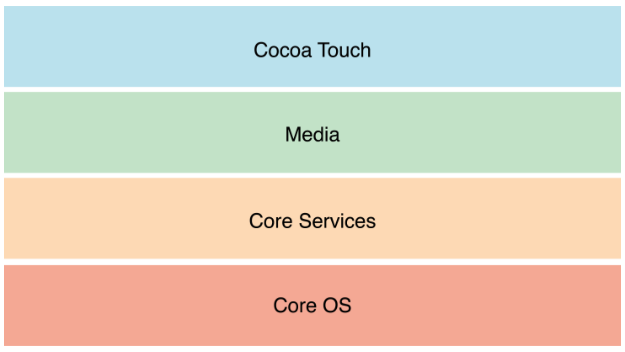
Core Motion

Using Device Motion in iOS

Lukas Welte

FRAMEWORK OVERVIEW

CoreMotion is a framework provided by Apple which you can use to access all motion based data available on the device. It is part of the Core Services Layer.



The framework provides access to the raw data of :

• Accelerometer

• Gyroscope

• Magnetometer

And provides sensor fusion data that delivers:

• Attitude

• Rotation Rate

• Gravity

• User Acceleration

• Calibrated Magnetic Field

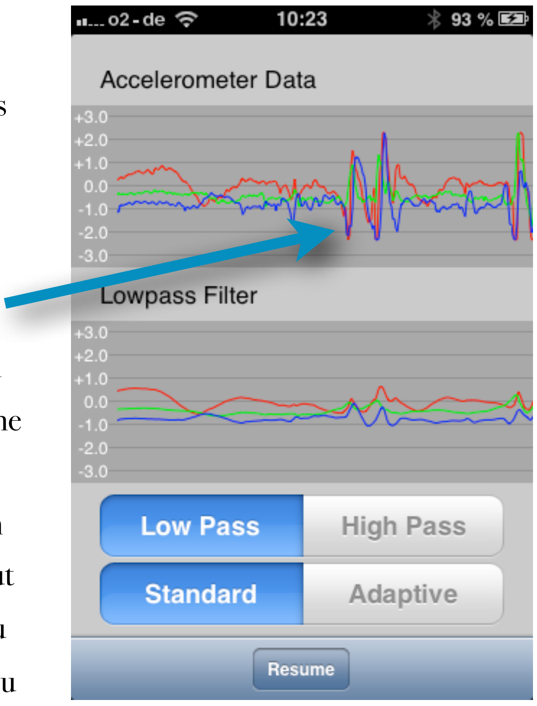
Motion Sensors

Available Motion Sensors in iOS Devices

With the release of the first iPhone there have been Motion Sensors built into the iPhone and all the other iOS devices. In the beginning it should only detect the orientation of the screen, so figuring out if the user is holding it in landscape or portrait mode. In order to detect this the first iPhone had just one Motion Sensor, an Accelerometer.

ACCELEROMETER

The Accelerometer is able to detect the amount of acceleration. Because the earth always accelerates objects in a thing called gravity, the Accelerometer actually measures the sum of gravity and user acceleration. Like mentioned before the Accelerometer is in iOS devices since the first iPhone, so in absolutely every iOS ever shipped. Although the sensor is quite responsive, it is not very precise.



The sensors values are quite noisy and fast movements cause instability. You can see this on the right image. The top graph represents the raw Accelerometer Data. You can see the noise by all the little edges on the graph. The instability can be seen at the point with high amplitudes in the graph. The device has been hold in the same Attitude, so it remained in the same angles towards the ground. That means that normally the gravity which you can see in the bottom graph should remain the same, but like you can see, it doesn‘t. In the bottom you can also see Low Pass and High Pass. Like you can get from the names we need to filter the Accelerometer Data to get either gravity or user acceleration out of the data. Low Pass filtering will give you gravity and high pass filtering gets you the user acceleration. But filtering always creates a delay. This delay increases the better the filter should work. Therefore the Accelerometer is not so fast in consequence of the delay.

Further we can't measure rotations about gravity, so rotations around axis vertical to the earth, because it doesn't affect any changes in the gravity. Spinning the device while it lies on a table would be an example. It wouldn't affect any changes to the Accelerometer's values.

