Ejercicios de sensores en iOS

Índice

1 Pantalla táctil: Implementando gestos....2
2 (*) Reconociendo varios gestos de manera simultánea....2
3 Probando el GPS....2
4 Otros sensores disponibles en iOS....5

1. Pantalla táctil: Implementando gestos

En este primer ejercicio vamos a implementar el reconocimiento de gestos multitáctiles en una aplicación sencilla. Para completar el ejercicio deberemos de realizar los siguientes pasos:

- 1) Creamos un nuevo proyecto en XCode que llamaremos `ejercicio-gestosios` de tipo `Single View Application`, y que utilice ARC.
- 2) Añadimos al proyecto una imagen (*cualquier imagen de tamaño mediano no sirve*)
- 3) Creamos el Outlet en la controladora, definimos el synthesize y lo relacionamos en la vista
- 4) Implementamos el gesto de pulsación simple: `UITapGestureRecognizer`
- 5) Implementamos el gesto de arrastre: `UIPanGestureRecognizer`
- 6) Implementamos el gesto de rotación: `UIRotationGestureRecognizer`
- 7) Implementamos el gesto de pellizco: `UIPinchGestureRecognizer`
- 8) Probamos el código y comprobamos que funcionan todos los gestos correctamente

2. (*) Reconociendo varios gestos de manera simultánea

Una vez realizado el ejercicio anterior deberemos de completarlo añadiéndole la funcionalidad de reconocimiento de varios gestos de manera simultánea, para ello deberemos de implementar el método correspondiente del protocolo `UIGestureRecognizerDelegate`.

3. Probando el GPS

En este ejercicio vamos a probar el funcionamiento del sensor GPS en un aplicación iPhone. Al finalizar el ejercicio podremos ver las coordenadas latitud y longitud que obtenemos del GPS y la velocidad a la que nos movemos. Comezamos creando un proyecto nuevo en XCode usando la plantilla `Single View Application` con las siguientes características:

- Product name: `ejercicio_gspios`
- Company Identifier: `es.ua.jtech`
- Class prefix: `UA`
- Device family: `iPhone`
- Marcar sólo la opción `Use Automatic Reference Counting`. El resto dejarlas desmarcadas.

1) Importamos el framework `CoreLocation.framework` al proyecto.

2) Creamos la clase encargada de gestionar los datos obtenidos por `Core Location` (posición, altitud, velocidad, etc). La clase la llamaremos: `ServicioCoreLocation` y será

de tipo `Objective-C class`. El fichero `ServicioCoreLocation.h` tendrá el siguiente código:

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

@protocol ServicioCoreLocationDelegate
@required
// Las actualizaciones de localizacion se gestionan aqui
- (void)locationUpdate:(CLLocation *)location;
- (void)locationError:(NSError *)error; // Cualquier error se gestiona
aqui
@end



@interface ServicioCoreLocation : NSObject <CLLocationManagerDelegate> {
    CLLocationManager *locMgr;
    id delegate;
}

@property (nonatomic, strong) CLLocationManager *locMgr;
@property (nonatomic, weak) id delegate;

@end
```

Por otro lado el fichero `ServicioCoreLocation.m` tendrá el siguiente código:

```
               #import "ServicioCoreLocation.h"

@implementation ServicioCoreLocation

@synthesize locMgr = _locMgr;
@synthesize delegate = _delegate;

// Metodo de inicialización
- (id)init {
    self = [super init];

    if(self != nil) {
        // Creamos una nueva instancia de locMgr
        self.locMgr = [[CLLocationManager alloc] init];
        self.locMgr.delegate = self; // Esta clase será el delegado
    }

    return self;
}

// Metodos del protocolo CLLocationManagerDelegate
- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation fromLocation:(CLLocation
*)oldLocation {
    if([self.delegate
conformsToProtocol:@protocol(ServicioCoreLocationDelegate)]) {
        [self.delegate locationUpdate:newLocation];
    }
}

- (void)locationManager:(CLLocationManager *)manager
didFailWithError:(NSError *)error {
    if([self.delegate
```

```
conformsToProtocol:@protocol(ServicioCoreLocationDelegate)]) {
        [self.delegate locationError:error];
    }
}

