# *Plugin* en QGIS para automatizar la completitud semántica vectorial a partir de datos OpenStreetMap®

**Trabajo Final de Grado**

**Carlos Lapuente Serra**
*Servicios basados en localización y espacios inteligentes*
Grado de Tecnologías de la Telecomunicación

Dirigido por la consultora: **Anna Muñoz Bollas**
Profesor resp. asignatura: **David Merino Arranz**

5 de junio de 2018

# FICHA DEL TRABAJO FINAL

| | |
|---|---|
| **Título del trabajo:** | **Plugin en QGIS para automatizar la completitud semántica vectorial a partir de datos OpenStreetMap®** |
| **Nombre del autor:** | **Carlos Lapuente Serra** |
| **Nombre del consultor:** | **Anna Muñoz Bollas** |
| **Fecha de entrega (mm/aaaa):** | ***06 /2018*** |
| **Área del Trabajo Final:** | **Servicios basados en localización y espacios inteligentes** |
| **Titulación:** | **Grado de Tecnologías de la Telecomunicación** |

**Resumen del Trabajo (máximo 250 palabras):**


Este documento es el resultado del trabajo final de carrera para el Grado de Tecnologías de la Telecomunicación, especialidad en Telemática por la *Universitat Oberta de Catalunya*, en concreto para el área de Servicios basados en localización y espacios inteligentes.

La realización de este proyecto de fin de carrera permitirá conocer el funcionamiento de QGIS 3.0. (Sistema de Información Geográfica de código libre) y de un plugin creado especialmente para poder automatizar la comprobación de geometrías vectoriales a partir de datos abiertos de OpenStreetMap (OSM) y los de otra base de datos topográfica, como por ejemplo la del *Institut Cartogràfic i Geològic de Catalunya* (ICGC).

El plugin ha sido creado en lenguaje Python aplicado a Sistemas de Información Geográfica (PYQGIS). La ejecución del plugin, permitirá a un usuario importar los puntos de interés (PoI o *Point of Interest*) con la información semántica (escuelas, hospitales, tiendas, etc) desde la Base Topográfica de OSM hacia otras bases topográficas, siempre y cuando dichos puntos coincidan con polígonos (edificios).

El resultado, es almacenado en una nueva capa o base de datos cuya información es susceptible a su vez de ser exportada a cualquier otra base de datos topográfica compatible con dicho formato.

**Abstract (in English, 250 words or less):**

In the current technological framework with the increase of the computational capacities of servers and personal computers, the bandwidth of Internet network connections and the working capacities in the cloud, has allowed to create new services and improve existing ones.

The improvement in visualization fluidity, loading of maps, vector semantic data layers of topographic databases and their operation engine have allowed a greater demand and growth of users from any electronic device and for a multitude of uses (agriculture, transport, fleet management, urban, etc.).
The biggest ideological change that has allowed us to exponentially improve the volume of existing data and constant improvement has been the collaborative work of open, shared data and open source platforms in which thousands of users can contribute disinterestedly data, modifications and corrections constantly for an improvement of existing information.

This project presents the possibility that, by means of a tool developed for QGIS in Python language, a user can easily import semantic data of points of interest from a topographic base of open data to another topographic base.

The reason for having addressed this problem has arisen because the largest and most powerful web map application server, Google Maps, even being free to use, his data and licenses are private and not open source. This is why this project has used open source geographic information systems such as the QGIS application together with topographic databases with open data such as OpenStreetMap or those of the *Cartographic and Geological Institute of Catalunya* (ICGC).

**Palabras clave:**

Matching vectorial, OpenStreetMap, plugin, Spatial Join, QGIS 3.0., ICGC, SIG, Python.

*Quisiera agradecer a todos los que me han dado su apoyo en estos últimos años durante el laborioso camino de la carrera hasta llegar aquí, al proyecto final de grado. Especialmente a Deby, a mis padres, hermanos, familiares y amigos que han tenido paciencia, comprensión y me han soportado en los momentos difíciles. Espero que el tiempo que no he podido dedicarles en este periodo pueda recuperarlo pronto y seguir disfrutando junto a ellos.*

*También un agradecimiento especial a mi tutora Anna por su soporte.*

# Índice

# Ilustraciones

# 1. INTRODUCCIÓN

En este primer capítulo se van a detallar las secciones más importantes que describen este proyecto así como sus objetivos, alcance, planificación y productos obtenidos, entre otros. En ellos se puede obtener una visión general del dominio que abarca el proyecto además de la familiarización de elementos que se van utilizar.

## 1.1 Descripción del proyecto y contexto

En este apartado se mostrarán cuáles son los objetivos, los resultados esperados y el alcance del proyecto.
Este Trabajo Final de Grado (TFG) consiste en implementar herramientas sencillas de *matching* vectorial entre los datos de la base de datos OpenStreetMap [1] (OSM) y los datos de alguna otra Base Topográfica con datos abiertos, como por ejemplo la del Instituto Cartográfico y Geológico de Cataluña, para enriquecer semánticamente los objetos geográficos de la base topográfica escogida.

Como objetivo general se analizaran e implementaran métodos para comparar la información vectorial de objetos de tipos punto, línea y polígono de forma que se calcule el grado de similitud o coincidencia entre los objetos.
Como objetivo mínimo del TFG, se implementarán métodos automáticos que permitan enriquecer elementos categorizados como edificios de la base topográfica a los que se pueda añadir semántica como por ejemplo indicando el tipo de edificio que representa: escuela, hospital, polideportivo, iglesia, etc.

La captura de información de manera voluntaria (VGI) y colaborativa es un fenómeno que se enmarca dentro del aumento en la disponibilidad de uso de los receptores GPS, ya sea integrados dentro de los teléfonos móviles o como dispositivos independientes. Uno de los proyectos más ambiciosos y con más recorrido dentro de la VGI es OpenStreetMap.

Para la realización de este proyecto se utilizarán, entre otros, un Sistema de Información Geográfica o SIG [2] *open source* como QGIS **[1]**. QGIS (anteriormente llamado también Quantum GIS) es un Sistema de Información Geográfica de código libre para plataformas GNU/Linux, Unix, Mac OS, Microsoft Windows y Android. Era uno de los primeros ocho proyectos de la Fundación OSGeo y permite manejar formatos *raster* y vectoriales a través de las bibliotecas GDAL **[2]** y OGR, así como bases de datos.

[1] OpenStreetMap, también conocido con el acrónimo OSM, es un proyecto colaborativo para crear mapas de contenido libre usando datos obtenidos mediante dispositivos GPS móviles, ortofotografías y otras fuentes de datos. Los usuarios registrados pueden subir sus trazas GPS y crear y corregir datos vectoriales mediante herramientas creadas por la comunidad OSM, como la aplicación JOSM.

[2] Un SIG o Sistema de Información Geográfica es un conjunto de herramientas que integra y relaciona diversos componentes (usuarios, hardware, software, procesos) que permiten la organización, almacenamiento, manipulación, análisis y modelización de grandes cantidades de datos procedentes del mundo real que están vinculados a una referencia espacial, facilitando la incorporación de aspectos sociales-culturales, económicos y ambientales para una toma de decisiones más eficaz.

## 1.2 Objetivos

Con la realización de este Trabajo Final de Grado se espera adquirir las capacidades necesarias para implementar algoritmos muy sencillos de *matching* entre geometrías de tipos punto, línea y polígono sobre datos OSM para enriquecer semánticamente una base de datos espacial externa. Por otro lado, adquirir las capacidades necesarias para crear, manipular y analizar la información espacial con un SIG, concretamente con QGIS 3.0 i alcanzar los siguientes objetivos generales y específicos:

### *1.2.1 Objetivos generales*

- Comprender los conceptos de la tecnología SIG **[3]** y su metodología.
- Conocer la estructura de los diferentes tipos de datos con los que trabaja un SIG y el concepto de topología.
- Aprender las habilidades necesarias para encontrar, generar y manipular datos geográficos.
- Saber plantear un proyecto SIG.
- Conocer de manera práctica las operaciones de análisis vectorial.
- Implementar algoritmos sencillos de *matching* entre geometrías sobre OSM.

### *1.2.2 Objetivos específicos*

- Analizar las nuevas funcionalidades de la versión 3.0 de QGIS.
- Conocer los diferentes mecanismos de consulta y descarga de la información de la base de OSM y del ICGC.
- Implementar algoritmos de *matching* entre las geometrías de OSM y las de otra base de datos espacial, en un *plugin* por QGIS 3.0 escrito en Python.

## 1.3 Resultados esperados

Una vez finalizado el proyecto, se espera haber alcanzado los siguientes resultados:

- Determinación de un *plugin* con un algoritmo funcional que permita determinar las similitudes entre geometrías vectoriales (punto, línea o polígono).
- Código del algoritmo escrito en Python del plugin para QGIS 3.0.
- Memoria del proyecto.
- Presentación virtual.

## 1.4 Alcance

Se considera dentro del alcance de este proyecto:

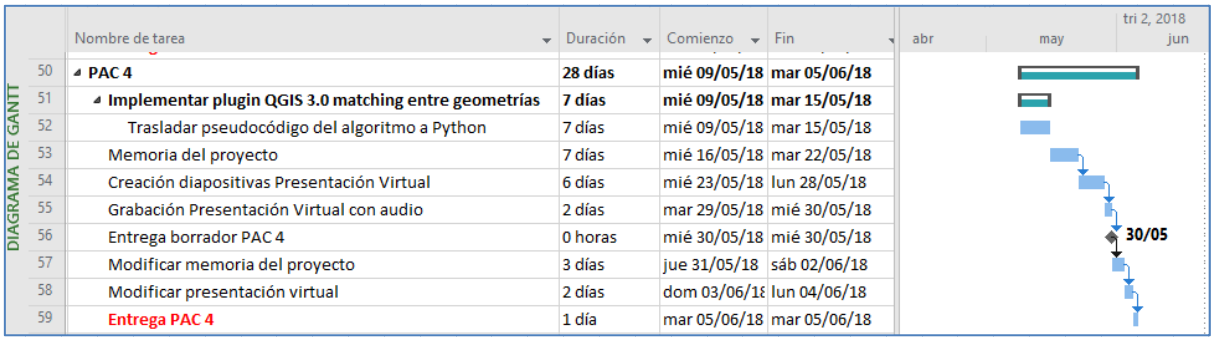
- Estudio de los SIG y de su metodología, así como entender diferentes aspectos de la representación cartográfica: escala, topografía, orientación, aspecto, etc. y de la estructura de los diferentes tipos de datos.
- Recopilación, búsqueda y tratamiento de los datos (mapas, ortofotomapas, etc.) que se utilizarán como base para el análisis.
- Familiarización con el diccionario de datos OSM.
- Análisis de los diferentes métodos de acceso a la información de la base de datos OSM.
- Análisis de la estructura de los datos de la base de datos OSM u otra Base Topográfica y de los tipos de elementos que contiene.
- Investigación otras fuentes de datos abiertos, como las del ICGC o la web de OpenData de la Generalitat de Cataluña.
- Investigación de algoritmos sencillos de *matching* entre varias geometrías.
- Investigación de metodologías para definir el grado de similitud entre las geometrías que se comparan.
- Investigación de plugins en QGIS que realicen tareas parecidas, como por ejemplo *Spatial Join* (antiguo *fTools*).
- Investigación de las herramientas de consulta y *matching* automatizados en QGIS.
- Desarrollo mediante el complemento o generador de plugins *Plugin Builder* de un *plugin* en Python para QGIS 3.0 que realice el *matching* entre geometrías.
- Redacción de la memoria del proyecto.

# 1.5 Planificación del Trabajo

En este apartado se muestra el calendario detallado para esta fase final de la memoria, la PAC 4.

## *1.5.1 Calendario de la PAC 4*

En la siguiente ilustración se muestra la programación establecida para la parte de la memoria del proyecto correspondiente a la PAC 4.

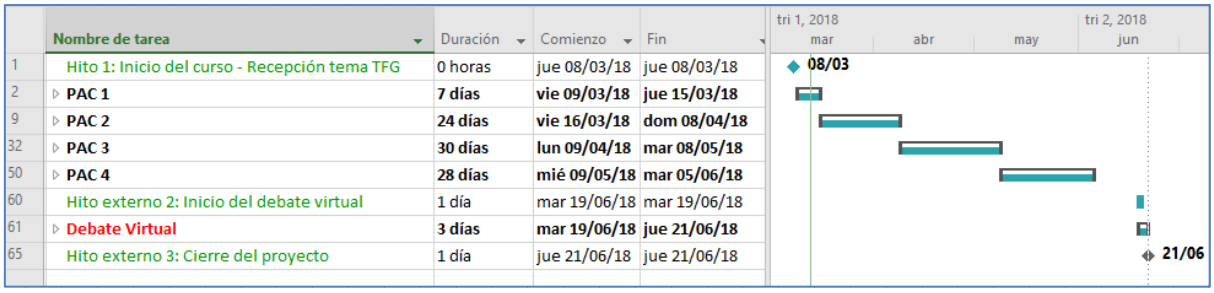


| | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| 50 | **PAC 4** | **28 días** | **mié 09/05/18** | **mar 05/06/18** |
| 51 | **Implementar plugin QGIS 3.0 matching entre geometrías** | **7 días** | **mié 09/05/18** | **mar 15/05/18** |
| 52 | Trasladar pseudocódigo del algoritmo a Python | 7 días | mié 09/05/18 | mar 15/05/18 |
| 53 | Memoria del proyecto | 7 días | mié 16/05/18 | mar 22/05/18 |
| 54 | Creación diapositivas Presentación Virtual | 6 días | mié 23/05/18 | lun 28/05/18 |
| 55 | Grabación Presentación Virtual con audio | 2 días | mar 29/05/18 | mié 30/05/18 |
| 56 | Entrega borrador PAC 4 | 0 horas | mié 30/05/18 | mié 30/05/18 |
| 57 | Modificar memoria del proyecto | 3 días | jue 31/05/18 | sáb 02/06/18 |
| 58 | Modificar presentación virtual | 2 días | dom 03/06/18 | lun 04/06/18 |
| 59 | Entrega PAC 4 | 1 día | mar 05/06/18 | mar 05/06/18 |



**Ilustración 1:** Calendario de la PAC 4.

## *1.5.2 Calendario resumen del proyecto*



| | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| 1 | Hito 1: Inicio del curso - Recepción tema TFG | 0 horas | jue 08/03/18 | jue 08/03/18 |
| 2 | **PAC 1** | **7 días** | **vie 09/03/18** | **jue 15/03/18** |
| 9 | **PAC 2** | **24 días** | **vie 16/03/18** | **dom 08/04/18** |
| 32 | **PAC 3** | **30 días** | **lun 09/04/18** | **mar 08/05/18** |
| 50 | **PAC 4** | **28 días** | **mié 09/05/18** | **mar 05/06/18** |
| 60 | Hito externo 2: Inicio del debate virtual | 1 día | mar 19/06/18 | mar 19/06/18 |
| 61 | **Debate Virtual** | **3 días** | **mar 19/06/18** | **jue 21/06/18** |
| 65 | Hito externo 3: Cierre del proyecto | 1 día | jue 21/06/18 | jue 21/06/18 |



**Ilustración 2:** Calendario resumen del proyecto.

## *1.5.3 Calendario completo del proyecto*

El calendario completo del proyecto se puede observar en el Anexo 1 adjunto al final de esta memoria.

## 1.6 Breve sumario de productos obtenidos

- El presente documento de **memoria** del proyecto del plugin de *matching vectorial.*
- **Código fuente** en lenguaje Python del algoritmo "*matching_vectorial.py*".
- Carpeta contenedora del **plugin** "*matchingvectorial*" con los archivos necesarios para su ejecución (archivos icon.png, *matching_vectorial.py*, *matching_vectorial_dialog.py*, *matching_vectorial_dialog_base.ui*, *metadata.txt, resources.py* y *resources.qrc*, como los más importantes).
- Documento de descripción de carga, uso y funcionamiento del plugin.
- Vídeo de presentación del producto y demostración del funcionamiento.

# 2. Preparación del entorno de trabajo

En esta sección se muestran los pasos a seguir para preparar el entorno de trabajo básico para poder desarrollar, personalizar y comprobar la funcionalidad del plugin.
Se dan por entendidos los requisitos mínimos de máquina (hardware) y programas (software) establecidos en el Plan de Trabajo en la sección "Requisitos de Material".

