MEMORIA
PROYECTO FIN DE CARRERA

"VISUALIZACIÓN DE DATOS RECOPILADOS EN REDES VEHICULARES MULTISENSOR"

Departamento de Ingeniería Matemática e Informática

Autor: Alberto Zurbano Gumuzio
Tutor: Jesus Villadangos
16 de Julio de 2013

ÍNDICE

INTRODUCCIÓN

Objetivo

El objetivo del este proyecto es crear un programa que muestre gráficamente datos relacionados a unas coordenadas, los cuales van a ser recogidos por sensores que transportan diferentes vehículos. Los datos recopilados pueden ser del tipo, nivel de humedad en ese punto, temperatura, nivel de CO2 en el aire, etc.

Antecedentes

Lo que he encontrado sobre el tema de Google Maps son las típicas aplicaciones para móviles para localizar móviles, como guía, GPS, etc.

Google Maps: Aplicación básica de Google Maps.

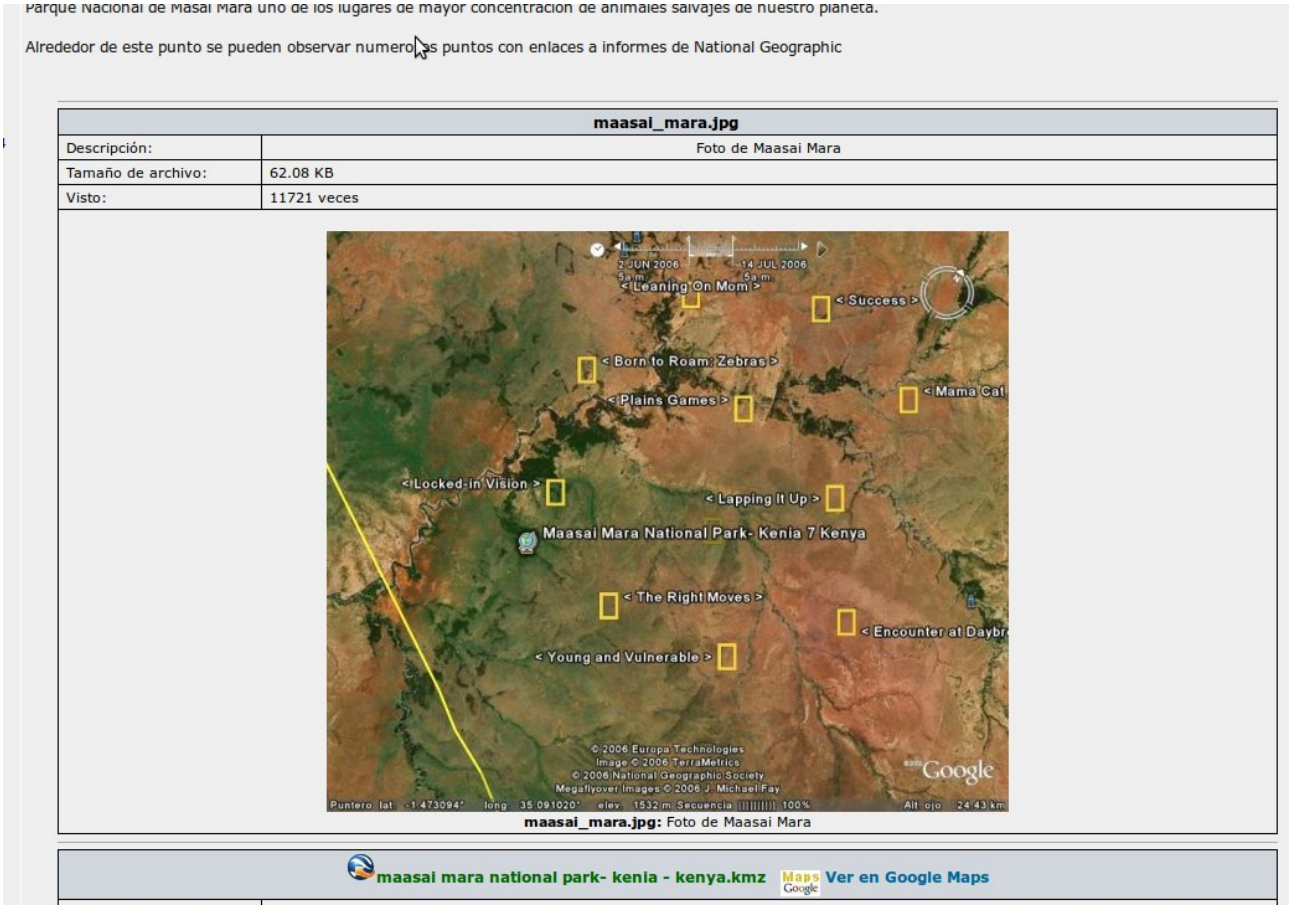
Google Sitios: Muestra lugares marcados por la persona que lo usa, muestra restaurantes, bares etc de la zona

Family Locator : Localizador de móviles familiares a tiempo real.

En esta página puedes encontrar mas aplicaciones para móviles de este tipo.
`https://play.google.com/store/search?q=maps&c=apps`

Lo que son estructuras gráficas, existen muchos enlaces en los que te muestran ejemplos, en foros, webs, tutoriales. El principal es este.

http://www.google-earth.es/foros.php?f=29
http://www.google-earth.es/foros.php?t=397



También he encontrado páginas donde se utiliza mucho el tema de la representación sobre mapas para mostrar datos políticos, cálculos de personas en manifestaciones, intención de voto, etc y para temas de agricultura. Alguna de esta páginas son:

http://www.scoop.it/t/los-mapas-del-15m/?tag=representacion
http://www.ub.edu/geocrit/aracne/aracne-169.htm
http://rtdibermatica.com/?tag=mineria-de-datos
http://imsturex.unex.es/ribera/index.php/proyecto

El programa Matlab nos da la posibilidad utilizar el Toolbox de Google Earth.(7).

Se trata de un conjunto de funciones de MATLAB con el que se puede visualizar espacialmente y temporalmente los datos distribuidos dentro de Google Earth. En conjunto, estas funciones se conocen como la Caja de herramientas de Google Earth. Se puede discutir que la conversión de los resultados de los modelos y las mediciones para poder verlos en Google Earth es innecesaria, ya que también se puede ver en MATLAB por sí mismos. Sin embargo, usando Google Earth para la evaluación de los datos puede tener algunas ventajas importantes, algunas de las cuales se enumeran a continuación:

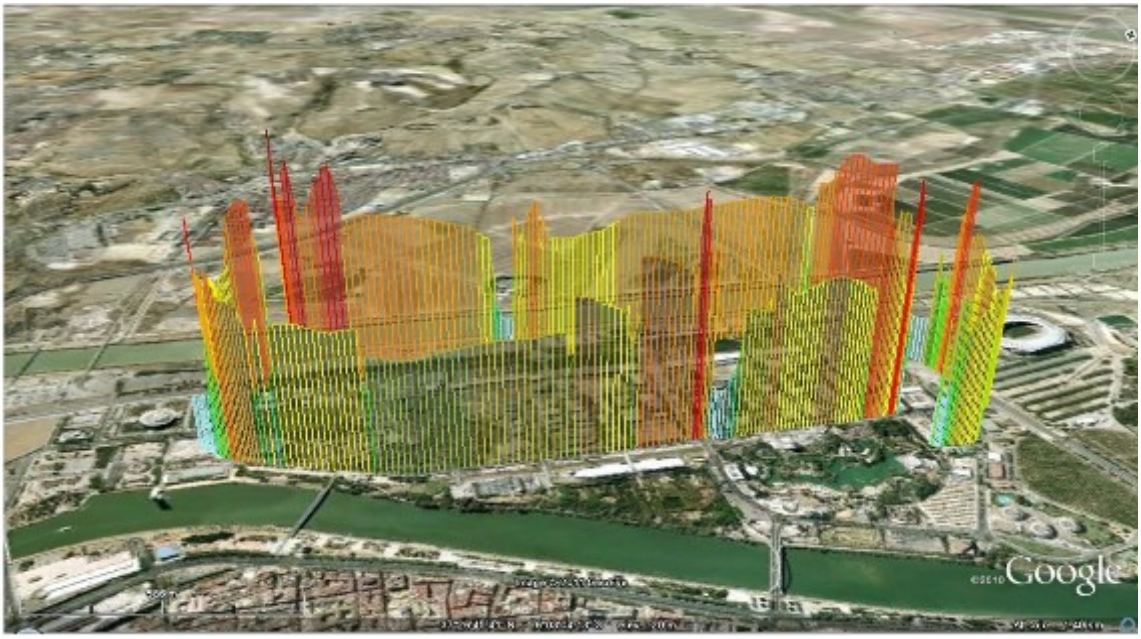
• Herramientas muy intuitivas para navegar de manera visual.

• Recuperación de datos de objetos en el visor.

• Reproducción rápida de archivos de gran tamaño.

•Elección de forma interactiva los objetos que se debe mostrar.

• Representación dinámica de datos variables - tiempo.



Fases del Proyecto

Las principales fases del proyecto son la recolección de datos, la representación gráfica y por último el análisis de esos datos.

La recolección de datos consta de dos partes, la primera parte es la recogida por parte de los sensores y la segunda, la creación de los ficheros de información, que se almacenarán en un servidor al que posteriormente se accederá para su análisis.

A la hora de crear el fichero hay que tener en cuenta varios factores. El primero es que debe tener una estructura adecuada, ya que será clave a la hora de la implementación. El fichero debe seguir un patrón uniforme que sea reconocido por todas las funciones del proyecto, de lo contrario sería imposible. El tipo de estructura que he utilizado esta basado en etiquetas, ya sea php, xml, etc... Otro factor es que de cada punto que nos envían, nos debe llegar toda la información necesaria para marcarlo (latitud, longitud) junto con sus datos representativos y la fecha en la que se recogió.

Para representar los datos hay que crear una serie de programas y funciones que los recojan y creen estructuras con ellos.

Por último lo que haremos será reconocer los datos y junto con los gráficos analizar si son valores representativos y si nos servirán.

Opciones de desarrollo

A la hora de representar los datos gráficamente se me sugirieron dos posibles herramientas. La utilización del lenguaje KML(2), que está basado en XML, junto con la aplicación Google Earth o la unión del lenguaje Javascript junto con la herramienta de Google Maps.

KML

La gran flexibilidad de Google Earth se deriva de su uso de archivos de texto basados en XML, conocido como archivos KML. Este tipode archivo por lo general contiene una serie de objetos, tales como polígonos, líneas y puntos. Cada uno de estos objetos se representa dentro de un archivo KML por sus etiquetas KML. Por ejemplo, podemos encontrar un objeto polígono:

```
<Polygon>
...
</Polygon>
```

O una línea de objetos:

```
<LinearRing>
...
</ LinearRing>
```

Propiedades tales como line width, coordinates, y polygon color se pueden especificar con el uso de etiquetas adicionales. A una línea se le puede asignar un estilo de línea de la siguiente manera:

```
<LineStyle>
<color> ffffffff </ color>
<width> 1,00 </ width>
</ LineStyle>
```

Lo que representa una opacidad total, la línea blanca del ancho de 1.

Por supuesto, la escritura de etiquetas KML manualmente en un editor de texto no es práctico cuando se trabaja con conjuntos de datos que contiene más de unos pocos objetos.

Para hacer mis ficheros de KML realicé algunas funciones en java para recoger los datos de las coordenadas y representar lo que quería. El resultado de esas funciones me construían los ficheros KML que después se verán con Google Earth. Por ejemplo:

```
  public  void generaUnCirculoKML(Double x, Double y,String color, double radio,int escala ,File datosKml){
try{
    String radioString=String.valueOf(radio);
    String punto="punto";
    String titulo="titulo";
    String nombre="nombre";
    int visibilidad=1;

    BufferedWriter escritura = new BufferedWriter(new FileWriter(datosKml));
    escritura.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
    escritura.flush();
    escritura.write("<kml xmlns=\"http://www.opengis.net/kml/2.2\">\n");
    escritura.flush();
    escritura.write("\t<Folder>\n");
    escritura.flush();
    escritura.write("\t\t<name>"+titulo+"</name>\n");
    escritura.flush();
    escritura.write("\t\t<visibility>"+visibilidad+"</visibility>\n");
    escritura.flush();
    escritura.write("\t\t<Placemark>\n");
    escritura.flush();
    escritura.write("\t\t\t<name>"+nombre+"</name>\n");
    escritura.flush();
    escritura.write("\t\t\t<visibility>"+visibilidad+"</visibility>\n");
    escritura.flush();
    escritura.write("\t\t\t<Style>\n");
    escritura.flush();
    escritura.write("\t\t\t\t<geomColor>"+color+"</geomColor>\n");
    escritura.flush();
    escritura.write("\t\t\t\t<geomScale>"+escala+"</geomScale>\n");
    escritura.flush();
    escritura.write("\t\t\t</Style>\n");
```

```
    escritura.flush();
    escritura.write("\t\t\t<LineString>\n");
    escritura.flush();
    escritura.write("\t\t\t\t<coordinates>\n");
    escritura.flush();

    Datos datos = new Datos();
    Double puntos[][]=new Double[5000][2];
    puntos=datos.generaCirculo(x,y,radio);
    int i=36;
    for (int e=0;e<=i;e++){
    if (puntos[e][0]!=null){
            escritura.write("\t\t"+puntos[e][0]+","+puntos[e][1]+",0\n");
              escritura.flush();

    }
    }
    escritura.write("\t\t\t\t</coordinates>\n");
    escritura.flush();
    escritura.write("\t\t\t</LineString>\n");
    escritura.flush();
    escritura.write("\t\t</Placemark>\n");
    escritura.flush();
    escritura.write("\t</Folder>\n");
    escritura.flush();
    escritura.write("</kml>\n");
    escritura.flush();
    escritura.close();
}
catch(Exception e){
System.out.println("Error "+e);
e.printStackTrace();}
  }
```