@end
```

Una vez implementada toda la gestión de los servicios de localización vamos a implementar nuestra controladora de la vista de la aplicación.

3) Modificamos la vista (fichero `UAViewController.xib`) añadiéndole un objeto `UILabel` que será donde mostremos los datos obtenidos. Creamos el Outlet del Label en la controladora y se lo asignamos en la vista.

4) Modificamos la definición de la controladora `UAViewController` para que implemente los métodos del protocolo `ServicioCoreLocationDelegate` que hemos creado en el paso 2.

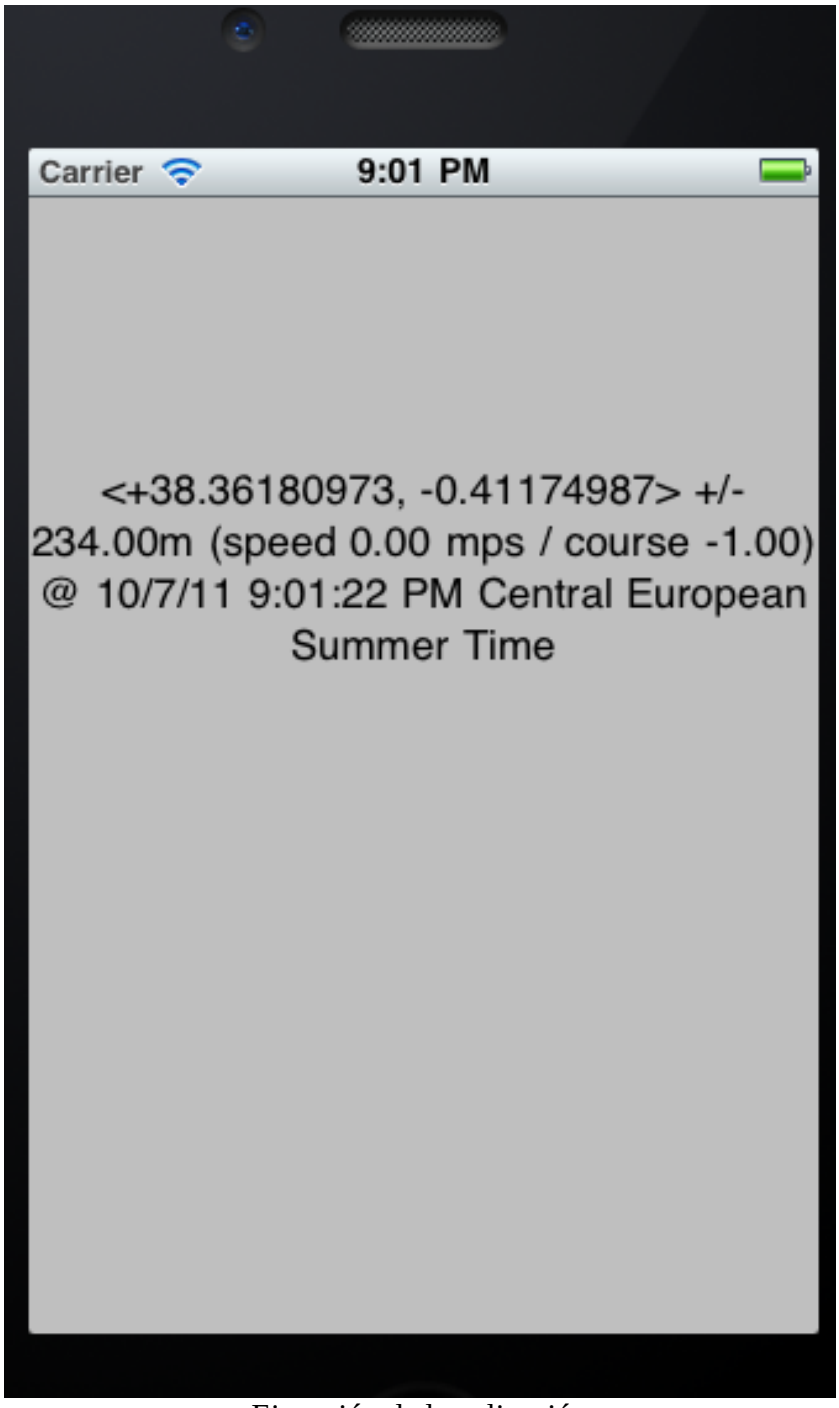
5) Implementamos los dos métodos del protocolo `ServicioCoreLocationDelegate`:

```
        #pragma mark Metodos del protocolo ServicioCoreLocationDelegate
- (void)locationUpdate:(CLLocation *)location {
    //TODO: Actualizar el label
}

- (void)locationError:(NSError *)error {
    //TODO: Actualizar el label
}
```

6) Por último deberemos inicializar el servicio de localización (`ServicioCoreLocation`) asignando la propiedad `delegate` a `self` e iniciando el servicio de esta forma: `[miServicioCoreLocation.locMgr startUpdatingLocation];`. Todo esto lo haremos en el método `viewDidLoad` la controladora `UAViewController`.

7) Probamos la aplicación y nos deberá aparecer una ventana solicitándonos permiso para obtener los datos del GPS. Si aceptamos se nos deberá actualizar el *Label* con los datos de localización básicos tal y como se muestra en la siguiente imagen:



Ejecución de la aplicación

4. Otros sensores disponibles en iOS

Responde a las siguientes cuestiones sobre los distintos sensores que podemos encontrar en iOS:

- 1) ¿Qué framework deberemos utilizar en nuestro proyecto para usar el sensor de la brújula?
- 2) ¿Qué framework deberemos utilizar en nuestro proyecto para usar el sensor del giroscopio? ¿Es compatible el giroscopio con el iPad 1?
- 3) Mediante la API de iOS podemos acceder al sensor de proximidad. ¿Que datos podemos obtener de este sensor?