MAGNETOMETER

With the iPhone 3GS Apple added another Sensor to the iPhone, to fill this gap of not being able to measure rotations about gravity. This sensor was the Magnetometer which is able to measure magnetic fields. So it can work as a compass, which like you all know can measure a spin without changing the attitude towards gravity, because it always can measure where the magnetic north of the globe is. With this information you always know the heading of the device, so in which cardinal direction the device points. And of course this also works while rotating about gravity. But the Magnetometer, which was added to CoreMotion with iOS 5.0, has one big problem. It's accuracy is very poor, because of the many magnetic fields we have around us and in the device. Compared to all these computers and refrigerators that create a magnetic field the earth magnetic field is very little. So it is really hard to get good values out of the Magnetometer.

GYROSCOPE

Although the magnetometer is quite useful to determine the heading of a device, it is completely useless to detect spins because of it's bad accuracy. A sensor that enables this is the Gyroscope which was added to the iOS devices with the introduction of iPhone 4. With the Gyroscope you can measure rotation rates independently from the devices orientation. This causes the Gyro to provide precise data. But also this data isn't perfect, because we have a bias over time caused by the earths rotation.

Device Motion

Sensor Fusion provided by CoreMotion

Although there are a couple of sensors in iOS devices, none of them individually provides a developer what they normally want to have when developing applications, this is the devices attitude, the devices spin, the devices heading and events like a shake . Either the sensor can't even measure it or the data is to poor. So CoreMotion takes all the available sensors of the device and does various calculations and operations to minimize the errors of the sensors and directly provide the developer the most needed things in a good quality. This results in an Object they call CMDeviceMotion. This device motion provides a couple of things:

- Attitude
- Rotation Rate
- Gravity
- User Acceleration
- Calibrated Magnetic Field

While all properties except for the Attitude can be retrieved via individual sensors you won't get them so easy and not in this quality Device Motion provides you. That's because the Sensor Fusion of CoreMotion, like mentioned before, also minimizes the error of each sensor by using the values of the other sensors. The property where Device Motion changes nearly everything is the Calibrated Magnetic Field. While the Magnetic Field provided by the Magnetometer is more or less completely useless, the Calibrated Magnetic Field can be used, because the values are much better and you can always check how precise these values are, but we will come back to this topic in the Data Types chapter.

The thing you won't get out of the sensors is the Attitude. This is the devices position towards a reference position, so basically how the device is oriented in the 3D field. This means you exactly know how the user holds device.

REFERENCE FRAMES

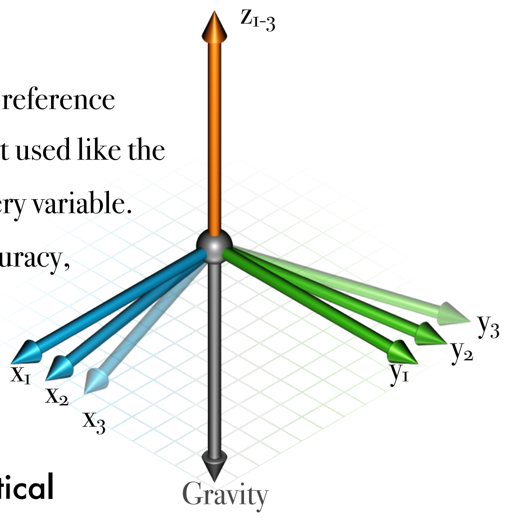
The reference position that is needed to calculate a Attitude can be chosen by a couple of options. Core Motion provides you 4 options that are listed in the following enum:

```
typedef enum {
  CMAttitudeReferenceFrameXArbitraryZVertical = 1 << 0,
  CMAttitudeReferenceFrameXArbitraryCorrectedZVertical = 1 << 1,
  CMAttitudeReferenceFrameXMagneticNorthZVertical = 1 << 2,
  CMAttitudeReferenceFrameXTrueNorthZVertical = 1 << 3
} CMAttitudeReferenceFrame;
```