Esta función es solo para crear un círculo en una coordenada dada. Y su resultado sería este fichero KML:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2"
xmlns:gx="http://www.google.com/kml/ext/2.2"
xmlns:kml="http://www.opengis.net/kml/2.2"
xmlns:atom="http://www.w3.org/2005/Atom">
<Folder>
 <name>Untitled</name>
 <visibility>10.000</visibility>
 <Placemark>
 <name>Untitled</name>
 <visibility>0</visibility>
 <Snippet maxLines="0"></Snippet>
 <description>POI</description>
 <styleUrl>http://www.metzgerwillard.us/rings/styles.kml#center</styleUrl>
```

<Style>
<IconStyle>
<color>ffffff00</color>
</IconStyle>
</Style>
<Point>
<coordinates>-1.674416397,42.805605127,0</coordinates>
</Point>
</Placemark>
<Placemark>
<name>0.25 km</name>
<visibility>0</visibility>
<Snippet maxLines="0"></Snippet>
<description>Untitled</description>
<StyleMap>
<Pair>
<key>normal</key>
<Style>
<BalloonStyle>
<text><![CDATA[<font style="font-size: 11px;">
$[name]<p>$[description]]]></text>
<textColor>ff000000</textColor>
<displayMode>default</displayMode>
</BalloonStyle>
<LineStyle>

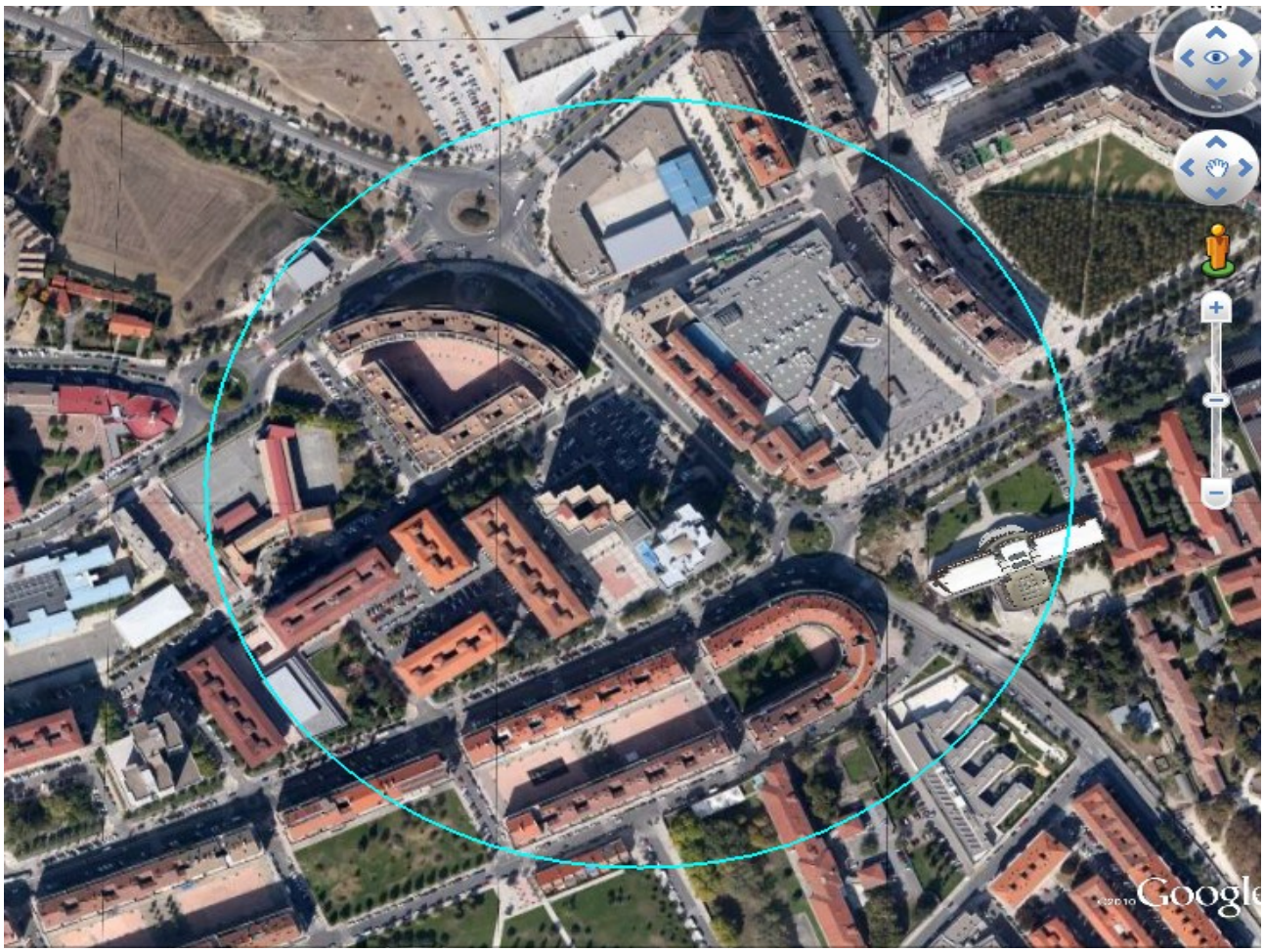
<color>ffffff00</color>
<width>2</width>
</LineStyle>
<PolyStyle>
<color>090000aa</color>
</PolyStyle>
</Style>
</Pair>
<Pair>
<key>highlight</key>
<Style>
<BalloonStyle>
<text><![CDATA[<font style="font-size: 11px;">
$[name]<p>$[description]]]></text>
<textColor>ff000000</textColor>

<displayMode>default</displayMode>
</BalloonStyle>
<LineStyle>
<color>ffffff00</color>
<width>3</width>
</LineStyle>
<PolyStyle>
<color>090000aa</color>
</PolyStyle>
</Style>
</Pair>
</StyleMap>
<LineString>
<tessellate>1</tessellate>
<coordinates>
-1.67441639699999,42.8078555754125,0
-1.67415001100637,42.8078470114687,0 -1.67388565258999,42.807821384821,0
-1.67362533389067,42.8077788905245,0
-1.67337103629005,42.8077198520203,0 -1.67312469532664,42.8076447186729,0
-1.67288818595931,42.8075540623492,0
-1.67266330829347,42.8074485730641,0
-1.67245177387668,42.8073290537274,0 -1.67225519267061,42.8071964140307,0
-1.67207506079585,42.8070516635221,0 -1.67191274914579,42.80689590392,0
-1.67176949295299,42.8067303207264,0 -1.6716463823907,42.806556174202,0
-1.67154435427842,42.8063747897725,0 -1.67146418495626,42.8061875479396,0
-1.67140648438103,42.8059958737727,0 -1.67137169148974,42.8058012260616,0
-1.67136007086472,42.8056050862137,0 -1.67137171072705,42.8054089469788,0
-1.67140652227126,42.8052143010878,0 -1.67146424034799,42.8050226298927,0
-1.67154442548865,42.8048353920933,0
-1.67164646725564,42.8046540126363,0 -1.6717695888942,42.8044798718722,0
-1.67191285324814,42.8043142950519,0 -1.67207516989621,42.8041585422424,0
-1.67225530345396,42.8040137987391,0 -1.67245188297704,42.8038811660479,0
-1.67266341239583,42.8037616535037,0 -1.67288828190062,42.8036561705919,0
-1.67312478019157,42.8035655200286,0 -1.67337110750017,42.8034903916537,0
-1.67362538928229,42.8034313571829,0
-1.67388569048012,42.8033888658583,0
-1.67415003024369,42.8033632410307,0
-1.67441639699999,42.8033546776998,0 -1.6746827637563,42.8033632410307,0
-1.67494710351987,42.8033888658583,0 -1.6752074047177,42.8034313571829,0
-1.67546168649982,42.8034903916537,0 -1.67570801380841,42.8035655200286,0
-1.67594451209947,42.8036561705919,0 -1.67616938160416,42.8037616535037,0

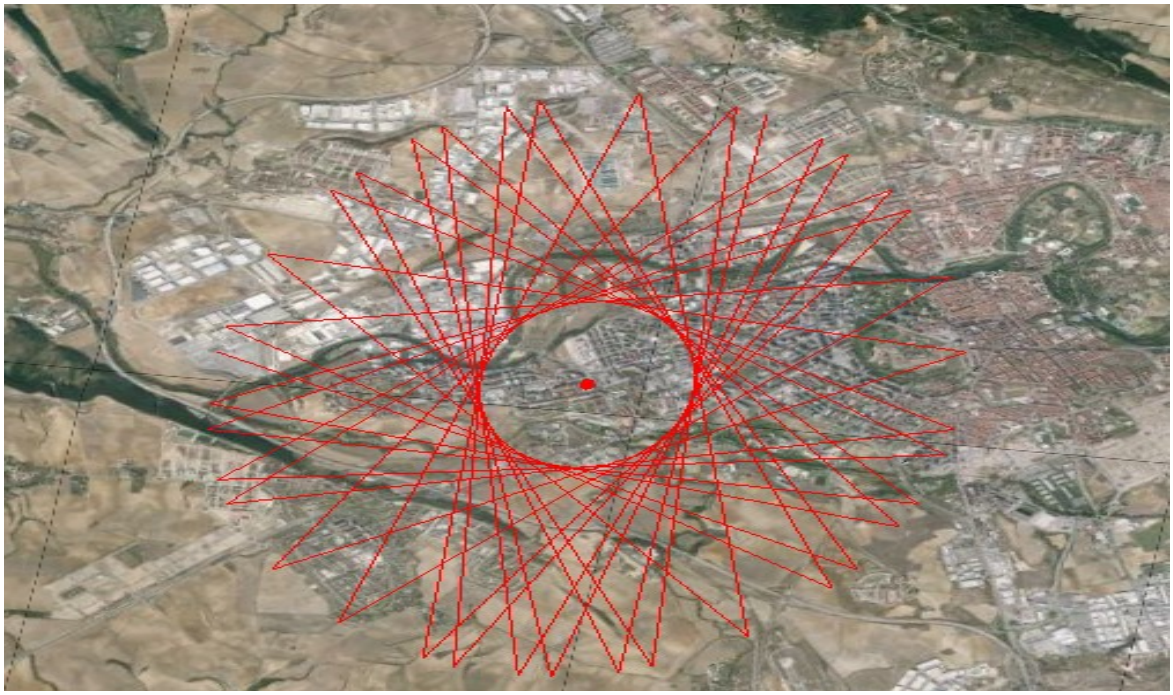
-1.67638091102295,42.8038811660479,0 -1.67657749054603,42.8040137987391,0
-1.67675762410378,42.8041585422424,0 -1.67691994075195,42.8043142950519,0
-1.67706320510579,42.8044798718722,0 -1.67718632674435,42.8046540126362,0
-1.67728836851144,42.8048353920933,0 -1.677368553652,42.8050226298927,0
-1.67742627172883,42.8052143010878,0 -1.67746108327294,42.8054089469788,0
-1.67747272313526,42.8056050862137,0 -1.67746110251025,42.8058012260616,0
-1.67742630961896,42.8059958737727,0
-1.67736860904373,42.8061875479396,0
-1.67728843972157,42.8063747897725,0 -1.67718641160929,42.806556174202,0
-1.67706330104699,42.8067303207264,0 -1.6769200448542,42.80689590392,0
-1.67675773320413,42.8070516635221,0 -1.67657760132938,42.8071964140307,0
-1.67638102012331,42.8073290537274,0 -1.67616948570651,42.8074485730641,0
-1.67594460804067,42.8075540623492,0
-1.67570809867335,42.8076447186729,0
-1.67546175770994,42.8077198520203,0
-1.67520746010942,42.8077788905245,0 -1.67494714141,42.807821384821,0
-1.67468278299372,42.8078470114687,0 -1.67441639699999,42.8078555754125,0

</coordinates>
</LineString>
</Placemark>
</Folder>
</kml>

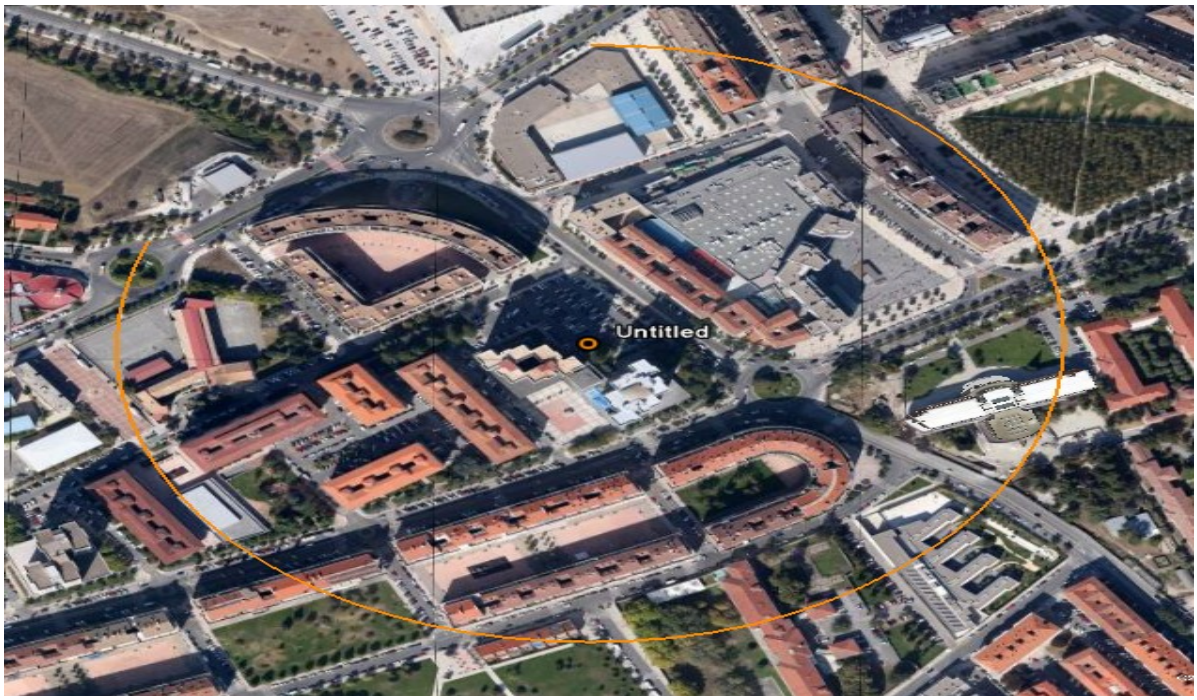
Y su representación en Google Earth sería:



Aquí se puede ver un ejemplo de dos figuras mal ajustadas al hacer una circunferencia.

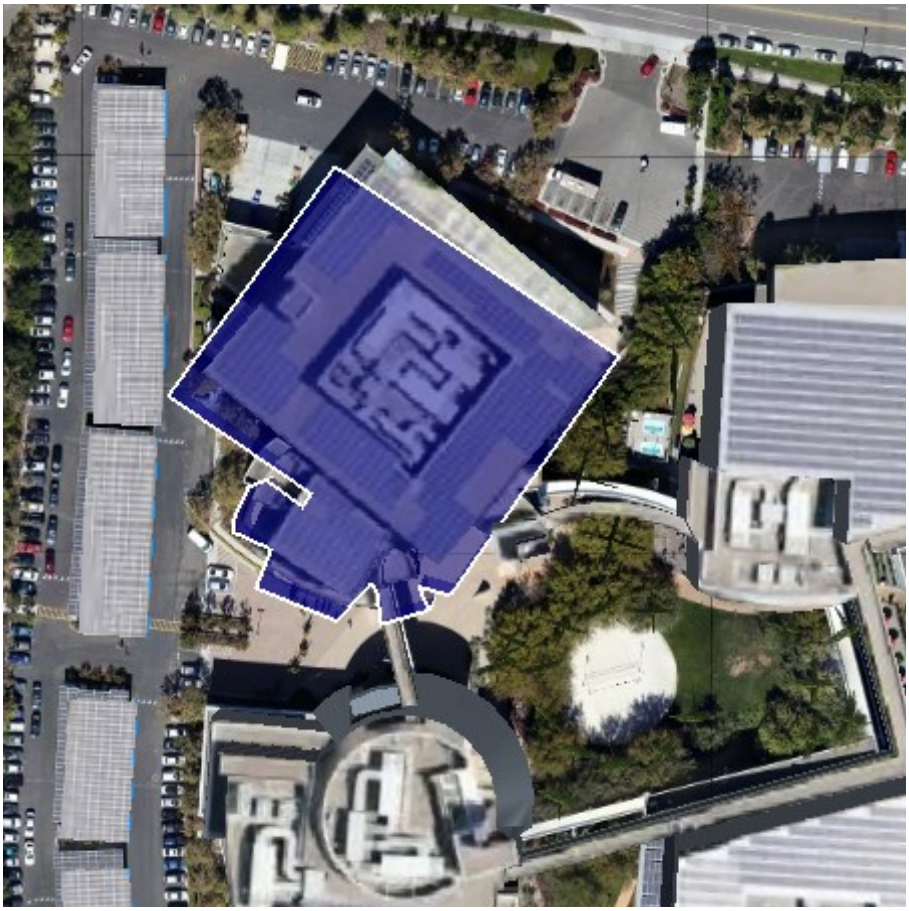


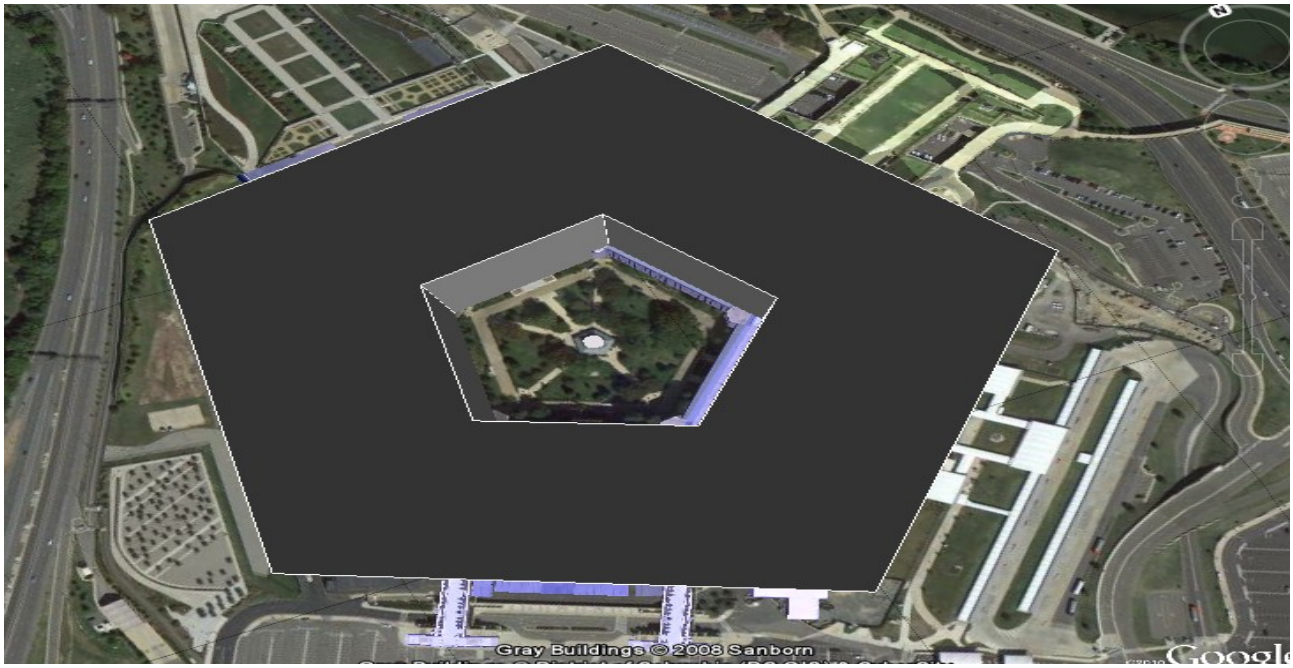
Nos puede interesar hacer figuras de este tipo también.



Por ejemplo, para seguir el trayecto que ha recorrido el vehículo, ponemos una marca sobre cada posición, podemos representarlo de manera que vayan apareciendo en orden de lectura, como si fuese una simulación. Podemos cambiar los tipos de marca e incluso asignarle la foto que nosotros queramos.

Ejemplo de como dibujar un área o una capa que hace efecto de de.





JavaScript

Vamos a explicar las características del lenguaje, realizando primero un recorrido por su historia y ver su creación y las diferencias y similitudes con java ya que puede llegar a confundirse y pensar que JavaScript es igual que java solo porque empiece por java.

JavaScript fue desarrollado originalmente por Brendan Eich de Netscape con el nombre de Mocha, el cual fue renombrado posteriormente a LiveScript, para finalmente quedar como JavaScript. El cambio de nombre coincidió aproximadamente con el momento en que Netscape agregó soporte para la tecnología Java en su navegador web Netscape Navigator en la versión 2.002 en diciembre de 1995. La denominación produjo confusión, dando la impresión de que el lenguaje es una prolongación de Java, y se ha caracterizado por muchos como una estrategia de mercadotecnia de Netscape para obtener prestigio e innovar en lo que eran los nuevos lenguajes de programación web.

«JavaScript» es una marca registrada de Oracle Corporation. Es usada con licencia por los productos creados por Netscape Communications y entidades actuales como la Fundación Mozilla.Microsoft dio como nombre a su dialecto de JavaScript «JScript», para evitar problemas relacionados con la marca. JScript fue adoptado en la versión 3.0 de Internet Explorer, liberado en agosto de 1996, e incluyó compatibilidad con el Efecto 2000 con las funciones de fecha, una diferencia de los que se basaban en ese momento. Los dialectos pueden parecer tan similares que los términos «JavaScript» y «JScript» a menudo se utilizan indistintamente, pero la especificación de JScript es incompatible con la de ECMA en muchos aspectos. Para evitar estas incompatibilidades, el World Wide Web Consortium diseñó el estándar Document Object Model (DOM, o Modelo de Objetos del Documento en español), que incorporan Konqueror, las versiones 6 de Internet Explorer y Netscape Navigator, Opera la versión 7, Mozilla Application Suite y Mozilla Firefox desde su primera versión.

En 1997 los autores propusieron8 JavaScript para que fuera adoptado como estándar de la European Computer Manufacturers 'Association ECMA, que a pesar de su nombre no es europeo sino internacional, con sede en Ginebra. En junio de 1997 fue adoptado como un estándar ECMA, con el nombre de ECMAScript. Poco después también como un estándar ISO.

Netscape introdujo una implementación de script del lado del servidor con Netscape Enterprise Server, lanzada en diciembre de 1994 (poco después del lanzamiento de JavaScript para navegadores web). A partir de mediados de la década de los 2000, ha habido una proliferación de implementaciones de JavaScript para el lado servidor. Node.js es uno de los notables ejemplos de JavaScript en el lado del servidor, siendo usado en proyectos importantes.

JavaScript se ha convertido en uno de los lenguajes de programación más populares en Internet. Al principio, sin embargo, muchos desarrolladores renegaban el lenguaje porque el público al que va dirigido lo formaban publicadores de artículos y demás aficionados, entre otras razones. La llegada de Ajax devolvió JavaScript a la fama y atrajo la atención de muchos otros programadores. Como resultado de esto hubo una proliferación de un conjunto de frameworks y librerías de ámbito general, mejorando las prácticas de programación con JavaScript, y aumentado el uso de JavaScript fuera de los navegadores web, como se ha visto con la proliferación de entornos JavaScript del lado del servidor. En enero de 2009, el proyecto CommonJS fue inaugurado con el objetivo de especificar una librería para uso de tareas comunes principalmente para el desarrollo fuera del navegador web.

Características de JavaScript:

JavaScript soporta gran parte de la estructura de programación de C (por ejemplo, sentencias if, bucles for, sentencias switch, etc.). Con una salvedad, en parte: en C, el ámbito de las variables alcanza al bloque en el cual fueron definidas; sin embargo en JavaScript esto no es soportado, puesto que el ámbito de las variables es el de la función en la cual fueron declaradas. Esto cambia con la versión de JavaScript 1.7, ya que soporta block scoping por medio de la palabra clave let. Como en C, JavaScript hace distinción entre expresiones y sentencias. Una diferencia sintáctica con respecto a C es la inserción automática de punto y coma, es decir, en JavaScript los puntos y coma que finalizan una sentencia pueden ser omitidos Tipado dinámico Como en la mayoría de lenguajes de scripting, el tipo está asociado al valor, no a la variable. Por ejemplo, una variable x en un momento dado puede estar ligada a un número y más adelante, religada a una cadena. JavaScript soporta varias formas de comprobar el tipo de un objeto, incluyendo duck typing. Una forma de saberlo es por medio de la palabra clave typeof.

JavaScript esta formado casi en su totalidad por objetos. Los objetos en JavaScript son arrays asociativos, mejorados con la inclusión de prototipos (ver más adelante). Los nombres de las propiedades de los objetos son claves de tipo cadena: obj.x = 10 y obj['x'] = 10 son equivalentes, siendo la notación con punto azúcar sintáctico. Las propiedades y sus valores pueden ser creados, cambiados o eliminados en tiempo de ejecución. La mayoría de propiedades de un objeto (y aquellas que son incluidas por la cadena de la herencia prototípica) pueden ser enumeradas a por medio de la instrucción de bucle for... in. JavaScript tiene un pequeño número de objetos predefinidos como son Function y Date.

JavaScript incluye la función eval que permite evaluar expresiones como expresadas como cadenas en tiempo de ejecución. Por ello se recomienda que eval sea utilizado con precaución y que se opte por utilizar la función JSON.parse() en la medida de lo posible, pues resulta mucho más segura.

JavaScript usa prototipos en vez de clases para el uso de herencia.Es posible llegar a emular muchas de las características que proporcionan las clases en lenguajes orientados a objetos tradicionales por medio de prototipos en JavaScript.

Las funciones también se comportan como constructores. Prefijar una llamada a la función con la palabra clave new crear una nueva instancia de un prototipo, que heredan propiedades y métodos del constructor (incluidas las propiedades del prototipo de Object).ECMAScript 5 ofrece el método Object.create, permitiendo la creación explícita de una instancia sin tener que heredar automáticamente del prototipo de Object (en entornos antiguos puede aparecer el prototipo del objeto creado como null).La propiedad prototype del constructor determina el objeto usado para el prototipo interno de los Nuevos objetos creados. Se pueden añadir nuevos métodos modificando el prototipo del objeto usado como constructor. Constructores predefinidos en JavaScript, como Array u Object, también tienen prototipos que pueden ser modificados. Aunque esto sea posible se considera una mala práctica modificar el prototipo de Object ya que la mayoría de los objetos en JavaScript heredan los métodos y propiedades del objeto prototype, objetos los cuales pueden esperar que estos no hayan sido modificados.

Diferencias con Java:

Compilador: Para programar en Java necesitamos un Kit de desarrollo y un compilador. Sin embargo, Javascript no es un lenguaje que necesite que sus programas se compilen, sino que éstos se interpretan por parte del navegador cuando éste lee la página. Orientado a objetos: Java es un lenguaje de programación orientado a objetos. (Más tarde veremos que quiere decir orientado a objetos, para el que no lo sepa todavía) JavaScript no es orientado a objetos, esto quiere decir que podremos programar sin necesidad de crear clases, tal como se realiza en los lenguajes de programación estructurada como C o Pascal.

Propósito: Java es mucho más potente que JavaScript, esto es debido a que Java es un lenguaje de propósito general, con el que se pueden hacer aplicaciones de lo más variado, sin embargo, con JavaScript sólo podemos escribir programas para que se ejecuten en páginas web.

Estructuras fuertes: Java es un lenguaje de programación fuertemente tipado, esto quiere decir que al declarar una variable tendremos que indicar su tipo y no podrá cambiar de un tipo a otro automáticamente. Por su parte JavaScript no tiene esta característica, y podemos meter en una variable la información que deseemos, independientemente del tipo de ésta. Además, podremos cambiar el tipo de información de una variable cuando queramos.

Otras características: Como vemos Java es mucho más complejo, aunque también más potente, robusto y seguro. Tiene más funcionalidades que JavaScript y las diferencias que los separan son lo suficientemente importantes como para distinguirlos fácilmente.

La aplicación de Google Maps V3, nos da la opción de utilizar funciones que ya tiene creadas por defecto y nos hace ahorrar tiempo a la hora de crear objetos. (https://developers.google.com/maps/documentation/javascript/tutorial)

Algún ejemplo:

```
function crear_map(){
try{
	var pam=new google.maps.LatLng(42.8124952, -1.649024000000054 );
	var myOptions = { zoom: 13, center:pam, mapTypeId: google.maps.MapTypeId.SATELLITE };
	var map = new google.maps.Map( document.getElementById("map_canvas"), myOptions);
	return(map);
	}
catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
  }
}
```

Esta función la utilizo para poder crear el mapa,con las características necesarias, que quiero incrustar en la web. En este caso creo una variable con un punto latitud, longitud de Pamplona.

Como se ve podemos capturar excepciones al igual que en java.

