基般

1 iPhone によるセンサプログラミング

沼田哲史
大阪電気通信大学 総合情報学部 デジタルゲーム学科

iPhone プログラミングの概要

この記事では，Apple 社より販売されている iPhone/iPad 用のアプリケーション開発において，本体に搭載された各種のセンサを利用する方法について解説します．

■ iOS に搭載されているセンサデバイス

2011 年 10 月 12 日に，iPhone/iPad 用の最新の OS である iOS 5 が公開されました．iOS 5 では，ユーザ向けにもプログラマ向けにも多くの新機能が追加され，さまざまな面で改良が施されました．

iOS 5 が対象とするのは，iPhone 3GS，iPhone 4，iPhone 4S，初代 iPad，iPad 2，そして，第 3 世代・第 4 世代（2009 年後期以降）の iPod touch です（図 -1）．これらのデバイスの多くでは，最も基本的な操作方法であるタッチパネルに加えて，加速度センサ，3 軸ジャイロセンサ，マイク，カメラ（静止画・動画に対応）が利用可能となっています．また，3G の電話回線が利用可能な iPhone/iPad については，衛星からの情報に基づく正確な位置測定が可能な GPS と，東西南北の方位が確認できる電子コンパスが搭載されています．

この記事では，自作アプリでこれらのセンサデバイスを活用するための方法について解説します．iPhone/iPad のプログラミングに必要な，Xcode 4 の使い方や，Objective-C の基本的な使い方をご存知の方を対象としています．

図 -1　iPhone 4, 第 4 世代 iPod touch, iPad 2（図中の Web ページは Wikipedia より「Gyroscope」「Accelerometer」の項目）

■ センサ利用のためのフレームワーク

iOS プログラミングの環境である Cocoa Touch は，NeXTSTEP からの流れを汲む，Mac OS X プログラミングのための Cocoa がベースになっています．Cocoa は Mac OS X のために 10 年以上も使われてきた実績のある環境です．そして Cocoa Touch は Cocoa と同様に，Objective-C という言語の特性を最大限に活用して，高度な MVC パラダイムをサポートするほか，多数の有用なフレームワークを iOS デバイス

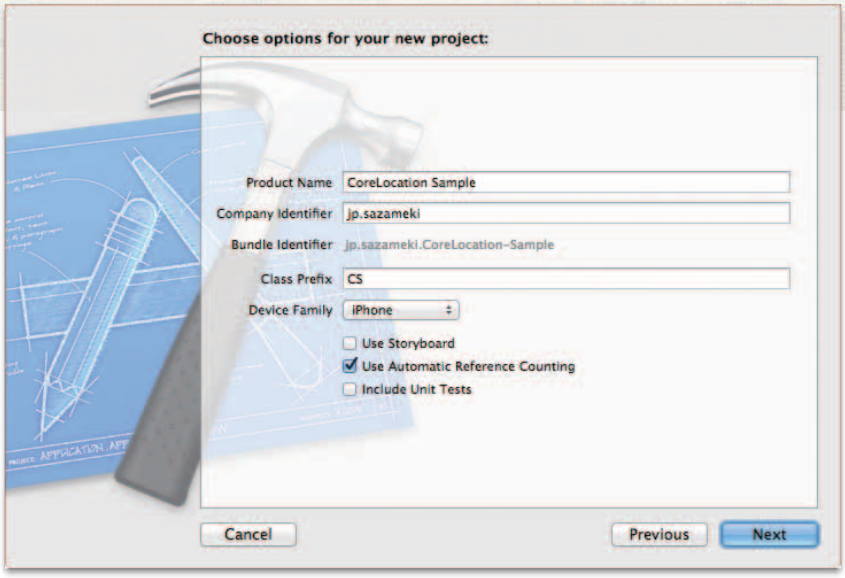

図 -2　新規プロジェクトの設定

用に提供しています．

それらのフレームワーク中でも，センサ活用のためには Core Location と Core Motion というフレームワークが利用可能です．Core Location は，GPS からの情報や利用中の携帯電話の基地局の情報，利用中の Wi-Fi ベースステーションの情報などを元に，現在の位置情報の取得をサポートします．また Core Motion は，加速度センサおよびジャイロセンサからの値の取得などをサポートします．