## *2.1 Instalar QGIS + otro software necesario (Plugin Builder)*

En primer lugar es imprescindible disponer de la plataforma de sistema de información geográfica. Para ello se debe acceder a la página web del proyecto QGIS (https://www.qgis.org/es/site/) y descargar la última versión del software (QGIS 3.0.2 'Girona, en el momento de la redacción de esta memoria). Como se observa en las siguientes imágenes, se ha elegido el instalador autónomo de QGIS versión 3.0 (64 bit) por la compatibilidad con el equipo con el que se va a trabajar:


**Ilustración 3:** Página de descarga de QGIS 3.0.


**Ilustración 4:** Selección de instalable adecuado al sistema operativo.

El instalador incorpora los siguientes paquetes: *QGIS, ECW Raster Plugin for GDAL, MrSID Raster Plugin for GDAL, SZIP compression library* y *Oracle Instant Client.*

GDAL[3] o ***Geospatial Data Abstraction Library*** (también conocida como GDAL/OGR) es una biblioteca de software para la lectura y escritura de formatos de datos geoespaciales. Algunas de las aplicaciones que usan GDAL para el acceso a datos geográficos son QGIS, Google Earth[4], GRASS GIS[5], Kosmo[6] o MapGuide Open Source[7].

[3] https://es.wikipedia.org/wiki/GDAL
[4] https://es.wikipedia.org/wiki/Google_Earth
[5] https://es.wikipedia.org/wiki/GRASS_GIS
[6] https://es.wikipedia.org/wiki/Kosmo
[7] https://es.wikipedia.org/wiki/MapGuide_Open_Source

Una de las ventajas que nos ofrece el SIG *open source* QGIS es la posibilidad de añadir diferentes tipos de "plugins" que aumentan y mejoran la capacidad que ya de por sí ofrece este SIG. Los "plugins" son pequeños programas que se acoplan y adaptan a otros mejorando su funcionalidad y complementándolos contribuyendo con nuevas características, facilitando de esta manera la labor al usuario.

Para comenzar a utilizarlos, primero se deben descargar, instalar y activar en QGIS.
En primer lugar es necesario tener instalado QGIS como se ha comentado anteriormente. A continuación, se añade el plugin llamado "Plugin Builder" que se tiene que descargar del repositorio de plugins de Python[8].

Una de las características que ofrece QGIS es que se pueden añadir los plugins directamente desde la opción "Complementos". Como se observa en la imagen a continuación, presionando en la opción de menú se encuentra la opción "Administrar e instalar complementos":

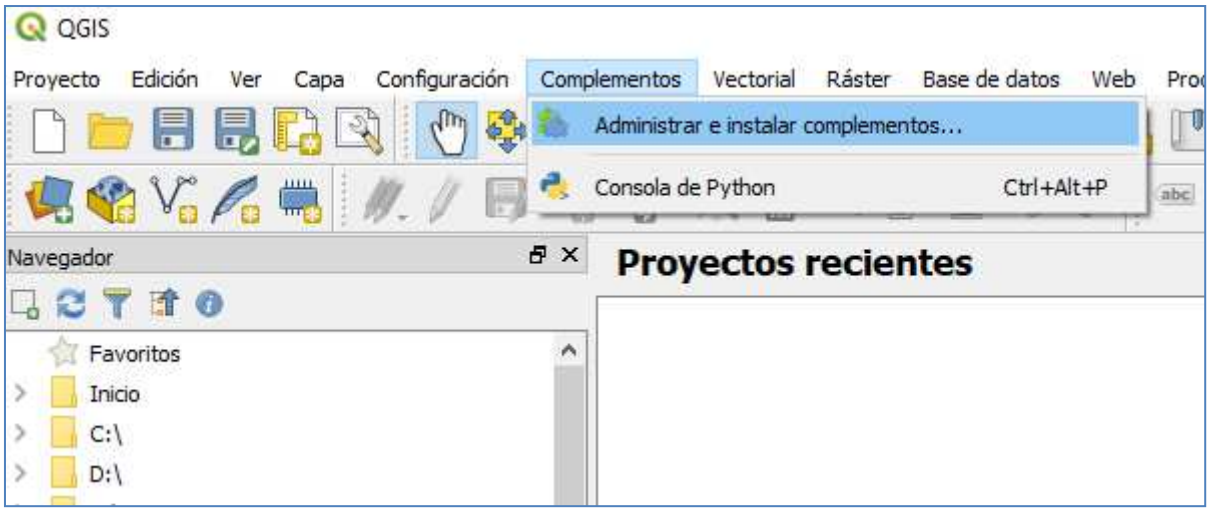



**Ilustración 5:** Menú del *Administrador de complementos* de QGIS.

Como muestra la siguiente ilustración, una vez abierta la ventana de menú es posible buscar en la lista el plugin "*Plugin Builder*" y seleccionar "*Instalar complemento*". Se puede visualizar en la columna derecha la información sobre el plugin seleccionado. En este caso la información concuerda con la función que se necesita.

[8] https://plugins.qgis.org/plugins/pluginbuilder/

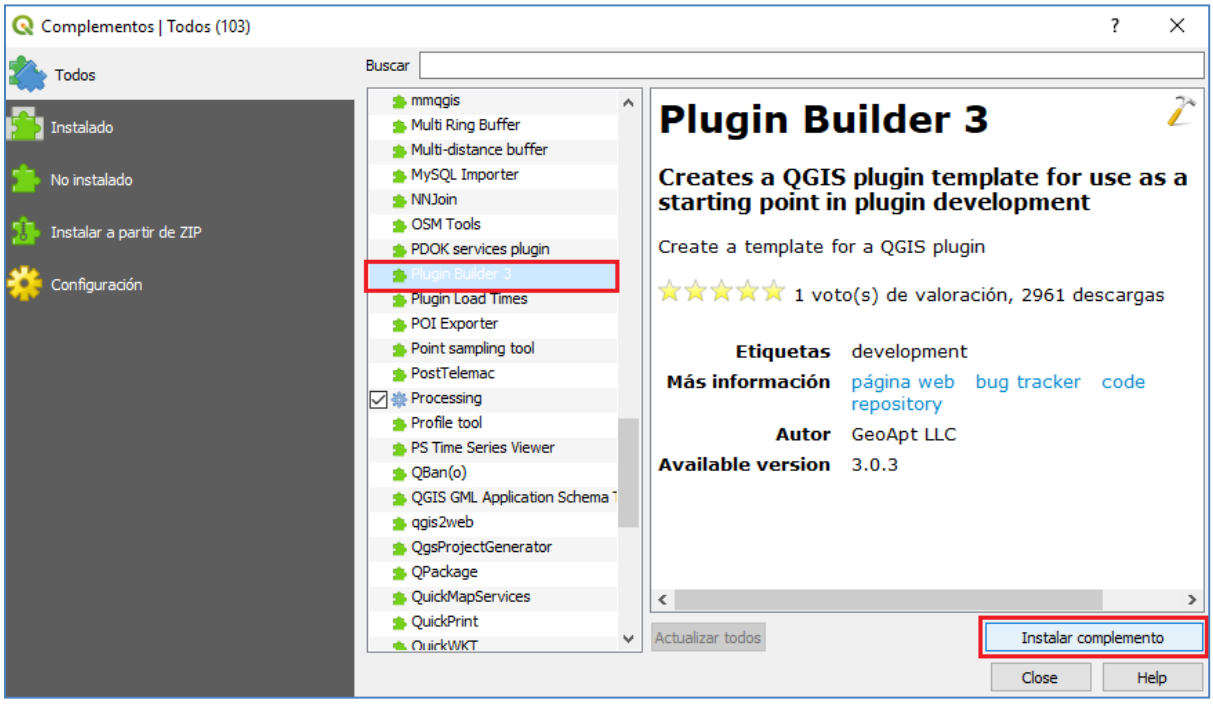



**Ilustración 6**: Ventana *Complementos* de QGIS para seleccionar el *Plugin Builder.*

Una vez realizado, se puede comprobar cómo ya aparece el plugin agregado en la opción de *Complementos*. En la siguiente imagen se observa la nueva opción:

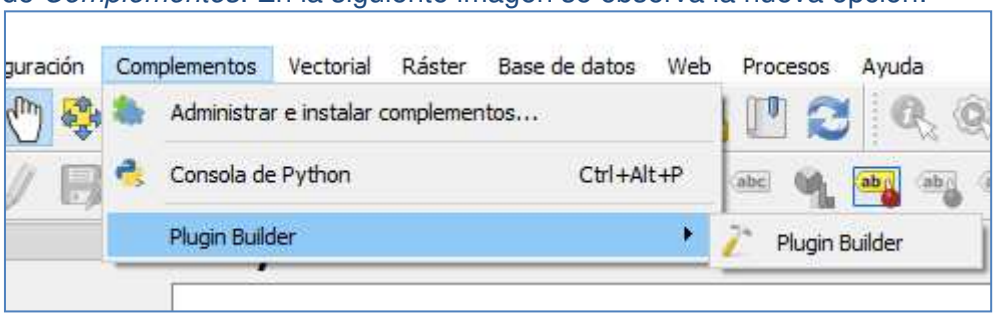



**Ilustración 7:** Selección del *Plugin Builder.*

Este nuevo plugin cargado permite construir o crear una plantilla o estructura básica en QGIS para usar como punto de partida del desarrollo del plugin. Se trata de la versión 2.18.0 que requiere la versión mínima de QGIS 2.0 (en este proyecto se dispone de la versión 3.0.1.). Se puede encontrar el código fuente del "Plugin Builder" en la web de Github[9] en la siguiente dirección: https://github.com/g-sherman/Qgis-Plugin-Builder.

Este plugin generador de plugins que está disponible en la librería, se ha utilizado en este proyecto para construir el esqueleto o plantilla del plugin que va a realizar el ejemplo de código "Hola Mundo". Este ejercicio puede ser de utilidad para el usuario novel que quiera inicializarse en el desarrollo de un plugin.

[9] https://github.com

## 2.2 Novedades de QGIS 3.0

Debido a que previo a la realización de este proyecto se desconocía la existencia y funcionamiento del software de QGIS, se ha hecho una serie de búsquedas en la red **[4]** para tratar de recopilar en foros y webs las diferentes novedades que se han incorporado en la versión 3.0 respecto a versiones anteriores. De entre ellas, las más destacables son las siguientes:

1) **Vistas de mapa múltiples en la misma interfaz**
Es posible trabajar con varias ventanas en la interfaz de *QGIS* y en diferentes escalas, y es posible asociar la vista principal del área de estudio. En la imagen siguiente se aprecia el detalle de las vistas en la parte derecha.

**Ilustración 8**: Detalle de la vista de mapas múltiples en QGIS.

2) **Vistas interactivas de mapas 3D**
Es posible realizar un alzamiento de una capa vectorial (extrusión) con la información de altitud proporcionada por un MET (modelo de elevación de terreno). Se observa en la ilustración cómo se ha formado en el centro una vista en 3 dimensiones de manera automática:

**Ilustración 9:** Detalle de vistas interactivas de mapas en 3D.

3) **Administrador de capas unificado**

Ahora es posible administrar los datos desde un único panel: el *Administrador de fuentes de datos* desde el que se pueden incluir diferentes tipos de datos desde la misma herramienta, evitando realizar algunos pasos. En la siguiente ilustración se puede visualizar que están organizadas todas las opciones en la columna izquierda

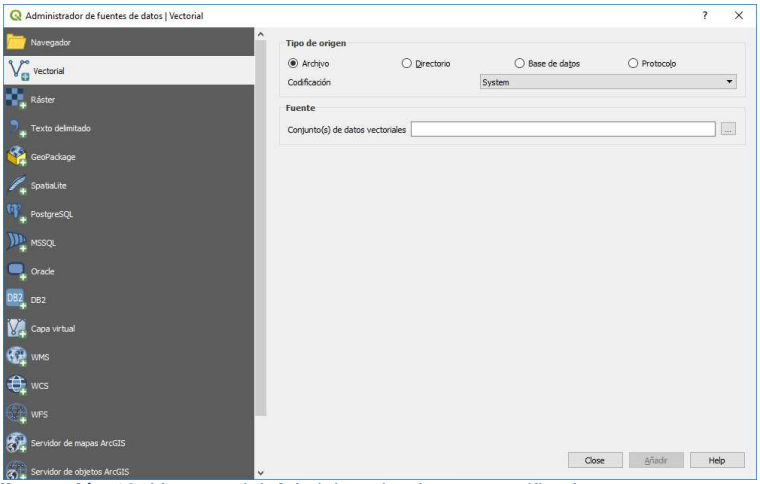



**Ilustración 10:** Ventana del *Administrador de capas unificado.*

4) **Herramienta de selección de entidades de atributos**.

Esta nueva opción permite concretar las búsquedas en una determinada entidad geométrica usando los valores guardados en la tabla de atributos de la capa. En la ilustración 10 aparece remarcada la opción de menú en un recuadro rojo y la ventana relacionada:

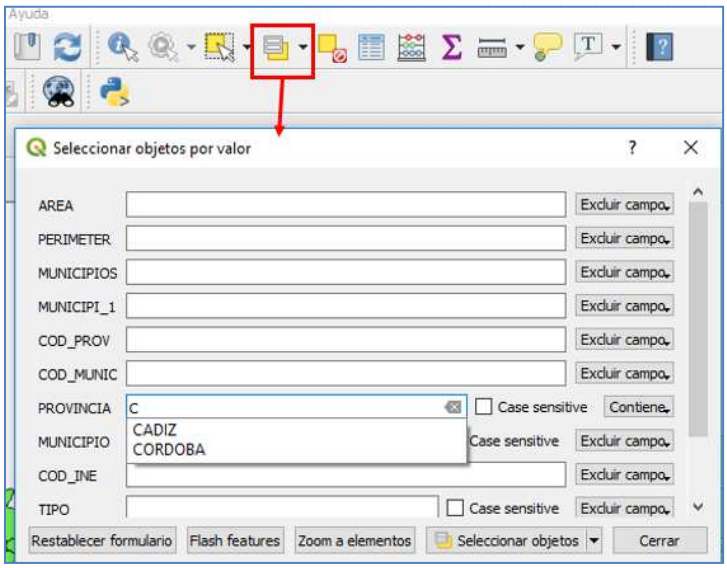



**Ilustración 11:** Selector de objetos por valor.

5) **Mejora en la herramienta de nodos**

Esta nueva mejora incorpora posibilidad de extender líneas de un modo más sencillo y es un cambio muy importante y muy demandado por los usuarios de QGIS ya que en versiones anteriores resultaba muy engorroso de utilizar. Se visualiza la extensión de puntos de color amarillo en la parte derecha inferior de la siguiente imagen:

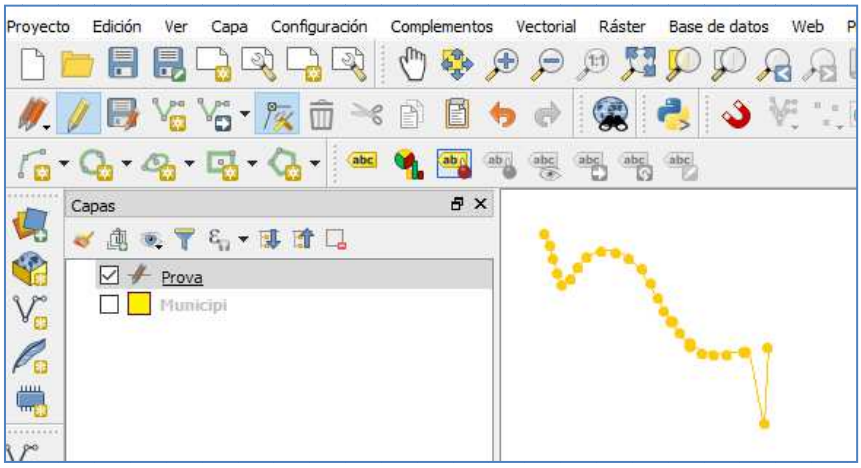



**Ilustración 12**: Herramienta de nodos.

## 6) **Edición de etiquetas**

Ahora la barra de herramientas *Etiqueta* permite realizar nuevas acciones como desplazar etiquetas, girarlas, etc. En la ilustración a continuación se observa remarcada en rojo la opción de menú y el movimiento de la etiqueta con la flecha roja:

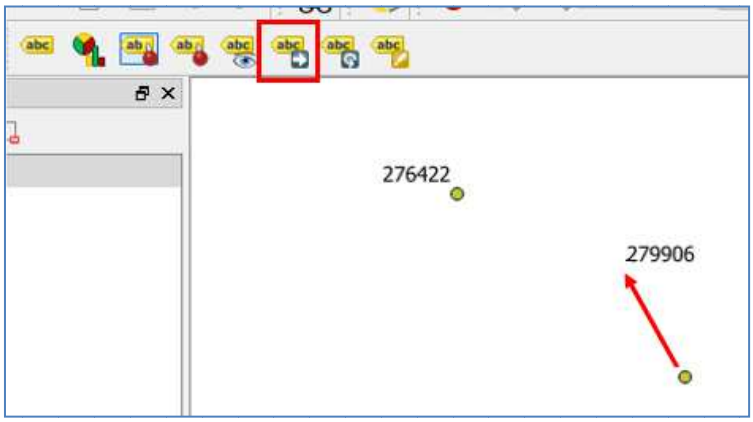



**Ilustración 13**: Edición de etiquetas de QGIS.

## 7) **Ejecución de operaciones en segundo plano**

Con esta función no se bloquea el flujo de trabajo cuando se trabaja con un volumen grande datos ya que, aunque su procesado pueda llevar varios minutos, se puede aprovechar para trabajar mientras se finaliza el algoritmo en ejecución. En la imagen siguiente se observa el buffer de memoria:

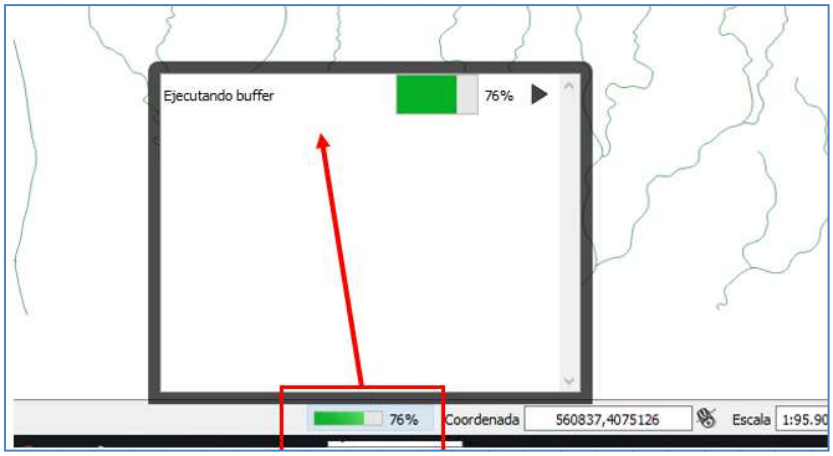



**Ilustración 14:** Ventana de visualización de ejecución de operaciones en segundo plano.

8) **Selector de Sistema de Referencia de Coordenadas (SRC)**
Ahora es posible visualizar de manera previa la extensión de la proyección seleccionada y es más sencillo seleccionar el SRC correcto a sus datos. En la ilustración siguiente se observan las diferentes proyecciones en el selector:

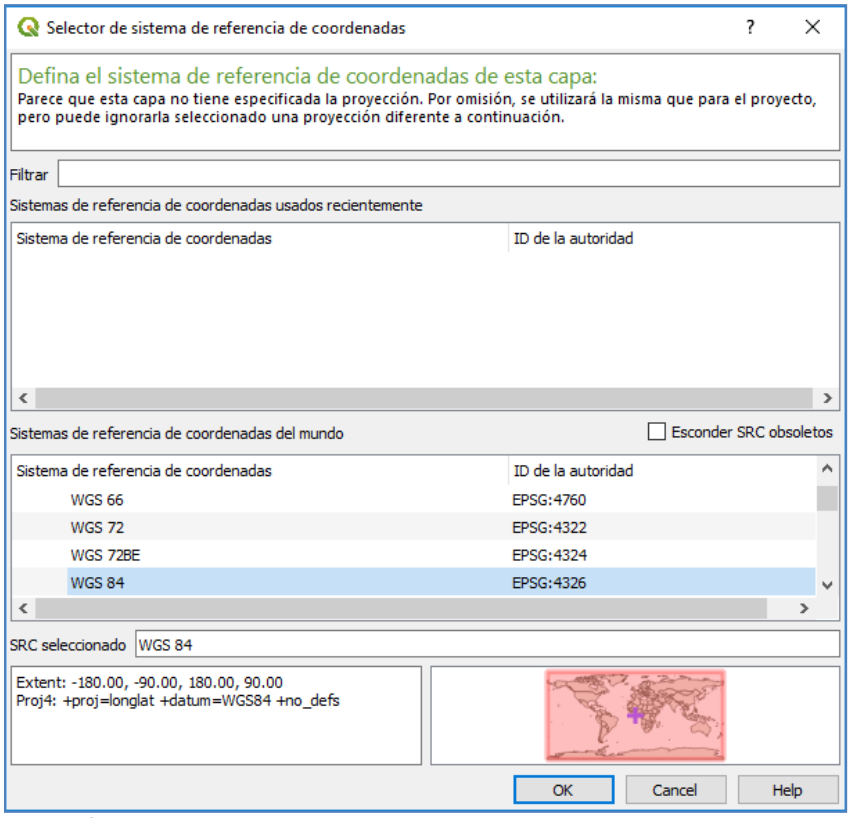


**Ilustración 15:** Ventana del selector de Sistema de Referencia de Coordenadas de QGIS.

9) **Actualización del código Python**
Se ha actualizado el código Python del núcleo de QGIS para que sea compatible con Python 3 y compatible con PyQt5 (implementación parcial para funcionalidades clave, por ejemplo, consola, complementos de Python core, etc.).

Una de las grandes ventajas de QGIS es su facilidad de interconexión con muchas bases de datos geoespaciales como GeoPackage[10], PostGIS[11], SpatiaLite[12], ORACLE Spatial[13] y otras. Además consta de soporte para la extensión espacial de PostgreSQL[14], PostGIS y manejo de archivos vectoriales Shapefile, ArcInfo coverages, Mapinfo, GRASS GIS, etc. Respecto a los archivos raster, tiene soporte para GRASS GIS[15], GeoTIFF[16], TIFF[17] y JPG, entre otros.

[10] https://en.wikipedia.org/wiki/GeoPackage
[11] https://en.wikipedia.org/wiki/PostGIS
[12] https://en.wikipedia.org/wiki/Oracle_Spatial_and_Graph
[13] https://en.wikipedia.org/wiki/PostgreSQL
[14] https://en.wikipedia.org/wiki/ArcInfo
[15] https://en.wikipedia.org/wiki/GRASS_GIS
[16] https://en.wikipedia.org/wiki/GeoTIFF
[17] https://en.wikipedia.org/wiki/TIFF

En la imagen siguiente se observa la vista de la ventana de menú de administrador de BBDD:

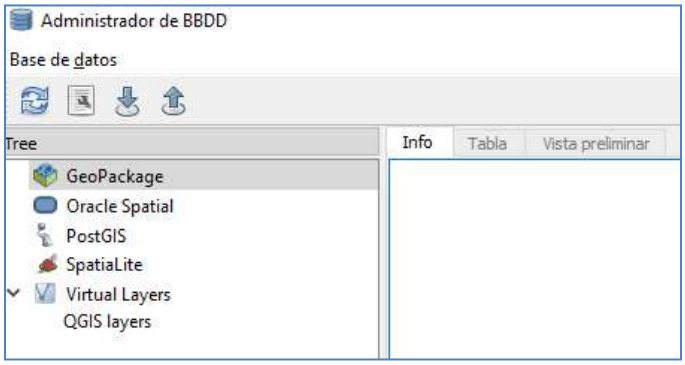



**Ilustración 16:** Ventana de administración de Bases de Datos.

Es posible encontrar hasta 100 mejoras que se han incluido en la versión 3.0 desde la versión anterior (versión 2.18). Para ampliar más información es posible revisarlas en el siguiente enlace:
https://mappinggis.com/2018/02/qgis-3-0-descarga-novedades-y-mucho-mas/

# 3. Descarga de Datos

En este apartado se presentan las bases de datos topográficas de datos abiertos que se van a utilizar en este proyecto así como el tratamiento de descarga, carga, estructura y diccionario de datos.

## *3.1 OpenStreetMap*

OpenStreetMap (también conocido como OSM) es un proyecto colaborativo para crear mapas libres y editables. Éstos se crean utilizando información geográfica capturada por los usuarios con dispositivos GPS móviles, ortofotografías y otras fuentes libres. La fundación OpenStreetMap es una organización internacional sin ánimo de lucro, dedicada a fomentar el crecimiento, desarrollo y distribución de datos geoespaciales libres y a proveer datos geoespaciales a cualquiera para utilizar y compartir.

Esta cartografía, las imágenes creadas y los datos vectoriales almacenados en su base de datos, se distribuyen bajo licencia abierta. El tamaño de la base de datos (llamada *planet.osm*) se situaba en julio de 2017 por encima de los 800 gigabytes.

En la mayoría de los países la información geográfica pública no es de libre uso y es por este motivo que surgieron movimientos con el fin de generar una plataforma de libre acceso a la información geográfica en la que cualquier usuario pudiese consultar, añadir o actualizar datos geográficos y cartográficos. En grandes ciudades dispone de una gran cantidad de datos aunque en las zonas rurales o de poca población todavía son escasos los datos de los que se dispone.

## *3.2 Diccionario de datos OSM*

OpenStreetMap utiliza una estructura de datos topológica. Los datos se almacenan en el *datum* WGS84 lat/lon (EPSG:4326) de proyección de *Mercator*. Los datos primitivos o elementos básicos de la cartografía de OSM son:

- Los **nodos** (*nodes*). Son puntos que recogen una posición geográfica concreta. Representa un punto específico en la superficie de la tierra definido por su latitud y longitud. Un ejemplo puede ser una señal de tráfico, un poste eléctrico o un árbol.

- Las **vías** (*ways*). Son una lista ordenada de nodos que representa una polilínea o un polígono (cuando una polilínea empieza y finaliza en el mismo punto). Un ejemplo claro podría ser un camino o una carretera para el caso de una polilínea abierta o un edificio para el caso de un polígono.

- Las **relaciones** (*relations*). Son grupos de nodos, vías u otras relaciones a las que se pueden asignar determinadas propiedades comunes. Un ejemplo puede ser, todas aquellas vías que forman parte del Camino de Santiago.

- Las **etiquetas** (*tags*). Se pueden asignar a nodos, caminos o relaciones y constan de una clave (*key*) y de un valor (*value*). Por ejemplo: *highway=trunk* define una vía como carretera troncal.

Es posible encontrar una lista extensa de las etiquetas **[9]** más comúnmente usadas en las siguientes direcciones de internet:

https://wiki.openstreetmap.org/wiki/Map_Features
https://taginfo.openstreetmap.org/tags

Como se puede encontrar en el último enlace mencionado, en la siguiente tabla se muestran algunas de las etiquetas más utilizadas:

| |
|---|
| building=yes<br>building=house |
| highway=residential<br>highway=service<br>highway=track<br>highway=unclassified |
| source=BAG<br>source=Bing |
| wall=no |
| waterway=stream |
| power=tower |
| natural=tree |

**Tabla 1:** Etiquetas OSM key-value más utilizadas.

Es posible ampliar más información en la web oficial de OpenStreetMap: http://www.openstreetmap.org. En la imagen siguiente se muestra un extracto de la información de etiquetas que aparece en dicha web:

| Key | Value | Element | Comment | Rendering | Photo |
|---|---|---|---|---|---|
| Types of aerialway | | | | | |
| aerialway | cable_car | | Cablecar or Tramway. Just one or two large cars. The traction cable forms a loop, but the cars do not loop around, they just move up and down on their own side, rolling along static cables over which they are suspended. | | |
| aerialway | gondola | | Gondola lift. Many cars on a looped cable. | | |
| aerialway | chair_lift | | Chairlift. Looped cable with a series of single chairs (typically seating two or four people, but can be more). Exposed to the open air (can have a bubble).<br>This implies oneway=yes (drawn upward). Any two-way chairlifts should be tagged oneway=no. | | |

**Ilustración 17:** Ejemplo de las etiquetas o *features* de OSM.

## 3.3 Estructura de la base de datos OSM

En la base de datos *Planet.osm* **[10]** está toda la información de OpenStreetMap en un solo archivo: todos los nodos, caminos y relaciones.
Para acceder a la base de datos principal para su edición se realiza a través de una API o interfaz. OpenStreetMap tiene una API (versión 0.6) de edición para extraer y guardar geoinformación en bruto desde/hacia la base de datos de OpenStreetMap. Actualmente Rails port es la implementación completa que se ejecuta desde la parte servidor.

Los dos principales formatos empleados son PBF[18] o XML OSM[19] comprimido.
PBF es un formato binario que es más pequeño y mucho más rápido de procesar que XML. La mayoría de herramientas que utilizan datos OSM soportan PBF. Los archivos tienen extensión **.osm.pbf*.
OSM XML es el formato proporcionado por la API. Las principales herramientas en el universo de OSM usan un formato XML que sigue una definición de esquema XML. Básicamente es una lista de instancias de las primitivas de datos (nodos, formas y relaciones).

Hay otros formatos compatibles como O5m[20], JSON bridge[21], Level0L[22]. Además existe en QGIS el complemento *QGIS OSM* que permite abrir archivos OSM y guardarlos en *Shapefile* o directamente en las últimas versiones es posible guardar la capa en otros formatos.

En la ilustración siguiente aparece representado el esquema de la bases de datos de Planet.osm de Railsport[23] en donde se visualizan las diferentes tablas, atributos y relaciones.

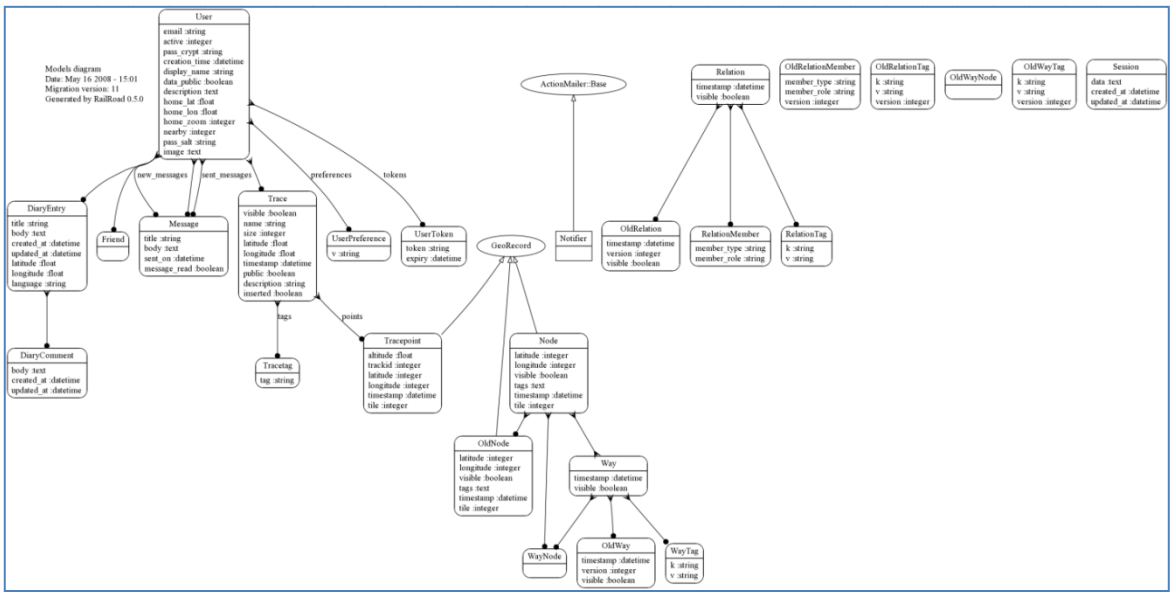



**Ilustración 18:** Esquema de la base de datos *Planet.osm* de Railsport. Fuente: wiki.openstreetmap.org.

[18] https://wiki.openstreetmap.org/wiki/PBF_Format

[19] https://wiki.openstreetmap.org/wiki/OSM_XML

[20] https://wiki.openstreetmap.org/wiki/O5m

[21] https://es.wikipedia.org/wiki/JSON

[22] https://wiki.openstreetmap.org/wiki/Level0L

[23] https://wiki.openstreetmap.org/wiki/The_Rails_Port

Es posible encontrar más información en la dirección: https://planet.openstreetmap.org.