```
function mi_matriz(p,s) {
	matriz = new Array(p);
	for (i = 0; i < p; i++) {
		matriz[i] = new Array(s)
		for (j = 0; j < s; j++) {
			matriz[i][j] = "["+i+","+j+"]"
		}
	}
}
```

Esta otra función crea una tabla doble, ya que si se hace con el constructor **new Array()** da problemas.

A continuación se ve un ejemplo de como podemos optimizar nuestro código si utilizamos las funciones del API Google Maps v3. Por ejemplo para crear un círculo y que se muestre en el mapa. En principio desarrollé mi propia función que con la entrada de una latitud, longitud (el centro de la circunferencia) y el radio (que no eran metros sino grados de separación con respecto al centro) me creaba una tabla con X puntos que yo quisiera y que pertenecían a la circunferencia. Dicha función es la siguiente:

```
function esquinas(corx,cory,radio){
            var i=0;
            var x=0.0;
            var y=0.0;
            var alpha=0;
            var tabla =new Array();
            while (alpha<360) {
                x=radio*Math.cos(alpha)+corx;
                y=radio*Math.sin(alpha)+cory;
                tabla[i]=new google.maps.LatLng(x,y);
                alpha=alpha+0.1;
                i++;
            }
            return (tabla);
}
```

Para poder crear dicho círculo se llama a la función y se crea el polígono con sus características. Lo que nos queda:

```
var esquina=esquinas(corx,cory,0.0005);
```

```
poligono = new google.maps.Polygon({paths: esquina, strokeColor:    "#FF0000",
                                    strokeOpacity:  0.8,
                                    strokeWeight:   2,
                                    fillColor:      "#FF0000",
                                    fillOpacity:    0.4   });
poligono.setMap(map);
```

Sin embargo con la función de Google Maps v3 se nos reduce a esto:

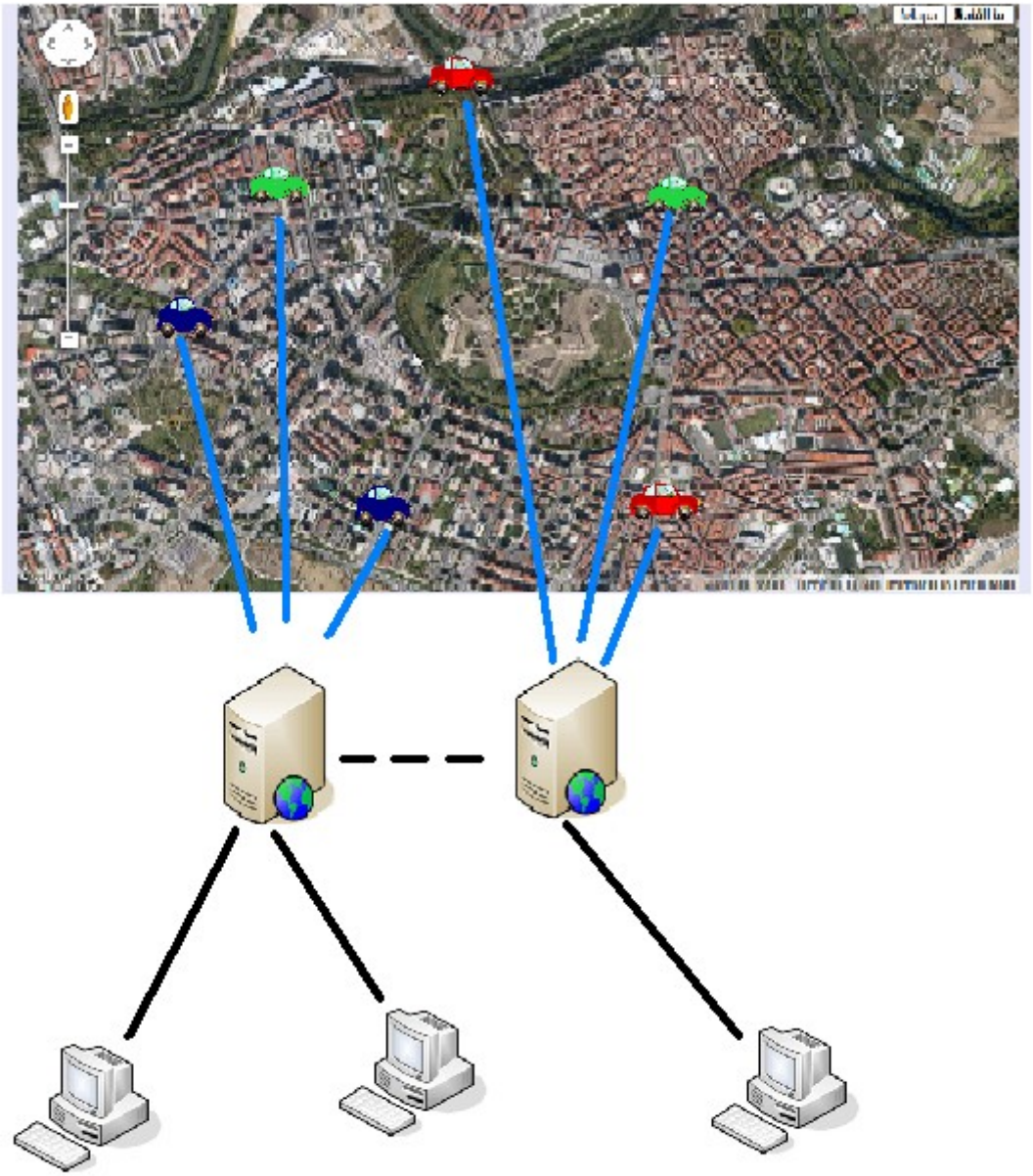
Ambos

```
var centro=new google.maps.LatLng(corx,cory);
var opciones={strokeColor: color,strokeOpacity: 0.8,
      strokeWeight: 2,fillColor: color,fillOpacity: 0.35 ,
      map: map,
      center:centro,
      radius: 500};
      Circle = new google.maps.Circle(opciones);
      Circle.setMap(map);
```

procedimientos daban con un resultado similar a esto:

Para la elección de que herramienta utilizar en un principio creí conveniente utilizar KML con Google Earth porque me parecía mas atractivo a la hora de su visualización, pero más adelante me di cuenta de que JavaScript y Maps es mas sencillo, te ofrece mas posibilidades en mi opinión y no dependes de la aplicación de Google Earth. A la hora de recoger los datos de las posiciones tenía que hacer un programa en java para crear el archivo KML que luego solo se podía ver abriendo Google Earth y que además si intentaba incrustar los diferentes ejemplos de KML en una web solo se podía hacer de forma estática. Y lo interesante es poder interactuar con la aplicación.

ARQUITECTURA DEL SISTEMA



Elementos del sistema

El sistema esta compuesto por unos sensores adaptados para vehículos, un servidor o servidores y los terminales de los usuarios que accedan al programa de forma local o remota.

Los sensores son los encargados de recoger los datos de las posiciones y enviarlos a los servidores.

Nuestros servidores son los encargados de recopilar los datos y generar el fichero con sus posiciones.

Desde el terminal nos conectaremos al servidor para poder acceder al fichero de datos del que sacaremos la información.

REQUISITOS

¿Qué es lo que se nos pide?

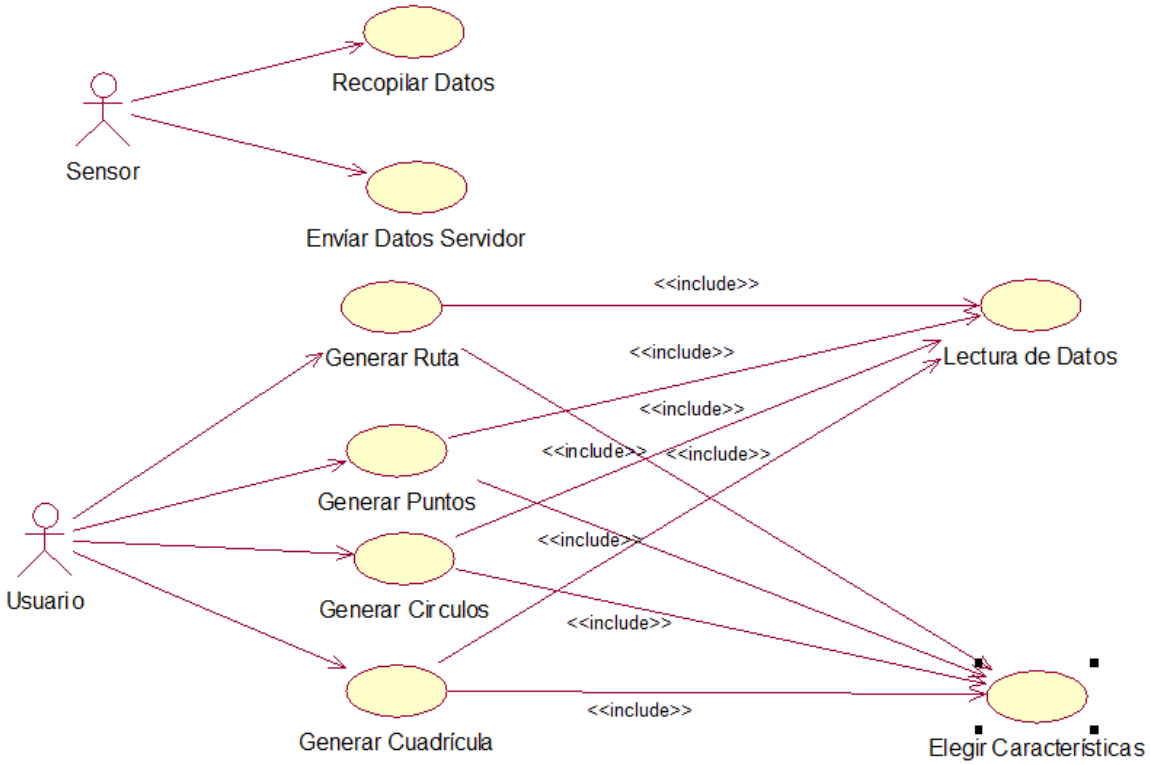
En esta fase del desarrollo nos vamos a centrar en establecer las ideas básicas que vamos a necesitar para cumplir con los objetivos de nuestro proyecto. Esto necesitará de una búsqueda de información que facilite la obtención de l objetivo planteado
.

El primer paso consiste en tener una idea clara de lo que se quiere conseguir. Esta fue tomando forma mediante una serie de reuniones con el tutor del proyecto en las que se fueron proponiendo ideas de forma conjunta hasta obtener una idea global, que ha sido descrita en el apartado de Objetivos.

En un principio el objetivo del proyecto era estudiar las distintas posibilidades que ofrecía la herramienta Google Earth a la hora de representar datos geográficos. El uso del lenguaje KML y su posibilidad de adaptarlo a HTML. El objetivo final fue cambiando a medida que me iba informando sobre el tema.

Una vez estudiado las posibilidades, la representación gráfica de datos sobre un mapa es otra cosa necesaria para cumplir el objetivo principal.

El diagrama de casos de uso quedaría de la siguiente forma.



DESARROLLO E IMPLEMENTACIÓN

Etapas del Desarrollo

Como el tema del proyecto trataba de ficheros KML tuve que informarme sobre qué tipo de lenguaje es , como se construyen, etc. Para ello había que entender de base como funciona. Al estar basado en XML fue lo primero de lo busqué información.

Lo que saqué en claro es que XML es un tipo de lenguaje que te permite describir información de forma estructurada con la utilización de etiquetas. Esa información te sirve como base de datos a la que podemos acceder de manera sencilla desde nuestro navegador para páginas web o para diferentes tipos de aplicaciones que puedan utilizar este lenguaje.

Al ser KML un tipo de archivo XML se entiende que el funcionamiento de Google Earth consiste en reconocer esta información y representarla. Así que el siguiente paso es conocer la estructura de KML para poder ser capaz de construir un archivo desde cero y que represente algo que yo quiera.

Como ya he dicho antes
<Kml xmlns="http://www.opengis.net/kml/2.2"> esta es la etiqueta principal de KML y a partir de ahí solo hay que construir
las diferentes estructuras, con las etiquetas que están definidas por defecto.

El primer “problema serio”

La construcción manual de estos ficheros no tiene ningún sentido ya que el trabajo para realizarlo te lo ahorras con programas para poder editarlos. Realicé estos programas en java y consistían en recoger esos datos y estructurarlos en KML. A medida que desarrollaba diferentes ejemplos y programas intenté acoplarlos a un HTML muy sencillo. Esto no resultó, o hasta el punto que yo llegue no fui capaz de realizarlo. Solo era capaz de incrustar elementos estáticos que creaba con una herramienta de Google. Esta herramienta te permite crear las imágenes que tu quieras sobre el mapa y a partir de eso te da el código que tienes que poner en tu HTML. Este fue mi primer “problema serio”.

Gadget Embed KML

Solución adoptada

Al ver que me quedaba estancado en este tema hablé con mi tutor para que me ayudara, me propuso que hablara con un ex alumno que trabajó sobre el tema a ver si él me podía guiar. Cuando por fin nos reunimos, me habló sobre lo que él había hecho con KML y me mostró alguno de sus ejemplos. Todo lo que me mostró tenía la misma idea que yo había desarrollado. Ficheros KML que ejecutados independientemente abren la aplicación Google Earth y te muestra el contenido gráfico de esa ejecución. Hasta ese punto no veía una solución al problema de que las figuras que se representen sean dinámicas. Sin embargo, me hablo sobre la posibilidad de usar el lenguaje JavaScript y en vez de Google Earth, Google Maps. Me enseñó varios ejemplos y me pareció la solución perfecta para la idea que yo tenía sobre el proyecto.

Nunca había trabajado con JavaScript, pero es un lenguaje muy parecido a java. En mi opinión es incluso mas sencillo y te da bastantes posibilidades para realizar funciones, tratamiento de datos y demás, muy útiles para mi proyecto.

Ahora ya que ya tengo la herramienta que quiero lo que tengo que hacer es implementar las distintas funciones que ya tenia para KML y pasarlas a JavaScript para que el código Html pudiera representarlo.

Antes de nada hay que explicar como son los tipos de datos y en que forma están para poder comprender las funciones y métodos del programa. La estructura es la siguiente:

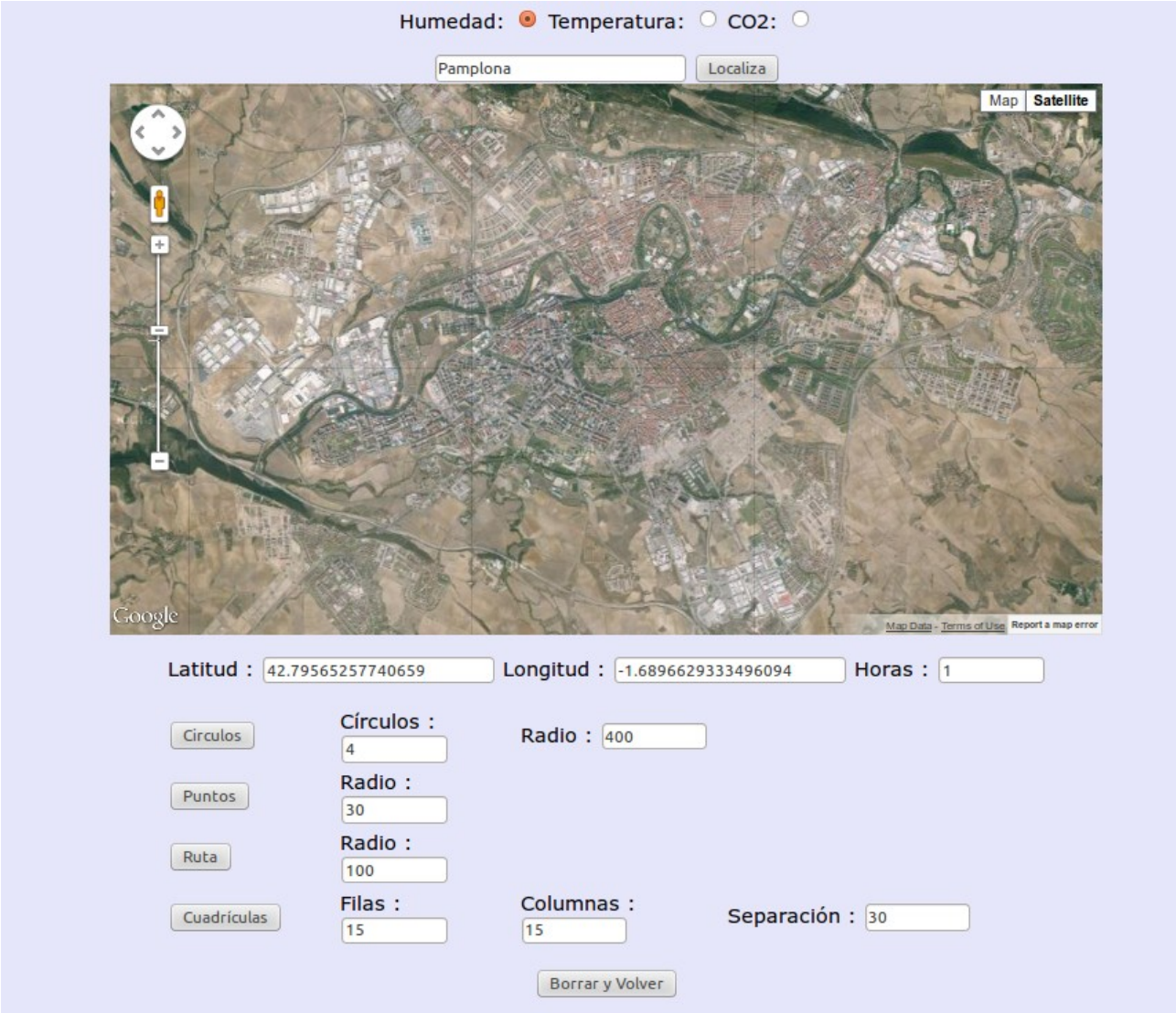
<Posicion><Latitud>42.8116</Latitud><Longitude>-1.61275</Longitude><Humedad>62</Humedad><Temperatura>31</Temperatura><Polucion>13</Polucion><Date>2013-03-15 11:59:59</Date></Posicion>

Se define la etiqueda principal de cada registro como **<Posicion>,** a partir de ahí ya podemos empezar a definir cada tipo de dato.

<Latitude**>**
<Longitude>
<Humedad>
<Temperatura>
<Polucion>
<Date>

Implementación

La página principal del proyecto es el fichero HTML llamado, Final.html, la que da como resultado la página donde se representará los datos.



Como podemos ver la página esta compuesta por el mapa de Google y por varios elementos que nos van a ayudar a describir como queremos que se representen los datos y que vamos a describir a continuación.

Al ser un fichero Html la cabecera que vamos utilizar tienen que ser una etiqueta Html con su descripción de formato y las llamadas a módulos que vamos a utilizar.

```
<html>
        <head>

        </head>
</html>
```

Estas son las etiquetas estándar para empezar a escribir el programa en Html, a partir de aquí ya podemos empezar a añadir los componentes del programa.

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
```

Lo primero que vamos a hacer es definir el tipo de texto que va a interpretar la página final, al poner esto luego no tendremos problemas con acentos o "ñ".

```
<script type="text/javascript"    src="http://maps.google.com/maps/api/js?sensor=false"></script>
<script type="text/javascript" src="http://maps.google.com/maps/api/js?libraries=geometry&sensor=false&language=en"></script>
<script language="javascript"  type="text/javascript" src="./funciones.js"></script>
```

Después tenemos que escribir llamada al modulo de Google Maps para que podamos utilizarlo y hacer referencia a sus funciones, en caso de necesitarlas. Esto es clave para poder utilizarlo, sino sería como llamar a una función externa sin haber declarado de donde se coge. Como se puede observar definimos el tipo de texto como JavaScript y después indicamos la dirección de las librerías.

Para poder utilizar las funciones que he desarrollado hay que hacer una llamada a ese fichero poniéndole el lenguaje utilizado, el tipo de texto y su nombre. En nuestro caso, **"funciones.js"**.

Ahora que ya hemos descrito como va a ser nuestra página lo que nos queda es desarrollar el cuerpo de la página.

```
<body onload="initialize()" bgcolor="e6e6fa" background="images1.jpg" >
      <center>
      <FONT FACE="Verdana" SIZE=4 COLOR="black">

      <form name="formulario0" id="formulario0">
            <tr>
              Humedad: <input type="radio" name="tipo" value="1" >
        </tr>
              <tr>
              Temperatura: <input type="radio" name="tipo" value="2 ">
              </tr>
              <tr>
              CO2: <input type="radio" name="tipo" value="3">
              </tr>
      </form>
              <div id="panel">


              <div id="map_canvas" style='height: 1px solid silver; height: 500px; width: 900px;'></div>


      <p>
      <form name="formulario01" id="formulario01">
          Latitud : <input type="text" name="Latitud"  size="18" value="42.8124952">
          Longitud :    <input type="text" name="Longitud"  size="18" value="-1.649024000000054">
          Horas :<input type="text" name="Horas"  size="5"  value="1">
      </form>
      </p>

      <table width="800" border="0">

  <p>
    <input name="button5" type="button" onClick="map=crear_map();stoper(timer)" value="Borrar y Volver" />
  </p>


      </FONT>

  </center>

</body>
```

La etiqueta encargada de definir el cuerpo es la etiqueta **<body>**, la cual podemos asignarle algunas características como la imagen de fondo o en este caso la función que vamos a cargar al inicializar la página. Esta función la explicaremos mas adelante.

Elementos Principales

El principal elemento de la página es el mapa de Google, para llamar a su función hay que describir la etiqueta que lo haga. Para ello hay que definir su id, “map-canvas” y algunas de sus características como altura y anchura.

```
<div id="map_canvas" style='height: 1px solid silver; height: 500px; width: 900px;'></div>
```

Lo que nos da como resultado la incrustación del mapa en la página.

Otro de los elemento principales del fichero es la función principal "**initialize()"**, en mi caso esta función se encarga de crear el objeto mapa que será al que le pasemos algunas de las funciones necesarias para crear las figuras. Es decir, sirve para unir los objetos que creo con el mapa ya descrito antes. Como podemos observar hay que definir una etiqueta de tipo script y decirle que el tipo que vamos a utilizar es el JavaScript, para que lo reconozca. Esto es común para todo uso de elementos JavaScript en un Html.

```
<script type="text/javascript">
var map;
var timer;
var poligono;
var pos = posicion();
var multipuntos;
var geocoder;
function initialize() {
        map = crear_map();
        geocoder = new google.maps.Geocoder();
}


</script>
```

Para llamar a esta función y asociarla a nuestro fichero Html hay que crear la etiqueta ya mostrada antes, de esta forma:

```
<body onload="initialize()" bgcolor="e6e6fa">
```

Antes de crear la función **initialize()** he definido unas variables que que se utilizarán mas tarde. Esta función tiene dos llamadas , la primera es la que nos interesa en estos momentos, **map=crear_mapa();** y la otra **geocoder = new google.maps.Geocoder();** que se describirá mas tarde.

La función **crear_mapa()** está descrita en el fichero de los métodos, funciones.js y es de esta forma:

```
function crear_map(){
try{
	var pam=new google.maps.LatLng(42.8124952, -1.649024000000054 );
	var myOptions = { zoom: 13, center:pam, mapTypeId: google.maps.MapTypeId.SATELLITE };
	var map 	= new google.maps.Map( document.getElementById("map_canvas"), myOptions);
	google.maps.event.addListener(map, 'click', function(event) {
		document.formulario01.Latitud.value=event.latLng.lat();
		document.formulario01.Longitud.value=event.latLng.lng();

		});
	return(map);
	}

catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
  }

}
```

Para empezar, he creado el try- catch para la captura de errores y que se me muestren por pantalla cuando ejecuto la función.(Esto es común para todas las funciones) En la parte del try, defino tres variables que son las encargadas de crear el objeto mapa.

- **pam**: Tupla de un punto geográfico (Latitud, Longitud) donde se centrará el mapa que creemos y que en este caso hace referencia a un punto de Pamplona. Para ello utilizamos la función predefinida por Google Maps, **LatLng().**

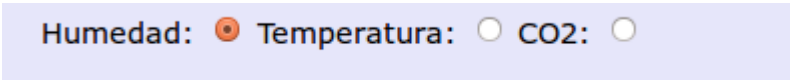
- **myOptions**: Aquí definiremos las características (en forma de array) del mapa que queremos crear. El zoom, punto donde debe apuntar con la variable pam y tipo de mapa.

- **Map:** Es el objeto mapa que queremos crear. Para ello utilizamos la función Map de Google Maps, y le pasamos el elemento "map-canvas" del documento Html y la variable myOptions.

También he generado un evento con la función **addListener()** sobre el mapa creado, que hace salte **function(event)** cuando hacemos un **'click'** (presionar el botón izquierdo del ratón) y que explicaremos mas tarde.

Elementos Secundarios

El primer elemento es un formulario del tipo **radioButton** que sirve para elegir el tipo de elemento que queremos observar. He escogido este tipo porque a parte de que es vistoso y funcional también te obliga que solo uno de estos elementos este seleccionado al mismo tiempo.