この記事では，Core Location と Core Motion の使い方について解説し，GPS 等に基づく位置情報の取得，加速度センサとジャイロセンサを利用したモーションの取得について見ていきます．

Core Location を利用した位置情報取得

■ Core Location の基本的な使い方

Core Location フレームワークを利用することで，ユーザの現在位置と方位を取得することができます．また，おおよその高度も計測可能です．

それではまず，Xcode 4.2 を使って，Core Location のサンプルコードを作成してみましょう．Xcode を起動して，メニューから「File」-「New」-「New Project. . .」を選択します．表示されるテンプレートの中から，「iOS Application」の「Single View Application」を選択します．「Product Name」に「CoreLocation Sample」と入力し，「Class Prefix」に「CS」と入力してください．またメモリ管理の簡単のため，「Use Automatic Reference Counting」にもチェックを入れておきましょう（図 -2）．こうして新しいプロジェクトを作成します．

プロジェクトが作成できたら，Xcode ウィンドウの左側のファイルリストから「CoreLocation Sample」を選択し，「TARGETS」を選択して，「Summary」の下の方にある「Linked Frameworks and Libraries」でフレームワーク一覧を表示します．左下の「+」ボタ

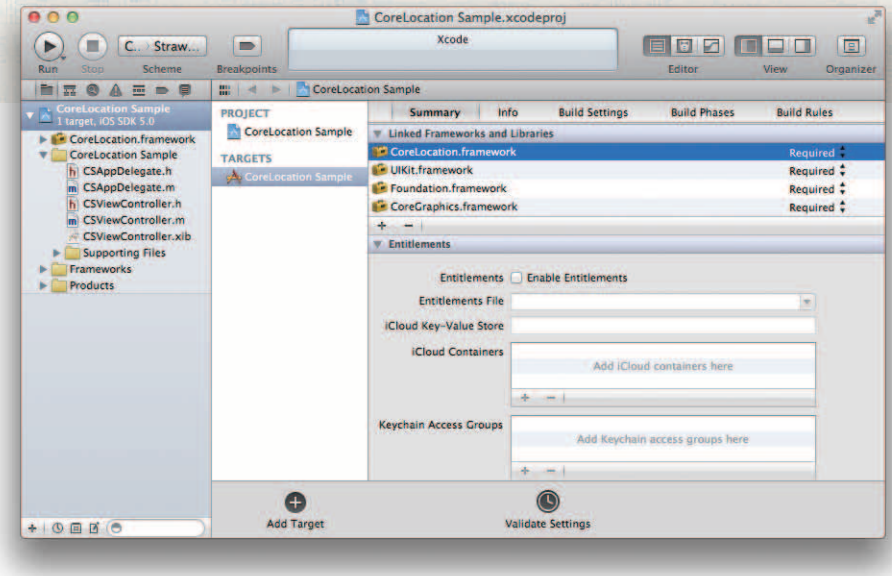

図 -3　Core Location フレームワークの追加

kCLLocationAccuracyBestForNavigation	ユーザに常に正確な位置情報を提供し，ナビゲーションを行うアプリケーションに使用するために最適な精度を使用します．
kCLLocationAccuracyBest	最も高い精度を使用します．
kCLLocationAccuracyNearestTenMeters	要求された対象に対して 10 メートル以内の精度を使用します．
kCLLocationAccuracyHundredMeters	100 メートル以内の精度を使用します．
kCLLocationAccuracyKilometer	キロメートル単位での精度を使用します．
kCLLocationAccuracyThreeKilometers	3 キロメートル内での精度を使用します．

表 -2　位置取得の精度の定数一覧

ンを押して，「CoreLocation.framework」を追加してください（図 -3）．

次のようにして，CSViewController.h ファイルに，Core Location のヘッダファイルを読み込み，CSViewController クラスが Delegate という仕組みで Core Location からの通知を受け取れるように，コードを追加します．また，Core Location の管理を行うマネージャ用の変数も追加します．

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>

@interface CSViewController : UIViewController<CLLocationManagerDelegate> {
    CLLocationManager  *locationManager;
}

