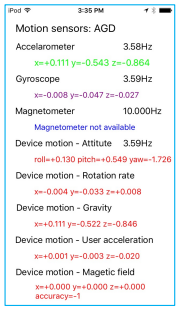


Right Hand Rule

45

Sensors – Push

- A simple demo app to display sensor data
  - Accelerometer
  - Gyroscope
  - Magnetometer
  - Device Motion
- Using the *push* approach
  - Show sensor readings in text labels



46

Sensors – Push

- The main parts in the *View Controller*

```
import UIKit
import CoreMotion

class ViewController: UIViewController {
    let motionManager = CMMotionManager()

    override func viewDidLoad() { … }
    override func viewWillAppear(_ animated: Bool) { … }
    override func viewWillDisappear(_ animated: Bool) { … }
    …
}
```

must be declared as a property

47

Sensors – Push
Determine Sensor Availability

```
override func viewDidLoad() {
    super.viewDidLoad()
    var s = ""
    if motionManager.isAccelerometerAvailable { s += "A" }
    if motionManager.isGyroAvailable { s += "G" }
    if motionManager.isMagnetometerAvailable { s += "M" }
    if motionManager.isDeviceMotionAvailable { s += "D" }
    if s == "" {
        availability.text = "Motion sensors: none"
    } else {
        availability.text = "Motion sensors: " + s
    }
}
```


48

Sensors – Push
Handle Accelerometer

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    let queue = OperationQueue()
    if motionManager.isAccelerometerAvailable {
        let accelStart = Date()
        var accelCount = 0
        motionManager.accelerometerUpdateInterval = 1/10
        motionManager.startAccelerometerUpdates(to: queue) {
            data, error in
            Handle acceleration data …
        }
    } else {
        accelValue.text = "Accelerometer not available"
    }
    …
}
```


49

Sensors – Push
The Handler for Accelerometer

```
motionManager.accelerometerUpdateInterval = 1/10
motionManager.startAccelerometerUpdates(to: queue) { data, error in
  accelCount += 1
  let accel: String
  if let data = data {
    accel = String(format: "x=%+6.3f y=%+6.3f z=%+6.3f",
      data.acceleration.x, data.acceleration.y, data.acceleration.z)
  } else { accel = "No accelerometer data" }
  DispatchQueue.main.async {
    self.accelValue.text = accel
    let elapsedTime = Date().timeIntervalSince(accelStart)
    self.accelFreq.text = String(format: "%4.2fHz",
                                 Double(accelCount)/elapsedTime)
  }
}
…
```


50

Sensors – Push
Handle Gyroscope

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    …
    if motionManager.isGyroAvailable {
        let gyroStart = Date()
        var gyroCount = 0
        motionManager.gyroUpdateInterval = 1/10
        motionManager.startGyroUpdates(to: queue) {
            data, error in
            Handle gyroscope data …
    } else {
        gyroValue.text = "Gyroscope not available"
    }
    …
}
```


51

Sensors – Push
The Handler for Gyroscope

```
motionManager.gyroUpdateInterval = 1/10
motionManager.startGyroUpdates(to: queue) { data, error in
  gyroCount += 1
  let gyro: String
  if let data = data {
    gyro = String(format: "x=%+6.3f y=%+6.3f z=%+6.3f",
      data.rotationRate.x, data.rotationRate.y, data.rotationRate.z)
  } else { gyro = "No gyroscope data" }
  DispatchQueue.main.async {
    self.gyroValue.text = gyro
    let elapsedTime = Date().timeIntervalSince(gyroStart)
    self.gyroFreq.text = String(format: "%4.2fHz",
                                Double(gyroCount)/elapsedTime)
  }
}
…
```


52

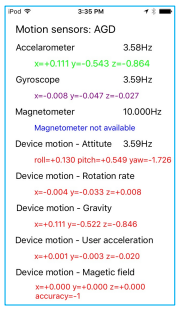
Sensors – Push
Stop Updates

```
override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    if motionManager.isAccelerometerActive {
        motionManager.stopAccelerometerUpdates()
    }
    if motionManager.isGyroActive {
        motionManager.stopGyroUpdates()
    }
    …
}
```