Y este es su código:

```
<form name="formulario0" id="formulario0">
      <tr>
        Humedad: <input type="radio" name="tipo" value="1" >
        </tr>
        <tr>
        Temperatura: <input type="radio" name="tipo" value="2 ">
        </tr>
        <tr>
        CO2: <input type="radio" name="tipo" value="3">
        </tr>
</form>
```

Hay que definir una etiqueta form con nombre e identificador, en este caso **“formulario0”** .Para cada radioButton se crea un input con el tipo, nombre y valor asociado a ese botón. Esta es una buena manera de hacerlo para pocos elementos, si fueran muchos yo preferiría crea una lista y de ahí elegir.

Para darle más forma y completar las funcionalidades del mapa he creado un elemento que sirve para introducir el sitio o dirección que queramos y que el mapa nos redirija a ese lugar. Es un modulo creado por Google Maps, se llama Geocoder. Este elemento es así:

Y su código es:

```
<div id="panel">
<input id="address" type="textbox" value="Pamplona">
<input type="button" value="Localiza" onclick="codeAddress()">
</div>
```

Son dos input de tipo textfield y button, que llaman a la función **"codeAddress()"** con el parámetro del textfield. La función que hace referencia a esta llamada esta implementada en el fichero de las funciones y es de esta forma:

```
function codeAddress() {
  try{
  var address = document.getElementById('address').value;
  geocoder.geocode( { 'address': address}, function(results, status) {
    if (status == google.maps.GeocoderStatus.OK) {
      map.setCenter(results[0].geometry.location);
      var marker = new google.maps.Marker({
          map: map,
          position: results[0].geometry.location
      });

        document.formulario01.Latitud.value=results[0].geometry.location.lat();
        document.formulario01.Longitud.value=results[0].geometry.location.lng();

    } else {
      alert('Localizador no se ha ejecutado por esta razón: ' + status);
    }

  });
    }
  catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);

  }
}
```

También tiene el try-catch para capturar errores. Esta función crea un parámetro address con el elemento recogido por el textfield de antes y genera una búsqueda con la función de Google

Maps Geocoder. Si la búsqueda, da resultado positivo genera un nuevo mapa con centro las coordenadas del lugar que hemos buscado. Si no encuentra ese lugar, salta una alerta explicándolo Esa función utiliza el elemento geocoder de la función initialize ya mostrada. A parte, en la función **codeAddress()** hay unos elementos que explicaré mas tarde.

```
function initialize() {
	map = crear_map();
	geocoder = new google.maps.Geocoder();
}
```

El último elemento que me parece de los importantes es el formulario principal que es común a todos. Es decir de aquí todos los botones cogen alguno o todos los datos y además ellos tienen formularios adicionales dependiendo de lo que necesiten.

Estos datos son la latitud, longitud y el rango de horas que queremos tener para saber si un dato o no nos interesa. El código de este formulario queda así:

```
<p>
<form name="formulario01" id="formulario01">
	Latitud : <input type="text" name="Latitud"  size="18" value="42.8124952">
	Longitud :  <input type="text" name="Longitud"  size="18" value="-1.649024000000054">
	Horas : <input type="text" name="Horas"  size="5"  value="1">
</form>
</p>
```

Un formulario normal, al que le he asignado el nombre e identificador **“formmulario01”,** con los tres input de los elementos y sus correspondientes características.

Métodos

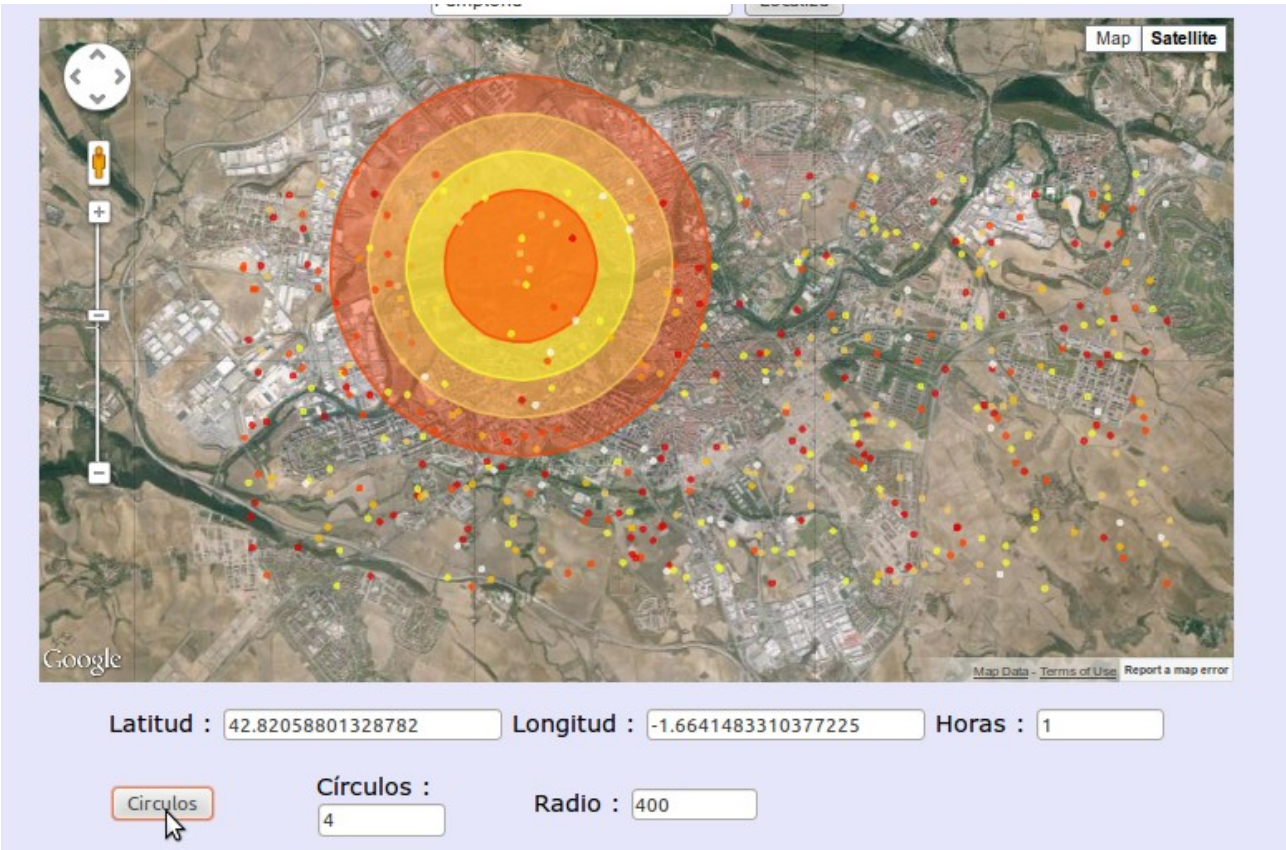
El sistema que he creído conveniente utilizar es la llamada a métodos a través de botones. En vez de que el botón redirija la página a otra y que esta función se ejecute en un script interno del código Html, el programa genera las modificaciones sobre la misma.

El primer elemento que vamos a ver sobre la llamada a métodos son los botones encargados de llamar a las funciones que muestran los datos y figuras en el mapa. El primer botón es el encargado de generar un numero de círculos con el radio que queramos en un punto.

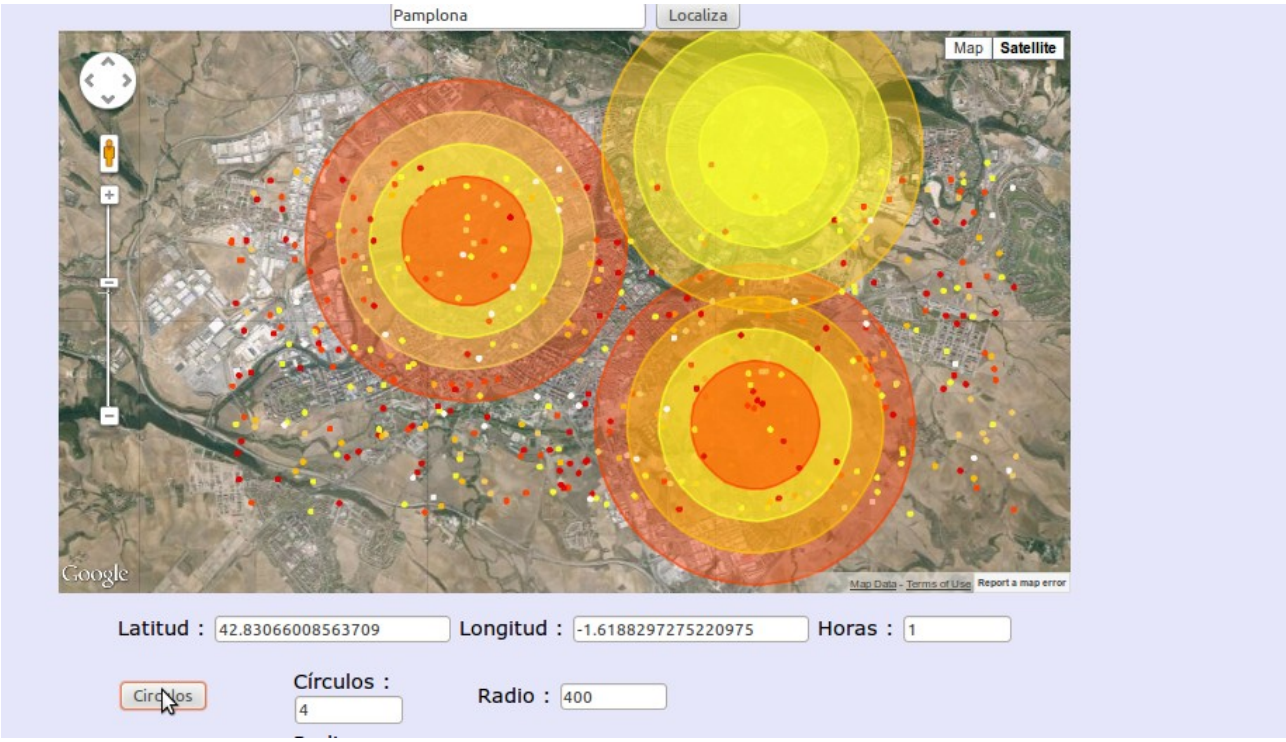
Este botón tiene 2 parámetros que debemos introducir para poder generar las formas. Son 2 texfieldel número de círculos y el radio del primer círculo. (El de los demás irá incrementando en base al radio del primero). La latitud y longitud del circulo que queremos generar se obtendrá del formulario común a todos. Su código es así:

```
      <form name="formulario1" id="formulario1">
<tr>
        <FONT FACE="Verdana" SIZE=4 COLOR="red">
 <td width="131"><input name="button2"       type="button"  onClick="creaCirculos()" value="Circulos" ></td>
 <td width="138"> <FONT FACE="Verdana" SIZE=4 COLOR="black">Círculos : </FONT> <input type="text" name="Circulos" size="5"
 value="4"></td>
 <td width="159"> <FONT FACE="Verdana" SIZE=4 COLOR="black">Radio : </FONT> <input type="text" name="Radio"  size="5"  value="400"></td>
 <td width="250"> </td>
        </FONT>
</tr>
        </form>
```

Para definir este elemento hay que crear un etiqueta **<form>**, con su nombre e identificador, en este caso **"formulario1"**. Para el botón se crea un input de tipo **"button"** con valor **"Circulos"** (etiqueta del botón) y la llamada a la función "**creaCirculos()**". Lo demás son simples textfields con sus nombres, tipo , tamaño y valor inicial. Este botón crea figuras de este tipo:



Como se puede observarse han generado 4 círculos en un punto, también se observa que hay puntos que están fuera de los círculos y que también están representados. En este caso muestra temperaturas muy diversas ya que los datos utilizados han sido generados aleatoriamente. Podemos generar los dibujos de este tipo que queramos puesto que se guardan hasta que se decide borrar. Como por ejemplo:



La función "**creaCirculos()**" es así:

```
function creaCirculos(){
try{
»
»
»       var corx=parseFloat(document.formulario01.Latitud.value);
»       var cory=parseFloat(document.formulario01.Longitud.value);
»       var tipo=getRadioButtonSelectedValue(document.formulario0.tipo);
»       if  (tipo==null){
»         alert("Debes elegir un tipo de dato");
»         return;
»       }
»       var circulos=parseFloat(document.formulario1.Circulos.value);
»       var radio=parseFloat(document.formulario1.Radio.value);
»       var rango=parseFloat(document.formulario01.Horas.value);
»
```

Para empezar definimos la función y recogemos los valores de los parámetros introducidos en los textfield. Para ello utilizamos el método document.formularioX"textfield".value. Corx y cory son latitud y longitud del centro de los círculos, tipo es humedad,temperatura o Co2, Si no se ha seleccionado ninguna de estas tres salta una alerta y se termina la ejecución. Círculos son el numero

de círculos que hay que generar, radio del primer círculo y rango de tiempo.

```
var r = new Array();
var radios=new Array();
var ahora=new Date();
var primero=ahora.getTime();
```

Después creamos “r” y radios, dos tablas que sirven para saber si las medias deben coger en dato como primero o hacer la media en caso de que ya haya leído mas y para describir el radio de cada uno de los círculos, estas dos dependen de la variable círculos, pues este será su tamaño. En la variable ahora creamos la fecha actual de la ejecución con la función **Date()** y en primero usamos la función **getTime(),** que nos devuelve el numero de milisegundos que han trascurrido desde el 1 de enero de 1970. Estos dos últimos valores nos servirán para el tema de acotar los datos en función del tiempo.

```
for (i=1;i<=circulos;i++){
    radios[i]=eval(radio)*(eval(i)+1);
    r[i]=false;
}
```

Como habíamos dicho las variables “r” y radios eran tablas que dependían del numero de círculos que queremos representar, Como el radio de los demás depende del primero haremos un pequeño calculo para asignar los nuevos radios y poniendo r[i]=false, lo que significa que todavía no se ha hecho cálculos con ninguno de los círculos.

```
var tabla = new Array();
var media  = new Array();
var txtFile= new XMLHttpRequest();
txtFile.open("GET", "posicion.php?", false);
txtFile.send( null );
var mySplitResult = txtFile.responseText.split("<Posicion>");
```

La siguiente parte crea una tabla donde se meterán elementos de las lecturas de datos, depende del numero de datos que hay por cada registro, en mi caso son 6, latitud, longitud, humedad, temperatura, Co2 y fecha. Media es la tabla de medias donde se calcularán las medias finales de cada círculo. Después hacemos la llamada al fichero de datos que queremos leer y hacemos una partición por cada registro que hay en el fichero de lectura. Estará separada por la etiqueta <Posicion> utilizando el método split y se lo asignamos a mySplitResult.

```
for  (j=1;j<mySplitResult.length;j++){

  var datos = mySplitResult[j].split("><");
      for(i = 0; i < 6; i++){

              var coor = datos[i].split(">");
              tabla[i] = coor[1];

      }
```

Ahora por cada registro actual, introducimos en la tabla de datos su parte de la cadena que nos interesa, separándola por los elementos **"><"** y **"<".**

```
var cx= parseFloat(tabla[0].split("<"));
var cy= parseFloat(tabla[1].split("<"));
var dat1= parseFloat(tabla[2].split("<"));
var dat2= parseFloat(tabla[3].split("<"));
var dat3= parseFloat(tabla[4].split("<"));
 var fech=tabla[5].split("<");
```

Ahora vamos leyendo por cada elemento que hemos separado y los volvemos a separar por subelementos de forma que solo nos queden los datos tal y como los queremos.

```
var fecha=creaFecha(fech[0]);

var segundo=fecha.getTime();
```

Para poder pasar la fecha del registro de datos que estamos analizando ahora hay que llamar a la función **"creaFechas()",** después calculamos el número de milisegundo de ese dato.

```
if ((primero-segundo)<=(rango*60*60*1000)){

    if (tipo==1){
      var dato=dat1;
    }
    else if  (tipo==2){
      var dato=dat2;
    }
    else if  (tipo==3) {
      var dato=dat3;
    }
```

Como el valor del rango que hemos introducido por pantalla está en horas y los valores de primero y segundo en milisegundos, multiplico el rango por 3600000 (rango*60 minutos* 60segundos *1000 milisegundos). Si la diferencia del momento actual y el del dato registrado es menor o igual que nuestro rango, podemos analizar ese dato, sino pasamos al siguiente. Dependiendo del la elección que hayamos echo a la hora de escoger el tipo que queremos observar, tendremos que asignarle a dato su correspondiente tipo.

```
var posicion1 =new google.maps.LatLng(corx,cory);
var posicion2 =new google.maps.LatLng(cx,cy);
var di=google.maps.geometry.spherical.computeDistanceBetween(posicion1,posicion2);
if (di<=radios[1]){
   if (r[1]==true){
    media[1]=sacarMedia(media[1],dato);
  }
  else {
    //media[1]=null;
    media[1]=dato;
    r[1]=true;
  }
}


for (k=2;k<=circulos;k++){

 if ((di<=radios[k])&&(di>radios[k-1])){
   if  (r[k]==true){
    media[k]=sacarMedia(media[k],dato);
  }
  else {
    //media[k]=null;
    media[k]=dato;
    r[k]=true;
  }

 }
}
```

Creamos dos puntos geográficos con la función de Google Maps **LatLng(),** el del primero siempre sera el centro de los círculos, y el segundo son las coordenadas de cada registro analizado. Creamos la variable di con la función **computeDistanceBetween()** de Google Maps. Nos devuelve la distancia que hay de un punto a otro en metros. Después toca asignar cada dato a cada círculo, dependiendo de la distancia que haya de ese dato al centro de los círculo se calculará su nueva media o no. Si la distancia de un punto al centro es menor que el radio del primer circulo, significa que ese punto hay que tenerlo en cuenta para ese círculo. Para el resto de círculos, solo tenemos en cuenta los puntos que pertenecen a ese círculo pero que no pertenecen a los círculos anteriores a este. Si el valor de la variable "r" para ese círculo es false, significa que todavía no se ha leído ningún dato para ese círculo y no tendremos que hacer la media sino que se asignará directamente. (ponemos la variable "r" de ese círculo a true para decir que ya han pasado datos por aquí) Si ya se ha leído algún valor, si que hacemos la media.

```
    if ( di>radios[circulos]     ) {
      if (tipo==2){
        var color=sacaColorTemperatura(dato);
      }
      else{
        var color=sacaColor(dato);
      }
     centro=new google.maps.LatLng(cx,cy);
      var pu={strokeColor: color,strokeOpacity: 0.8,strokeWeight: 2,fillColor: color,fillOpacity: 0.35 ,map: map,center:centro,radius: 25};
      Circle = new google.maps.Circle(pu);
      Circle.setMap(map);
    }
}
```

Además, los que su distancia sea mayor que la del radio del último círculo se crearan puntos para representarlos. Utilizamos la función **Circle()** de Google Maps**.**

```
      for (k=k;k>=1;k--){
       if (tipo==2){
        var color=sacaColorTemperatura(media[k]);
      }
      else{
        var color=sacaColor(media[k]);

      }

        centro=new google.maps.LatLng(corx,cory);
        var pu={strokeColor: color,strokeOpacity: 0.8,strokeWeight: 2,fillColor: color,fillOpacity: 0.6-k*0.05 ,map: map,center:centro,radius:
        radios[k]};
        Circle = new google.maps.Circle(pu);
        Circle.setMap(map);

      }

}
```

Por último, generamos cada círculo con su correspondiente color en función de su media, su centro, radio en metros, etc, utilizando la función de Google Maps **Circle()**, y se lo asignamos al objeto map. Dependiendo del tipo de dato que sea tendremos que utilizar una función u otra para sacar el color. Las veremos mas tarde.

```
      catch(err)
            {
      txt="There was an error on this page.\n\n";
      txt+="Error description: " + err.message +","+i+ "\n\n";
      txt+="Click OK to continue.\n\n";

      /*alert*/(txt);

      }
    }
```

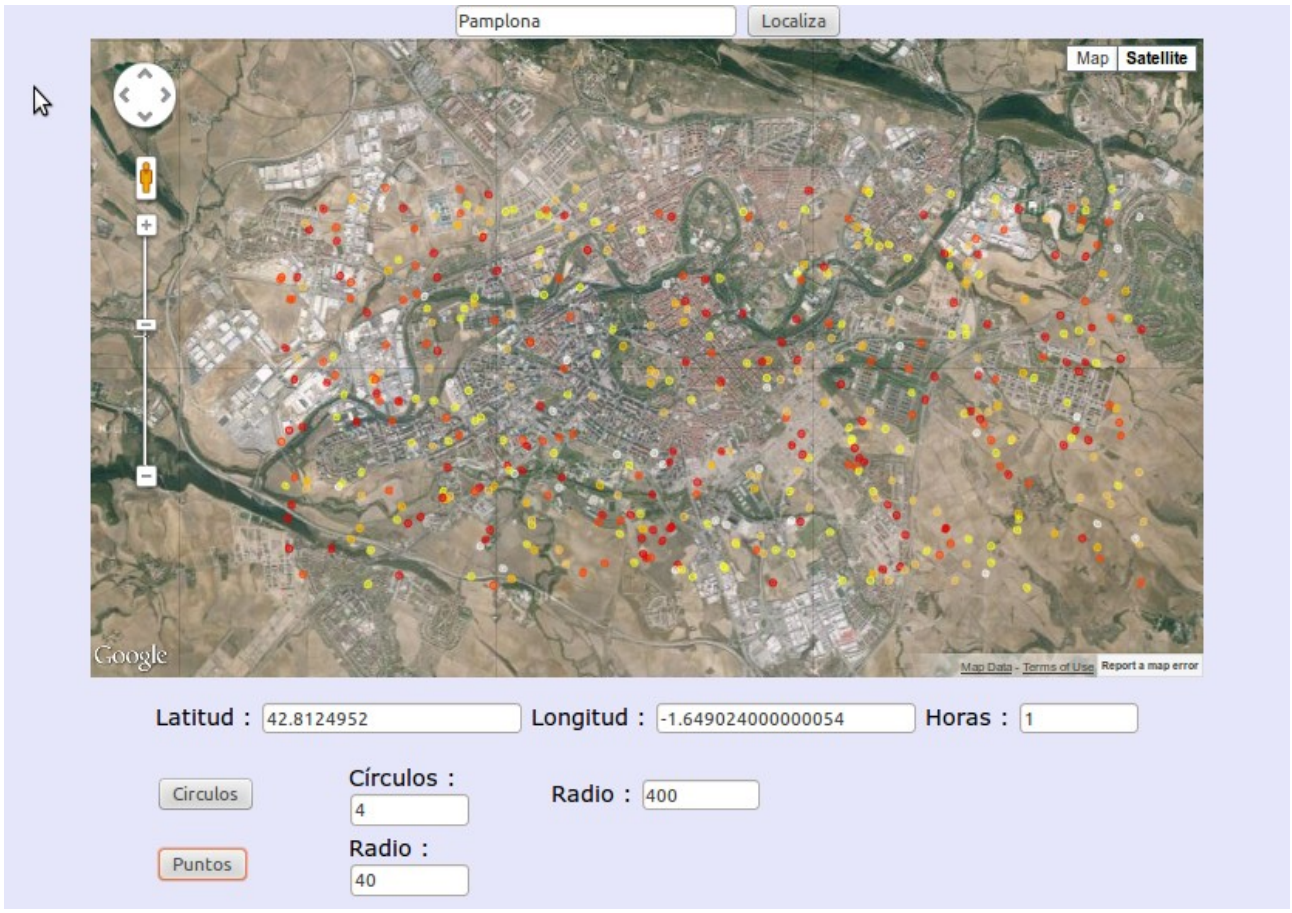
Por último el captador de errores.

