TÍTULO
DESARROLLO DE UNA APLICACIÓN WEB DE LOCALIZACIÓN MEDIANTE GPS Y JAVA

AUTOR
FRANCISCO JOSÉ GARCÍA RICO

TUTOR
MIGUEL ÁNGEL CAZORLA QUEVEDO

DEPARTAMENTO
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

CURSO
2003-2004

ÍNDICE

INTRODUCCIÓN

El proyecto consistía en desarrollar una aplicación web para la localización de un usuario en el entorno de la Universidad de Alicante mediante la utilización de un dispositivo GPS y un elemento móvil como puede ser un portátil. La idea inicial era que el usuario accediera a una página web y que esta le indicará la posición exacta. Posteriormente el usuario sería capaz de elegir un destino al que desee llegar y la aplicación le indicará una ruta para llegar a este punto final.

La comunicación entre el dispositivo GPS y el portátil se realiza a través del puerto serie, que es el tipo de conexión que permiten la mayoría de los dispositivos GPS. Sin embargo, los últimos portátiles empiezan a no disponer de un puerto serie y esta comunicación se debe realizar a través del puerto USB. Para ello, la señal procedente del dispositivo GPS se debe convertir para ser adaptada a un puerto USB mediante un conversor de serie DB-9 a USB.

El portátil ya comienza a recibir datos procedentes del dispositivo GPS, sin embargo necesitamos establecer un protocolo de comunicación entre ambos. Existen dos protocolos comúnmente utilizados por los dispositivos GPS: NMEA y GARMIN. Para el desarrollo del proyecto se dispone del modelo de GPS *SportTra Pro de Magellan* que permite el envío de datos utilizando el protocolo NMEA. El dispositivo GPS se dedica a enviar ristras de caracteres en este protocolo que posteriormente deberán ser interpretadas por el ordenador. Esta interpretación de los resultados se realiza mediante unas librerías de libre distribución para fines no comerciales llamadas Chaeron. Mediante su utilización podemos olvidarnos de tener que interpretar esas ristras de caracteres y nos ceñiremos a realizar llamadas para recuperar la información necesaria del dispositivo GPS. Estas librerías las podemos obtener en la dirección http://www.chaeron.com

Los datos recibidos empiezan a tener sentido. El siguiente paso consiste en calibrar un mapa de la Universidad de Alicante para posteriormente situar al usuario en su interior. La información necesaria del dispositivo GPS es la latitud y la longitud, que es recibida en coordenadas geográficas y en formato decimal. Por ejemplo, latitud podría ser 38,3332 y la longitud, 30,7833. Estas coordenadas geográficas necesitan ser convertidas a otro tipo de coordenadas conocidas como UTM (Universal Transversor Mercator) que nos permite de forma más sencilla la calibración del mapa y situar posteriormente al usuario en el mapa. Además, también convertiremos estas coordenadas a formato de grados, minutos y segundos para mostrárselas al usuario.

El usuario ya se encuentra localizado en el mapa y este puede elegir un destino. El mapa se encuentra representado en una capa inferior como un grafo conectado con aristas bidireccionales. Para calcular el camino más corto entre un punto inicial y otro final se optó por desarrollar el algoritmo A-estrella. Además, el usuario tiene la posibilidad de que la aplicación le vaya indicando en cada momento la acción a realizar a medida que va siguiendo la ruta.

La aplicación ha sido desarrollada por completo en un applet en Java que además utiliza un servidor de base de datos MySQL, de donde obtiene los datos referentes a algunos destinos posibles.

DESARROLLO

En este apartado se explica como se han desarrollado las diferentes partes del proyecto, empezando por la comunicación entre el dispositivo GPS y el portátil, la conversión de los datos para su utilización en la aplicación, el cálculo de caminos óptimos, el seguimiento de una ruta calculada y por último la construcción del applet.

CONEXIÓN CON EL DISPOSITIVO GPS

El modelo del dispositivo GPS con el que se ha trabajado durante el desarrollo del proyecto ha sido el *SportTrak Pro de Magellan*. En principio, la aplicación sólo ha podido ser probada con este dispositivo, por ser el único del cual se disponía, no obstante, la utilización de un protocolo estándar como es el NMEA (National Marine Electronics Association) permitiría el empleo de cualquier otro tipo de dispositivo GPS capaz de transmitir datos mediante este protocolo.