## 3.4 Investigación de fuentes de datos abiertas como ICGC

El siguiente paso es realizar una descarga de datos de fuentes abiertas de bases de datos topográficas como la del *Institut Cartogràfic i Geològic de Catalunya (ICGC).*
Se descargará en formato Shapefile una hoja de la Base Topográfica a escala 1:5000 del ICGC.
Para ello, es necesario dirigirse a la página web oficial: http://www.icgc.cat.
Como se puede visualizar en la siguiente ilustración, accediendo a la sección de descarga de la Base Topográfica (en adelante, BT) se observan las características principales que ofrece. En el remarcado en azul junto a la flecha, se observa la opción que lleva la sección de *Descarga de Hojas*:


**Ilustración 19:** Página web oficial del Instituto Cartográficos y Geológico de Catalunya. Área de descarga.

Una vez en la sección mencionada, se accede al mapa de Cataluña donde muestra la cuadrícula que representa las hojas. A continuación es necesario seleccionar la zona de ejemplo que se quiere descargar. En la figura siguiente se observa en detalle el proceso de selección:

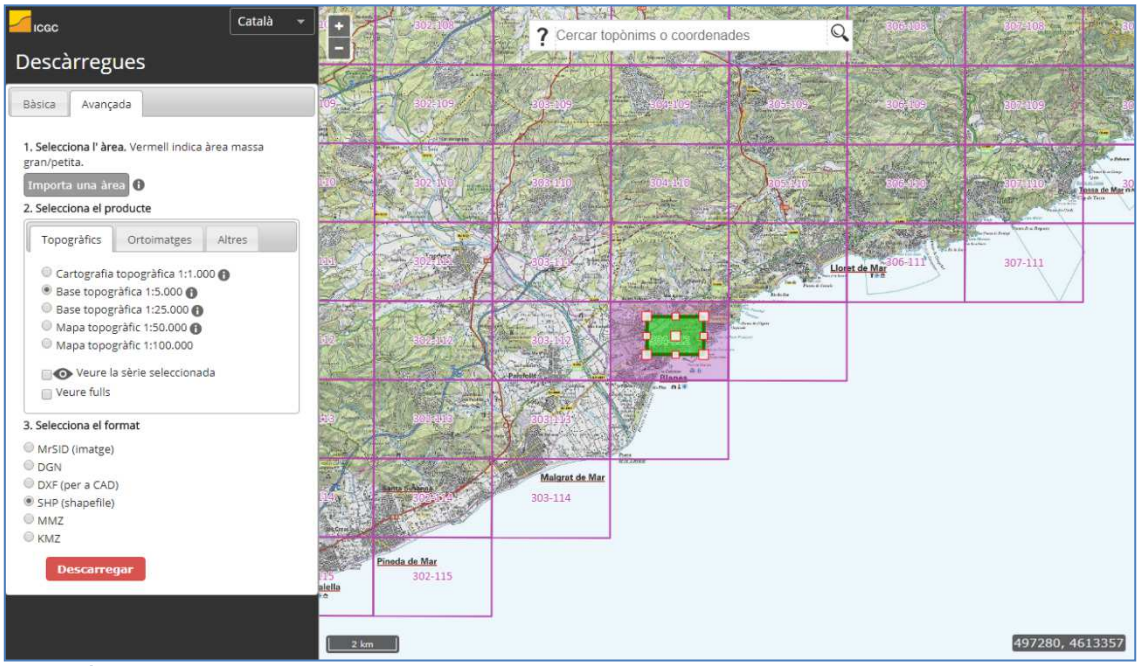



**Ilustración 20:** Sección de descarga de hojas de la BT del ICGC.

Como se observa, es posible seleccionar la escala y el formato. En este caso, se selecciona la escala 1:5000 y formato SHP (Shapefile) **[11]**. Respecto a la zona, se elige a modo de ejemplo la hoja que a descargar que comprende al centro del municipio de Blanes, en Girona.

La escala 1:5000 representa que 1 unidad del mapa son 5000 unidades de la realidad. Por ejemplo, 1 cm del mapa representarán 5000 cm de la realidad, o lo que es lo mismo, 50 metros.

El formato ESRI Shapefile (SHP) es un formato de archivo informático de datos espaciales desarrollado por la compañía ESRI[24], quien crea y comercializa software para Sistemas de Información Geográfica como Arc/Info o ArcGIS. Originalmente se creó para la utilización con su producto ArcView[25] GIS, pero actualmente se ha convertido en formato estándar de facto para el intercambio de información geográfica entre Sistemas de Información Geográfica por la importancia que los productos ESRI tienen en el mercado SIG y por estar muy bien documentado.

Un Shapefile[26] es un formato vectorial de almacenamiento digital donde se guarda la localización de los elementos geográficos y los atributos asociados a ellos. No obstante carece de capacidad para almacenar información topológica. La geometría de una característica (*feature*) se almacena como una forma (*shape*) que comprende un conjunto de coordenadas de vectores. Debido a que los *shapefiles* no tienen la sobrecarga de proceso de una estructura de datos topológicos, tienen ventajas sobre otras fuentes de datos, como una mayor velocidad de dibujo y capacidad de edición. También suelen requerir menos espacio en disco y son más fáciles de leer y escribir.

[24] https://es.wikipedia.org/wiki/Esri
[25] https://es.wikipedia.org/wiki/ArcView_3.x
[26] https://es.wikipedia.org/wiki/Shapefile

Los registros pueden admitir funciones de punto, línea y área. Las características del área se representan como polígonos de circuito cerrado. Los atributos se mantienen en un archivo de formato dBASE[27]. Cada registro de atributo tiene una relación de uno a uno con el registro de forma asociado.

Es un formato multiarchivo, es decir está generado por varios ficheros informáticos. El número mínimo requerido es de tres. En la descarga de la hoja de la BT del ICGC del municipio de Blanes, se puede observar que el archivo ZIP contiene diversos tipos de archivo que corresponden a los ficheros Shapefile. Se pueden observar en la imagen siguiente:

| Nombre | Fecha de modifica... | Tipo | Tamaño |
|---|---|---|---|
| bt5mv20sh0f304112al1r060.dbf | 13/02/2018 12:53 | Archivo DBF | 115 KB |
| bt5mv20sh0f304112al1r060.prj | 06/10/2010 18:07 | Archivo PRJ | 1 KB |
| bt5mv20sh0f304112al1r060.shp | 13/02/2018 12:52 | Archivo SHP | 1.581 KB |
| bt5mv20sh0f304112al1r060.shx | 13/02/2018 12:52 | Archivo SHX | 37 KB |
| bt5mv20sh0f304112an1r060.dbf | 13/02/2018 12:52 | Archivo DBF | 36 KB |
| bt5mv20sh0f304112an1r060.prj | 06/10/2010 18:07 | Archivo PRJ | 1 KB |
| bt5mv20sh0f304112an1r060.shp | 13/02/2018 12:52 | Archivo SHP | 62 KB |
| bt5mv20sh0f304112an1r060.shx | 13/02/2018 12:52 | Archivo SHX | 12 KB |
| bt5mv20sh0f304112ap1r060.dbf | 13/02/2018 12:53 | Archivo DBF | 1 KB |
| bt5mv20sh0f304112ap1r060.prj | 06/10/2010 18:07 | Archivo PRJ | 1 KB |

**Ilustración 21:** Archivos descargados de la hoja del mapa del ICGC en formato Shapefile.

Se pueden encontrar 4 tipos de archivos:

**.dbf** - es la base de datos, en formato dBASE, donde se almacena la información de los atributos de los objetos en formato alfanumérico. Se pueden considerar como una tabla de base de datos relacional.
**.shp** - es el archivo que almacena las entidades geométricas de los objetos.
**.shx** - es el archivo que almacena el índice de las entidades geométricas.

Además de estos tres archivos requeridos, opcionalmente se pueden utilizar otros para mejorar el funcionamiento en las operaciones de consulta a la base de datos, información sobre la proyección cartográfica, o almacenamiento de metadatos. En este caso, el archivo es:

**.prj** - Es el archivo que guarda la información referida al sistema de coordenadas del *Shapefile* en formato WKT (*Well Known Text*) **[12]**. WKT es una codificación en formato ASCII estandarizada diseñada para describir objetos espaciales expresados de forma vectorial. Los objetos que es capaz de describir el formato WKT son los siguientes: puntos, líneas, polígonos y sus multiplicidades.

Como se observa en la ilustración siguiente, es posible descargar de la web del ICGC **[5]** los archivos complementarios de especificaciones técnicas para ampliar información sobre la BT, el diccionario de la BBDD y las especificaciones del formato SHP (Shapefile):

[27] https://es.wikipedia.org/wiki/DBase

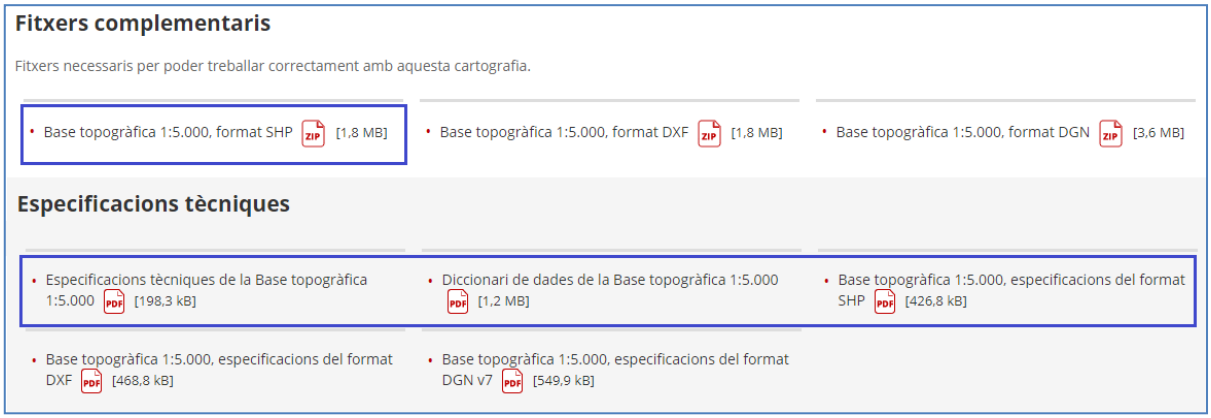



**Ilustración 22:** Zona de descarga de ficheros de ayuda, especificaciones técnicas, diccionario de datos, etc.

Una vez descargada la hoja del mapa topográfico del ICGC en formato Shapefile se debe proceder a cargarla en una capa de QGIS. Para ello se selecciona en QGIS, como se observa a continuación, la opción *Capa → Administrador de fuentes de datos:*

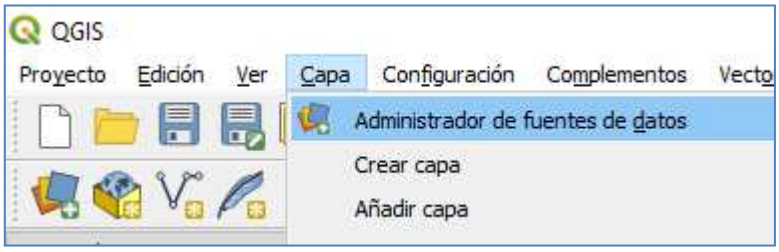



**Ilustración 23:** Opción *Administrador de fuentes de datos* del menú *Capa.*

Se selecciona la opción *Vectorial* como tipo de datos y en *Tipo de Origen* se marca la opción de botón radial *Directorio.* Esto es así debido a que se necesitan cargar todos los datos de Shapefile que hay la carpeta de la hoja descargada de la BT correspondiente al municipio de Blanes. Como son muchos archivos es interesante realizarlo de esta manera para que los cargue todos automáticamente.
Como muestra la figura siguiente, en *Conjunto de datos vectoriales* se selecciona la carpeta del ordenador donde están los archivos descargados del ICGC. En este caso la carpeta tiene el nombre "bt5mzt202108823566" que corresponde al formato de nombre de la BT 1:5000 v2 asignado por el ICGC:

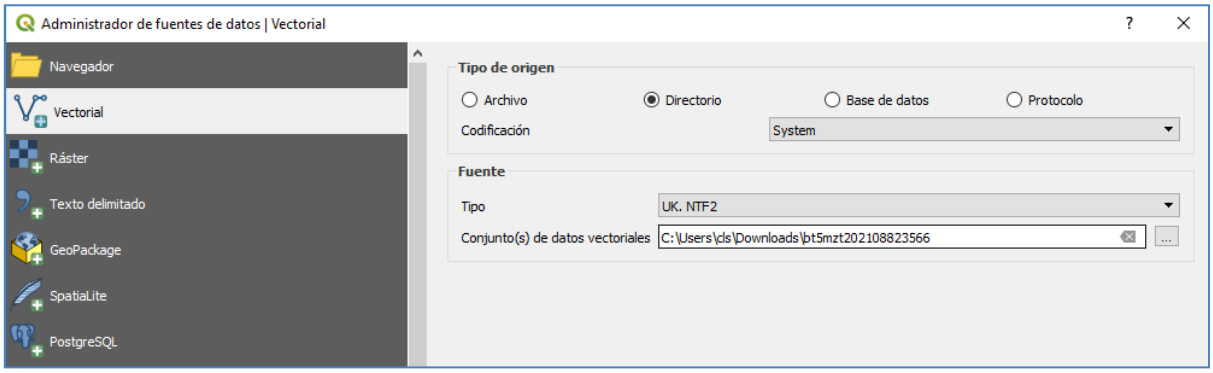



**Ilustración 24:** Ventana del Administrador de fuentes de datos vectoriales de QGIS.

Haciendo clic en *Añadir* aparecerá una ventana con las capas disponibles donde se debe marcar la opción *Seleccionar todo* y *OK,* tal como se observa a continuación:

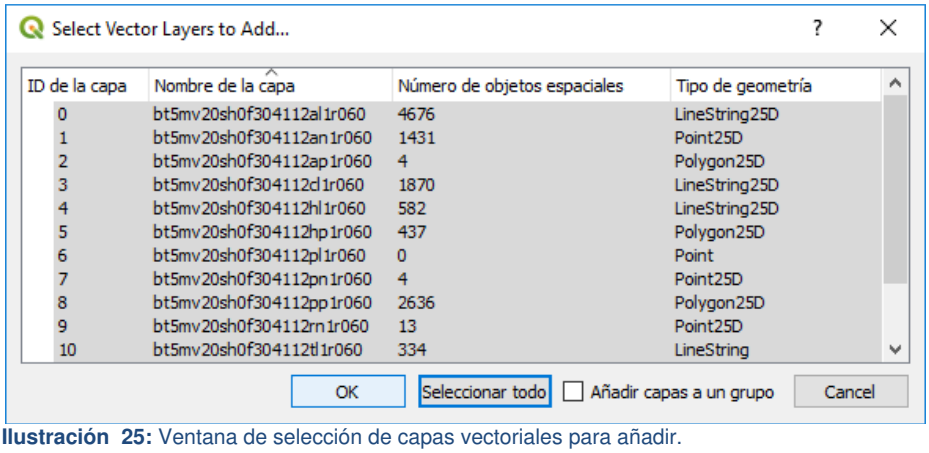



**Ilustración 25:** Ventana de selección de capas vectoriales para añadir.

Se observa en la parte central de la siguiente captura de pantalla de QGIS, que se ha cargado el mapa y en la columna de abajo a la izquierda se observan las capas. Es posible desmarcar capas que no interese tener a la vista:

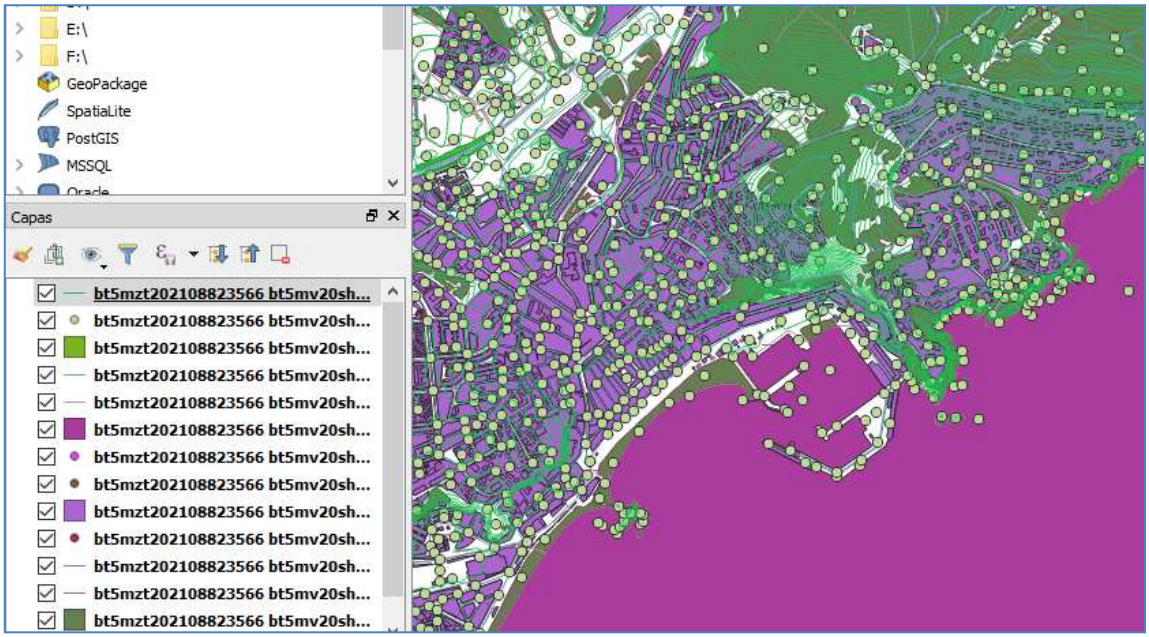



**Ilustración 26:** Capas cargadas en QGIS de la hoja descargada de la BT del ICGC.

En el siguiente paso, es necesario seleccionar la herramienta de selección de objetos espaciales, tal como se muestra en la figura siguiente:


**Ilustración 27:** Botón *Identificar Objetos* de QGIS.

Posteriormente, se puede realizar una comprobación marcando un punto cualquiera del mapa. En este caso, se prueba con un punto (nodo) en el final del dique o espigón del puerto de Blanes, tal como se observa a continuación:


**Ilustración 28:** Selector de identificación de elementos: nodos, vías o áreas.

Es posible identificar el punto o nodo (o a la polilínea que pertenezca), el área en el que esté contenido o identificarlo todo. Se observa en ilustración 29, en la columna derecha la información de este punto: el título, COT02, que equivale a "Cota altimétrica: /singular" según el diccionario del ICGC, las coordenadas *x* e *y* donde está situado en el plano, así como su altura o cota sobre el nivel del mar (eje Z = 5, 82 metros):

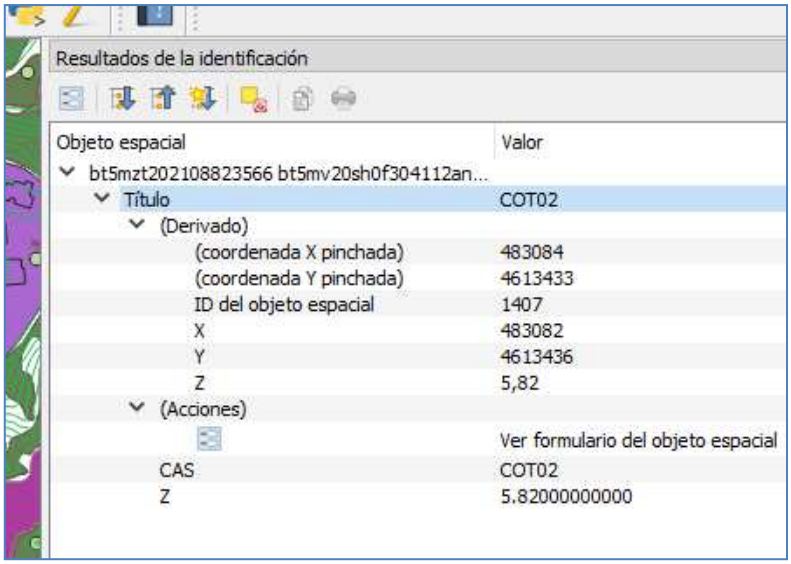



**Ilustración 29:** Ventana de información de QGIS.

Esta información que facilita el panel de *Resultados de la identificación* es muy útil para utilizarla, por ejemplo, en las comparaciones o *matching vectorial* que se vayan a realizar a continuación. Hay que recordar que se puede seleccionar la capa que interese (mapa de puntos, mapa de líneas, etc.) para identificar más rápidamente los elementos ya que suelen estar agrupados semánticamente por capas para facilitar las vistas. Esta característica será interesante posteriormente para leer los datos del mapa cargado.

## *3.5 Carga de datos OSM*

En este apartado se va a investigar acerca de la carga de datos de la base topográfica abierta de OpenStreetMap. Para ello hay que dirigirse a la página web de OSM **[6]**:
http://www.openstreetmap.org/

Como se observa en la siguiente captura de pantalla, en la web de OSM se muestra el visor de OpenStreetMap. Se debe hacer zoom en el mapa para seleccionar el área que se desea exportar hasta que el botón *Exportar*, en el menú superior de la izquierda, esté activo:

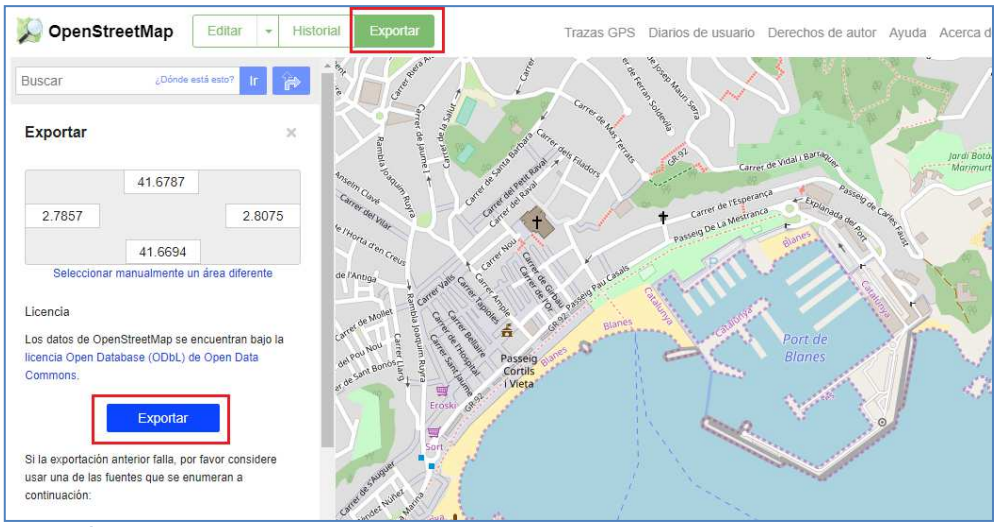


**Ilustración 30:** Mapa de OpenStreetMap y sus opciones de configuración de descarga.

Haciendo clic en el botón *Exportar***,** comienza la descarga del archivo *map.osm***.**

Si la exportación anterior falla, es posible usar una de las fuentes que se enumeran a continuación:

**Overpass API**[28]**:** Descargar este cuadro delimitador desde una réplica de la base de datos de OpenStreetMap.
**Planeta OSM**[29]**:** Copias actualizadas regularmente de la base de datos completa de OpenStreetMap.
**Descargas de Geofabrik**[30]**:** Extractos actualizados regularmente de los continentes, países, y ciudades seleccionadas.
**Otras fuentes:** Fuentes adicionales que aparecen en la wiki de OpenStreetMap.

Una vez se ha descargado el archivo map.osm se procede a realizar la conversión del archivo *.osm* a Shapefile (*.shp)* con QGIS.

En QGIS, clic sobre el botón *Añadir capa vectorial*. 
Seleccionar el archivo *.osm* que se acaba de descargar. En la siguiente imagen se puede observar el archivo descargado:

[28] https://wiki.openstreetmap.org/wiki/Overpass_API
[29] https://wiki.openstreetmap.org/wiki/Planet.osm
[30] https://www.geofabrik.de/data/

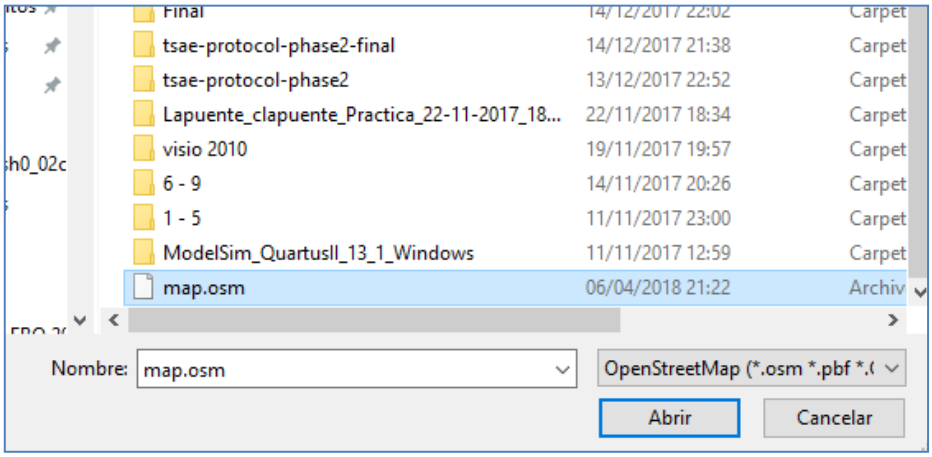



**Ilustración 31:** Archivo de mapas de OSM descargados.

Al hacer clic en *Abrir* se observarán las capas de OpenStreetMap en QGIS.
Haciendo clic en *Seleccionar todo* y a continuación *OK* se importarán las capas tal como se muestra en la siguiente figura:

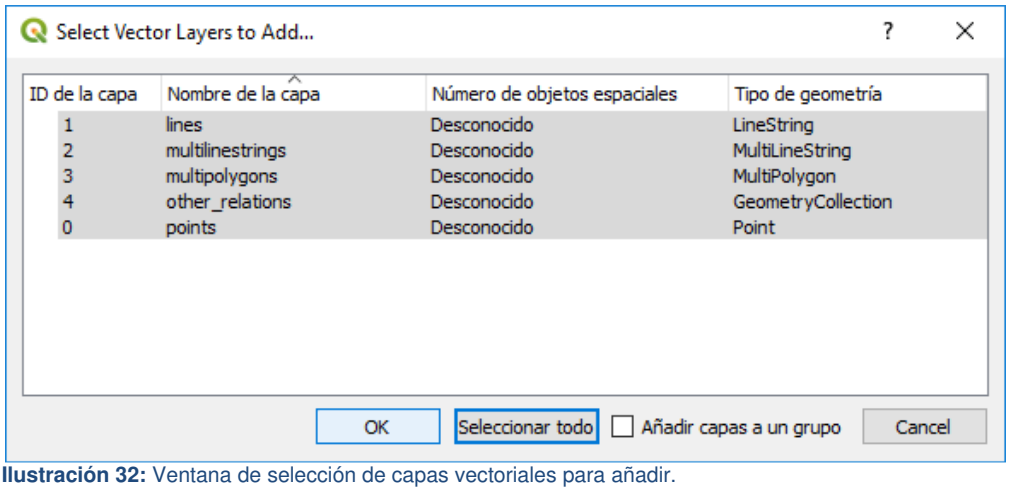



**Ilustración 32:** Ventana de selección de capas vectoriales para añadir.

En el paso siguiente, se selecciona la capa que se pretende transformar en otro formato y desde el menú superior de QGIS, clic en *Capa > Guardar como*...
Tal como se observa en la ilustración 33, se marca el formato vectorial admitido por OGR, en este caso *Archivo shape de ESRI*:

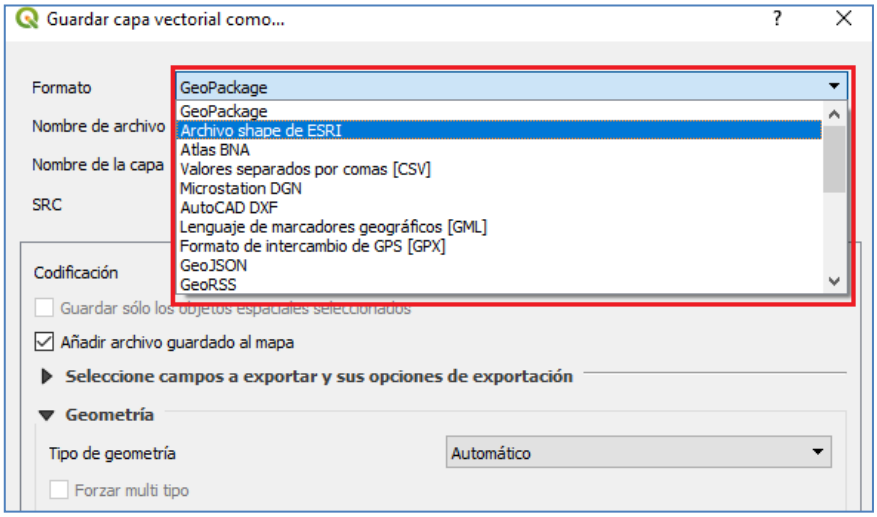



**Ilustración 33:** Ventana de exportación de tipo de capa vectorial.

Una vez convertidas todas las capas de OSM a *shapefile* ya se dispone de ellas en este formato y se pueden visualizar en la columna de la izquierda. Para poder gestionar mejor las capas es interesante agrupar las que son de OSM en un grupo y las que son del ICGC, en otro. Se les asigna el nombre correspondiente a los grupos ya que de esta manera se tiene un mayor control y comodidad a la hora de hacer visibles o no las capas de los mapas:

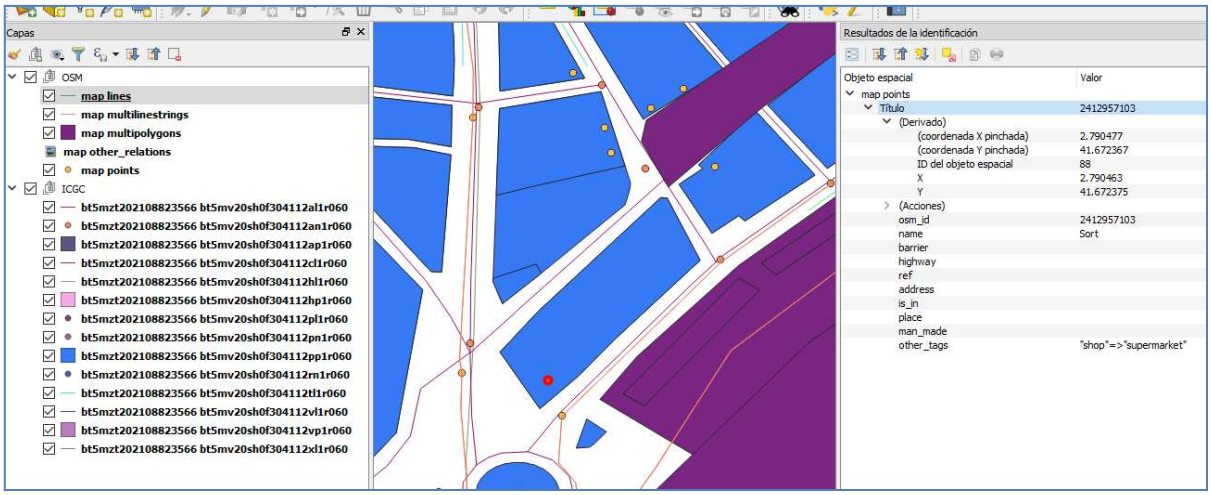



**Ilustración 34:** Resultado de la importación de capas de mapas del ICGC y OSM.

Se puede observar en la ilustración 34 que las capas se han cargado correctamente. Si se marca sobre un punto de interés (POI o *Point of Interest*) del mapa de OSM (nodo remarcado en rojo en la imagen) se puede observar en la información de la columna derecha de QGIS que se trata de una etiqueta *shop=supermarket (*corresponde al atributo *'other tags')*. En la ilustración 35 se observa que coincide con la información de valor añadido que se había importado de OSM:

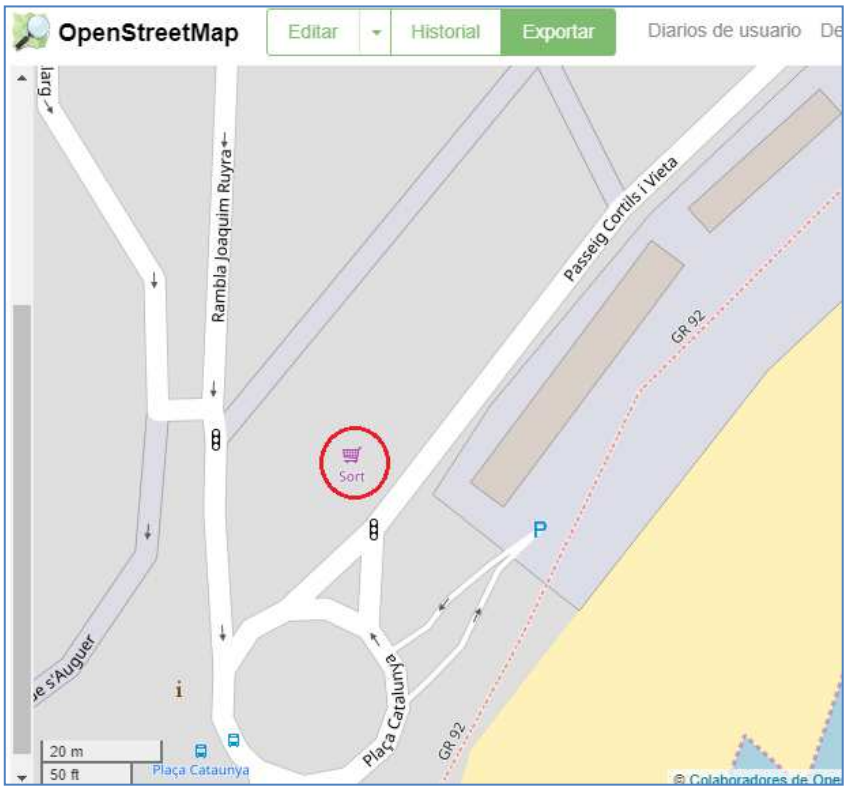



**Ilustración 35:** Visualización de información semántica en un elemento del mapa.

Efectivamente corresponde al supermercado de *Sort*.