El siguiente botón se encarga de crear por cada punto que leemos de los datos un circulo con radio el que queramos y color según su valor.

Y su código es así:

```
        <form name="formulario2" id="formulario2">
<tr>
    
  <td><input name="button" type="button"  onClick="creaCirculosPuntos();" value="Puntos" /></td>
  <td> <FONT FACE="Verdana" SIZE=4 COLOR="black">Radio :</FONT>    <input type="text" name="Radio"  size="5"
  value="5"></td>
  <td> </td>
  <td> </td>
            </FONT>
</tr>
        </form>
```

Está formado por una etiqueta form con nombre e identificador, “**formulario2”**, un input para el botón, de tipo **“button”**, su etiqueta y la función a la que llama, “**creaCirculosPuntos();**”, después esta la descripción del textfield con sus parámetros. El valor del parámetro radio está medido en metros, ya que la función Circle de Google Maps así lo define. Este es un ejemplo del resultado:



La función "**creaCirculosPuntos();"** está definida de la siguiente manera:

```
function creaCirculosPuntos(){
  try{
        var tipo=getRadioButtonSelectedValue(document.formulario0.tipo);
        if  (tipo==null){
         alert("Debes elegir un tipo de dato");
          return;
        }
        var rango=parseFloat(document.formulario01.Horas.value);
        var radio=parseFloat(document.formulario2.Radio.value);
        var ahora=new Date();
```

```
var primero=ahora.getTime();
var tabla = new Array();
var position = new Array();
var txtFile = new XMLHttpRequest();
txtFile.open("GET", "posicion.php?", false);
txtFile.send( null );
var mySplitResult = txtFile.responseText.split("<Posicion>");
var cont=mySplitResult.length;

var temp= new Array();
var hum= new Array();
var pol= new Array();
for( i = 1; i < mySplitResult.length; i++ ){
      var datos = mySplitResult[i].split("><");
      for(j = 0; j < 6; j++) {
            var coor = datos[j].split(">");
            tabla[j] = coor[1];
      }
var coorx = tabla[0].split("<");
var coory = tabla[1].split("<");
var dat1  = tabla[2].split("<");
var dat2  = tabla[3].split("<");
var dat3  = tabla[4].split("<");
var fech=tabla[5].split("<");

 var fecha=creaFecha(fech[0]);

 var segundo=fecha.getTime();
```

En principio la función realiza los mismos primeros pasos que la función **creaCirculos()** con la excepción de que en esta creo diferentes variables como por ejemplo position , temp,hum o pol. Estas variables son tablas donde meteremos los datos de cada uno de los puntos que queremos generar.

```
      if ((primero-segundo)<=(rango*60*60*1000)){
position[i-1] = new google.maps.LatLng(coorx[0], coory[0]);
temp[i-1] = dat1[0];
hum[i-1] = dat2[0];
pol[i-1] = dat3[0];
cont=i;
}
}
```

En este caso vamos a realizar los puntos una vez analizados los datos, en vez de leer, generar

y siguiente. Para ello vamos a ir asignado los valores de cada dato a su tabla correspondiente con el valor de i-1 para que empiece desde 0.

```
for( i = 1; i < cont; i++ ){
    var corx=position[i].lat();
    var cory=position[i].lng();
    var color;

if (tipo==1){

  var color=sacaColor(temp[i]);
}
else if  (tipo==2){

  var color=sacaColorTemperatura(hum[i]);
}
else if  (tipo==3) {

  var color=sacaColor(pol[i]);

}
```

Ahora, una vez analizado todos los puntos asignamos sus valores a los parámetros necesarios para crear la figura. En función del tipo de dato que hayamos elegido utilizará un método diferente para calcular su color.

```
centro=new google.maps.LatLng(corx,cory);
var pu={strokeColor: color,strokeOpacity: 1,strokeWeight: 1,fillColor: color,fillOpacity: 0.4 ,map: map,center:centro,radius: radio};
Circle = new google.maps.Circle(pu);
Circle.setMap(map);
}
```

Generamos el centro del círculo con los parámetros “corx” y “cory”, radio y color el correspondiente y se lo asignamos al objeto map.

```
  }
  ctach(err)
        {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);

  }
}
```

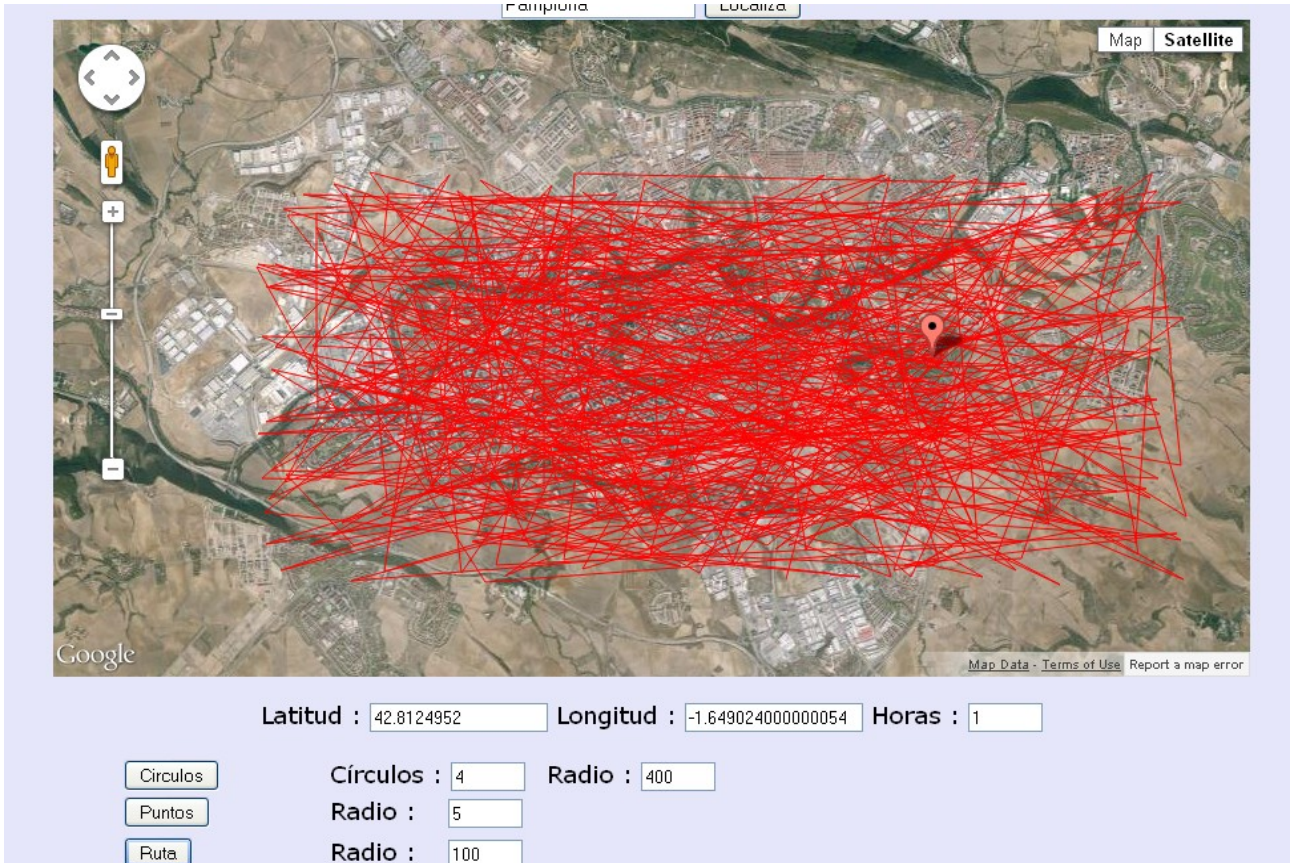
Y por último el captador de errores.

El siguiente botón se encarga de realizar una ruta uniendo todos los puntos por donde ha pasado el vehículo y crea un circulo en el ultimo punto registrado con la media de los valores que están dentro de él. Para este botón existe dos parámetros que se introducen a través de dos textfield, radio del círculo final y el valor del tiempo de hasta hace cuento quieres ver.

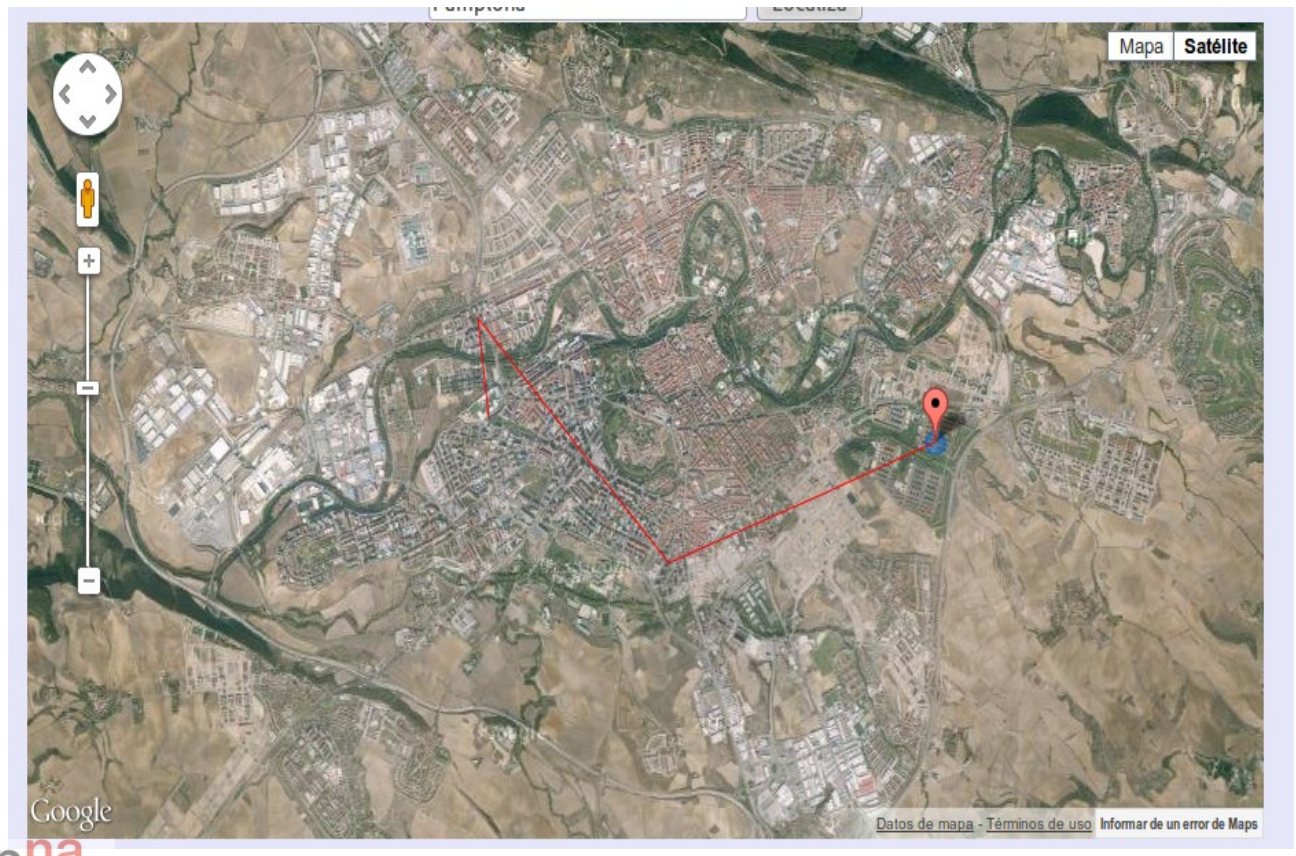
Su código es así:

```
        <form name="formulario3" id="formulario3">
<tr>
            <FONT FACE="Verdana" SIZE=4 COLOR="red">
  <td height="32"><input name="button3" type="button" onClick="timer=ruta_con_circulo()" value="Ruta" /></td>
  <td> <FONT FACE="Verdana" SIZE=4 COLOR="black">Radio :</FONT>    <input type="text" name="Radio"  size="5"
  value="100"></td>
  <td> </td>
  <td> </td>
            </FONT>
</tr>
        </form>
```

Al igual que los demás se crea utilizando una etiqueta form con su nombre e identificador,formulario3 y los textfield correspondientes. Un ejemplo de como quedaría es:



Como se pude observar al final sería una locura ver la trayectoria del vehículo a lo largo del tiempo. Para eso está el campo horas que no permite ver los datos sobre un rango de tiempo.



Al estar en función de horas, si se quiere acotar el tiempo hay que hacer lo de esa forma.

El botón llama a la función **"ruta_con_circulo()",** que se define de la siguiente forma:

```
function ruta_con_circulo(){
      try
          {
      var tipo=getRadioButtonSelectedValue(document.formulario0.tipo);
      if  (tipo==null){
       alert("Debes elegir un tipo de dato");
         return;
      }

      var radio=parseFloat(document.formulario3.Radio.value);
      var rango=parseFloat(document.formulario01.Horas.value);
      var tabla            = new Array();
      var position    = new Array();

      var ahora=new Date();
```

Como todas la funciones, empezamos por definir su nombre y coger los parámetros que le hemos introducido por pantalla, en este caso, el tipo, radio del circulo final y rango de tiempo que queremos observar. Los dos últimos parámetros los pasaremos a tipo Float porque después van a ser manipulados. Creamos las mismas variables que la función anterior para recoger en una tabla, llamada **tabla**, los diferentes datos de cada registro de posiciones, una tabla, **position**, que los valores de latitud, longitud de cada punto. Sin embargo, esta vez vamos a crear la variable ahora, que es de tipo Date y tiene como valor el momento actual.

```
var txtFile    = new XMLHttpRequest();
txtFile.open("GET", "posicion.php?", false);
txtFile.send( null );
var mySplitResult     = txtFile.responseText.split("<Posicion>");
for( i = 1; i < mySplitResult.length; i++ ){
      var datos       = mySplitResult[i].split("><");
      for(j = 0; j < 6; j++) {
            var coor  = datos[j].split(">");
            tabla[j]  = coor[1];
      }
var coorx       = tabla[0].split("<");
var coory       = tabla[1].split("<");
var fech=       tabla[5].split("<");

var primero=ahora.getTime();
var fecha=creaFecha(fech[0]);
var segundo=fecha.getTime();
```

Como podemos ver, ahora seguimos el mismo método de anteriores funciones para la lectura de datos. Esta vez queremos que lea hasta la partición 6 para coger el dato fecha que cada registro. Introducimos en coorx y coory la latitud y longitud y en fech la fecha de cada registro de nuestro fichero de datos.

```
if ((primero-segundo)<=(rango*60*60*1000)){

  position[i-1]= new google.maps.LatLng(coorx[0], coory[0]);

}

}
```

Para poder representar los datos en función de una fecha debemos calcular si la diferencia entre la fecha actual y la fecha del registro que estamos analizando es menor o igual que el rango introducido por pantalla y si es mayor, no guardarlo. Para ello, he creado tres variables mas, fecha, donde se introduce la fecha del registro actual con la función **creaFecha()** y primero y segundo. Con el fin de poder manipular y hacer cálculos con fechas, he utilizado la función **getTime().** Esta función me devuelve el número de milisegundos que han pasado desde el 1 de enero de 1970 hasta el momento actual. En primero meto los milisegundo que han pasado hasta el momento de ahora y en segundo el número de milisegundos que han pasado hasta la fecha del registro que estoy analizando. Como estoy introduciendo el rango por horas, debo multiplicarlo por 60 (minutos) por 60 (segundos) y por 1000 (milisegundos). Si la resta del primero menos el segundo es menor o igual que el rango pasado a milisegundos, entonces introduzco esa posición en la tabla position.

```
corx = posicion().lat();
cory = posicion().lng();
poligono=circulo(corx,cory,radio,tipo,rango,ahora);
poligono.setMap(map);
marker = new google.maps.Marker({ position: posicion(), map: map, title:"Último lugar donde estuvo!!!", animation: google.maps.Animation.DROP});
multipuntos = new google.maps.Polyline({ path: position, strokeColor: "#FF0000", map: map, strokeOpacity: 1.0, strokeWeight: 1,});
```

Ahora lo que vamos a hacer es crear las figuras que nos interesan. Esta vez vamos allamar a otras funciones para realizar las figuras. Creamos la posición corx y cory y llamamos a la función posicion(), esta función nos devuelve el último punto registrado en forma de tupla de latitud y longitud, generada por la función de Google Maps **LastLng()**. Esta función nos permite hacer referencia a cada dato de la tupla de la forma que describo, utilizando la función lat() y lng() sobre la tupla de posiciones. Una vez obtenidos esos datos llamamos a la función **circulo(),** con los parámetros que nos interesa y lo asignamos al objeto map ya creado. También he querido poner un objeto de tipo marcador que señala la posición actual del vehículo, o su último registro. La función de Google Maps **Marker()** nos lo permite de esa forma, introduciéndole la posición donde queremos, el mapa al que va asignado un pequeño comentario si ponemos el ratón sobre él y el tipo de animación. Ya solo nos queda crear la trayectoria del vehículo uniendo todos los puntos de la tabla position. La funcion de Google Maps **Polyline()** nos crea este elemento, indicándole los puntos que debe unir (position), el mapa al que queremos asociarlo y algunas características mas , como el color o intensidad de este.

```
        }
            catch(err){
                txt="There was an error on this page.\n\n";
                txt+="Error description: " + err.message + "\n\n";
                txt+="Click OK to continue.\n\n";
                alert(txt);
            }
}
```

Como esta vez hemos utilizado otra forma de análisis de datos, llamando a funciones que lo realizan en vez de la propia función, vamos a explicar que es lo que hacen. La primera función que se utiliza es **posicion()** y es de la siguiente manera.

```
function posicion(){ // Me devuelve última posición que he recibido
try{
        var tabla = new Array();
        var tabla1 = new Array();
        var txtFile= new XMLHttpRequest();
        txtFile.open("GET", "posicion.php?", false);
        txtFile.send( null );
        var mySplitResult = txtFile.responseText.split("<Posicion>");
        var datos = mySplitResult[1].split("><");
        for(i = 0; i < 2; i++){
                var coor      = datos[i].split(">");
                tabla[i]      = coor[1];
        }
        var coorx     = tabla[0].split("<");
        var coory     = tabla[1].split("<");
        var position = new google.maps.LatLng(coorx[0], coory[0]);
        return (position);
}
catch(err)
{
   txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
}
        }
```

Lectura del fichero de posiciones que da como único resultado la posición final que se ha registrado. Es mas rápido que esperar a leer todos los datos por que solo lee el primer registro. La otra función a la que hacemos referencia es **circulo().**

```
function circulo(corx,cory,radio,tipo,rango,ahora){
try{


      var distancia=0;
      var media=media_ultimo_circulo(radio/1000,tipo,rango,ahora);
      if (tipo==2){
        var color=sacaColorTemperatura(media);
      }
      else {
        var color=sacaColor(media);
      }

      centro=new google.maps.LatLng(corx,cory);
      var pu={strokeColor: color,strokeOpacity: 1,strokeWeight: 1,fillColor: color,fillOpacity: 0.4 ,map: map,center:centro,radius: radio};
      Circle = new google.maps.Circle(pu);

      return Circle;
}
 catch(err)
       {
 txt="There was an error on this page.\n\n";
 txt+="Error description: " + err.message +","+i+ "\n\n";
 txt+="Click OK to continue.\n\n";

 alert(txt);

 }

}
```

Esta función llama a su vez a otra función que calcula la media de todos los datos que están dentro del círculo final. Evalúa el tipo de dato que quiere representar y devuelve un objeto círculo con las características que nos interesan y han dado como resultado. La función que calcula esa media se llama **media_ultimo_circulo().** Sus parámetros son el radio del círculo final, el tipo de dato, rango y el momento de ahora para acotar la representación. Es de la siguiente forma:

```
function media_ultimo_circulo(radio,tipo,rango,ahora){
  try
  {

       var media =null;
       var tabla1 = new Array();
       var txtFile= new XMLHttpRequest();
       txtFile.open("GET", "posicion.php?", false);
       txtFile.send( null );
```

Creamos media, la variable donde se introducirá el resultado final. Tabla1 para separar los datos del registro que estamos analizando y hacemos la llamada al fichero de lectura.

```
var primero=ahora.getTime();
var mySplitResult = txtFile.responseText.split("<Posicion>");
for  (j=1;j<mySplitResult.length;j++){
        var datos =  mySplitResult[j].split("><");

        for(i = 0; i < 6; i++){
            var coor      = datos[i].split(">");
            tabla1[i]     = coor[1];

        }
        var cx      = parseFloat(tabla1[0].split("<"));
        var cy      = parseFloat(tabla1[1].split("<"));
        var dat1    = parseFloat(tabla1[2].split("<"));
        var dat2    = parseFloat(tabla1[3].split("<"));
        var dat3    = parseFloat(tabla1[4].split("<"));
        var fech    = tabla1[5].split("<");
```

Creamos la variable primero con los milisegundo hasta la fecha que le hemos introducido como parámetro y empezamos a leer los registros uno a uno.

```
    var fecha=creaFecha(fech[0]);
    var segundo=fecha.getTime();


if ((primero-segundo)<=(rango*60*60*1000)){

        if (tipo==1){
          var dato=dat1;
        }
        else if  (tipo==2){
          var dato=dat2;
        }
        else if  (tipo==3) {
          var dato=dat3;
        }
```