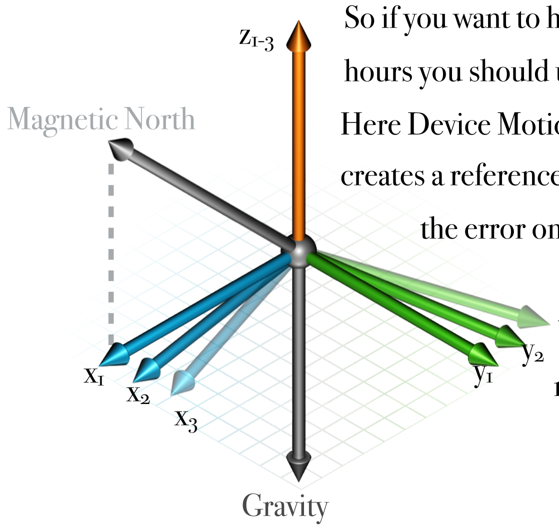
All of them like you can see are Z vertical, so the Z axis always is vertical oriented. The changing part is just the X axis that can be set to the following options.

CMAttitudeReferenceFrameXArbitraryZVertical

If you don't specify a reference frame Device Motion will use this reference frame to calculate the attitude. X Arbitrary means the X axis is just used like the device's X axis is when starting Device Motion. So the X axis is very variable. But this arbitrary selection causes that over the time your yaw accuracy, so the accuracy for a device spin around the Z axis, decreases.



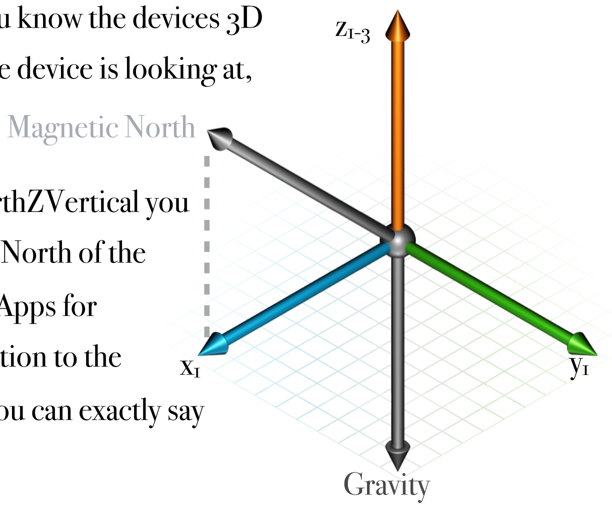
CMAttitudeReferenceFrameXArbitraryCorrectedZVertical



So if you want to have a very correct yaw when the user is using the app a couple of hours you should use CMAttitudeReferenceFrameXArbitraryCorrectedZVertical. Here Device Motion uses the Magnetometer to determine the Magnetic North and creates a reference with it towards the arbitrary chosen X axis. With this reference the error on yaw accuracy can be corrected. But of course this causes higher CPU usage, because Device Motion needs to calculate the error and correct it. So I would recommend to use it just if it is really necessary to have very accurate yaw over longterm.

CMAttitudeReferenceFrameXMagneticNorthZVertical

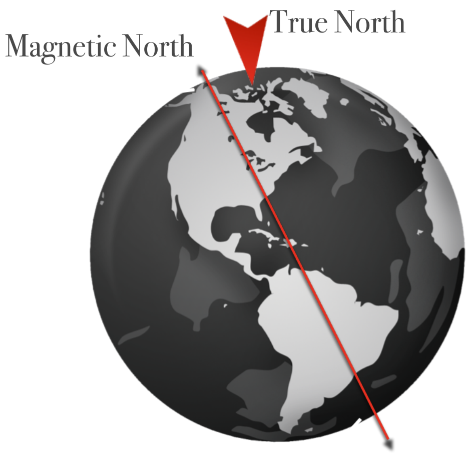
With the reference frames mentioned till now, you know the devices 3D orientation, but you can't determine exactly where device is looking at, because of the arbitrary chosen X axis.

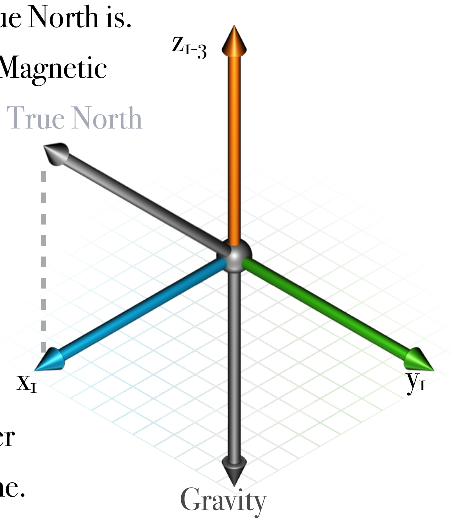


With CMAttitudeReferenceFrameXMagneticNorthZVertical you get an X axis which always points to the Magnetic North of the earth. This is very helpful for Augmented Reality Apps for example, because if you know the location in addition to the attitude calculated towards this reference frame you can exactly say where the user looks at.