@end
```

次に，CSViewController.m の viewDidLoad を次のように修正してください．ここでは，CLLocationManager というクラスのオブジェクトを作成します．このクラスは，位置情報の取得のすべての管理を行います．CLLocationManager の Delegate を登録した後，どのくらいの精度で現在位置の情報を取得するかを，desiredAccuracy プロパティで定数を指定して設定します（表 -2 参照）．この例では，ナビゲーションに最適な精度を指定しています．そして，位置情報の更新を開始するための startUpdatingLocation メソッドと，startUpdatingHeading メソッドを呼び出します．これで位置情報の取得と，方位情報の

取得が開始されます．

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    locationManager = [[CLLocationManager alloc] init];
    locationManager.delegate = self;
    locationManager.desiredAccuracy = kCLLocationAccuracyBestForNavigation;
    [locationManager startUpdatingLocation];
    [locationManager startUpdatingHeading];
}
```

指定された精度に応じた位置情報の更新があった場合に，Delegate のメソッドが呼び出されます．次のように，2 つのメソッドを実装してください．CSViewController.m ファイル末尾の「@end」の直前に，次のコードを挿入します．

```
- (void)locationManager:(CLLocationManager *)manager
   didUpdateToLocation:(CLLocation *)newLocation
          fromLocation:(CLLocation *)oldLocation
{
    CLLocationCoordinate2D coord = newLocation.coordinate;
    NSLog(@"coord=(%.2f,%.2f)", coord.latitude, coord.longitude); // 緯度・経度
    NSLog(@"c_acc=(%.2f,%.2f)",     // 精度
       newLocation.horizontalAccuracy,
       newLocation.verticalAccuracy);
}

- (void)locationManager:(CLLocationManager *)manager
      didUpdateHeading:(CLHeading *)newHeading
{
    NSLog(@"dir=%.2f", newHeading.magneticHeading);
    NSLog(@"d_acc=%.2f", newHeading.headingAccuracy);
}
```

位置情報が更新されると，Delegate の locationManager:didUpdateToLocation:fromLocation: メソッドが呼び出されます．最新の位置情報が newLocation 引数に，以前の位置情報が oldLocation 引数に入って渡されます．位置に関係する各種の情報は，上記のコードのようにして取得できます．緯度と経度を表す CLLocationDegrees の型は，double 型で，度単位で値が表されます．精度を表す CLLocationAccuracy の型は，double 型で，メートル単位で精度が表されます．

同様にして，方位の情報が更新されたときには，Delegate の locationManager:didUpdateHeading: メソッドが呼び出されます．本体の向いている方角を示す magneticHeading プロパティの CLLocationDirection 型は double 型で，真北が 0.0，真東が 90.0，真南が 180.0 といったような値が入ります．headingAccuracy プロパティで取得できる精度情報も同様の値で，磁場によって変化し，小さい値ほど方向の精度が高いことを示します．

最後に，このプログラムを実行すると，次のように，緯度・経度と精度，本体の向きと精度が取得できます．なお，電子コンパスによる本体の向きは，実機上で実行していないと取得できません．