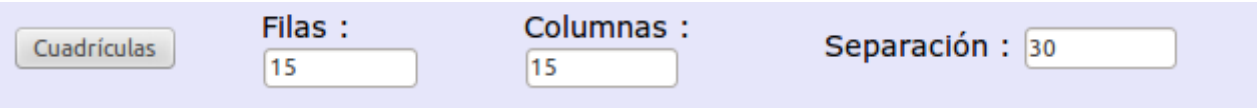
Hacemos lo mismo de la fecha actual pero con la fecha del registro y calculamos si esta dentro de nuestro rango. Dependiendo del tipo de dato que queremos observar se lee su correspondiente.

```
                var di=Dist(cx,cy,cx,cy);
                if ((di<=radio) && (media!=null) ) {
                    media=sacarMedia(dato,media);


                }
                else {
                media=dato;

                }
          }
         }

return (media);
  }
```

Ahora como hemos echo otras veces calculamos la distancia que hay entre los puntos que estamos analizando y el punto donde se va a generar el círculo final. Si la distancia es null es que es el primer dato y no tenemos que hacer la media si no cogerlo directamente. Y por último generamos el captador de errores.

```
catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
  }
}
```

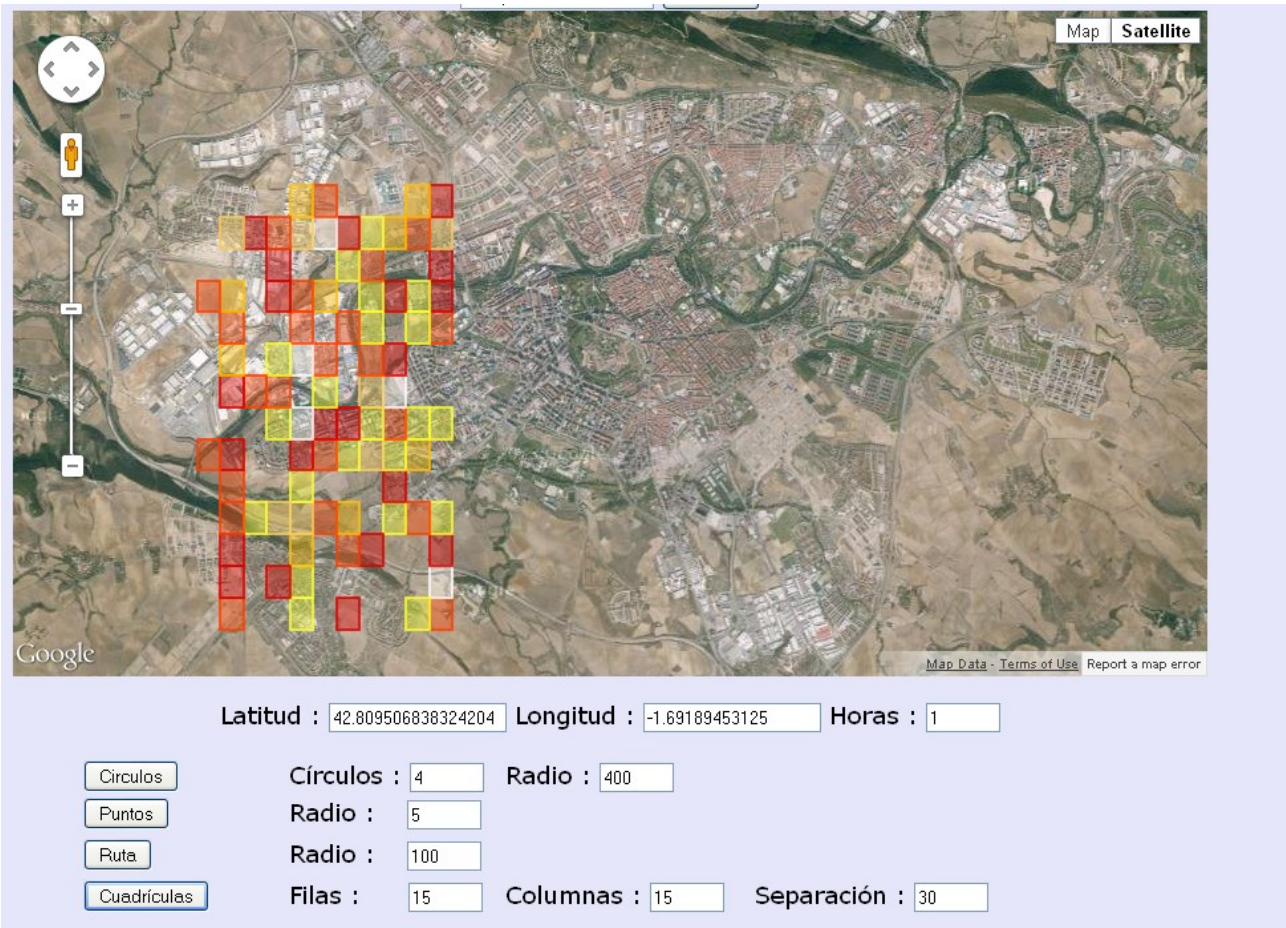
La última de las figuras que he desarrollado es una cuadrícula sobre el mapa, es el botón "**Cuadrículas".** Esta función recoge los valores de los parámetros del formulario común a todos y genera la cuadrícula en ese punto.



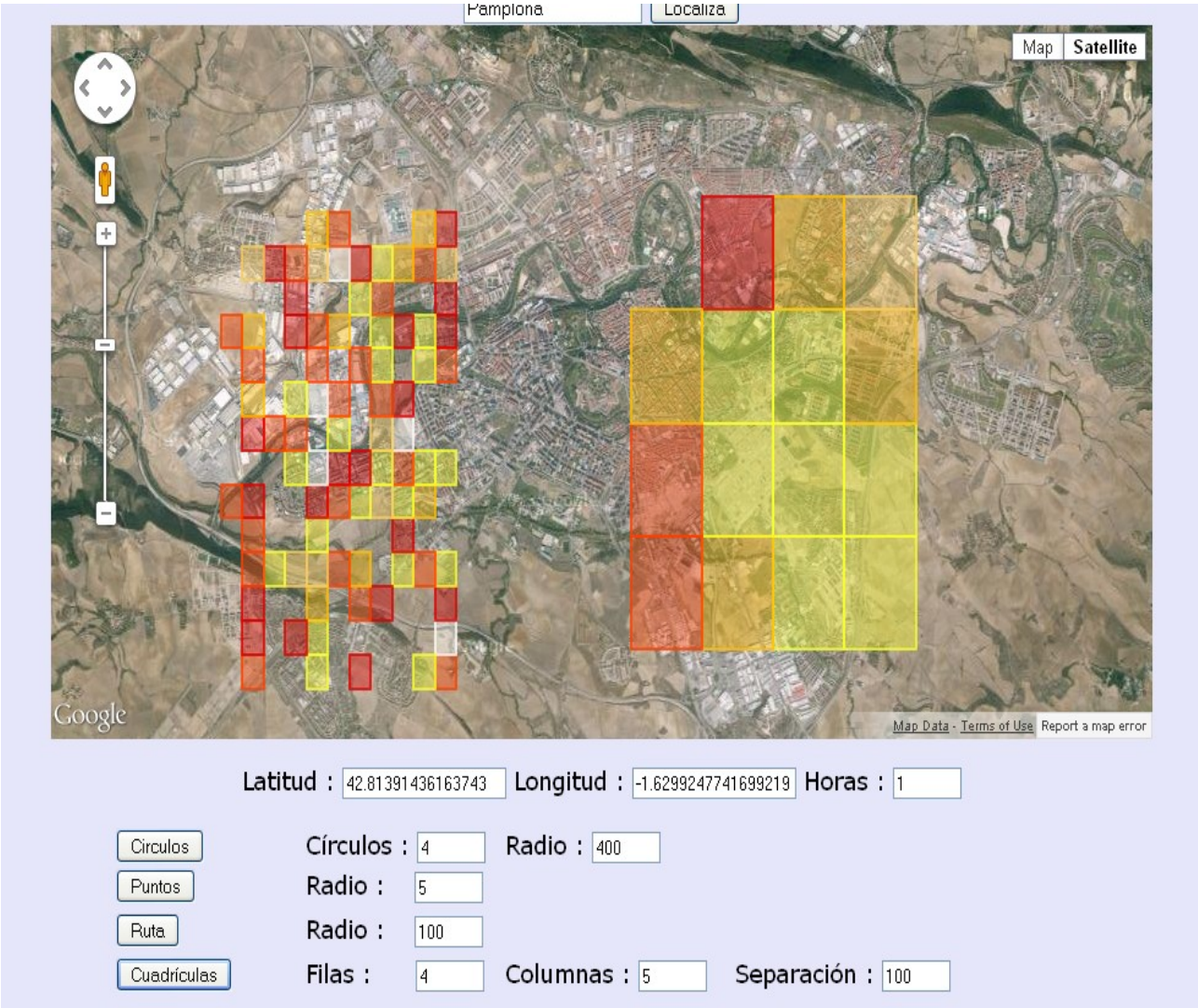
A parte de los parámetro generales también hay que introducir los valores para definir como va ser la cuadrícula, las filas, columnas y la separación en grados entre un punto y otro de las cuadrículas. El código Html correspondiente es así:

```
        </form>
         <form name="formulario4" id="formulario4">
<tr>
 <FONT FACE="Verdana" SIZE=4 COLOR="red">
 <td><input name="button4" type="button" onClick="creaCuadricula()" value="Cuadrículas" /></td>
 <td><FONT FACE="Verdana" SIZE=4 COLOR="black">Filas :</FONT>     </FONT>
 <input type="text" name="Filas"  size="5"  value="15"></td>
 <td> <FONT FACE="Verdana" SIZE=4 COLOR="black">Columnas : </FONT><input type="text" name="Columnas"  size="5"  value="15"></td>
 <td>  <FONT FACE="Verdana" SIZE=4 COLOR="black">Separación :</FONT> <input type="text" name="Separacion"  size="5"  value="30"></td>
 </FONT>
</tr>
</form>
```

Este formulario tiene como nombre e identificador, **"formulario4",** aparece la descripción del botón y de los textfield. El botón llama a la función **"creaCuadricula()"**, que da como resultado los siguientes ejemplos:



En este caso hay celdas de la cuadrícula que no se han pintado, eso es debido a que no hay los datos necesarios en esa zona para hacer el cálculo. También podemos pintar varias cuadrículas y de diferentes tamaños al mismo tiempo.



La función **creaCuadrícula()** esta implementada de la siguiente forma:

```
function creaCuadricula(){
  try
  {
 var filas=parseFloat(document.formulario4.Filas.value)+1;
 var columnas=parseFloat(document.formulario4.Columnas.value);
 var tipo=getRadioButtonSelectedValue(document.formulario0.tipo);
      if  (tipo==null){
       alert("Debes elegir un tipo de dato");
         return;
      }
 var corx=parseFloat(document.formulario01.Latitud.value);
 var cory=parseFloat(document.formulario01.Longitud.value);
 var rango=parseFloat(document.formulario01.Horas.value);

var separacion=parseFloat(document.formulario4.Separacion.value)/10000;
```

Como todas las funciones, lo primero que hago es la declaración del nombre y recogida de parámetros para la función. Filas, columnas, tipo, latitud, longitud , rango y separación. Esta última la dividimos entre 10000 para hacer los cálculos sobre decenas de milésimas de grados.

```
corx=corx-(filas/2*separacion);
cory=cory+(columnas/2*separacion);
```

Este tratamiento lo hago para que el punto que genero sea un punto céntrico de la cuadrícula y no uno de los vértices.

```
 var ahora=new Date();
var primero=ahora.getTime();
var cuadrados=(filas)*(columnas);
```

Creamos el valor de hora con la fecha actual y su correspondiente valor en mili segundos. Definimos el número de cuadrados que se van a generar con los datos que se han recogido.

```
var cuadricula= new Array(cuadrados);
      for (i = 0; i < cuadrados; i++) {
            cuadricula[i] = new Array(4)
            for (j = 0; j < 5; j++) {
                  cuadricula[i][j] = "["+i+","+j+"]"
            }
      }
```

La estructura cuadrícula está formada por 4 puntos, necesitaremos na tabla de dos dimensiones con el índice de cuadrícula y sus correspondientes puntos geográficos. Como en algunos casos la construcción de una tabla de dos dimensiones daba problemas lo que hice fue utilizar este método. Sabiendo cuantos cuadrados vamos y que cada cuadrado esta formado por cuatro elementos generamos las tablas.

```
var punto=new Array();
var e=1;
var lat=0.0;
var lng=0.0;
var puntosFila=filas+1;
var puntosColumna=columnas+1;
for (j=0;j<(puntosColumna);j++){
for (i=0;i<(puntosFila);i++){

      lat=(corx+i*separacion);

      lng=(cory+(-j*separacion));

      punto[e]=new google.maps.LatLng(lat,lng);

      e++;

   }
}
```

Lo siguiente que tenemos que hacer es calcular los puntos geográficos que se utilizarán para definir los vértices de los cuadrados. Definimos una tabla punto, que será una dupla de latitud, longitud. Inicializamos el índice "e" a 1 y las variables lat y lng a 0.0. Calculamos los puntos por filas y puntos por columna que vamos a necesitar y empezamos a calcular. La latitud del primer punto será la de la coordenada x que antes hemos calculado mas la separación por el numero de fila actual. Esto nos va construyendo los puntos por columnas. Y hacemos lo mismo para la longitud. El punto final e, será la variable creada con la función de Google Maps **LatLng()** y los parámetros

actuales de latitud y longitud. Es importante saber que por cada fila y columna tiene que haber mas puntos que estas ya que el numero de filas no será igual al número de puntos que se necesitan para representarlos y lo mismo para las columnas.

```
var media=new Array();
var tabla= new Array();
var position    = new Array();
var txtFile     = new XMLHttpRequest();
txtFile.open("GET", "posicion.php?", false);
txtFile.send( null );
var mySplitResult     = txtFile.responseText.split("<Posicion>");
for (c=1;c<=cuadrados;c++){
  if (c%(filas+1)!=0){

    cuadricula[c]=[punto[c-1],punto[c],punto[c+filas+1],punto[c+filas]];

  }
}
```

Preparamos las variables de lectura de datos, de cálculo de medias y definimos el fichero de lectura y su separación por elementos de registro. Ahora como queremos ver que puntos están dentro de cada cuadrícula hacemos un recorrido por cada una de ellas leyendo los datos y si están dentro de esta hacemos su análisis. Para ello comenzamos calculando por cada cuadrícula cual serán sus puntos y los introducimos en la tabla de cuadrículas. Hay que tener en cuenta que si las cuadrículas empiezan de 1 no nos interesa representar la cuadrícula número de filas +1 por que representaría una fila desde arriba del todo y se uniría con los puntos de abajo del rectángulo formado.

```
for (c=1;c<=cuadrados;c++){

 if (c%(filas+1)!=0){

      var a = parseFloat(punto[c].lat());
      var b = parseFloat(punto[c+1].lat());
      var n = parseFloat(punto[c+filas].lng());
      var d = parseFloat(punto[c].lng());
```

Una vez creadas las cuadriculas y los puntos, comenzamos la búsqueda de puntos internos. Asignamos a a,b,n,c los valores de los puntos de la cuadrícula actual para luego hacer el cálculo.

```
var m=false;
for( i = 1; i < mySplitResult.length; i++ ){


   var datos      = mySplitResult[i].split("><");
   for(j = 0; j < 6; j++) {
        var coor      = datos[j].split(">");
        tabla[j]      = coor[1];
   }
     var cx    = parseFloat(tabla[0].split("<"));
     var cy    = parseFloat(tabla[1].split("<"));
     var dat1  = parseFloat(tabla[2].split("<"));
     var dat2  = parseFloat(tabla[3].split("<"));
     var dat3  = parseFloat(tabla[4].split("<"));
        var fech=tabla[5].split("<");

        var fecha=creaFecha(fech[0]);

        var segundo=fecha.getTime();
```

Como muchas veces hemos visto, “n” será el indicador de primera o posterior lectura. Leemos los datos del registro actual y generamos su fecha tipo Date con la función creaFecha(). Segundo será los mili-segundos que han pasado desde la fecha estándar.

```
      if ((primero-segundo)<=(rango*60*60*1000)){
       if (tipo==1){
            var dato=dat1;
          }
          else if  (tipo==2){
            var dato=dat2;
          }
          else if  (tipo==3) {
            var dato=dat3;
          }
```

Si la fecha del registro actual está dentro del rango que nos interesa, ese dato se podrá analizar. El dato que mediremos estará en función del tipo que hayamos escogido.

```
      if ((cx>= a) && (cx<=b) && (cy>=n) && (cy<=d)){

        if (m==true){
          media[c]=sacarMedia(dato,media[c]);
        }
        else {
          media[c]=dato;
          m=true;
        }
      }
}
  }
```

Ahora, si la latitud del punto actual es mayor que la mínima de la cuadrícula y menor que la máxima, y su longitud es mayor que que la mínima y menor que la máxima, entonces ese punto pertenece al rectángulo actual. Sino, ese punto no nos interesa. Repetimos este proceso con todos los puntos y calculamos la media.

```
for (c=1;c<=cuadrados;c++){

  if ((c%(filas+1)!=0) && (media[c]!=null)){

     if (tipo==2){
    var color=sacaColorTemperatura(media[c]);
  }
  else{
    var color=sacaColor(media[c]);
  }
    poligono = new google.maps.Polygon({paths: cuadricula[c],strokeColor:color,strokeOpacity:0.8,strokeWeight:2,fillColor:color,fillOpacity:0.4});
    poligono.setMap(map);

  }
 }
}
```

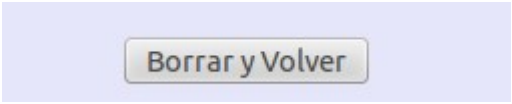
Para cada cuadrado hayamos su color en función de su media y tipo de dato y creamos una

estructura llamada polígono. Este polígono se genera con la función de Google Maps **Polygon().** Le introducimos como parámetros los cuatro puntos que debe unir de cada rectángulo, el color y otras características del color y lo asignamos al objeto mapa actual.

```
            catch(err){
                  txt="There was an error on this page.\n\n";
                  txt+="Error description: " + err.message + "\n\n";
                  txt+="Click OK to continue.\n\n";
                  alert(txt);
            }

}
```

Por último generamos el catador de errores.

El último botón que he creado es el botón "**Borrar y Volver",** se encarga de dejar el mapa tal y como estaba al principio, sin imágenes y apuntando al sitio inicial. Este botón es útil porque el programa esta hecho de forma que si presionas dos botones de los descritos anteriormente se superponen las imágenes. Me pareció interesante poder ver varios resultados al mismo tiempo.



Y su código es así:

```
<p>
      <input type="button" value="Borrar y Volver" onclick="map=crear_map() />
</p>
```

Es un input de tipo button con su nombre para la etiqueta, que cuando se presiona llama a la función **crear_map().** Esta función ya la hemos visto antes.

Otros Métodos

A continuación voy a describir funciones que se han utilizado en mas de un sitio. Este es el caso de la función **Dist(),** que calcula la distancia que hay entre dos puntos. Devuelve el valor en metros y tiene como entrada las latitudes y longitudes de los dos puntos que queremos analizar. Esta función ha sido intercambiada por la función **calculateDistanceBetween()** , aunke hacen lo mismo.

```
function Dist(lat1, lon1, lat2, lon2)
      {
        try
          {

                rad = function(x) {return x*Math.PI/180;}
                var R     = 6378.137;
                var dLat  = rad( lat2 - lat1 );
                var dLong = rad( lon2 - lon1 );
                var a = Math.sin(dLat/2) * Math.sin(dLat/2) + Math.cos(rad(lat1)) * Math.cos(rad(lat2)) * Math.sin(dLong/2) * Math.sin(dLong/2);
                var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
                var d = R * c;
                return d.toFixed(4);
          }
            catch(err){
                txt="There was an error on this page.\n\n";
                txt+="Error description: " + err.message + "\n\n";
                txt+="Click OK to continue.\n\n";
                alert(txt);
          }
}
```

Otra función que utilizo es **sacarMedia(),** obviamente para hallar la media de dos valores x e y.

```
function  sacarMedia( x, y){
  try{
        var media=(x+y)/2;
        return (media);
      }
      catch(err){
            txt="There was an error on this page.\n\n";
            txt+="Error description: " + err.message + "\n\n";
            txt+="Click OK to continue.\n\n";
            alert(txt);
          }
}
```

La función **sacaColor(),** me devuelve el colore correspondiente a la media introducida en hexadecimal, que es como los lee el programa. Esta función va de 0 a 100, porcentaje de tipo de dato sobre el medio ambiente.

```
function sacaColor(media){
  try{
    var color="";
    if (media >= 90){
        color="#FF0000";
      }
       else  if ((90>media) && (media>=70)) {
        color="#FF6600";
      }
       else if  ((70>media) && (media>=50)){
       color="#FFFF00";
      }
       else if  ((50>media) && (media>=30)){
       color="#009900";
      }
        else if  ((30>media) && (media>=10)){
        color="#0066CC";
      }
      else {color="#000000"; }
        return color;

  }
  catch(err){
   txt="There was an error on this page.\n\n";
   txt+="Error description: " + err.message + "\n\n";
   txt+="Click OK to continue.\n\n";
   alert(txt);

  }
}
```

Como la temperatura la medimos de diferente forma he creado otra función que hace lo mismo pero los valores y tipo de color cambian, se llama **sacaColorTemperatura().**

```
function sacaColorTemperatura(media){
    try{
    var color="";
    if (media >= 40){
         color="#DF0101";
        }
         else  if ((40>media) && (media>=35)) {
          color="#FF4000";
        }
         else if  ((35>media) && (media>=30)){
         color="#FF4000";
        }
         else if  ((30>media) && (media>=25)){
         color="#FFBF00";
        }
          else if  ((25>media) && (media>=15)){
          color="#F7FE2E";
        }
          else if  ((15>media) && (media>=5)){
          color="#F7CC5D";
        }
          else if  ((5>media) && (media>=-5)){
          color="#FDFCF9";
        }
        else if  (-5>media){
          color="#1F73EA";
        }

          return color;
}
 catch(err){
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);

 }
}
```

En varias de las funciones principales se puede ver que para coger la elección del tipo de elemento que queremos observar hay una función especial. Se llama **getRadioButtonSelectedValue(),** tiene como parámetro de entrada una tabla que hace referencia al RadioButton de la página Html, Va recorriendo las distintas opciones y devuelve el valor asignado a la elección escogida. La función está definida de la siguiente forma.

```
function getRadioButtonSelectedValue(ctrl)
{
  try
  {
    for(i=0;i<ctrl.length;i++)
        if(ctrl[i].checked) return ctrl[i].value;
        }
  catch(err)
        {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message +","+i+ "\n\n";
  txt+="Click OK to continue.\n\n";

  alert(txt);

  }
}
```