CMAttitudeReferenceFrameXTrueNorthZVertical

Now we have a reference frame that has a fixed X axis, but normally we don't deal with the Magnetic North. Instead we use the True North, which we call North Pole normally, as reference point, like in Maps for example. To figure out the True North, Device Motion needs to know the Magnetic North and the rough device location. With these two values it can calculate where the True North is. The rest is identical to the Magnetic North Reference Frame. The Reference for the X axis just changes from the Magnetic North to the True North. This reference is easier to use, but needs even more battery than the Magnetic North Reference Frame, because Device Motion needs to figure out the rough location of the device. So if you don't use the location in some other purpose in the App consider if you really need this Reference Frame.





SENSOR FUNCTION OVERVIEW

	Accelerometer	Magnetometer	Gyro	Device Motion
Attitude	⚠	✗	✗	✓
Heading	✗	⚠	✗	✓
Spin	✗	✗	✓	✓
Shake	⚠	✗	✗	✓

The Magnetometer is able to figure out the heading but, with the Magnetic Field you get out of the Raw Magnetometer Data, the heading you calculate with these data is far from accurate. Also the Accelerometer gives you the possibility to get a shake event or a rough attitude of the device. But also here the attitude you can calculate is not very good and also the possibility to detect shake events is very restricted. To detect this stuff in a precise manner you need more than one sensor.

SENSOR DEVICE AVAILABILITY

	iPhone 3GS	iPhone 4 - 5	iPad	iPad 2	iPod touch 3G	iPod touch 4G	Simulator
Accelerometer	✓	✓	✓	✓	✓	✓	✗
Gyro	✗	✓	✗	✓	✗	✓	✗
Magnetometer	✓	✓	✓	✓	✗	✗	✗
Device Motion	✗	✓	✗	✓	✗	⚠	✗

On all iPod touches there is no Magnetometer built in. That‘s why all iPod touches since the 4th generation iPod touch have no complete Device Motion. All these iPod touches have Gyroscopes built in, but because they have no Magnetometer, not all Reference Frames of Device Motion are available and of course Core Motion doesn‘t provide Magnetometer Data and a calibratedField via CMDeviceMotion.

Retrieve Motion Data

Get the information out of the framework

Retrieving the samples from the motion sensors, or getting the sensor fusion device motion is really simple with CoreMotion. There are basically 3 steps:

1. Create or configure CMMotionManager
2. Check if desired source is available
3. Start the updates by pushing or pulling

That's why the following examples for each sensor look quite familiar.

ACCELEROMETER DATA

1. Include CoreMotion.framework in your Project
2. Import <CoreMotion/CoreMotion.h> to your file where you want to use CoreMotion
3. If there is no singleton (for example in AppDelegate) create a CMMotionManger, but be save it's just one single instance running. If there is CMMotionManager singleton (for example in AppDelegate) just use this one
4. Check if Accelerometer is available via if ([self.motionManager isAccelerometerAvailable])
5. Set the desired update interval (30-60 Hz for games recommended)
6. Start the accelerometer updates in push or pull mode

MAGNETOMETER DATA

1. Include CoreMotion.framework in your Project
2. Import <CoreMotion/CoreMotion.h> to your file where you want to use CoreMotion
3. If there is no singleton (for example in AppDelegate) create a CMMotionManger, but be save it‘s just one single instance running. If there is CMMotionManager singleton (for example in AppDelegate) just use this one
4. Check if Magnetometer is available via if ([self.motionManager isMagnetometerAvailable])
5. Set the desired update interval (30-60 Hz for games recommended)
6. Start the magnetometer updates in push or pull mode

GYROSCOPE DATA

1. Include CoreMotion.framework in your Project
2. Import <CoreMotion/CoreMotion.h> to your file where you want to use CoreMotion
3. If there is no singleton (for example in AppDelegate) create a CMMotionManger, but be save it‘s just one single instance running. If there is CMMotionManager singleton (for example in AppDelegate) just use this one
4. Check if Gyroscope is available via if ([self.motionManager isGyroAvailable])
5. Set the desired update interval (30-60 Hz for games recommended)
6. Start the gyro updates in push or pull mode

PUSH MODE - PULL MODE

Core Motion provides two options to get samples from the Motion Sensor and Device Motion.