El SportTrak Pro permite la conexión al puerto serie de un ordenador, pero sin embargo, los portátiles recientes ya no disponen de este tipo de conexión, así que se debe convertir la señal procedente del GPS para que pueda ser interpretada por un puerto USB. Para ello se utiliza un convertidor de serie a USB.

Una vez conectado todo correctamente debemos seleccionar en las opciones del dispositivo GPS el método de envío de datos a través del protocolo NMEA con una velocidad de 9600 baudios. Las secuencias de caracteres NMEA envía secuencias similares a las siguientes:

$GPGLL,3823.1833,N,00030.7505,W,093818.026,A*2E

$GPGGA,093818.03,3823.1833,N,00030.7505,W,1,04,13.3,00088,M,,,,*09

$GPGLL,3823.1834,N,00030.7505,W,093820.030,A*25

$GPGGA,093820.03,3823.1834,N,00030.7505,W,1,04,13.3,00088,M,,,,*05

Existen muchos tipos de secuencias NMEA pero las más importantes son las que nos informan sobre las coordenadas geográficas. En Internet se encontró una librería de libre distribución que permitía de forma transparente la interpretación de las sentencias NMEA y obtener de forma sencilla las coordenadas geográficas enviadas por el GPS, así como otros datos del mismo. Esta librería se puede obtener desde la dirección web http://www.chaeron.com/gps.html y gracias a un programa ejemplo que venía con estas librerías, se logró conseguir la comunicación del dispositivo GPS con la aplicación Java.

Para utilizar esta librería correctamente se desarrollo la clase *GPSJava* que implementaba un *GPSListener* (clase de la librería) para obtener los datos del GPS. Esta clase es la principal que se debe utilizar para recoger la información del GPS. En ella se desarrollaron métodos para obtener información como latitud, longitud y elevación. Además, también se implementa un método (*onGPSEvent*) en el que se diferencian las sentencias NMEA y se actúa en consecuencia.

Existen varios tipos de GPSEvent (eventos), entre los que se encuentra el GPS_EVENT_POSITION que es la sentencia NMEA que transmite la información del usuario en cuanto a latitud, longitud y elevación. Una vez se detecta que se ha recibido este tipo de información, se actualizan los valores correspondientes a la latitud, longitud y elevación de la instancia GPSJava.

Los otros tipos de GPSEvent son ignorados en la aplicación. Entre estos se encuentran otro tipo de información como el tratamiento de errores, la velocidad del usuario, direccionamiento, número de satélites activos y otros menos relevantes.

Posteriormente también se necesita sobrecargar otros métodos. Estos son *initGPS(), pollGPS() y disconnectGPS().* El método *initGPS(),* como su nombre indica se utiliza para inicializar la comunicación del computador con el dispositivo GPS. En este método se le especifica al programa que tipo de protocolo se va a emplear, así como el puerto a utilizar y la velocidad de envío de datos. También, es necesario especificar que tipos de GPSEvent se desea estar recibiendo.

El método *pollGPS()* se utiliza para realizar una pregunta al dispositivo GPS sobre los datos enviados. Por último, *disconnectGPS()* se utiliza en el momento en que deseemos terminar la conexión con el dispositivo GPS, bien porque se haya producido un error o bien por el usuario haya terminado su utilización.

Un esquema muy simple de cómo se debería utilizar esta librería podría ser el siguiente. En él se inicializa un objeto de la clase GPSJava y posteriormente se van realizando llamadas periódicas para obtener los datos necesarios del usuario.

```
GPSJava.initGPS();//inicializamos el GPS
Cada 2 segundos hacer{
        GPSJava.pollGPS();//preguntamos al GPS
        GPSJava.getLongitud();//obtenemos la longitud
        GPSJava.getLatitud();//obtenemos la latitud
}
```

La librería Chaeron, además de disponer de un ejemplo de utilización de la misma, viene también con una completa API de utilización. Las librerías necesarias para la aplicación son *gps_windows.jar* y *gps_common.jar*. Además de estas librerías que vienen con la distribución de Chaeron, también fue necesario ampliar la máquina virtual de Java con unas librerías utilizadas para el acceso al puerto serie que no vienen con la distribución base de Java. Estas librerías se conocen como javacomm y se puede obtener más información sobre ellas en la dirección web oficial de Java en Sun http://java.sun.com/products/javacomm/ .