53

Sensors – Pull

- A simple demo app to display sensor data
  - Accelerometer
  - Gyroscope
  - Magnetometer
  - Device Motion
- Using the *pull* approach
  - Show sensor readings in text labels



54

Sensors – Pull

```
import UIKit
import CoreMotion

class ViewController: UIViewController {
    let motionManager = CMMotionManager()
    var timer: Timer?
     var timeStart = Date()
     var count = 0

    override func viewDidLoad() { … }
    override func viewWillAppear(_ animated: Bool) { … }
    override func viewWillDisappear(_ animated: Bool) { … }
    …
}
```

must be declared as a property

55

Sensors – Pull
Start Updates

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    if motionManager.isAccelerometerAvailable {
        motionManager.accelerometerUpdateInterval = 1/10
        motionManager.startAccelerometerUpdates()
    } else { accelValue.text = "Accelerometer not available" }
    …
    timeStart = Date()
    count = 0
    timer = Timer.scheduledTimer(timeInterval: 1/10, target: self,
                    selector: #selector(handleSensors),
                    userInfo: nil, repeats: true)
}
```


56

Sensors – Pull
The Handler

```
func handleSensors(_ timer: Timer) {
  count += 1
  let elapsedTime = Date().timeIntervalSince(timeStart)
  let freq = Double(count)/elapsedTime
  if motionManager.isAccelerometerAvailable {
    let accel: String
    if let data = motionManager.accelerometerData {
      accel = String(format: "x=%+6.3f y=%+6.3f z=%+6.3f",
        data.acceleration.x, data.acceleration.y, data.acceleration.z)
      } else {
        accel = "No accelerometer data"
      }
      self.accelValue.text = accel
      self.accelFreq.text = String(format: "%4.2fHz", freq)
    }
…
```


57

Sensors – Pull Stop Updates