Para crear las fechas que vamos leyendo de registro hay que hacerle un tratamiento de cadenas porque en el fichero de datos vienen con caracteres que sobran. La función **creaFecha()** recoge esta tarea y es de esta forma.

```
function creaFecha(fecha){
  try {
    var m=fecha.split(" ");
    var a=m[0].split("-");
    var b=m[1].split(":");
```

Definimos la función con su nombre un parámetro que es la cadena fecha del registro, es de esta forma, 2013-03-15 11:59:52. A nosotros nos interesa tener cada uno de los elementos por separado. Lo primero es dividir por espacios para después separar por “-” y “:”. Utilizando la función **split()** metemos las dos particiones en las variables a y b, tablas con los elementos

separados por sus correspondientes caracteres.

```
var año=a[0];
var mes=a[1];
var dia=a[2];
var horas=b[0];
var minutos=b[1];
var segundos=b[2];
```

Asignamos a cada variable el factor que le corresponde de las tablas creadas anteriormente.

```
  var fecha=new Date(año,mes,dia,horas,minutos,segundos);
return fecha;
```

Y creamos la variable final fecha con ayuda de la función **Date()** y nuestros parámetros nos devolverá el objeto tal y como nos interesa para poder manipularlo con la función **getTime(),** por ejemplo. Y por último el captador de excepciones.

```
  catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
  }

}
```

Otro elemento que me pareció interesante y bastante funcional es que a la hora de introducir los datos de las latitudes y longitudes no solo puedas introducirlo como número, es decir a mano, sino que el programa te diera la opción de que tu señales o indiques un lugar y esas celdas de datos se rellenaran solas. Anterior mente hemos visto las funciones que lo implementaban. Eran las

funciones **crear_map()** y **codeAddress().**

```
function crear_map(){
try{
        var pam=new google.maps.LatLng(42.8124952, -1.649024000000054 );
        var myOptions = { zoom: 13, center:pam, mapTypeId: google.maps.MapTypeId.SATELLITE };
        var map       = new google.maps.Map( document.getElementById("map_canvas"), myOptions);
        google.maps.event.addListener(map, 'click', function(event) {
                document.formulario1.Latitud.value=event.latLng.lat();
                document.formulario1.Longitud.value=event.latLng.lng();


               });
        return(map);
        }

catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
  }

}
```

```
function codeAddress() {
  try{
  var address = document.getElementById('address').value;
  geocoder.geocode( { 'address': address}, function(results, status) {
    if (status == google.maps.GeocoderStatus.OK) {
      map.setCenter(results[0].geometry.location);
      var marker = new google.maps.Marker({
          map: map,
          position: results[0].geometry.location
      });

        document.formulario1.Latitud.value=results[0].geometry.location.lat();
        document.formulario1.Longitud.value=results[0].geometry.location.lng();


    } else {
      alert('Localizador no se ha ejecutado por esta razón: ' + status);
    }

  });
    }
  catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);

  }
}
```

Como podemos ver ambas funciones tienen lineas de código que hemos visto pero no hemos explicado, son:

```
google.maps.event.addListener(map, 'click', function(event) {
        document.formulario1.Latitud.value=event.latLng.lat();
        document.formulario1.Longitud.value=event.latLng.lng();


       });
```

Sabemos que en la función **crea_map()** teníamos un evento que a la hora de hacer 'click' con el botón izquierdo de ratón ejecutaba la función evento. Esta función introduce en el espacio reservado para la latitud y longitud el valor de la latitud y longitud correspondiente al punto geométrico sobre el que estamos ejecutando el evento. Se hace con la función de Google Maps **event.latlng.lat()** y **lng().**

```
document.formulario1.Latitud.value=results[0].geometry.location.lat();
document.formulario1.Longitud.value=results[0].geometry.location.lng();
```

Y en el caso de **codeAddress()** asigna el valor de la búsqueda realizada por la función a los textfields correspondientes.

En definitiva, puedes buscar un lugar y que te aparezcan sus coordenadas, o pinchar con el ratón sobre un punto o introducirlas directamente.

ANÁLISIS, CRITERIOS Y RESULTADOS

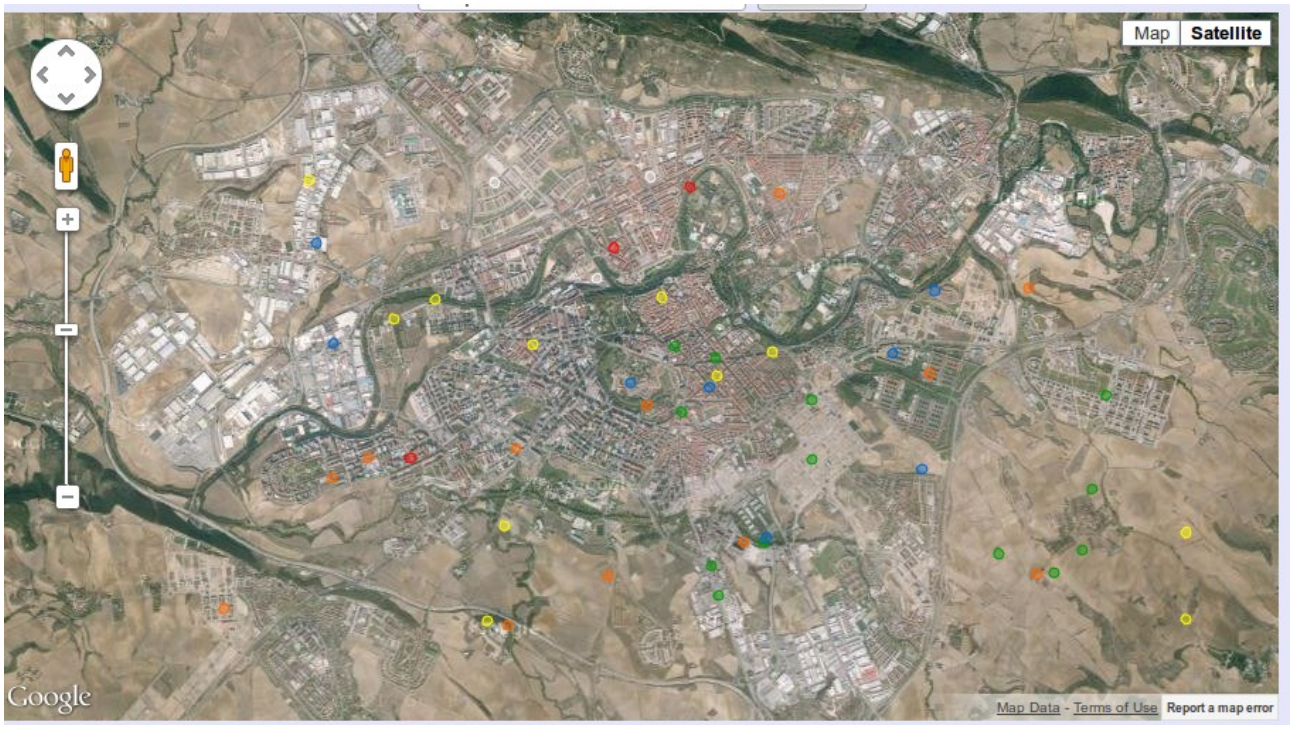
Una vez hecha la representación gráfica tenemos que tener en cuenta que esas formas y figuras tienen que representar algo que se ajuste a la realidad, Si por ejemplo quiero ver la media de la temperatura de una zona tendré que tener datos suficientes para que eso sea representativo. No es lo mismo la media de temperatura de Pamplona teniendo 500 datos registrados que la media de toda España con esos mismos datos. Al final será mas o menos valida en función de estos.

Criterios para evaluar

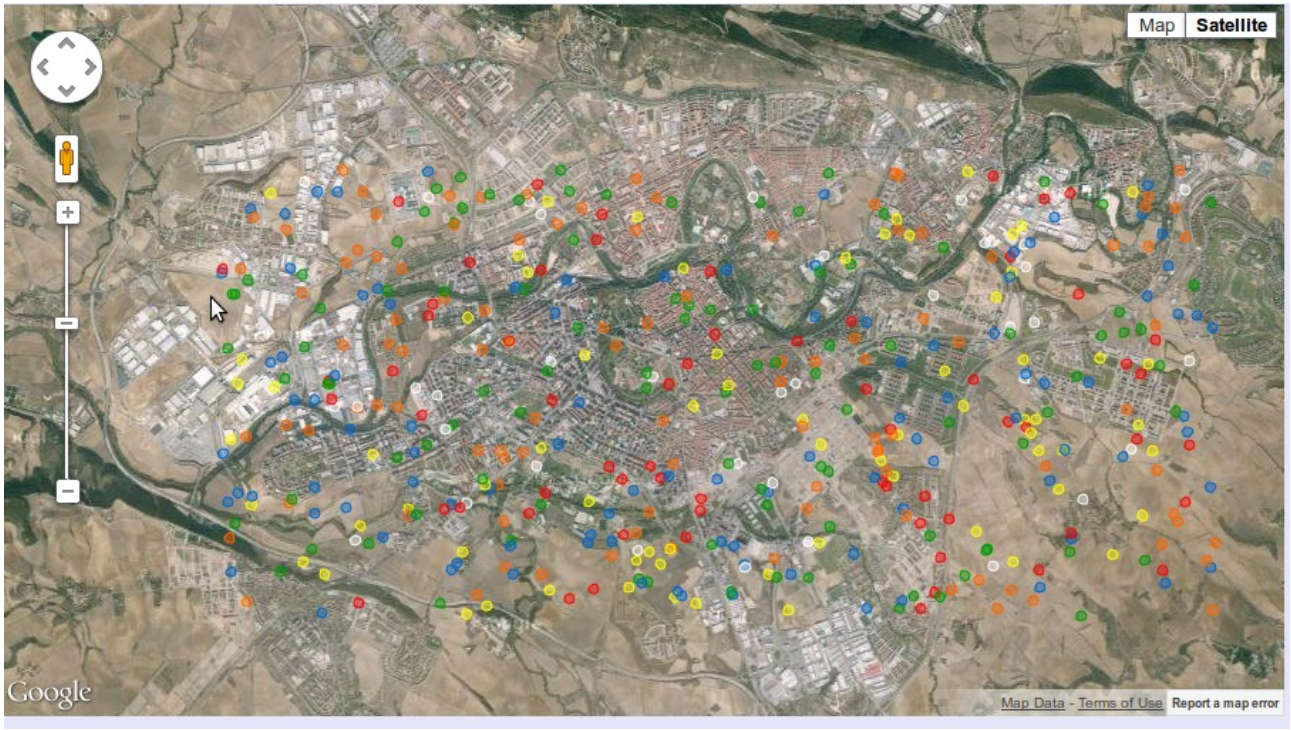
Los principales parámetros que debemos tener en cuenta son el número de datos por zona y como de distribuidos están. Hay que saber diferenciar una buena representación en función de estos valores. Para ello vamos a utilizar las herramientas que hemos desarrollado.

Por ejemplo, entre estas tres diferentes representaciones de puntos, existe una gran diferencia en cuanto a validez de estos a la hora de medirlos gráficamente.

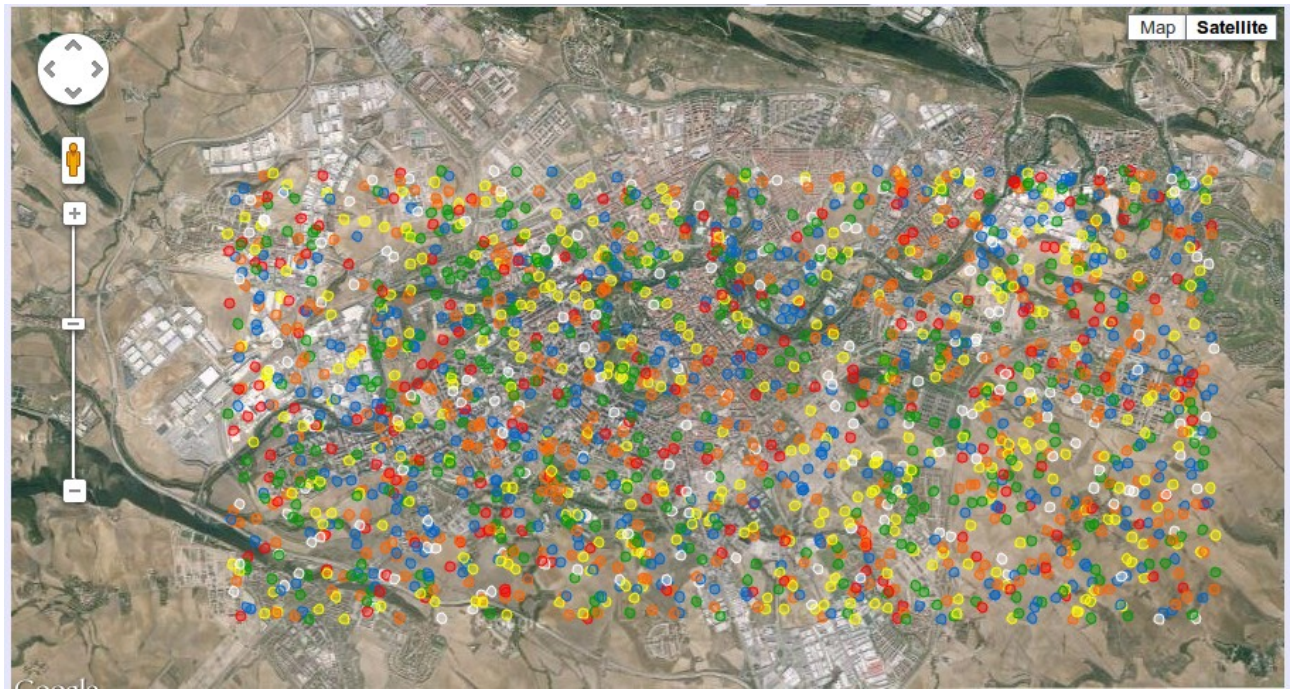
Ejemplo 1



Ejemplo 2



Ejemplo 3



Diferentes estrategias a seguir

A partir de estos datos generaremos diferentes gráficos y aplicando diferentes tipos de parámetros para cada ejemplo veremos que resultados nos dan.

Círculos

Nº Círculos	Radio	Ej.1	Ej. 2	E. 3
1	Pequeño	**X**	**X**	**X**
1	Medio	**X**	**X**	**ok**
1	Grande	**ok**	**ok**	**ok**
5	Pequeño	**X**	**X**	**ok**
5	Medio	**X**	**ok**	**ok**
5	Grande	**ok**	**X**	**X**
10	Pequeño	**X**	**ok**	**ok**
10	Medio	**X**	**X**	**X**
10	Grande	**X**	**X**	**X**

- X, no es buen resultado
- OK, es buen resultado

Teniendo en cuenta lo que hablábamos antes, he hecho unos ejemplos en función del número de círculos, su radio sobre los distintos ejemplos.

El principal problema cuando los datos de los que disponemos no son suficientes es que aunque una de las figuras consiga recoger uno de esos datos puede no ser representativo para el resto de la zona.

Por ejemplo, cuando generamos un círculo pequeño (de unos 50 metros de radio) sobre los distintos ejemplos, existe la posibilidad de que coja un solo dato o incluso ninguno. También puede suceder que la figura sea tan grande, en el caso de recoger muchos puntos en una zona abarque tanto que representa otra zona de la que no tenemos tantos datos. Ese es el caso de los ejemplos de 10 círculos con radio medio y grande.

Sin embargo, hay unos ejemplos que si que me han parecido representativo respecto a los ejemplos. Al generar un círculo grande sobre una zona, sabes que no es representativo porque las zonas pueden variar, pero te sirve para hacer una vista intuitiva,

En el ejemplo de 5 círculos de radio pequeño, no me parecía buena manera de hacerlo ya que ses como el ejemplo de un solo círculo, no hay datos, hasta que lo he hecho cuando había muchos datos. Aunque había algunas franjas que no se rellenaban, las que si lo estaban eran zonas con pocos datos pero muy pequeñas y las que no cogían datos, se podría hacer una estimación con respeto a sus contiguas. Lo mismo pasa cuando hacemos 5 círculos de radio medio y 10 círculos de radio pequeño.

Cuadrículas

Para el caso de las cuadrículas no he hecho un estudio como el anterior porque creo que es mas simple. Cuanto menos error quieres tener hay que hacer mayor número de celdas con menor separación. De las zonas que no hay datos se quedarán vacías. Esta forma de verlo me parece mas interesante cuando el tipo de dato que estamos observando tiene un margen de varianza mayor. Por ejemplo en una misma zona de la ciudad puede existir una zona verde donde el nivel de Co2 en el aire sea muy bajo pero al lado suya pasa una carretera principal donde ese nivel se dispara. Hacer una media de los dos sería una dato muy general, habría que dividirlo en sus zonas.

Resultados

Como resultado, yo diría que el programa te da una pequeña información y que hay que saber interpretarla. Dependiendo del tipo de dato que queremos observar, el número de muestras y como están repartidos hay que saber analizarlo. En este caso los círculos de forma general, y las cuadrículas para una vista mas detallada.

CONCLUSIONES

Una vez finalizado, se ha comprobado que hemos cumplido los requisitos que habíamos definido previamente. Es una programa muy sencillo, sin la necesidad de grandes capacidades de conocimiento para usarlo, que es capaz de reconocer datos recolectados a través de un sensor y tratarlos para después con el construir una estructura que los pueda representar.

El otro objetivo del proyecto era estudiar distintas posibilidades para la representación de esos datos y se ha con seguido con la ayuda de Google Maps V3 y Google Earth. Son dos herramientas que a priori no parecen muy sencillas pero si investigas cual es su estructura se hace muy sencillo y de manera amigable.

LINEAS FUTURAS

Hay que decir que el programa aquí descrito es una versión muy básica y siempre se le van a poder añadir nuevas funcionalidades o mejoras.

Para empezar, se podría diseñar una página Html más moderna, con mas funciones y mas amigable. Diseñando imágenes para el fondo, botones,etc. Creando foros, enlaces o sitios de interés.

A la hora de la implementación se podría analizar la estructura del código para poder optimizar la velocidad de ejecución de las funciones. Por ejemplo creando unos algoritmos de búsqueda o recorrido mas rápidos que los actuales. Hay que tener en cuenta que JavaScript consume recursos de memoria de los navegadores, si tardan mucho en ejecutarse pueden ralentizarlos o incluso bloquearlos.

Sobre las funcionalidades del sistema podemos decir que se pueden ampliar o mejorar. Podemos construir nuevas estructuras gráficas, que nos den resultados estadísticos de los cuales nos sea mas fácil sacar conclusiones, a parte del mero echo de informarnos.

Por último, lo que mas me interesa es saber que a estas estructuras podemos aplicarles unos algoritmos de mayor complejidad para poder hacer estimaciones sobre zonas de las que no tenemos datos. Calcular valores anómalos dentro de un rango o incluso calcular rutas por las cuales nos interese ir según los datos.

BIBLIOGRAFÍA

Manual Xml

1-http://www.desarrolloweb.com/manuales/desarrollo-con-api-de-google-maps.html

Kml y Google Earth

2-https://developers.google.com/kml/documentation/kml_tut?hl=es
3-http://earth.google.es/intl/es/outreach/tutorial_kmlembed.html

JavaScript Y Google Maps

4-http://www.desarrolloweb.com/manuales/desarrollo-con-api-de-google-maps.html
5-https://developers.google.com/maps/documentation/javascript/?hl=es

Otros

6-http://www.w3schools.com/
7-http://bibing.us.es/proyectos/abreproy/5013/fichero/Capitulo+5.pdf

Proyecto Fin De Carrera: "Visualización de Datos Recopilados en Redes Vehiculares Multisensor"

Autor: Alberto Zurbano Gumuzio

Objetivo Principal

- Crear un programa que muestre gráficamente datos respectivos a unas coordenadas.

Posibles Herramientas

- Google Earth y Kml.
- Google Maps con JavaScript.