Push Mode

In Push Mode you need to provide the CoreMotion source a queue and a handler block. As soon as a Motion Sensor or Device Motion processes a new sample it will push the sample in the queue, and your handler block will process the queue as fast as he can. So Push Mode will provide you absolutely every sample that is processed. You won‘t miss any. But for normal you don‘t need really every sample. You can easily drop many of them. That‘s why push mode has a big overhead. Therefore I would recommend you to use it only for Data Collection Apps.

Pull Mode

In Pull Mode Core Motion processes everything in the background, and you just ask your Motion Manager when you want a sample. Then the Motion Manager will deliver you the latest sample. This causes less code and a little overhead, because you only grab and handle samples you really need. The only thing you might need to do, if your program not already has one, is to provide a method that is periodically called to grab the samples periodically. The best way to do this is grabbing the data in the method that draws your view. Because normally just user interaction which you will display on the screen is interesting for your application. To sum it up I would recommend you to use Pull Mode in all interactive Apps and absolutely all Games.

Push - Pull Mode Comparison

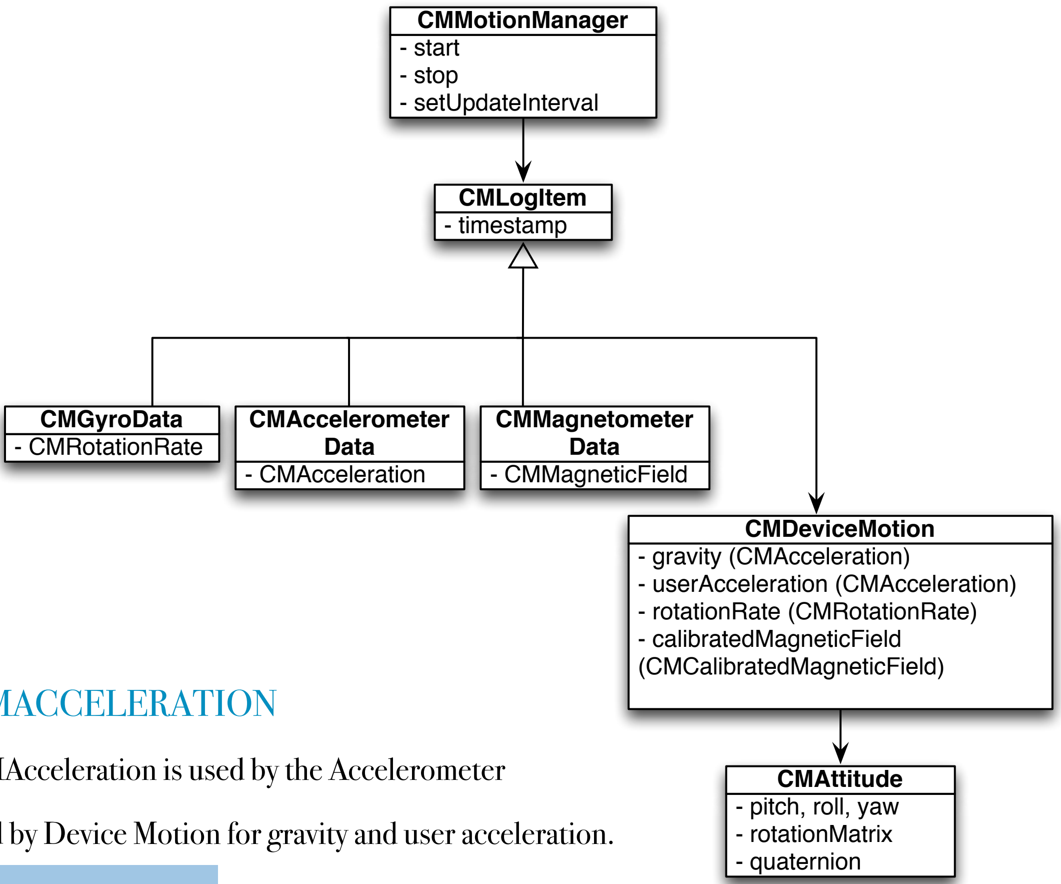
	Push	Pull
Positive	Absolutely all samples	More efficient Less code
Negative	Increased Overhead	Eventually additional Timer necessary
Usage	Data Collection Apps	Most Apps and Games

Data Types

Data Structures of the Motion Sensor Data

When you grab the values of the Motion Sensors or Device Motion from your Motion Manager it will give you a couple of types.