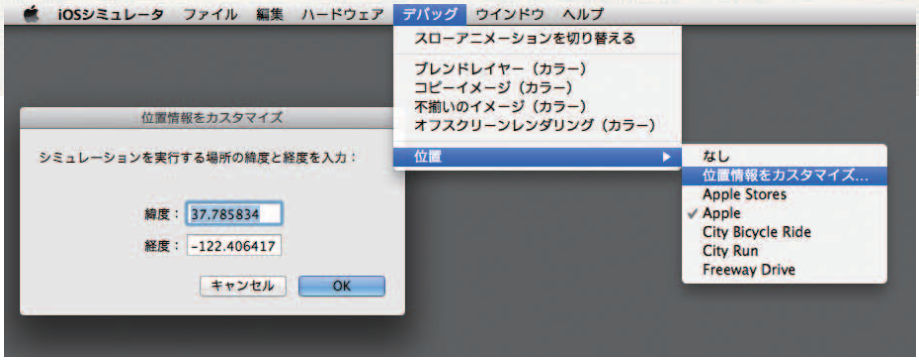

図 -4　シミュレータ上での位置情報のカスタマイズ

```
2011-10-26 05:06:15.474 CoreLocation Sample[76725:f803] coord=(37.33,-122.03)
2011-10-26 05:06:15.475 CoreLocation Sample[76725:f803] c_acc=(30.00,-1.00)
2011-10-26 05:06:15.475 CoreLocation Sample[76725:f803] altitude=104.50
2011-10-26 05:06:16.476 CoreLocation Sample[76725:f803] dir=21.64
2011-10-26 05:06:16.476 CoreLocation Sample[76725:f803] d_acc=25.00
...
```

なお，アプリケーションの開発時には，効率の問題で常に実機を使って開発することは難しいでしょう．そのため，iOS シミュレータには，現在位置の情報をシミュレーションすることもできるようになっています．iOS シミュレータのメニューから「デバッグ」-「位置」のサブメニューを表示して，プリセットの場所を選択するか，「位置情報をカスタマイズ...」を選択して，任意の緯度と経度を入力することで，現在位置の情報をシミュレートできます（図 -4）．

■ Core Location の精度

Core Location における位置情報の取得には，GPS からの情報，利用中の携帯電話の基地局の情報，利用中の Wi-Fi ベースステーションの位置データベースが組み合わされて使用されます．そのため，GPS を搭載していない iPod touch などでも，Core Location フレームワークを利用して位置情報を取得することが可能です．指定された位置取得の精度によって，これらの組合せの割り合いが変わってきますが，どのように組み合わされるかは明文化されていません．

図 -5　位置情報取得に関するアラートと環境設定

■位置情報取得の許可について

なお，ユーザの現在位置の取得は，個人のセキュリティやプライバシーにかかわる問題でもありますので，あらかじめ位置情報を取得することをユーザに許可してもらわなければいけません（図 -5）．そして，ユーザが環境設定で特定のアプリが位置情報を取得できないように設定している場合がありますので，あらかじめ

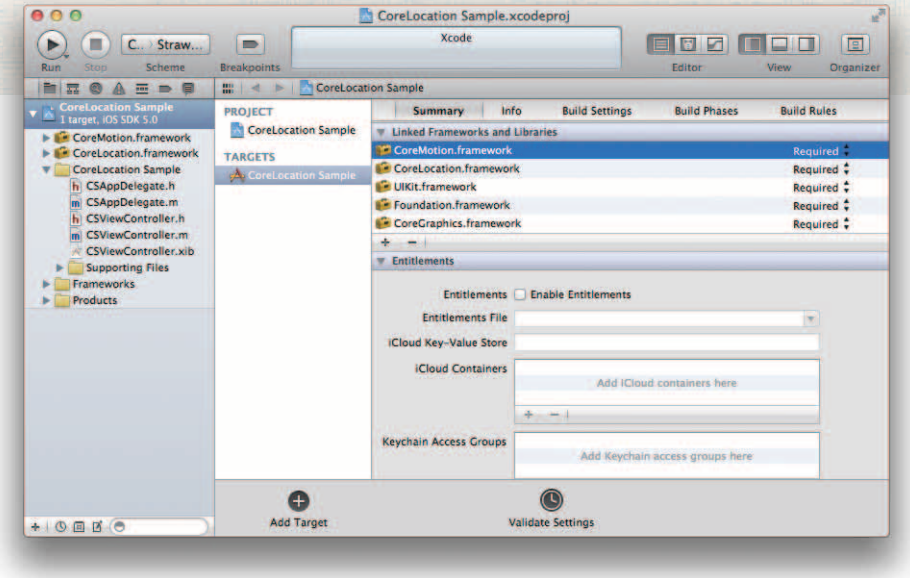

図 -6　Core Motion フレームワークの追加

CLLocationManager クラスの locationServicesEnabled メソッドを呼び出して，自分のアプリで位置情報の取得が許されていることを確認しておく必要があります．

```
if ([CLLocationManager locationServicesEnabled]) {
    // 位置情報の取得が許可されている
}
```