Clases involucradas en la etapa de conexión con el dispositivo GPS:

- *GPSJava:* en esta clase se desarrollan métodos para el acceso a la longitud, latitud y elevación así como los métodos anteriormente explicados de initGPS(), pollGPS() y disconnectGPS(). Con una instancia de esta clase, podremos realizar la comunicación entre dispositivo GPS y aplicación Java.

CALIBRACIÓN DEL MAPA

Sin duda está fue la etapa más costosa de llevar a cabo en todo el proyecto. Para realizar esta calibración se requerían unos conceptos avanzados de cartografía de los cuales no se disponía y resultó bastante complicado conseguir unos resultados satisfactorios para localizar a un usuario en el entorno de la Universidad de Alicante. Además esta deficiente etapa ha supuesto un lastre para el resto de partes del proyecto.

En principio los datos que necesitamos obtener del dispositivo GPS son la latitud y la longitud. Estos datos son recogidos gracias a la clase GPSJava explicada anteriormente. El formato en el que son recibidos estos datos es el decimal, como por ejemplo 38,23183, y son convertidos a formato de grados, minutos y segundos para mostrárselos al usuario, por ejemplo 38º22'50''. Sin embargo, la calibración del mapa resulta más sencilla si se utilizan otro tipo de coordenadas conocidas como *UTM (Universal Transversor Mercator).* Este tipo de coordenadas entienden al mundo como un plano en dos dimensiones, con lo que se asemeja a la representación en píxeles de nuestro plano de la Universidad.

De esta forma, para referenciar a un usuario dentro del plano de la Universidad de Alicante, necesitamos conocer las coordenadas UTM de las esquinas de dicho plano. Con conocer las coordenadas de tres esquinas, la cuarta se puede deducir fácilmente calculando diferencias verticales y horizontales con las coordenadas conocidas, tal y como se muestra en el ejemplo:

Figura 1: Calibración del mapa utilizado en la aplicación

En el mapa presentado se conocen las coordenadas UTM de todas las esquinas, salvo la superior derecha. Sin embargo, conociendo las relaciones entre las otras esquinas, se pueden obtener estos valores. Por ejemplo, entre las esquinas inferiores se aprecia una diferencia en la Y de 9 unidades, por lo que en las esquinas superiores también se debe producir esta diferencia. Por otro lado, la diferencia en las esquinas inferiores para la X es de 455 unidades, con lo que el dato que nos falta también lo podemos obtener calculando esa diferencia entre las esquinas superiores.

Una vez tenemos el mapa calibrado tan sólo nos queda referenciar un punto cualquiera en coordenadas UTM (convertidas a partir de coordenadas geográficas). Para ello, necesitamos saber la relación vertical y horizontal existente entre un píxel de la imagen y las coordenadas UTM. Por ejemplo, si la imagen tiene de ancho 258 píxeles y la diferencia horizontal en coordenadas UTM es de 355 (717454 – 717099), sabemos que cada píxel horizontal de la imagen se corresponde con 355/258 coordenadas UTM. Se debe proceder de la misma forma para las coordenadas verticales.

Las fórmulas para la obtención de un punto en la imagen a partir de sus coordenadas UTM son las siguientes:

$$px? \frac{\left| Xesquina\sup eriorizquierda\ ?\ xpunto \right|}{\left| diferenciaHorizontal UTM \right| / tamaño\,\mathrm{Im}\, gHorizontal}$$

$$py? \frac{\left| Yesquina\sup eriorizquierda\ ?\ ypunto \right|}{\left| diferenciaVerticalUTM \right| / tamaño\,\mathrm{Im}\, gVertical}$$

Ecuación 1: Conversión de coordenadas UTM a un punto en la imagen

Por ejemplo, si tenemos las coordenadas UTM X = 717255 e Y = 4251511, se calcularía la posición en el plano de la Universidad de Alicante de la siguiente forma:

$$px\ ? \frac{\left| 717102\ ?\ 717255 \right|}{\left| 355 \right| / 258} ? \frac{153}{1'37}\ ?\ 111 \qquad py\ ? \frac{\left| 4251889\ ?\ 4251511 \right|}{\left| 611 \right| / 383} ? \frac{378}{1'59}\ ?\ 237$$

El punto en la imagen es el píxel (111, 237). ¡Ya tenemos localizado al usuario en el mapa!.