```
override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    if motionManager.isAccelerometerActive {
        motionManager.stopAccelerometerUpdates()
    }
    if motionManager.isGyroActive {
        motionManager.stopGyroUpdates()
    }
    …
    timer?.invalidate()
}
```

DePaul University 55

58

Device Motion

- Class `CMDeviceMotion`
  - Sensor fusion: accelerometer + gyroscope + magnetometer
  - Processed motion data, highly accurate
  - Calibrated and corrected for bias
- Properties:

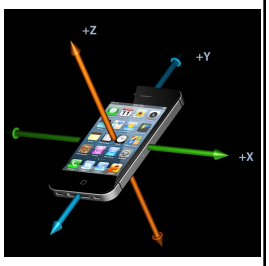
```
var attitude: CMAttitude { get }
var rotationRate: CMRotationRate { get }
var gravity: CMAcceleration { get }
var userAcceleration: CMAcceleration { get }
var magneticField: CMCalibratedMagneticField { get }
```

DePaul University 56

59

Device Attitude

- The orientation of a device
  - In three dimensional space
  - Relative to an external reference frame
- Are you interested in
  - the relative changes in device attitude?
  - or the true orientation?



DePaul University 57

60

Reference Frames of Attitude

- You choose the initial reference frame
- The attitude reading is relative to the initial reference frame
- The initial reference frame assumes
  - A device lying on a flat, i.e., horizontal, surface; face up
  - X-Y axes are on a horizontal plane
  - Z-axis is vertical, points up, i.e., towards the sky
- Which direction does the X-axis point to?
  - Impact on the accuracy and power consumption

DePaul University 58

61

Reference Frame CMAttitudeReferenceFrame

- `.xArbitraryZVertical`
  - X-axis points to an "arbitrary" direction.
  - Fixed to the orientation of the device when you *first* start device motion updates.
  - Most energy efficient option.
- `.xArbitraryCorrectedZVertical`
  - Same as above with improved long-term yaw accuracy.
  - Uses the magnetometer, when available and calibrated, to correct bias in gyroscope.
  - Increased CPU usage and power consumption.

For relative attitude changes

DePaul University 59

62

Reference Frame CMAttitudeReferenceFrame

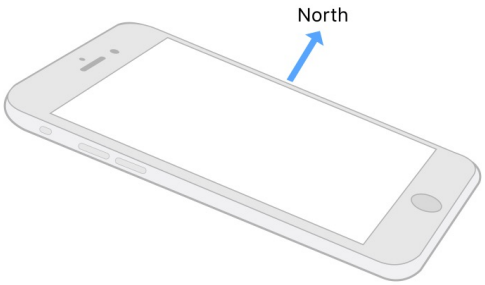
- `.xMagneticNorthZVertical`
  - X-axis points to the magnetic north.
  - Uses the magnetometer. Requires calibration of the magnetometer.
  - Increased CPU usage and power consumption.
- `.xTrueNorthZVertical`
  - X-axis points to the true north
  - Requires the magnetometer and the location data (GPS) to calculate the difference between magnetic and true north.
  - Most costly in CPU usage and power consumption.

For true device orientation

DePaul University 60

63

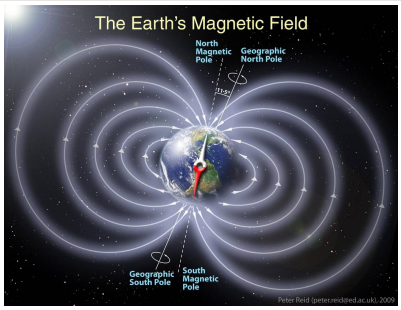
Reference Frame – True North



DePaul University 61

64

Earth's Magnetic Field

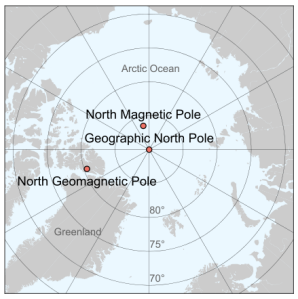


Copyright: Peter Reid, The University of Edinburgh

DePaul University 62

65

Geographic North Pole vs. Magnetic North Pole



DePaul University 63

66

Magnetometer Data

- Class `CMCalibratedMagneticField`
  - Property

```
var accuracy: CMMagneticFieldCalibrationAccuracy
var field: CMMagneticField
```

- Accuracy of magnetic field measurement

```
enum CMMagneticFieldCalibrationAccuracy {
    case uncalibrated
    case low
    case medium
    case high
}
```

DePaul University 64

67

Magnetometer Data

- Struct `CMMagneticField`
  - Properties:

```
var x: Double
var y: Double
var z: Double
```

- Measured in *microteslas* (μT): strength of magnetic field
  - 31.869 μT: strength of Earth's magnetic field at 0° latitude, 0° longitude

DePaul University 65

68

Attitude Data

- Class `CMAttitude`
  - Properties:

```
var roll: Double
```

Euler angles measured in radians (≈ 57.2958°)