■ より詳細な Core Location のサンプル

Core Location を使用する詳細なサンプルコードは，Apple 社の iOS Dev Center（http://developer.apple.com/devcenter/ios/）の Sample Code のページに，3 個登録されています（2011 年 10 月 12 日現在）．Teslameter，LocateMe，GeocoderDemo です．中でも，この記事の説明に沿った形で緯度・経度の情報が取得できるのは LocateMe のサンプルコードです．ぜひダウンロードして，実機で実行してみてください．

Core Motion を利用したモーション取得

■ ジャイロセンサの場合

加速度センサとジャイロセンサの活用のための Core Motion は，Core Location のように Delegate を使用した値取得は行いません．CMMotionManager という値取得のためのマネージャオブジェクトを作成したら，適宜そのオブジェクトにセンサ値を問い合わせます．

「Core Location」のために作成したプログラムに，ジャイロセンサを利用するコードを追加してみましょう．まずは，プロジェクトに Core Motion フレームワークを追加します（**図 -6**）．

そして CSViewController.h に，Core Motion のためのコードを追加します．

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>
#import <CoreMotion/CoreMotion.h>

@interface CSViewController : UIViewController<CLLocationManagerDelegate> {
    CLLocationManager   *locationManager;
    CMMotionManager     *motionManager;
}

@end
```

次に，viewDidLoad メソッドに，マネージャ作成のためのコードを書きます．iOS デバイスによってはジャイロセンサの搭載されていないものもありますので，まずは gyroAvailable プロパティを参照して，ジャイロセンサが利用可能かどうかを確認する必要があります．ジャイロが利用可能な場合には，gyroUpdateInterval プロパティでセンサからの値取得の更新頻度を秒単位の小数点の値で設定します．この例では，0.5 秒ごとに値を更新するように設定しています．更新頻度の設定後，startGyroUpdates メソッドを呼び出して，ジャイロセンサからの値取得を開始します．

あとはタイマーを利用して，任意のタイミングで，gyroData プロパティを参照して，センサの値を参照します．ここでは，0.5 秒ごとに繰り返し motionProc: メソッドを呼び出すタイマーを作成しています．

```
- (void)viewDidLoad
{
    // （前略。ここまで Core Location のコード）
    CMMotionManager *manager = [[CMMotionManager alloc] init];
    if (manager.gyroAvailable)  {
        manager.gyroUpdateInterval = 0.5;
        [manager startGyroUpdates];
    }
    [NSTimer scheduledTimerWithTimeInterval:0.5
                                     target:self
                                   selector:@selector(motionProc:)
                                   userInfo:nil
                                    repeats:YES];
}
```

それでは，CSViewController.m ファイルの末尾の「@end」の直前に，motionProc: メソッドの実装を追加しましょう．このようにして，double 型の値として，X・Y・Z の 3 軸の値が取得できます．ジャイロセンサのそれぞれの値の単位は，ラジアン／秒です．値は右手系の座標に沿って出てきます．

```
- (void)motionProc:(NSTimer *)timer
{
    CMGyroData *data = motionManager.gyroData;
    CMRotationRate rate = data.rotationRate;
    NSLog(@"rotation=(%.2f, %.2f, %.2f)", rate.x, rate.y, rate.z);
}
```

ジャイロセンサを使った場合の実行結果は次のようになります．

```
2011-10-26 05:40:05.048 CoreLocation Sample[728:707] rotation=(0.00, 0.00, 0.00)
2011-10-26 05:40:05.549 CoreLocation Sample[728:707] rotation=(0.00, 0.00, 0.00)
2011-10-26 05:40:06.049 CoreLocation Sample[728:707] rotation=(0.02, -0.00, -0.02)
2011-10-26 05:40:06.549 CoreLocation Sample[728:707] rotation=(-0.00, 0.04, 0.04)
2011-10-26 05:40:07.048 CoreLocation Sample[728:707] rotation=(-0.30, 2.19, -0.02)
2011-10-26 05:40:07.548 CoreLocation Sample[728:707] rotation=(-0.13, 1.92, 0.05)
2011-10-26 05:40:08.049 CoreLocation Sample[728:707] rotation=(0.02, 2.48, 0.01)
2011-10-26 05:40:08.548 CoreLocation Sample[728:707] rotation=(0.10, 1.17, 0.33)
2011-10-26 05:40:09.048 CoreLocation Sample[728:707] rotation=(-0.01, 0.36, 0.11)
```

■ 加速度センサの場合

加速度センサを利用する場合も，ジャイロセンサとほぼ同様の方法となります．加速度センサが利用可能かどうかを accelerometerAvailable プロパティで確認してから，加速度センサからの値取得を開始します．