Las clases involucradas en esta etapa son las siguientes:

- *DatosMapa*: esta clase contiene la información referente a los datos en coordenadas UTM de las esquinas del mapa de la Universidad de Alicante, así como el ancho de la imagen en píxeles. De esta forma, para la conversión de coordenadas podemos utilizar esta clase para recoger la información referente al mapa utilizado.
- *PuntoImagen*: se representa una clase para almacenar la información de puntos sobre el mapa. Contendrá datos referentes a coordenadas x e y de un punto en el mapa.
- *UTMCoord*: es la clase equivalente a PuntoImagen pero en este caso con coordenadas UTM. Contiene datos sobre X y Y en coordenadas UTM.
- *Calibrar*: es la clase encargada de realizar la conversión de una coordenada UTM a un punto de la imagen del mapa.
- *LatLonConvert*: esta clase se utiliza para convertir unas coordenadas geográficas en formato decimal en otras en formato de grados, minutos y segundos o viceversa.

A todo esto hay que añadirle la conversión de coordenadas geográficas en coordenadas UTM. Esto se realiza con una librería conocida como geotransform.jar

que permite la conversión de un tipo de coordenadas (en nuestro caso geográficas) en otras como UTM. Esta librería se puede descargar desde la web http://www.geovrml.org/geotransform/download.shtml y es muy práctica para realizar este tipo de conversiones.

CÁLCULO DE LA RUTA

Esta es una de las partes más importantes de la aplicación y consiste en encontrar un camino desde un punto inicial hasta un punto final. Esta parte de la aplicación suponía representar el mapa utilizado de tal forma que se pudiera emplear un algoritmo cualquiera para la búsqueda de caminos entre dos puntos del mismo. Entre las diversas posibilidades estudiadas, finalmente se optó por utilizar un grafo conectado para posteriormente calcular el camino mediante el algoritmo A-estrella.

Sobre el mapa se situaron diversos puntos clave (vértices) y se conectaron entre sí formando un grafo. Como se comentaba anteriormente, el cálculo del camino más corto entre dos vértices cualesquiera se realiza utilizando el algoritmo A-estrella. La idea consiste en ir formando caminos desde un vértice inicial hasta llegar a otro vértice final. Para cada camino se calculara el coste del mismo, calculado como la suma de la distancia recorrida entre cada par de vértices (*función g*), así como también se necesitará obtener una heurística (*función h*). La heurística utilizada consiste en calcular la distancia euclídea desde el último vértice del camino analizado hasta el vértice destino. Con esto, iríamos formando una lista de caminos ABIERTOS ordenados de menor a mayor coste(función f = función g + función h), para ir analizando por orden cada camino (ACTUAL) de esta lista. Una vez analizado un camino, este se pasará a una lista de caminos CERRADOS para no volver a analizar un camino que ya haya sido calculado.

El esquema en pseudocódigo del algoritmo podría ser algo así:

1. Hacer ABIERTOS la cola formada por el nodo inicial (es decir, el nodo cuyo estado solo contiene el vértice inicial, cuya función g es 0 y función h es la distancia euclídea desde el vértice inicial hasta el final).
2. Mientras que abiertos no esté vacía,
    2.1. Hacer ACTUAL el primer nodo de ABIERTOS
    2.2. Hacer ABIERTOS el resto de ABIERTOS
    2.3. Poner el nodo ACTUAL en CERRADOS
    2.4. Si el nodo ACTUAL llega al vértice destino
        2.4.1. Devolver el nodo ACTUAL y terminar
        2.4.2. en caso contrario

2.4.2.1. Hacer NUEVOS-SUCESORES la lista de nodos de SUCESORES(ACTUAL) para los que no existe ni en ABIERTOS ni en CERRADOS un nodo con el mismo estado y menor coste

2.4.2.2. Hacer ABIERTOS el resultado de incluir NUEVOS-SUCESORES en ABIERTOS y ordenar todo en orden creciente de la función f

3. Devolver FALLO

La representación del mapa en un grafo supone la creación de un grafo con 74 vértices y 101 aristas bidireccionales. Los datos de este grafo se encuentran en un fichero de texto con un determinado formato, en el que se especifican los vértices que forman el grafo y posteriormente, las aristas del mismo.