# 4. Tratamiento de datos

En esta sección se presenta una breve investigación de las herramientas de comparación con datos georreferenciados así como su tratamiento.

## *4.1 Investigar comparación de datos georreferenciados con coordenadas*

Una vez cargadas las capas de las BT5m tanto de ICGC y OSM en QGIS, se procede a comparar datos geométricos con coordenadas y matching vectorial con geometrías. En los siguientes pasos se pretende mostrar cómo poder traspasar los POIs de OSM a las capas de ICGC.
El concepto se basa en tener abierto un proyecto de QGIS donde se hayan cargado manualmente las capas de datos de las BT que se quieren comparar. Por ejemplo, cargando una capa de edificios (polígonos) de la BT5Mv20 del IGCG y una capa de puntos de OSM que representen POIs (puntos de interés) que representen escuelas, hospitales, bares, etc. Para facilitar la tarea en el plugin, se pueden renombrar las capas implicadas con nombres representativos (como puntos POI o edificios). El plugin ejecutará el algoritmo que realizará todos los procesos automáticamente por lo que no será necesaria una interfície gráfica que solicite las capas entrada (INPUT) y unión (JOIN) ya que las rutinas irán a buscar las capas con los nombres directamente.

En este proceso, se compararán los puntos de la capa de OSM junto con las geometrías de polígonos de edificios de la capa destino (JOIN) de la BT del ICGC y si hacen *matching* se asignaran las etiquetas o el atributo de interés obtenido de OSM.

Para la realización de pruebas es interesante utilizar herramientas vectoriales de consultas espaciales de que disponen las librerías de Python como las que se pueden encontrar en QGIS en el menú *Procesos* → *Caja de herramientas* que corresponden a las antiguas herramientas *fTools* **[7]** que se podían encontrar en versiones anteriores de QGIS. En la ilustración 36 se observa cómo acceder a las herramientas mencionadas:

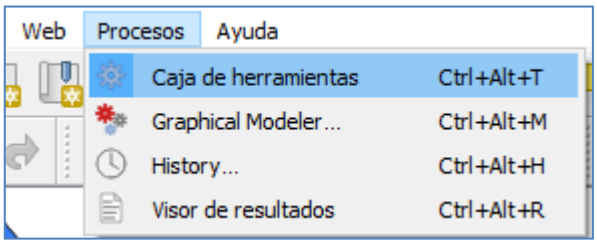



**Ilustración 36:** Opción *Caja de Herramientas* de la opción de menú *Procesos*.

Una vez abierta la *Caja de herramientas de Procesos* se pueden encontrar las diversas herramientas agrupadas por familias. En este caso se utilizaran las de carácter vectorial ya que se necesita precisión vectorial.

Como muestra la ilustración 37, se pueden encontrar en la *Caja de herramientas de procesos* los operadores y funciones de comparación de geometrías que se han visto en el tema *Bases de dades geogràfiques* de los materiales de la asignatura *SIG y Geotelemática* **[8]** de la UOC. Entre ellas están: Diferencia, diferencia simétrica, intersección, unión, distancia, vecino más próximo, etc.

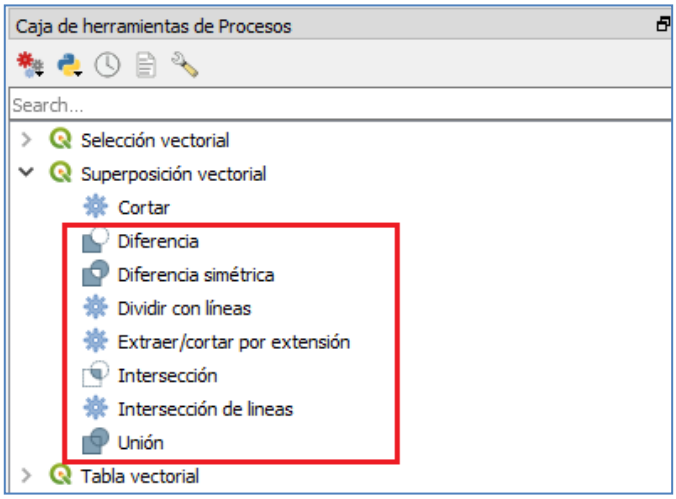



**Ilustración 37:** Operadores y funciones de superposición vectorial más importantes.

Es de interés ver cómo funcionan estas operaciones o herramientas internamente ya que dichas librerías puede servir como referencia o ser llamadas desde otros métodos Python.

# 5. Investigar y conocer el lenguaje Python

Un buen comienzo para trabajar con Python[31] es utilizar el plugin instalado en QGIS en los pasos anteriores, el *Plugin Builder* **[13]**. Este plugin constructor de plugins (o plugins plantilla) servirá para generar el código estructura del plugin en Python.

## *5.1 Construir la estructura del plugin*

A continuación se comenzará por desarrollar una aplicación sencilla que muestre por pantalla un "*Hello World" (*Hola Mundo) al pulsar un botón en QGIS.
Para ello, hay que seleccionar la herramienta de *Plugin Builder.* Se puede observar en la siguiente imagen (ilustración 38) el icono de la aplicación.

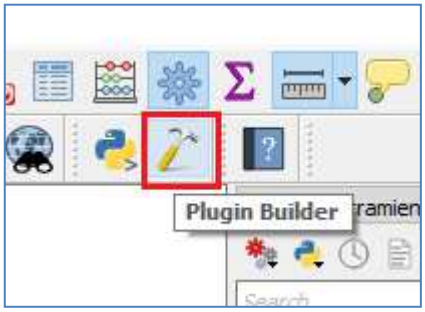



**Ilustración 38:** Botón de *Plugin Builder.*

A continuación se sucederán una serie de ventanas de información que se deben rellenar para que el constructor se capaz de generar la plantilla. En las ilustraciones 39 y 40 se pueden observar algunos de estos detalles:

[31] https://es.wikipedia.org/wiki/Python

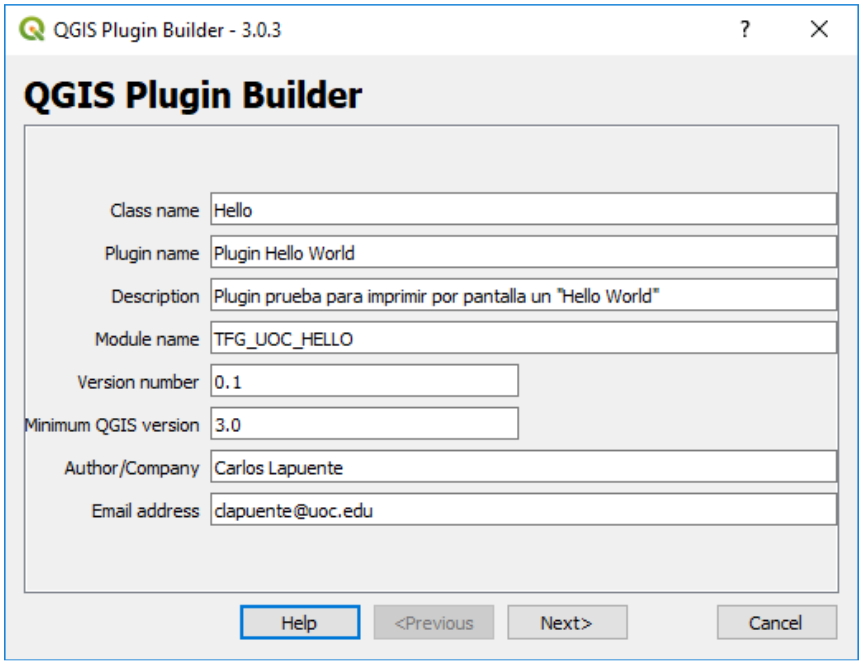



**Ilustración 39:** Ventana del primer paso de configuración del constructor de plugins.

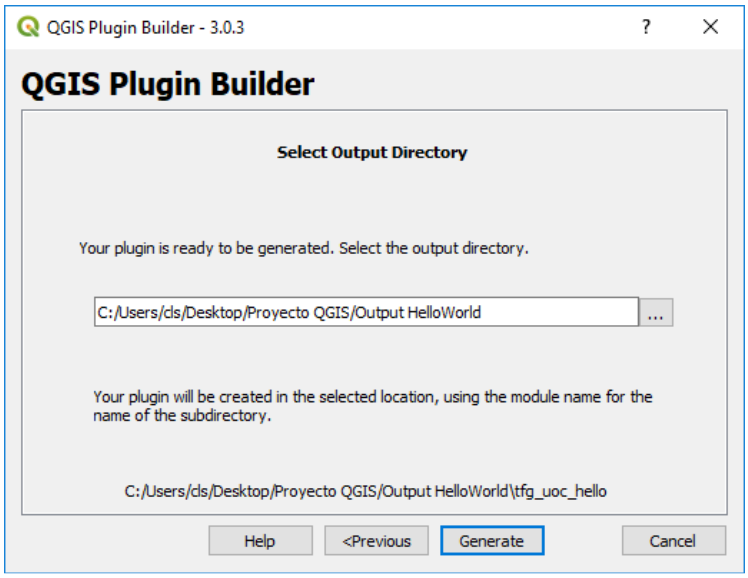



**Ilustración 40:** Ventana de selección del directorio de salida del Plugin Builder.

Finalmente se presenta una ventana resumen con la información y resultados de los pasos realizados, tal como se ve en la imagen 41:

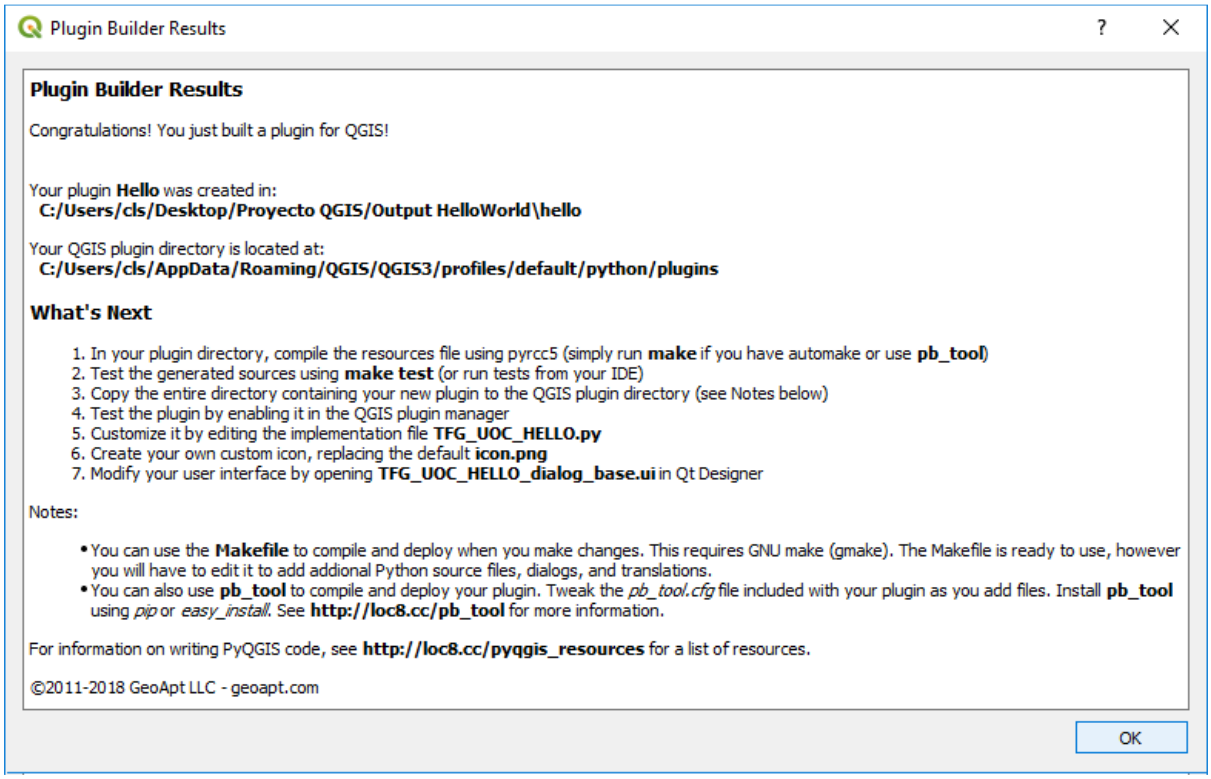



**Ilustración 41:** Ventana de resultado de los pasos realizados en la construcción del plugin.

El plugin ***HelloWorld*** fue creado en: **C:/Users/cls/Desktop/Hello_World/helloworld**

El directorio QGIS de plugins se encuentra en:
**C:/Users/cls/AppData/Roaming/QGIS/QGIS3/profiles/default/python/plugins**

A continuación se aconseja realizar las siguientes acciones que indica la ventana de resultados del *Plugin Builder*. Son las siguientes:

*1.* En el directorio de plugins, compilar el archivo de recursos usando pyrcc5 (simplemente ejecutar ***make*** si se tiene *automake* o usar ***pb_tool***).

*2.* Probar las fuentes generadas con ***make test*** (o ejecutar pruebas desde su IDE).

*3.* Copiar todo el directorio que contiene el nuevo plugin en el directorio de complementos de QGIS.

*4.* Probar el complemento habilitándolo en el administrador de complementos de QGIS.

*5.* Personalizarlo editando el archivo de implementación **hello_world.py**

*6.* Crear tu propio icono personalizado, reemplazando el icono predeterminado.

*7.* Modificar la interfaz de usuario abriendo **hello_world_dialog_base.ui** en Qt Designer.

Para el caso que presenta este proyecto solamente es imperativo realizar las acciones 1, 3, 4 y 5 ya que el resto son opcionales (recomendadas).
Debido a que se va a personalizar el icono del Plugin respecto al que viene por defecto, se van a realizar los siguientes pasos de acuerdo a las acciones recomendadas:

Paso 1: Agregar la imagen del icono deseado en la carpeta del Plugin.

Modificar los archivos **resources.qrc**, **metadata.txt** y **hello_world.py** con la información del nombre del archivo del icono. A continuación se muestran remarcadas las modificaciones en los fragmentos de código respectivos:

**resources.qrc**

```
<RCC>
   <qresource prefix="/plugins/hello_world" >
      <file>hello_world.png</file>
   </qresource>
</RCC>
```

**metadata.txt**

```
homepage=
category=Plugins
icon=hello_world.png
# experimental flag
experimental=False
```

**hello_world.py**

```
def initGui(self):
   """Create the menu entries and toolbar icons inside the QGIS GUI."""
   icon_path = ':/plugins/hello_world/hello_world.png'
   self.add_action(
      icon_path,
      text=self.tr(u'hello_world'),
      callback=self.run,
      parent=self.iface.mainWindow())
```

Paso 2: Una vez modificados los archivos, se deben compilar los recursos.
Una de las características de las que dispone el lenguaje de Python es que no necesita ser compilado ya que es un lenguaje interpretado. Aún así, para el caso de desear cambiar o personalizar algunos elementos como el icono por defecto (icon.png) del plugin en QGIS, es necesario “compilar” los recursos.