```
if (manager.accelerometerAvailable) {
    manager.accelerometerUpdateInterval = 1.0 / 60;
    [manager startAccelerometerUpdates];
}
```

加速度センサの値も，次のようにして，double 型の値として，X・Y・Z の 3 軸の値が取得できます．加速度センサのそれぞれの値の単位は，重力の G です．

```
CMAccelerometerData *data = manager.accelerometerData;
CMAcceleration accel = data.acceleration;
double x = accel.x;
double y = accel.y;
double z = accel.z;
```

■ 複合センサによる向きの検出

加速度センサとジャイロセンサの両方が使える環境では，2 つのセンサからの値を組み合わせることで，デバイスの方向を取得できるようになっています．使い方はほとんど同じです．

```
if (manager.deviceMotionAvailable) {
    manager.deviceMotionUpdateInterval = 1.0 / 60;
    [manager startDeviceMotionUpdates];
}
```

複合センサ値を格納している CMDeviceMotion というオブジェクトには，ユーザの加速度情報等も含まれていますが，最も興味深い方向の値は，attitude プロパティで取得できます．attitude プロパティからは，いくつかの方法で角度情報を取得できます．この角度情報は，ある参照時点からの相対的な本体の姿勢を表す値となります．AR（Augmented Reality）などに活用しやすいのは，この情報でしょう．

ロール・ピッチ・ヨーの 3 つの double 型の値として角度を取得することもできますし，回転行列やクォータニオンの形で回転表現を取得することもできます．

```
CMDeviceMotion *motion = manager.deviceMotion;
CMAtttude attitude = motion.attitude;
double roll = attitude.roll;
double pitch = attitude.pitch;
double yaw = attitude.yaw;
CMRotationMatrix mat = attitude.rotationMatrix;   // 3x3 の回転行列
CMQuaternion quat = attitude.quaternion;     // クォータニオンによる回転表現
```

図 -7　本体の向きに合わせて公園の名前を重ね合わせて表示する pARk サンプルコード

■ より詳細な Core Motion のサンプル

Core Motion を使用するより詳細なサンプルコードは，iOS Dev Center（http://developer.apple.com/devcenter/ios/）の Sample Code のページに，1 個登録されています（2011 年 10 月 12 日現在）．pARk という AR のサンプルコードで，本体の方向に加えて方角も参照しながら，ライブカメラの上に，各地の公園の名前を重ね合わせて表示するというものです（図 -7）．

おわりに

この記事で紹介した，Core Location や Core Motion といった，iOS デバイスに搭載されたセンサを活用するためのフレームワークは，地図情報を表示できる Map Kit や 3D の CG 描画が可能な OpenGL ES，3D 空間でのオーディオ再生のための OpenAL などのフレームワークと組み合わせて利用しやすいように工夫されています．さらに iOS 5 で導入された Core Image というフレームワークでは，人の顔認識ができる画像処理フィルタが用意されていたりもします．さまざまなセンサとフレームワークを組み合わせることで，今後面白い応用がされていくことを期待しています．

（2011 年 10 月 14 日受付）

沼田哲史（正会員）　numata@dg.osakac.ac.jp
1978 年 1 月生．2005 年大阪大学大学院情報科学研究科にて博士（情報科学）取得．同年より大阪電気通信大学総合情報学部デジタルゲーム学科講師．著書に「実践 iPad/iPhone ゲームプログラミング」（秀和システム），「iPad/iPhone アプリを作る前に知っておきたい 70 の常識」（秀和システム），「15 歳からはじめる iPhone わくわくゲームプログラミング教室」（ラトルズ）．