Las estructuras de datos y clases desarrolladas para el uso del grafo han sido las siguientes:

?? *Grafo:* es la clase fundamental y a partir de la cual se forma toda la estructura de grafo necesaria para la ejecución del módulo para calcular la ruta. Se encarga de leer de un fichero de datos con un determinado formato, los datos de este grafo utilizado en el mapa. El formato de este fichero es el siguiente:

;vértices

<identificador> <coordenada x> , <coordenada y>

<identificador> <coordenada x> , <coordenada y>

;aristas

<vertice1>, <vertice2>

<vertice1>, <vertice2>

Cabe resaltar que las aristas se consideran que son bidireccionales, de tal forma que no hay que indicar arista (1,2) y arista (2,1).

En esta clase también se implementa un método para calcular el algoritmo A* entre dos vértices del grafo.

El grafo estará formado por una serie de vértices (clase Vertice).

?? *Vertice*: esta clase contiene la información necesaria de cada vértice. Identificador, coordenada x, coordenada y una lista de las aristas con las que se relaciona este vértice.

- *VerticeAsociado*: esta clase se utiliza para indicar la relación entre dos vértices. Se guarda el identificador del vértice, así como el peso entre ambos vértices.
- *ListaAristas*: un objeto de esta clase está ubicado en cada uno de los Vertices para albergar la información de las aristas con las que se relaciona cada Vertice. En el fondo es un Vector con VerticesAsociados.

Con estas clases representadas, podríamos solucionar el problema de encontrar el camino más corto entre dos vértices del grafo. Pero, ¿qué pasa cuando se quiere encontrar el camino más corto entre dos puntos cualesquiera del mapa? La solución pasa por añadir vértices temporales para calcular desde estos vértices el camino más corto. A continuación se explica de forma más detallada el problema que se presenta y como se puede solucionar.

El cálculo del camino más corto se realiza entre dos vértices del grafo, sin embargo en la aplicación se debe permitir que el camino comience o termine en un punto del mapa que no se corresponda con ningún vértice. Para solucionar este problema, se deben añadir algunos vértices temporales. Para un punto cualquiera del mapa se debe comprobar que este no coincida con ningún punto del mapa. Si es así, se añade un nuevo vértice con el punto indicado en el mapa y se comprueba que arista del mapa es la más cercana al nuevo vértice añadido. Si la arista más cercana pasa sobre el nuevo vértice, es decir, la distancia es 0, no se deben añadir más vértices, pero si la arista más cercana no pasa sobre este nuevo vértice, se debe crear un nuevo vértice en el punto más cercano al primer vértice creado que pase por la arista más cercana. Esto se debe hacer tanto para el punto origen como para el destino. La siguiente figura muestra gráficamente como se procede en cada caso:

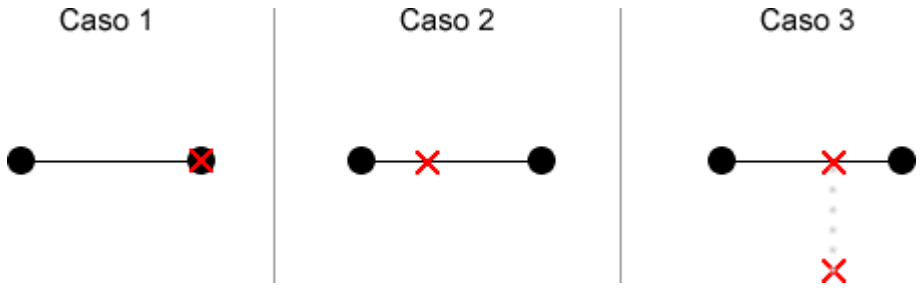

Figura 2: Casos para añadir vértices temporales al grafo original

En el caso 1 no se debe añadir ningún vértice nuevo, en el caso 2 se debe añadir tan sólo un nuevo vértice, mientras que en el tercero se añaden dos nuevos

vértices temporales. Estos vértices temporales deben ser eliminados una vez calculado el camino más corto entre un punto origen y otro destino del mapa.

Para facilitar y acelerar este proceso se añaden las siguientes clases:

- *Recta:* contiene la información de una recta del grafo. Esta recta tiene información sobre sus vértices origen y destino, así como los términos que acompañan a las variables x e y, y el término independiente. En esta clase también se han implementado métodos para calcular la Recta perpendicular a otra que pasa por un punto determinado o el punto en el que intersectan dos rectas. También es necesario calcular la distancia desde un punto dado hasta una recta.
- *ListaRectas:* contiene información sobre todas las rectas del grafo. De esta forma se puede acelerar, a partir de un punto dado, que Recta es la más cercana a ese punto.