OVERVIEW



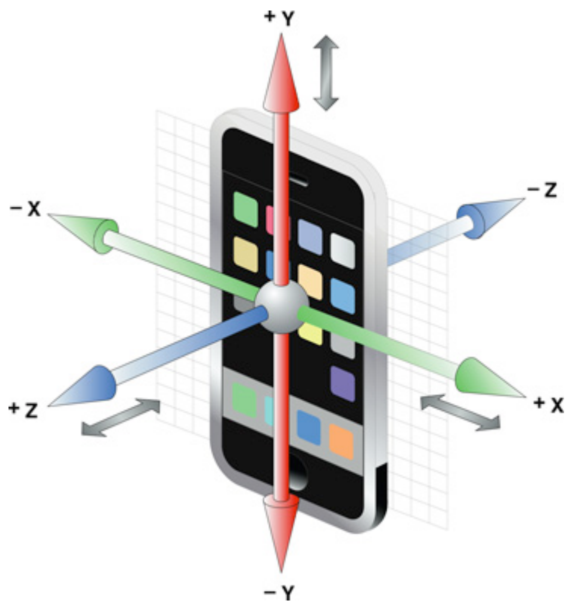
CMACCELERATION

CMAcceleration is used by the Accelerometer

and by Device Motion for gravity and user acceleration.

```
typedef struct {
  double x;
  double y;
  double z;
} CMAcceleration;
```

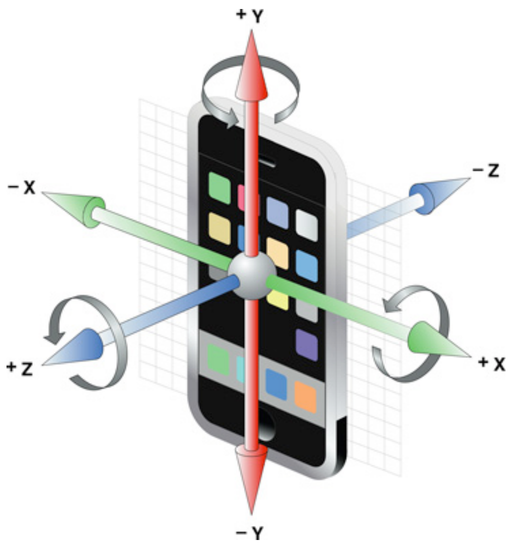
It represents the force on each axis. Force is measured in G's. You can get values plus minus 2G.



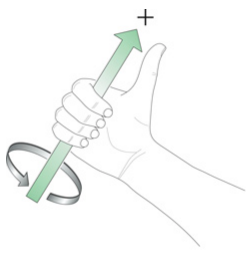
CMROTATIONRATE

CMRotationRate is used by the Gyro and by Device Motion for rotationRate.

```
typedef struct {

   double x;

   double y;

   double z;

} CMRotationRate
```



It represents the rotation in radians per second around each axis. All values follow the right hand rule, which works like you know from your last physics class.