```
var pitch: Double
var yaw: Double
```

```
var rotationMatrix: CMRotationMatrix
```

Rotation matrix – a 3x3 matrix

```
var quaternion: CMQuaternion
```

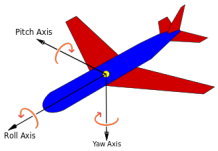
Quaternion – a 4-dimentional vector

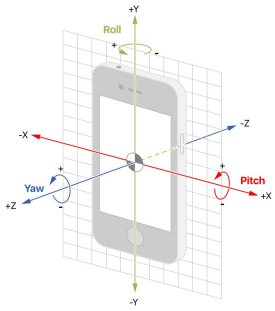
DePaul University 66

70

Euler Angles

- Roll, pitch, yaw
- Intuitive
  - Easily visualized
  - Familiar to user



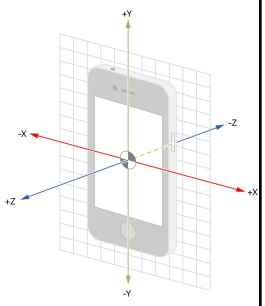


DePaul University 67

71

Demo: Always Up

- Hold device in up-right position
- Turn on a vertical plane
- The arrow maintains upward position
- Using gravity on X-Y plane

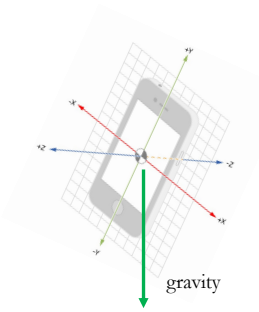


DePaul University 68

72

Demo: Always Up

- Hold device in up-right position
- Turn on a vertical plane
- The arrow maintains upward position
- Using gravity on X-Y plane



DePaul University 69

73

Demo: Always Up – Drawing the Arrow

```
class UpArrow: UIView {
  override func draw(_ rect: CGRect) {
    if let context = UIGraphicsGetCurrentContext() {
      let tip = CGPoint(x: bounds.width / 2, y: bounds.height * 0.1)
      let center = CGPoint(x: bounds.width / 2, y: bounds.height / 2)
      let left = CGPoint(x: bounds.width * 0.1, y: bounds.height * 0.9)
      let right = CGPoint(x: bounds.width * 0.9, y: bounds.height * 0.9)

      context.move(to: tip)
      context.addLine(to: left)
      context.addLine(to: center)
      context.addLine(to: right)
      context.closePath()

      context.setFillColor(UIColor.green.cgColor)
      context.fillPath()
    }
}
```

DePaul University 70

74

Demo: Always Up – The Gravity Reading

```
class ViewController: UIViewController {
  @IBOutlet weak var arrow: UpArrow!
  let motionManager = CMMotionManager()
  let updateFrequency = 60.0

  override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    if motionManager.isDeviceMotionAvailable {
      motionManager.deviceMotionUpdateInterval = 1/updateFrequency
      motionManager.startDeviceMotionUpdates(to: .main) { data, error in
        if let gravity = data?.gravity {
          let rotation = atan2(gravity.x, gravity.y) - Double.pi
          self.arrow.transform =
            CGAffineTransform(rotationAngle: CGFloat(rotation))
        }
      }
    }
  }
  …
```

DePaul University 71

75

Vector and Trigonometry

- A vector has an angle and a magnitude
- The magnitude of a vector

$$\sqrt{x^2 + y^2 + z^2}$$

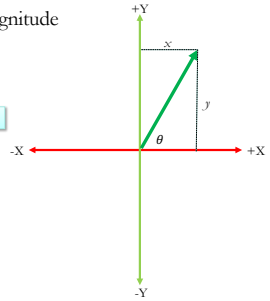
```
sqrt(x * x + y * y + z * z)
```

- The *tangent* function

$$\tan\theta = \frac{y}{x}$$

- The inverse tangent function

$$\theta = \arctan\frac{y}{x}$$



DePaul University 72

76

Arctangent Function in Swift

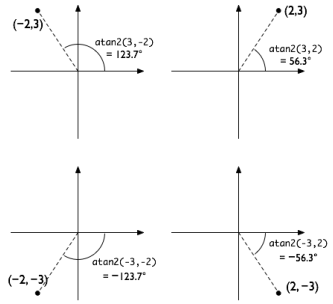
- Two options:
  `func atan(_ a: CGFloat) -> CGFloat`
  Result: $(-\pi/2, \pi/2)$
  `func atan2(_ x: CGFloat, _ y: CGFloat) -> CGFloat`
  Result: $(-\pi, \pi]$
- Equivalent *some of the time*
  `atan(y / x)` = `atan2(y, x)`
  - If $y > 0$ and $x \neq 0$
- `atan` is limited to 2 quadrants.
- `atan2` supports all four quadrants.
- `atan2` works on all values of x and y, including when $x = 0$

Half a circle

Full circle

77

Arctangent Function in Swift



78

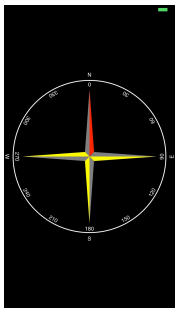
Demo: Always Up – Stop Updates & No Auto Rotate