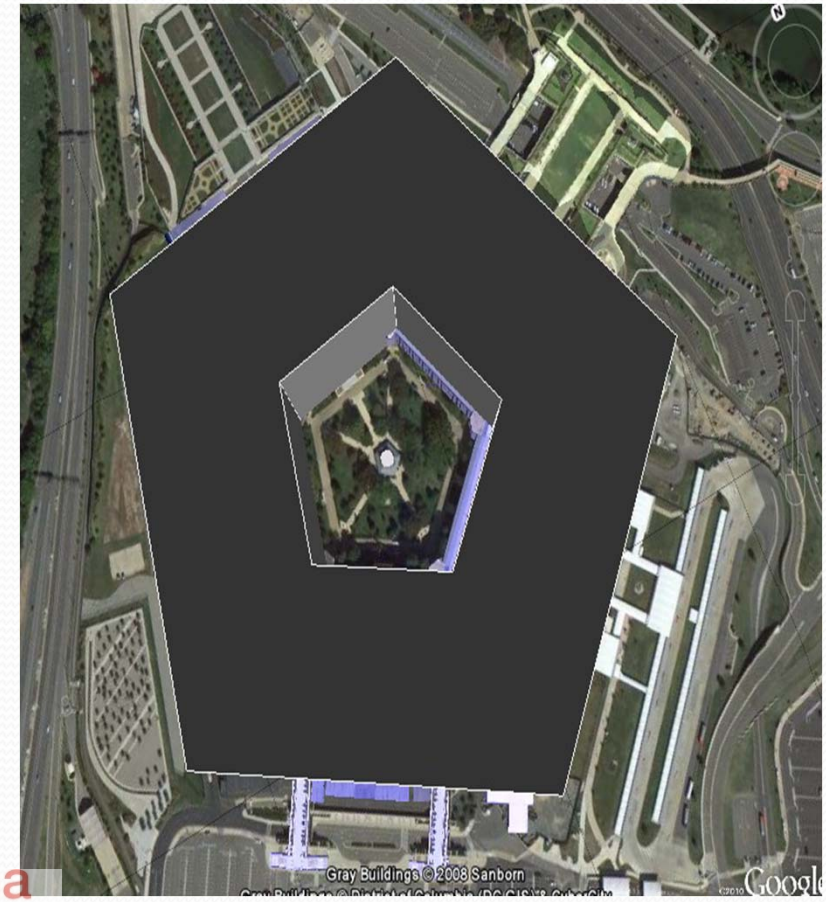
Google Earth

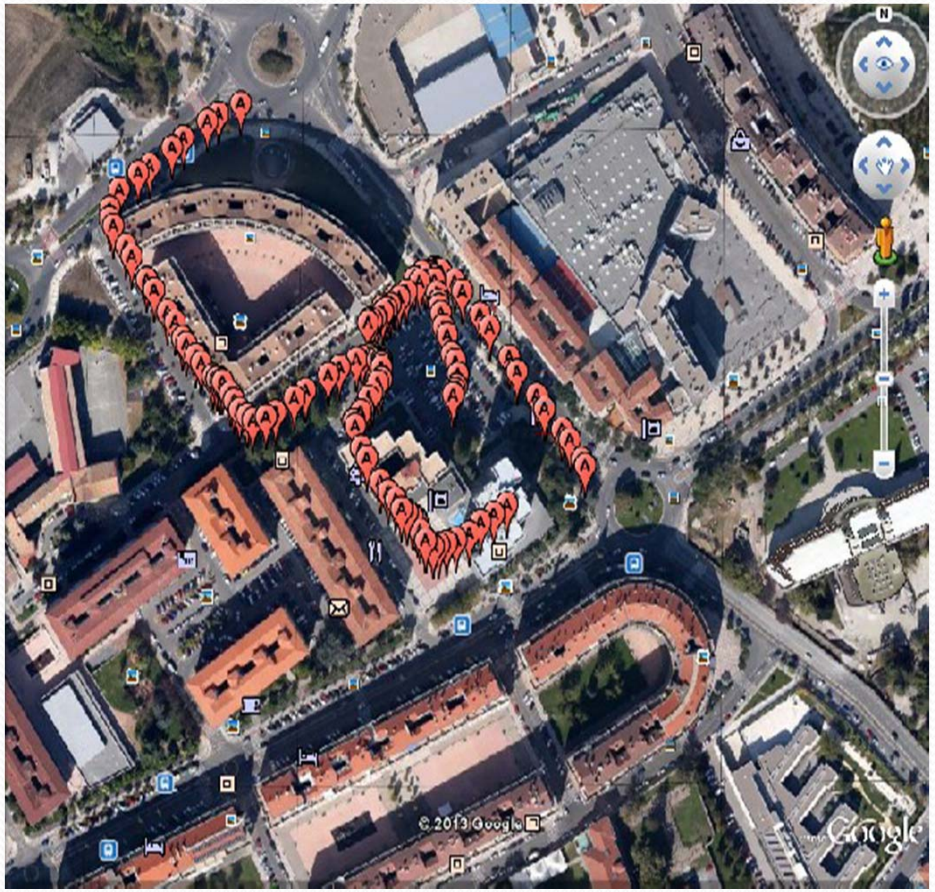
- Keyhole -> Google Earth
- Maps, Street View, 3D, Simulador de Vuelos
- Multilenguaje
- Windows, Mac y Linux
- http://www.google.com/earth/index.html
- Otros: Marble, World Wind, Bing Maps

KML

- **K**eyhole **M**arkup **L**anguage
- Desarrollado para Keyhole LT
- Basado en XML
- Etiquetas:
  - **<Placemark>**
  - **<Point>**
  - **<coordinates>**
- Editor Usuario Google Earth
- Compresión zip, KMZ
- Google Maps (para móviles), Adobe PhotoShop, AutoCAD y otros.

```
<Placemark>
    <name>0.25 km</name>
    <visibility>0</visibility>
    <Snippet maxLines="0"></Snippet>
    <description>Untitled</description>
    <StyleMap>
        <Pair>
            <key>normal</key>
            <Style>
        </Pair>
        <Pair>
            <key>highlight</key>
            <Style>
                <BalloonStyle>
                    <text><![CDATA[<font style="font-size: 11px;">
                  <b>$[name]</b><p>$[description]]]></text>
                    <textColor>ff000000</textColor>
                    <displayMode>default</displayMode>
                </BalloonStyle>
                <LineStyle>
                    <color>ffffff00</color>
                    <width>3</width>
                </LineStyle>
                <PolyStyle>
                    <color>090000aa</color>
                </PolyStyle>
            </Style>
        </Pair>
    </StyleMap>
    <LineString>
        <tessellate>1</tessellate>
        <coordinates>
            -1.67441639699999,42.8078555754125,0 -1.67415001100637,42.8078470114687
        </coordinates>
    </LineString>
</Placemark>
```





Google Maps

- Creado por Google
- Imágenes Desplazables
  - Mapas
  - Satélite
- IE, Firefox, Opera, Safari
- App Móviles
- Gran uso de JavaScript

JavaScript

- Netscape, Mozilla
- Multiparadigma
- Complemento del navegador
- Normalmente Client-Side
- Varias Influencias (C, Java)

Ejemplo

```
function posicion(){
    var tabla = new Array();
    var tabla1 = new Array();
    var txtFile= new XMLHttpRequest();
    txtFile.open("GET", "posicion2.php?", false);
    txtFile.send( null );
    var mySplitResult = txtFile.responseText.split("<Posicion>");
    var datos = mySplitResult[1].split("><");
    for(i = 0; i < 2; i++){
        var coor    = datos[i].split(">");
        tabla[i]    = coor[1];
    }
    var coorx       = tabla[0].split("<");
    var coory       = tabla[1].split("<");
    var position = new google.maps.LatLng(coorx[0], coory[0]);
    return (position);
    }
```

Ejemplo

```
<script type="text/javascript">
var map;
var timer;
var poligono;
var pos = posicion();
var multipuntos;
var geocoder;
function initialize() {
    map = crear_map();
    geocoder = new google.maps.Geocoder();
}
</script>
```

Ejemplo

```
<form name="formulario1" id="formulario1">
    <input type="button" value="Circulo"  onclick="creaCirculos()" >
    Latitud : <input type="text" name="Latitud"  size="15" value="42.8124952">
    Longitud : <input type="text" name="Longitud"  size="15" value="-1.649024000000054">
    Círculos : <input type="text" name="Circulos" size="2"  value="4">
    Radio : <input type="text" name="Radio"  size="5"  value="400">
</form>
```

Datos

- Estructurados
- Ordenados
- Etiquetados

```
<Posicion><Latitud>42.8012</Latitud><Longitude>-1.61115</Longitude><Humedad>50</Humedad>
<Temperatura>22</Temperatura><Polucion>3</Polucion><Date>2013/03/15 12:0:1</Date></Posicion>
<Posicion><Latitud>42.821</Latitud><Longitude>-1.58149</Longitude><Humedad>44</Humedad>
<Temperatura>22</Temperatura><Polucion>4</Polucion><Date>2013/03/15 12:0:2</Date></Posicion>
<Posicion><Latitud>42.8167</Latitud><Longitude>-1.69056</Longitude><Humedad>51</Humedad>
<Temperatura>21</Temperatura><Polucion>5</Polucion><Date>2013/03/15 12:0:3</Date></Posicion>
<Posicion><Latitud>42.7946</Latitud><Longitude>-1.62028</Longitude><Humedad>40</Humedad>
<Temperatura>2</Temperatura><Polucion>11</Polucion><Date>2013/03/15 12:0:4</Date></Posicion>
```

Características

- Html y Javascript
- Un solo fichero para cada uno
- Elementos
  - Google Maps V3
    - Geocoder
    - Funciones (círculo, polígono, marcador, mapa)
  - Propios
    - Captura de Latitud Longitud
    - Factor Tiempo
    - Basado en Botones

Humedad: Temperatura: CO2:

Pamplona Localiza

Latitud : 42.79565257740659 Longitud : -1.6896629333496094 Horas : 1

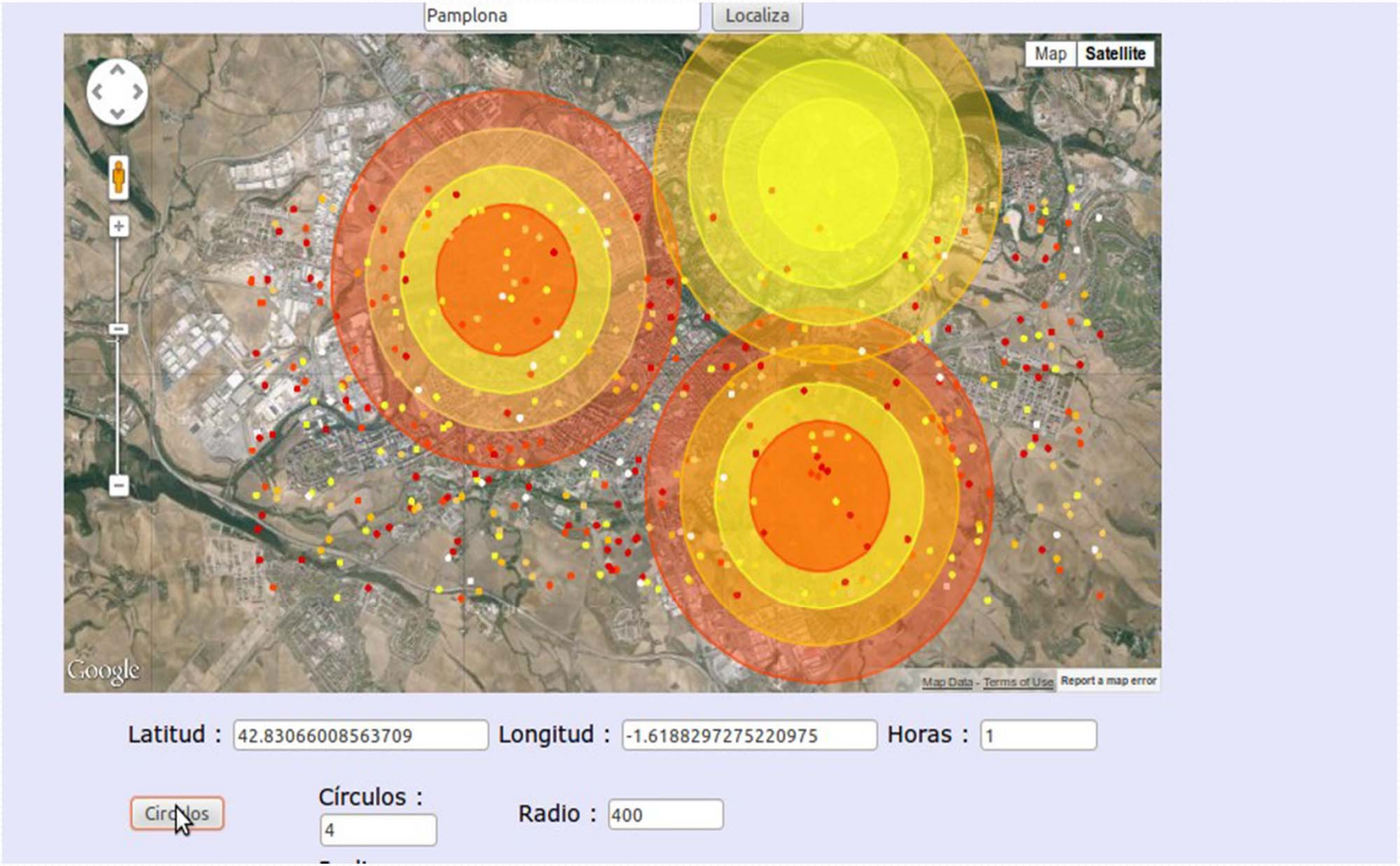
Circulos Círculos : 4 Radio : 400

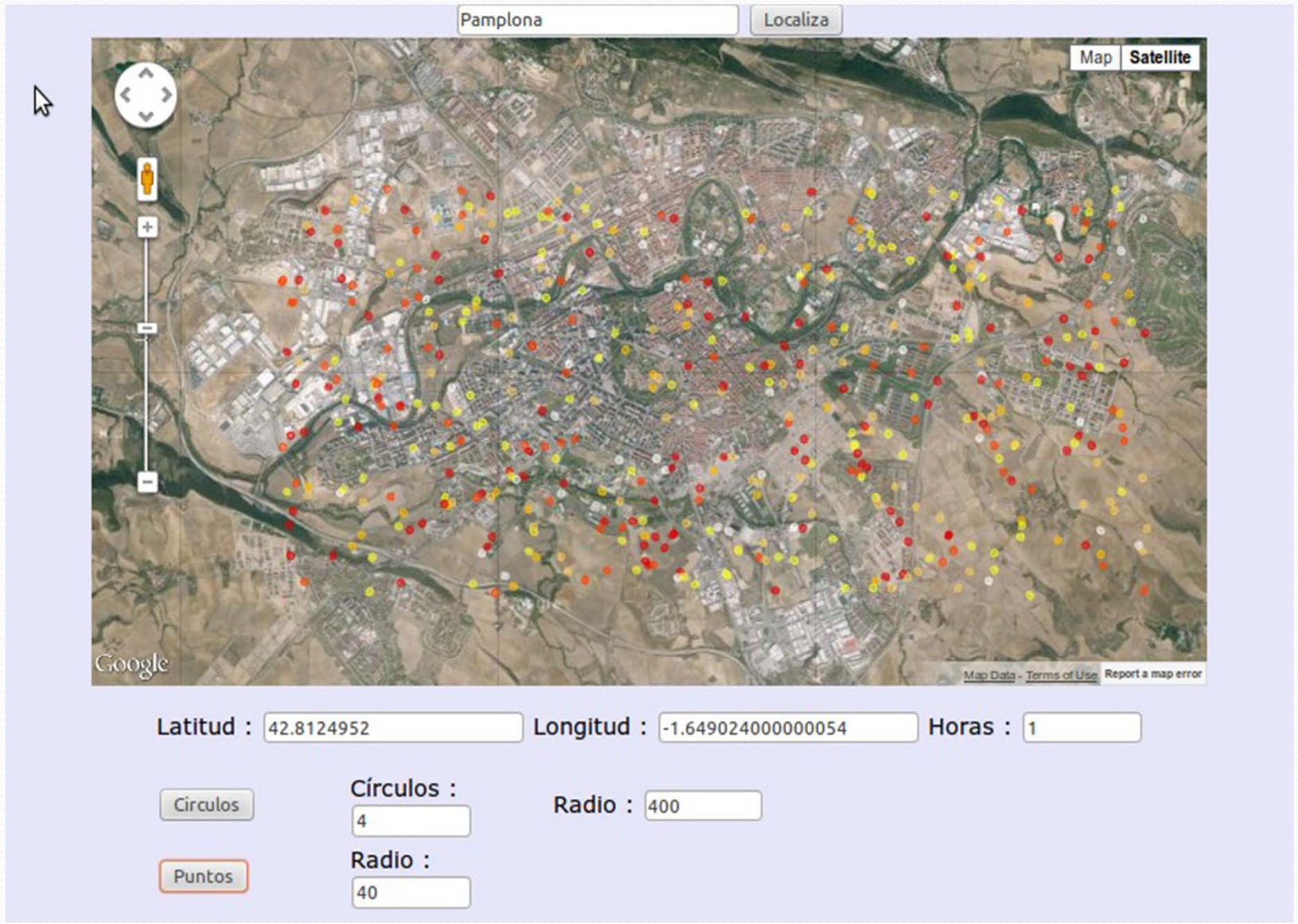
Puntos Radio : 30

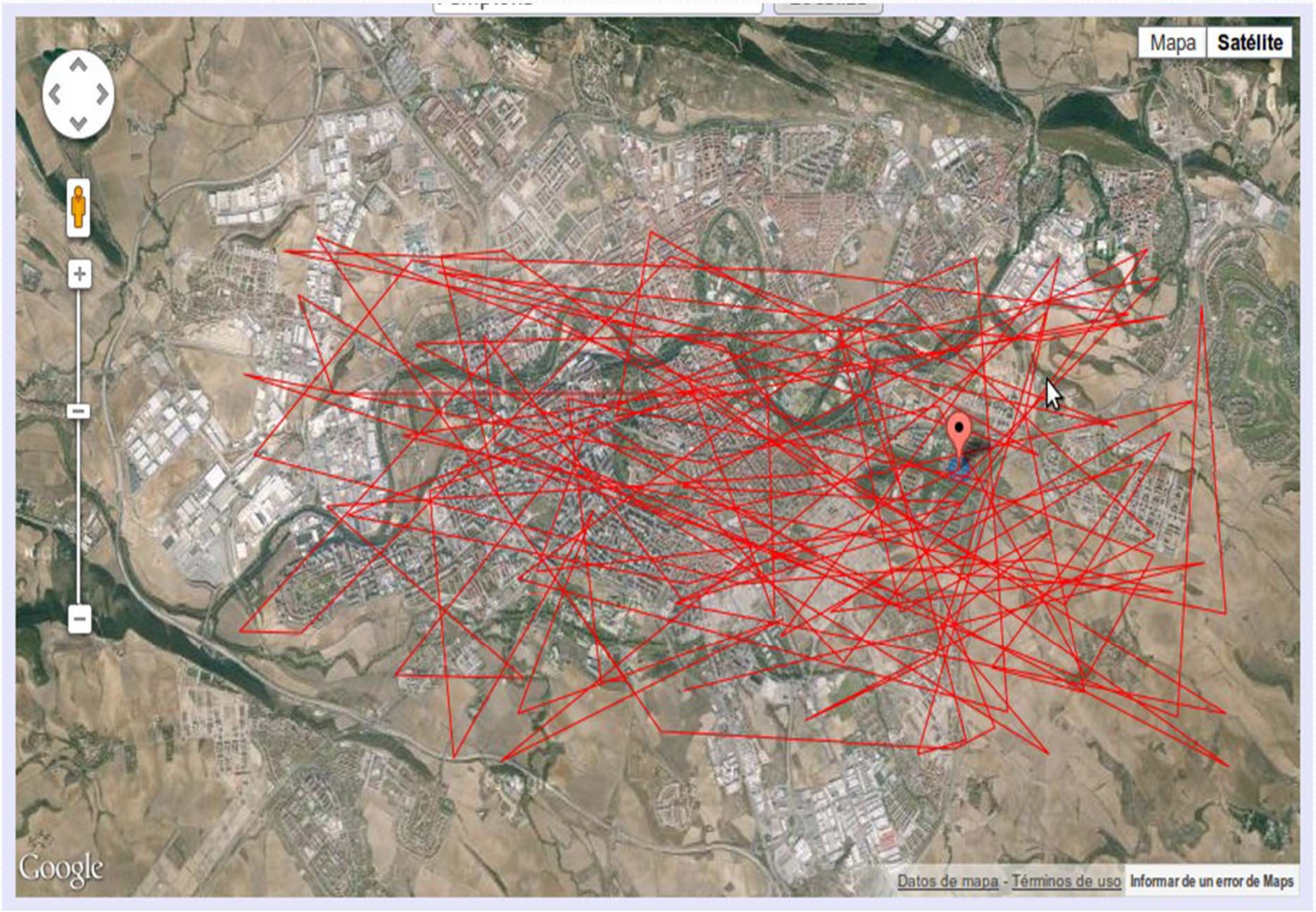
Ruta Radio : 100

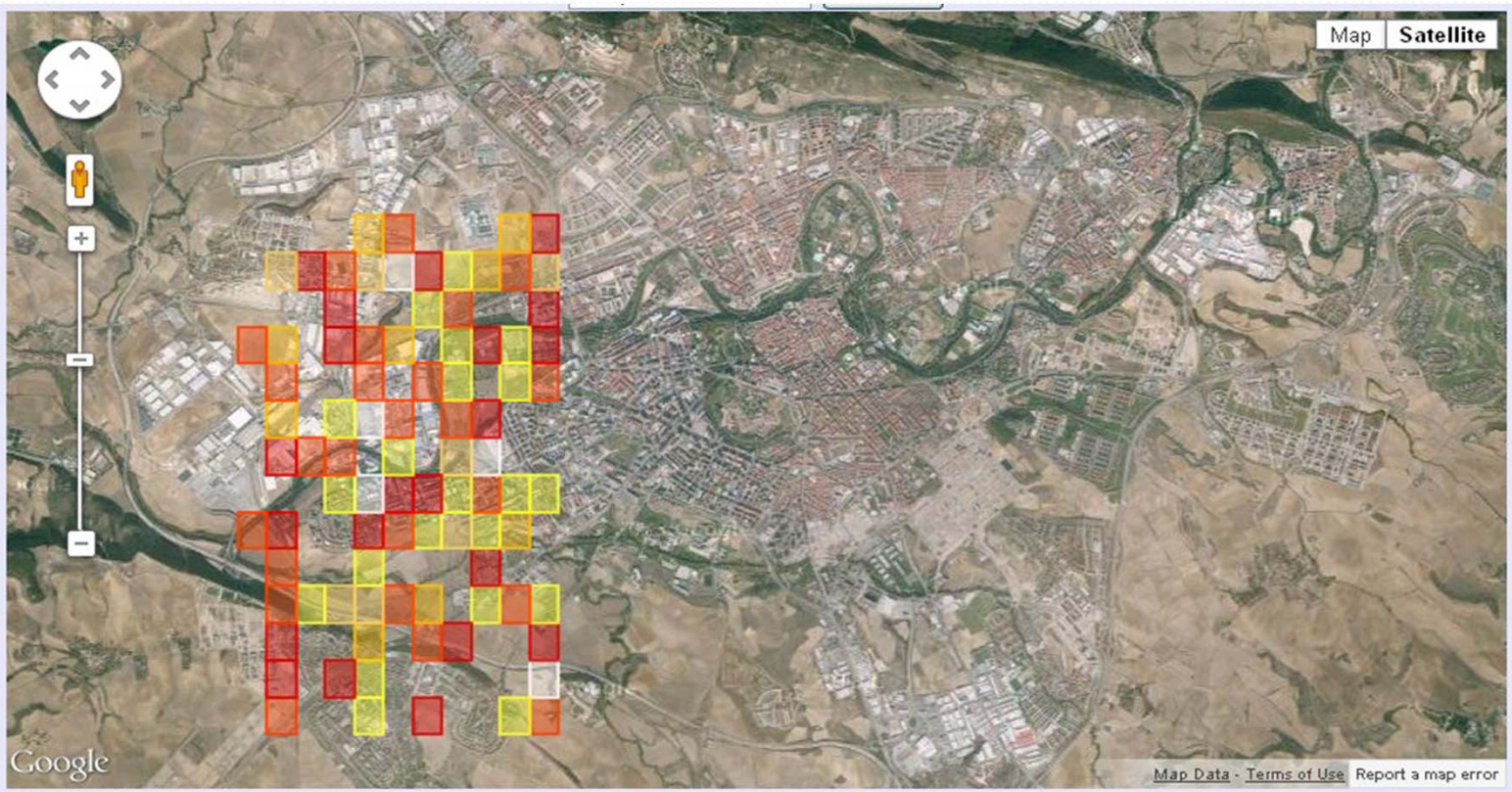
Cuadrículas Filas : 15 Columnas : 15 Separación : 30

Borrar y Volver









Latitud : 42.809506838324204 Longitud : -1.69189453125 Horas : 1

Circulos Círculos : 4 Radio : 400

Puntos Radio : 5

Ruta Radio : 100

Cuadrículas Filas : 15 Columnas : 15 Separación : 30

Mejoras Futuras

- Web.
- Optimizar Código.
- Ampliar o mejorar funcionalidades actuales.
- Nuevas estructuras gráficas.
- Estadísticas.