La clase Grafo también se encarga de implementar un método para calcular el algoritmo A* a partir de dos puntos cualesquiera del mapa. Este método analiza si los puntos coinciden con algún vértice del grafo y crea los vértices temporales necesarios para el cálculo del A* entre dos vértices cualesquiera. Una vez calculado el camino, se debe proceder a la eliminación de los vértices temporales creados para tal efecto.

El algoritmo A* debe ir formando caminos que se irán ordenando en función de su mayor o menos valia para obtener el camino final. Este algoritmo ordenará estos caminos de mayor a menor función f y los irá seleccionando para ir analizándolos. Las clases implicadas en este proceso son las siguientes:

- *Camino*: implementa un Vector con elementos que indica el orden de visitas de los vértices al calcular una ruta.
- *NodoCamino*: además de un objeto de tipo Camino, tiene el cálculo de las funciones g y h que determinarán un orden para analizar los caminos formados. Se debe sobrecargar la función *compareTo()* para posteriormente establecer un orden.
- *ListaOrdenada:* ordena los elementos NodoCamino formados en función de sus valores g y h. Se hace uso de la sobrecarga de la función compareTo() anteriormente comentada. El algoritmo A* irá seleccionando de mayor a menor los elementos de esta lista.

SEGUIMIENTO DE LA RUTA

La aplicación permite, una vez calculado el camino más corto hasta un destino, el seguimiento de una ruta. Esto quiere decir que el usuario recibirá información puntual en cada instante de su ruta de lo que debe hacer. Además, si el usuario se desvía demasiado de la ruta, esta volverá a ser calculada de nuevo. La idea fundamental para poder implementar este módulo consiste en analizar cada par de rectas que forman la ruta en cuanto a la diferencia de pendientes se refiere. De esta forma, si las pendientes de dos rectas consecutivas de la ruta se parecen lo suficiente, se le indicará al usuario que debe seguir recto hasta nueva orden. En caso contrario, se debe analizar el sentido de las rectas en el camino para indicar que debe realizar un giro a la izquierda o a la derecha. A continuación se presentan algunos ejemplos en el análisis de la ruta:

Figura 3: Ejemplos de seguimiento de la ruta

Se procede de la siguiente forma. Para cada recta de la ruta calculada se obtienen una serie de datos que nos indicarán la forma que tiene. Se debe conocer si el sentido vertical de la recta es Norte o Sur, y si el sentido horizontal de la recta es Oeste o este. También es necesario conocer, cual es la dirección predominante, esto es si la recta es vertical u horizontal. Una vez especificada cada recta de la ruta, se debe calcular la diferencia de pendientes entre una recta y la siguiente para comprobar la acción que debe realizar el usuario al acercarse al punto de corte entre estas dos rectas. Si estas pendientes se parecen, se le indicara al usuario que siga recto hasta nueva orden, mientras que si la diferencia es superior a un umbral establecido, se deben analizar las rectas implicadas con los datos obtenidos de la dirección vertical, horizontal y dirección predominante.

Las siguientes tablas muestran los resultados de analizar dos rectas consecutivas con pendientes diferentes, dependiendo de la dirección predominante de la primera recta:

Dirección predominante recta 1 = Vertical		
	Recta 2 = OESTE	Recta 2 = ESTE
Recta 1 = NORTE	*Gira a la izquierda*	*Gira a la derecha*
Recta 1 = SUR	*Gira a la derecha*	*Gira a la izquierda*

Dirección predominante recta 1 = Horizontal		
	Recta 2 = NORTE	Recta 2 = SUR
Recta 1 = OESTE	*Gira a la derecha*	*Gira a la izquierda*
Recta 1 = ESTE	*Gira a la izquierda*	*Gira a la derecha*

A continuación se muestra como se analizaría el primer ejemplo de la figura 3:

Datos	Recta 1	Recta 2
Dirección predominante	*VERTICAL*	*HORIZONTAL*
Dirección vertical	*NORTE*	*NORTE*
Dirección horizontal	*INDEFINIDO*	*ESTE*
Resultado	**GIRO A LA DERECHA**	

El texto para cada acción se mostrará cuando el usuario se acerque lo suficiente al siguiente vértice de la ruta a visitar. Para ello se comprueba la distancia desde el punto actual hasta el siguiente nodo a visitar. Si esta distancia es inferior a un umbral, se muestra al usuario la siguiente acción a realizar y se actualiza el siguiente nodo a visitar. Cuando se llega a las cercanías del destino final, se mostrará al usuario que ha llegado correctamente al destino solicitado.