```
class ViewController: UIViewController {
  …
  override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    if motionManager.isDeviceMotionActive {
      motionManager.stopDeviceMotionUpdates()
    }
  }

  override var shouldAutorotate: Bool { return false }

}
```


79

Demo: Compass

- Hold device in horizontal position
- The arrows point to N-E-S-W
- Using the magnetic field on X-Y plane



80

Demo: Compass

```
class ViewController: UIViewController {
  @IBOutlet weak var compassView: CompassView!
  let motionManager = CMMotionManager()

  override func viewWillAppear(_ animated: Bool) {
    if motionManager.isDeviceMotionAvailable {
      motionManager.deviceMotionUpdateInterval = 1/10
      motionManager.startDeviceMotionUpdates(
          using: .xTrueNorthZVertical, to: .main) { data, error in
        if let mdata = data?.magneticField.field {
          self.compassView.transform =
            CGAffineTransform(rotationAngle:
                CGFloat(atan2(mdata.x, mdata.y)))
        }
      }
    }
  }
  …
```


81

Process Raw Sensor Data

82

Filters for Raw Acceleration Data

- The raw accelerometer data was jittery, i.e., noisy.
  - Gravity, user action, environment,
- Filters can be applied to
  - Smooth out noises
- Isolate specific behaviors, e.g.,
  - The gravity – constant
  - User action – sudden, variable

83

Low-Pass Filter

- A low-pass filter weighs the previous readings more than the current.
- A low pass filter has the effect of changing slowly based on the current raw data reading.
  - smoothing out sudden changes
  - isolates constant acceleration, i.e., gravity

84

Low-Pass Filter

- A simple low-pass filter

```
filteringFactor = 0.1
value = currentReading * fileteringFactor +
    previousValue * (1.0 – filteringFactor)
previousValue = value
```

- 10% current reading, 90% previous readings

85

High-Pass Filter

- A high-pass filter weighs the current readings more than the previous
- A high pass filter has the effect of changing rapidly based on the current raw data reading.
  - highlighting sudden changes, i.e., user movement

86

High-Pass Filter

- A simple high-pass filter

```
filteringFactor = 0.1
lowPassValue = currentReading * fileteringFactor +
    previousValue * (1.0 – filteringFactor)
value = currentReading – lowPassValue;
previousValue = lowPassValue;
```

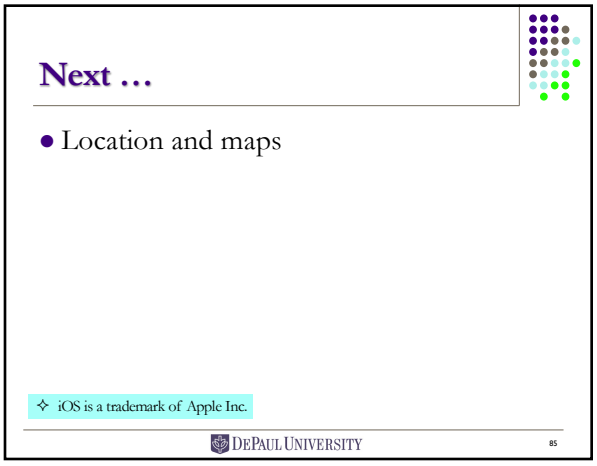
- current reading – smoothed value (obtained by low-pass filter)

87

Sample Code

- Shaking – Shaking gesture
- Orientation – Device orientation changes
- Sensor – Push – All motion sensors
- Sensor – Pull – All motion sensors
- Always Up – Device Motion, Gravity
- Compass Demo – Device Motion, Magnetic Field
- Accelerometer Graph – Filters

88


Next …
Location and maps
iOS is a trademark of Apple Inc.
DePaul University
85

89