Para realizar este paso se ha de compilar con **pyrcc5** el archivo de *resources* (archivo con el nombre del plugin creado y extensión *.py*). Se puede compilar con algunas de las herramientas aconsejadas como *automake* o *pb_tool*). En este caso se va a utilizar un proceso *batch* (.bat) con las instrucciones necesarias para compilar de manera sencilla y automática. Este fichero se ubicará dentro del plugin para evitar tener que introducir las rutas absolutas o *paths* en la compilación.
Las instrucciones son las siguientes:

```
@echo off
call "C:\Program Files\QGIS 3.0\bin\o4w_env.bat"
call "C:\Program Files\QGIS 3.0\bin\qt5_env.bat"
call "C:\Program Files\QGIS 3.0\bin\py3_env.bat"

@echo on
pyrcc5 -o resources.py resources.qrc
```

Mediante estos comandos y, ejecutando el *batch*, se compilarán automáticamente dichos recursos. En la ilustración 42 se observan los diversos ficheros generados:

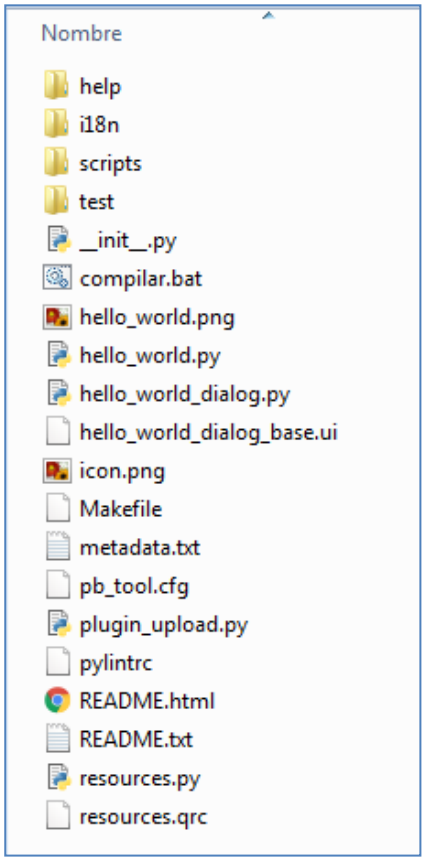



**Ilustración 42:** Archivos generados por el *Plugin Builder* en la carpeta del Plugin.

Paso 3: Modificar el código. En este caso se va a modificar ligeramente el código generado por el *Plugin Builder* para que el plugin realice alguna acción. En este caso se pretende que realice un "Hola Mundo" por pantalla.
Para que el plugin pueda realizar alguna acción, todo código nuevo a añadir en el plugin se debe incluir en la sección o "método" *run*.
En la sección del código *def run (self)* del archivo Python *matching_vectorial.py* se debe añadir un *print("Hola Mundo")* y retirar el *pass* que hace que se acabe la ejecución del *run* y salga del programa:

```
def run(self):
    """Run method that performs all the real work"""
    # show the dialog
    self.dlg.show()
    # Run the dialog event loop
    result = self.dlg.exec_()
    # See if OK was pressed
    if result:
        print("Hola Mundo")
```

Paso 4: Una vez realizada la modificación del código y compilados los recursos se ha de copiar la carpeta completa del plugin en la ruta de plugins de QGIS (se puede revisar en el fichero **README.txt**). Seguidamente se ha de lanzar QGIS y añadir el plugin en el menú de **Complementos -> Administrador de complementos -> hello_world**. Al habilitarlo, y hacer clic sobre su icono se obtiene algo similar a la siguiente imagen (ilustración 45):

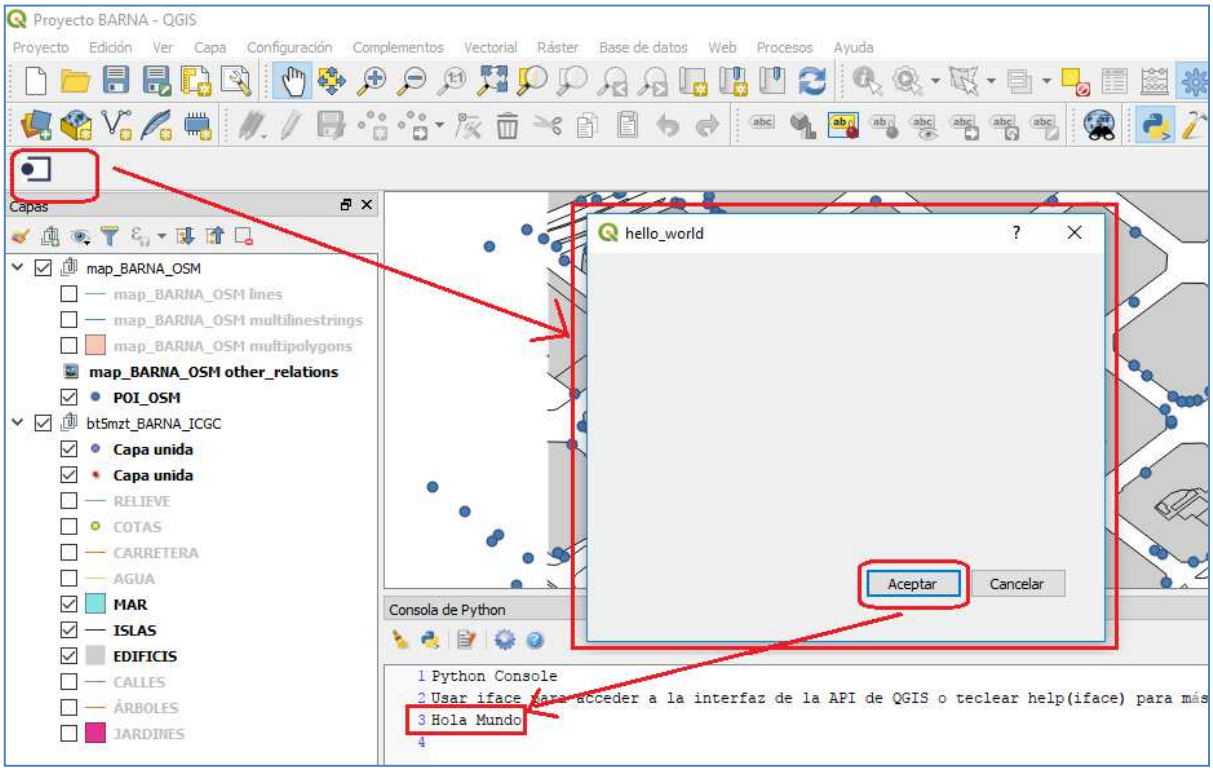



**Ilustración 43:** Carga y ejecución del plugin "Hello World" generado desde el *Plugin Builder.*

Una vez realizados estos pasos, comprendida la mecánica de operatividad de QGIS con los plugins de Python y comprobado el correcto funcionamiento del plugin *Hola Mundo* con la impresión por pantalla del mensaje, el siguiente paso es reutilizar este código "plantilla" para implementar el código del plugin de matching vectorial.
En primer lugar hay que retirar los *Print()* añadidos.

## *5.2 Algoritmo de matching*

Una de las ventajas que tiene el uso de herramientas GIS como QGIS 3.0. es que dispone de múltiples librerías de geoproceso implementadas por la comunidad colaborativa del proyecto *open source* de QGIS.

En el desarrollo del plugin de este proyecto se ha utilizado una de las librerías de geoproceso de QGIS a fin de aprovechar el trabajo realizado por la comunidad. En este caso, se ha hecho uso de la librería *Spatial Join*.

A continuación se muestra un fragmento del código Python de la herramienta *Spatial Join* donde se observa que en el método *name* la llamada a esta librería responde al nombre "*joinattributesbylocation*" **[14] [15]** al que se ha de realizar:

```
    def name(self):
        return 'joinattributesbylocation'

    def displayName(self):
        return self.tr('Join attributes by location')

    def tags(self):
        return self.tr("join,intersects,intersecting,touching,within,contains,overlaps,relation,spatial").split(',')

    def processAlgorithm(self, parameters, context, feedback):
        source = self.parameterAsSource(parameters, self.INPUT, context)
        join_source = self.parameterAsSource(parameters, self.JOIN, context)
        join_fields = self.parameterAsFields(parameters, self.JOIN_FIELDS, context)
        method = self.parameterAsEnum(parameters, self.METHOD, context)
        discard_nomatch = self.parameterAsBool(parameters, self.DISCARD_NONMATCHING, context)

        source_fields = source.fields()
        fields_to_join = QgsFields()
        join_field_indexes = []
        if not join_fields:
            fields_to_join = join_source.fields()
            join_field_indexes = [i for i in range(len(fields_to_join))]
        else:
            for f in join_fields:
                idx = join_source.fields().lookupField(f)
                join_field_indexes.append(idx)
                if idx >= 0:
                    fields_to_join.append(join_source.fields().at(idx))
```

**Ilustración 44:** Fragmento del código de la librería de QGIS *spatialjoin.py* ".

El código completo de la librería del archivo Python "spatialjoin.*py*" se puede encontrar en el anexo 2 adjunto al final de esta memoria.

En el algoritmo del plugin que se va a implementar en este proyecto se fijarán unos parámetros concretos establecidos sin necesidad de una ventana gráfica de *input*.
En el código del plugin se utilizarán las siguientes configuraciones para el paso de los parámetros:

- **Descartar registros que no hicieron match:** *'DISCARD_NONMATCHING' : True,*
- **Capa de entrada:** 'INPUT' : *'C:/Users/cls/Desktop/Proyecto QGIS /BARNA /POI_OSM.shp'*
- **Capa a unir:** 'JOIN' : *'C:/Users/cls/Desktop/Proyecto QGIS/BARNA/MAPASBASE /bt5mzt_BARNA_ICGC*
- **Nombre real de la capa:** *layername=bt5mv20sh0f289126pp1r060'*
- **Campos a añadir. Por defecto, todos:** *'JOIN_FIELDS' : [],*
- **Crear objeto separado para cada objeto localizado:** *'METHOD' : 0*
- **Capa unida:** *'OUTPUT' : 'memory:'*
- **Predicado geométrico:** 'dentro'= *'PREDICATE' : [5]*

Con toda esta información ya se disponen de todos los elementos necesarios para la creación del plugin objeto de este proyecto.
Como se ha comentado, para facilitar esta tarea, en la configuración del plugin dichos parámetros no vendrán dados desde la ventana gráfica ya que estarán fijados desde el propio código Python. Esto es así debido a que no se va a desarrollar interfície gráfica por la dificultad y tiempo de trabajo que conllevaría y debido a que no está al alcance de este proyecto.

# 6. Plugin

En este apartado se muestra el código implementado así como la explicación de los pasos realizados y su correspondiente traducción a lenguaje Python aplicado a QGIS (PYQGIS).

## *6.1 Código algoritmo*

Tal como se ha realizado en la estructura generada por el *Plugin Builder* para el plugin de *Hello World*, también aquí se debe introducir todo el código nuevo en la estructura *def run (self)*.

La ejecución de los procesos que realizará el algoritmo del plugin está basada en 5 pasos muy bien diferenciados. Cada uno de ellos ha sido codificado en Python **[16] [17]** y probado por separado y una vez se ha comprobado individualmente su correcto funcionamiento tanto en GIS como en la consola de Python integrada, se ha procedido a incluirlo en el código.

A continuación se muestran detalladamente dichos pasos a fin de facilitar su comprensión.

**Paso inicial:**

Se debe cambiar el código en la línea de "*result*". Si el valor de *result* es igual a 1 (lo que significa que se ha pulsado el botón *Aceptar* en la ventana del plugin) se ejecutaran las líneas de código posteriores:

```
193        # Comprueba si se ha pulsado Aceptar (OK)
194        if result == 1:
```

**Paso 1:** Listar todas las capas cargadas en QGIS.

Con esta sentencia de Python se guarda en la variable *layers* el listado de todas las capas cargadas en el proyecto abierto en QGIS.

```
196            # Hace un listado de todas las capas cargadas en QGIS
197            layers = QgsProject.instance().mapLayers().values()
```

**Paso 2:** Buscar las capas objeto del *matching*

Buscará la capa que coincida con el nombre EDIFICIOS y la capa con el nombre PUNTOS ya que serán las capas que realizarán el matching. Se ha programado de esta manera para que así el plugin sea compatible con cualquier base de datos topográfica siempre y cuando las capas estén en formato *Shapefile* y renombradas de esta manera.
Se puede observar en el bucle *for* que si la capa buscada corresponde con una capa llamada EDIFICIOS, averiguará la ruta del equipo donde está esa capa y obtiene la ruta donde está guardada hasta el nombre de la carpeta contenedora:

```
199            # Busca cual es la capa con el nombre EDIFICIOS...
200            for layer in layers:
201                if layer.name() == "EDIFICIOS":
202                    # ... averigua la ruta en el disco duro de esta capa ... y
203                    myfilepath_EDIFICIOS = layer.dataProvider().dataSourceUri()
204                    # ... deduce el nombre de la ruta hasta la carpeta que la contiene y el nombre del
                       fichero shape
205                    (myDirectory_EDIFICIOS,nameFile_EDIFICIOS) = os.path.split(myfilepath_EDIFICIOS)
```

Igual que en el anterior pero para la capa PUNTOS:

```
207            # Lo mismo que el bloque anterior de código pero para la capa PUNTOS (PoI)
208            for layer in layers:
209                if layer.name() == "PUNTOS":
210                    myfilepath_PUNTOS = layer.dataProvider().dataSourceUri()
211                    (myDirectory_PUNTOS,nameFile_PUNTOS) = os.path.split(myfilepath_PUNTOS)
```

**Paso 3:** Paso de parámetros:

Como se ha comentado en el apartado anterior, se pasarán los parámetros directamente por código con los valores mencionados. Para ello se hace servir una variable *parameters* donde se pasarán los parámetros que interesan.
Como se ha comentado, en este caso la capa INPUT será la capa de puntos de OSM llamada "PUNTOS", la capa "JOIN" será la capa de polígonos de ICGC llamada "EDIFICIOS", el predicado geométrico será el del valor que corresponde a "dentro" que es el [5], etc.
Se puede observar en las líneas de código a continuación:

```
216            # Parametros del geoproceso de union de atributos por localizacion.
217                #INPUT: Capa de entrada (puntos).
218                #JOIN: Capa que hará el matching (edificios).
219                #PREDICATE: [5]= Predicado geométrico "dentro".
220                #DISCARD_NONMATCHING [True]= Descartará los que no cumplan el predicado geométrico.
221            parameters = {'INPUT':myfilepath_PUNTOS,'JOIN': myfilepath_EDIFICIOS,'PREDICATE':[5],
               'JOIN_FIELDS':[],'METHOD':0,'DISCARD_NONMATCHING':True,'OUTPUT':OUTPUT}
222            feedback = QgsProcessingFeedback()
```

**Paso 4:** Ejecución del geoproceso de librería:

Como se comentó en el apartado anterior en el código de *SpatialJoin*, para hacer la llamada a la librería de QGIS de geoproceso se realiza mediante esta línea de código que va a buscar en la biblioteca de procesos "*processing*" de QGIS. Se puede observar que se pasan los parámetros con la variable utilizada en el paso anterior:

```
224            # Ejecución del geoproceso y carga en el proyecto de la capa resultante
               Capa_matchimp_vectorial
225            processing.runAndLoadResults("qgis:joinattributesbylocation", parameters, feedback =
               feedback)
```

**Paso 5:** Guardado de capa resultante y aplicación del estilo de simbolización:

Este paso final previo al *pass* de Python, guarda la capa OUTPUT resultante del geoproceso en la ruta marcada. En caso de haber marcado en QGIS sobre el grupo de capas de OSM o ICGC, se guardará en esa ubicación. En caso de no haber marcado nada en QGIS, se guardará en la misma ruta del proyecto:

```
213            # Ruta donde se guardará la capa resultante de la union de atributos por localizacion
               Capa_matching_vectorial (en formato shapefile).
214            OUTPUT = str(myDirectory_EDIFICIOS) + '/' + 'Capa_matching_vectorial.shp'
```

La capa guardada se muestra como "*capa unida*" en memoria de QGIS y además se guarda con el estilo de simbolización marcado. En este caso como "capa_matching_vectorial" en formato *Shapefile.* Utiliza la misma ruta marcada en el OUTPUT:

```
227            # Dar el estilo de simbolización deseado (como se almacenará la capa en QGIS y en el disco
               duro).
228            STYLE = str(myDirectory_EDIFICIOS) + '/' + 'Capa_matching_vectorial.qml'
229            Capa_unida = QgsProject.instance().mapLayersByName('Capa unida')[0]
230            Capa_unida.loadNamedStyle(STYLE)
231            pass
```

El código completo del archivo "*matching_vectorial.py*" sección *def run* se puede encontrar en el anexo 3 adjunto al final de esta memoria.

## 6.2 Parte gráfica

En esta sección se desarrolla el apartado gráfico del plugin, en la ventana gráfica de ejecución como en la imagen del icono de la aplicación.

### *6.2.1 Ventana de ejecución*

El apartado gráfico de este proyecto es muy básico pues se ha pretendido que su sencillez sea pilar básico. No era el objetivo de este proyecto así que simplemente se ha personalizado la ventana gráfica que genera el *Plugin Builder*.
Para ello se han utilizado dos herramientas. Por un lado el programa editor *QT Creator* y por el otro lado con la modificación directa sobre el archivo "*matching_vectorial_dialog_base.ui"* con un programa editor de texto como Notepad++.

Qt Creator[32] es un IDE o entorno de desarrollo multiplataforma programado en C++, JavaScript y QML creado por Trolltech el cual es parte de SDK para el desarrollo de aplicaciones con Interfaces Gráficas de Usuario (GUI).
Notepad++[33] es un editor de texto y de código fuente libre con soporte para varios lenguajes de programación. Tiene soporte nativo con Microsoft Windows. Es similar al Bloc de notas en cuanto al hecho de que puede editar texto sin formato y de forma simple.

Con el programa QT Creator en su versión de demo descargado e instalado, se realiza básicamente toda la personalización. En primer lugar se abre desde QT Creator el archivo a modificar desde la opción de menú "File". Como se ha comentado anteriormente, se debe abrir el archivo "*matching_vectorial_dialog_base.ui"* ya que es el archivo que corresponde al apartado gráfico generado por el *Plugin Builder*.