Las clases desarrolladas en este módulo son las siguientes:

?? *RectaRuta*: esta clase representa la información de cada recta formada en la ruta. Esta información es la pendiente de la recta, la dirección

predominante (vertical u horizontal), dirección vertical (norte o sur) y dirección horizontal (este u oeste).

?? *SeguirRuta*: es la clase que se encarga, de una vez calculada una ruta, construir todas las acciones necesarias para llegar al objetivo de la misma, a partir de la información de las rectas de la misma. Ella misma, es la encargada de construir la información sobre cada de una de las rectas de la ruta y posteriormente las analiza dos a dos, para organizar todas las acciones.

APPLET

La aplicación final se presenta en un applet el cual podrá ser accedido desde un sitio web en internet. Este applet debe tener acceso al puerto serie (o en su defecto al puerto usb). Para ello se debe utilizar una librería adicional que permite este acceso al puerto serie y que no viene con la distribución del jre común. Esta librería es *comm.jar* que debe ser copiada en el directorio *lib\ext* de nuestra máquina virtual. Además, también es necesario copiar el archivo win32com.dll al directorio *bin* de la misma máquina virtual.

También se deben copiar al directorio *lib\ext* los siguientes archivos que permiten utilizar las librerías chaeron para el acceso al dispositivo GPS:

- *gps_common.jar*
- *gps_windows.jar*

Al acceder a la página web que contiene el applet, nos aparecerá una información de seguridad, la cual se debe aceptar para poder ejecutar el applet sin problemas. Este mensaje de seguridad se debe a que el applet debe ser estar firmado, ya que sino no se podría acceder a la información del puerto serie.

La interfaz del applet muestra el mapa de la Universidad de Alicante a su izquierda, mientras que a la derecha aparece toda la información necesaria para el manejo de la aplicación. Información sobre la latitud y la longitud, posibilidad de seleccionar un destino directamente sobre el mapa o elegir el destino de una lista, la posición destino y final, y por último, información sobre las acciones a realizar para seguir una ruta.

Esta interfaz utiliza 3 clases para su representación:

- *AppletGPSJava*: es la principal de todo y contiene la inicialización de los datos, así como la organización de la información a mostrar en el applet. A la izquierda se mostrará el mapa y a la derecha la botonera.
- *MapaCamino*: clase para dibujar el mapa de la Universidad de Alicante. También se aprovecha para ejecutar aquí la actualización de los datos procedentes del GPS aprovechando el hilo para dibujar el mapa.
- *Controles*: se refiere a toda la botonera necesaria para interactuar con la aplicación. En ella se representa la información de las coordenadas

geográficas, los destinos posibles y los botones para calcular y seguir la ruta.

Esta es una captura de pantalla de la interfaz de usuario del applet.

Figura 4: Interfaz de usuario del applet

También cabe destacar que la lista de destinos posibles que se presenta, se extrae de una base de datos en MySql. Con lo que debe haber una instalación de MySql para poder extraer esta información.

EJEMPLOS DE EJECUCIÓN

A continuación se presentan un par de ejemplos de utilización de la aplicación. En el primero de ellos se pretende llegar desde las cercanías de la Politécnica II hasta la facultad de Ciencias Empresariales. La elección del punto destino se realizó pinchando directamente sobre el mapa.