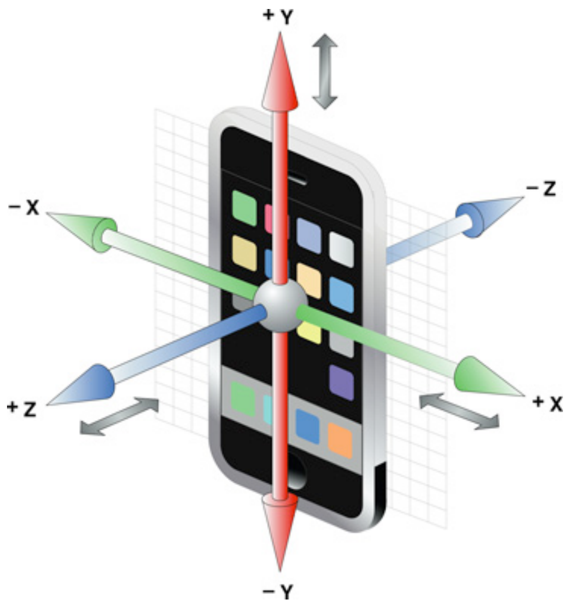


If you point with your thumb in the direction of the axis-vector your fingers show you in what direction the rotation is positive. Accordingly you can see in the image on the left that a spin counterclockwise around the green arrow (which is demonstrates by the arrow on the bottom of the graphic) comes to a positive value.

CMMAGNETICFIELD

CMMagneticField is used by the Magnetometer.

```
typedef struct {

   double x;

   double y;

   double z;

} CMMagneticField;
```



It represents the magnetic field measured on each axis. The field is measured in microteslas.

CMCALIBRATEDMAGNETICFIELD

CMCalibratedMagneticField is used by DeviceMotion for the calibratedMagneticField Property. It consists of a CMMagneticField struct and a typedef that shows the precision of the measured values.

```
typedef struct {
  CMMagneticField field;
  CMMagneticFieldCalibrationAccuracy accuracy;
} CMCalibratedMagneticField;
```

The CMMagneticField struct can be seen above, while the the typedef for the accuracy CMMagneticFieldCalibrationAccuracy looks like this:

```
typedef enum {
  CMMagneticFieldCalibrationAccuracyUncalibrated = -1,
  CMMagneticFieldCalibrationAccuracyLow,
  CMMagneticFieldCalibrationAccuracyMedium,
  CMMagneticFieldCalibrationAccuracyHigh
} CMMagneticFieldCalibrationAccuracy;
```

The accuracy of the raw Magnetometer Data would always be uncalibrated, therefore CMMagneticFieldCalibrationAccuracyUncalibrated. The Sensor Fusion improves the Magnetic Field a lot, but it also can‘t perform miracles. That‘s why it is great to have the accuracy property: You are always able to check wether the data is good enough for your use usage.

Clean-Up CoreMotion

Save CPU and battery

Every CoreMotion source needs CPU and costs battery live while running. The Sensor of course needs power to run and the CPU is needed to sample the sensor data, or even for the whole sensor fusion process.

To save battery and minimize CPU usage turn of all CoreMotion sources in the minute you don‘t need them any longer. To stop a CoreMotion source simply call

```
[self.motionManager stopAccelerometerUpdates];
```

Or

```
[self.motionManager stopMagnetometerUpdates];
```

Or

```
[self.motionManager stopGyroUpdates];
```

Or

```
[self.motionManager stopDeviceMotionUpdates];
```

And if you no longer need any of the CoreMotion sources stop all the Sources and remove the CMMotionManger by setting it to nil

```
self.motionManager = nil;
```

This saves you CPU that you might want to use in your Application, but also if you aren‘t in need of this extra CPU power the user is happy with a longer battery life, so he can keep on using your Application the whole day.

Summary

Summary of the CoreMotion Framework

COREMOTION IN GAMES

CoreMotion can be used in every game to make the game play intuitive. So just replace the joypad by tilting the device. But you could also really give the user the impression to be part of the game by merging the game with the reality via CoreMotion. You can easily turn whole game worlds according to the devices attitude what creates the feeling that the user really is the controlled character. So the user needs to rotate himself to look into the other direction and not just rotating a character on screen.

But you can also image various other places where you could use the huge advantage of having the knowledge how the device moves in the real world.

LECTURE SUMMARY

This lecture provided you a brief overview about CoreMotion. The Framework provides you the raw MotionSensor data for own calculations and additionally a processed device motion which you can directly use.

When you like to use Motion Events in your application CoreMotion is really the framework to use.

REFERENCES

- Apple CoreMotion Framework Reference
    - CMAccelerometerData Reference
    - CMAttitude Reference
    - CMDeviceMotion Reference
    - CMGyroData Reference
    - CMLogItem Reference
    - CMMagnetometerData Reference

    - CMMotionManager Reference

- WWDC Videos about CoreMotion

- Event Handling Guide for iOS