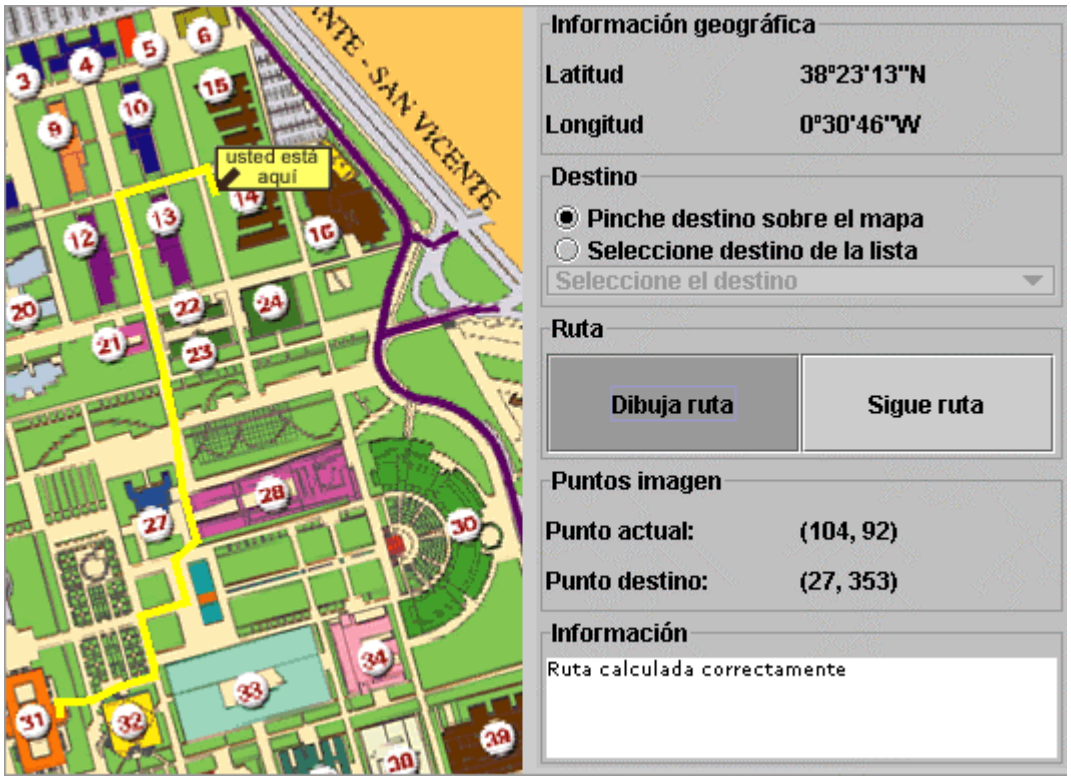

Figura 5: Primer ejemplo de cálculo del camino a seguir

En el segundo ejemplo el usuario se encuentra en la misma posición indicada anteriormente y en esta ocasión selecciona llegar hasta la Facultad de Ciencias III por medio de la lista desplegable.

En ambos ejemplos, el seguimiento de la ruta se realizaba correctamente salvo por algunos problemas para recibir información de la posición de manera constante que situaran al usuario en cada instante en la posición correcta, así como los problemas derivados de la calibración del mapa, que sitúan al usuario en otro punto distinto al que verdaderamente se encuentra. Estos fallos de calibración no suelen ser muy grandes, sin embargo, son suficientes para que la aplicación entienda que el usuario se ha desviado de la ruta y vuelva a recalcular el camino a seguir.

CONCLUSIONES

Durante todo el desarrollo de la aplicación se han encontrado varias dificultades. Empezando por la necesidad de tener acceso por medio de un applet a la información que circulaba por el puerto serie. Esto supone que el usuario debe tener la librería comm.jar correctamente instalada en su máquina virtual. Además, por motivos de seguridad, Java no permite el acceso al puerto serie por medio de un applet, salvo que sea un applet firmado y tengamos configurada la máquina virtual del usuario de una determinada forma.

Por otro lado, el proyecto necesitaba de unos determinados conocimientos de cartografía para la conversión de datos. Esta parte fue sin duda la más dura de todo el proyecto, ya que necesité mucho tiempo para poder obtener unos resultados óptimos en la calibración del mapa. Además, esta etapa ha provocado deficiencias en el resto de etapas, principalmente para seguir correctamente una ruta.

Considero que en este tipo de proyectos sería necesario la colaboración entre diversos departamentos de la Universidad para compartir conocimientos y obtener mejores resultados.

Al comienzo del proyecto se esperaban alcanzar objetivos más elevados que los que se han llegado a conseguir, sin embargo, y después de todo el trabajo que he realizado, considero que los resultados obtenidos son óptimos y me encuentro bastante satisfecho con los mismos.

BIBLIOGRAFÍA

- http://www.magellangps.com
- http://www.mundogps.com
- http://www.chaeron.com
- http://www.uco.es/~bb1rofra/ftpgps.html
- http://forum.java.sun.com
- *Thinking in Java*, Bruce Eckel. Second Edition. Prentice Hall