Una vez abierto el archivo se muestra en pantalla el diseño de la ventana por defecto que genera el *Plugin Builder*. Se trata de una ventana de color gris con dos botones: OK (aceptar) y CANCEL (cancelar).
Para realizar la personalización se pueden modificar los valores que se encuentran en la columna derecha de color amarillo pastel. En la imagen siguiente (Ilustración 45) se puede observar el programa *QT Creator* en funcionamiento:

---

[32] https://es.wikipedia.org/wiki/Qt_Creator
[33] https://es.wikipedia.org/wiki/Notepad%2B%2B

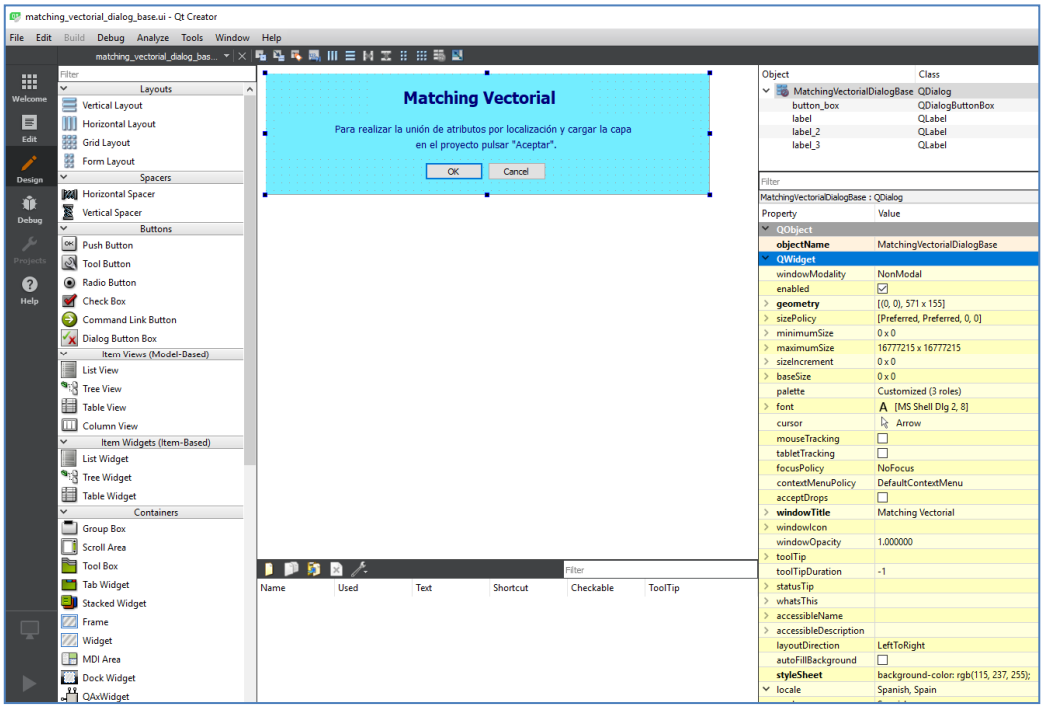



**Ilustración 45:** Ventana del programa QT Creator donde se puede personalizar el apartado gráfico del plugin.

Se puede observar que las secciones de los parámetros están separadas por grupos.
En el caso del color de fondo de la ventana, éste se ubica en la sección ***styleSheet: background-color*** y un código de color en RGB asociado.
Este se puede cambiar según el gusto del usuario. Para saber el código de color deseado, se puede utilizar una herramienta de Google[34] en la que simplemente introduciendo en el buscador las palabras clave "*rgb color picker*" aparece un selector de color con su código RGB correspondiente, y viceversa:

En la imagen siguiente (Ilustración 46) se observa el selector e identificador de codificación de colores RGB incrustado en el navegador en forma de *applet*:


**Ilustración 46:** Applet incrustado en el navegador Chrome el cual realiza las funciones de identificador de colores RGB.

[34] https://goo.gl/Yc42LL

En este ejemplo es el color elegido para la ventana gráfica de este proyecto.

En la columna de parámetros de QT Creator se pueden modificar, entre muchos otros, valores como el color de fondo, el tipo y color de letra, formas y tipos de botones, el tamaño de la ventana, etc.

En la ilustración 47 se observan algunos de los más importantes utilizados en este proyecto (remarcados en rojo):

Filter

MatchingVectorialDialogBase : QDialog

| Property | Value |
|---|---|
| windowModality | NonModal |
| enabled | ☑ |
| **geometry** | [(0, 0), 571 x 155] |
| sizePolicy | [Preferred, Preferred, 0, 0] |
| minimumSize | 0 x 0 |
| maximumSize | 16777215 x 16777215 |
| sizeIncrement | 0 x 0 |
| baseSize | 0 x 0 |
| palette | Customized (3 roles) |
| font | A [MS Shell Dlg 2, 8] |
| cursor | Arrow |
| mouseTracking | ☐ |
| tabletTracking | ☐ |
| focusPolicy | NoFocus |
| contextMenuPolicy | DefaultContextMenu |
| acceptDrops | ☐ |
| **windowTitle** | Matching Vectorial |
| windowIcon | |
| windowOpacity | 1.000000 |
| toolTip | |
| toolTipDuration | -1 |
| statusTip | |
| whatsThis | |
| accessibleName | |
| accessibleDescription | |
| layoutDirection | LeftToRight |
| autoFillBackground | ☐ |
| **styleSheet** | background-color: rgb(115, 237, 255); |
| locale | Spanish, Spain |
| Language | Spanish |
| Country | Spain |

**Ilustración 47:** Parámetros y sus valores correspondientes en QT Creator.

Por otro lado, como se ha comentado al inicio de este apartado, también se pueden modificar los valores del apartado gráfico editándolos directamente en el archivo desde un programa editor de texto como Notepad++.

En este caso basta con modificar sobre el texto y guardar el archivo. A continuación se muestra un recorte (ilustración 48) del código del archivo "*matching_vectorial_dialog_base.ui*" donde se aprecian los valores y la posibilidad de modificación:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ui version="4.0">
3   <class>MatchingVectorialDialogBase</class>
4   <widget class="QDialog" name="MatchingVectorialDialogBase">
5    <property name="geometry">
6     <rect>
7      <x>0</x>
8      <y>0</y>
9      <width>571</width>
10     <height>155</height>
11    </rect>
12   </property>
13   <property name="windowTitle">
14    <string>Matching Vectorial</string>
15   </property>
16   <property name="styleSheet">
17    <string notr="true">background-color: rgb(115, 237, 255);</string>
18   </property>
19   <widget class="QDialogButtonBox" name="button_box">
20    <property name="geometry">
21     <rect>
```

**Ilustración 48:** Recorte del archivo *matching_vectorial_dialog_base.ui* en modo edición con el Editor Notepad++.

Se pueden observar los valores codificados como el tamaño que tendrá la ventana, el color RGB del fondo, el nombre de la ventana, el tipo de botones, etc.

### *6.2.2 Personalización del icono.*

Tal como se ha realizado para el icono del plugin de prueba "Hola Mundo", se debe copiar el archivo de imagen para el icono en la carpeta del plugin. Debe tener un tamaño apropiado a un archivo correspondiente de icono (en este caso de 45x40 pixeles).

Debe nombrarse de la misma manera como se ha comentado en el ejemplo (en este caso "icon.png") y compilarse.
En este caso, para la imagen del plugin se ha elegido la de la nueva imagen de la UOC, tal como se puede observar en la ilustración 49:

**Ilustración 49:** Icono elegido para el plugin (diseño UOC).

# 7. Guía de uso y carga del plugin

En este apartado se pretende ofrecer una breve guía en sencillos pasos para la carga y funcionamiento del plugin. Se da por hecho que se ha realizado la instalación del software SIG QGIS 3.0.

**a)** Copiar la carpeta del plugin "*matchingvectorial*" en la ruta personal de plugins Python de QGIS:
*C:\Users\[usuario]\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins*.

**b)** Abrir QGIS y crear un proyecto nuevo. Guardar el proyecto en la ruta deseada (preferiblemente una ruta en la que el usuario tenga privilegios de escritura).
Cargar el plugin nuevo desde *Complementos --> Administrar e instalar complementos --> Matching Vectorial*. En la ilustración 50 se puede observar cómo instalar el plugin.

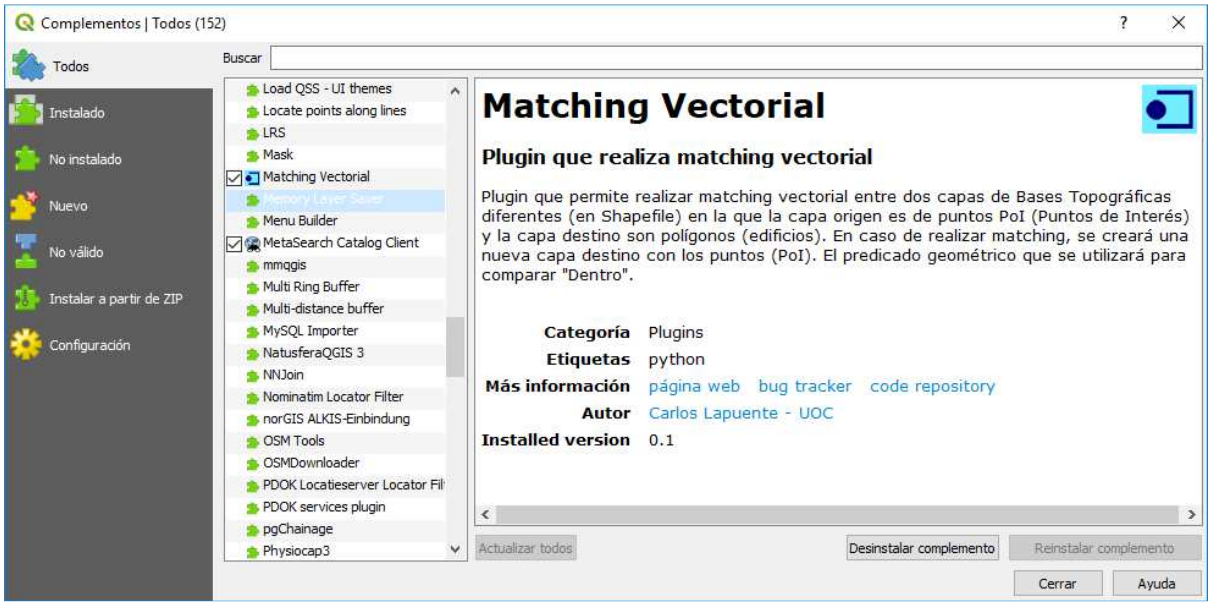



**Ilustración 50:** Carga del plugin Matching Vectorial en el menú de complementos de QGIS.

**c)** Descargar una hoja de cartografía topográfica en formato *Shapefile* de una base de datos topográfica. Por ejemplo de la BT del ICGC 1:5000 de la zona deseada de Catalunya.

**d)** Descargar la misma zona de la BT de OpenStreetMap.

**e)** Cargar estos archivos con sus capas correspondientes en QGIS desde *Capa --> Administrador de Fuentes de datos --> Vectorial.*

**f)** Guardar o convertir la capa de puntos de OSM en formato *ESRI Shapefile* (la capa suele llamarse *map points).* Para ello, en QGIS, hay que situarse encima de la capa e ir al menú: *Capa --> Guardar Como --> Formato.*

**g)** Renombrar la capa *map points* de OSM convertida del punto anterior con el nombre "PUNTOS" y la capa de polígonos de edificios del ICGC como "EDIFICIOS". Ésta última es la capa que se llama "bt5mv20sh0fxxx**pp**1r060**".**

**h)** Ejecutar el icono del plugin cargado y pulsar *Aceptar* en la ventana informativa

emergente. Esperar unos segundos (entre 3 y 5 segundos) y se observará que aparece una capa en memoria en QGIS llamada "Capa unida" y además se habrá guardado una copia en la ruta de carpetas donde estuvieran las BT o el proyecto (dependiendo de si se había marcado encima de algún grupo de capas antes de ejecutar el plugin).

En este punto, una vez finalizada la ejecución del plugin y comprobado que ha aparecido la nueva capa “Capa unida”, es aconsejable pulsar encima de ella haciendo doble clic y personalizar el icono de muestra de los puntos a fin de mejorar la visualización del resultado (simbología). Resulta útil seleccionar el icono de la estrella o rosa de los vientos de color roja que viene en los iconos predeterminados ya que permite resaltar de una manera visible el *join* realizado, tal como se observa en la imagen a continuación (Ilustración 51):

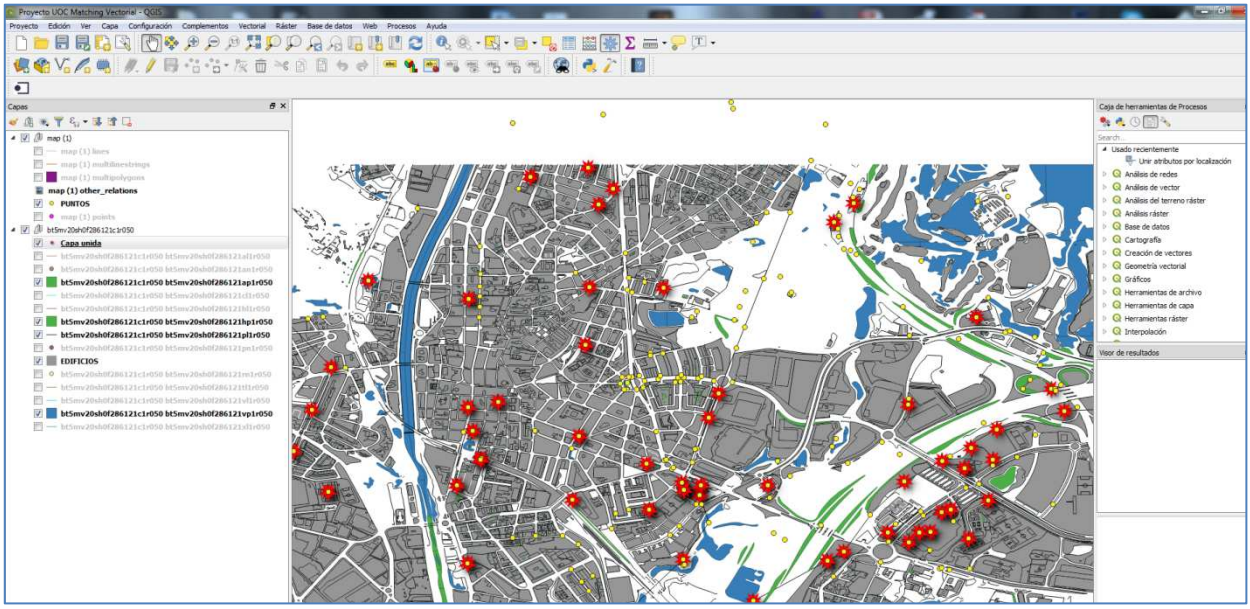



**Ilustración 51:** Resultado del Join o ejecución del plugin Matching Vectorial.

# 8. Conclusiones

Para finalizar la memoria del proyecto, en este capítulo se describirán los resultados obtenidos durante el desarrollo del proyecto además de analizar los posibles problemas encontrados, lecciones aprendidas y proponer futuras líneas de trabajo.

## *8.1 Resultados*

Para poder afirmar que el proyecto ha cumplido con los objetivos propuestos en su inicio es suficiente con comprobar las funcionalidades del producto final:

- El producto está basado 100% en el uso de tecnología *open source*: QGIS, Plugin Builder, OpenStreetMap, datos abiertos del ICGC, etc.

- Se ha implementado un plugin que realiza el matching vectorial entre una capa *Shapefile* de puntos de interés de la BT de OSM y una capa *Shapefile* de polígonos (edificios) de otra BT, como ICGC.

- El producto almacena el resultado del matching no solo en memoria en QGIS susceptible de ser exportado a cualquier formato de BT sino que además guarda la capa obtenida en formato *Shapefile* a fin de que el usuario final pueda utilizar esta información semántica obtenida para cualquier otra finalidad.

Con estas funcionalidades implementadas en el producto final, se puede afirmar que el proyecto ha alcanzado con éxito los objetivos planteados al inicio de éste.

## *8.2 Lecciones aprendidas*

En el camino y aventura de la realización de este proyecto se han podido extraer algunas lecciones interesantes.

En primer lugar se ha podido comprobar la alta calidad que proporcionan las herramientas GIS de código abierto como QGIS y las bases de datos topográficas de datos abiertos como OpenStreetMap o las del ICGC.
En segundo lugar, el uso de Python como lenguaje interpretado de desarrollo aplicado a GIS (PYQGIS) ha resultado ser una experiencia interesante por su simplicidad así cómo el poder encontrar una comunidad de usuarios y foros con información que va creciendo a un ritmo casi exponencial.

En tercer lugar se ha podido observar que con dichas herramientas se abre todo un mundo de posibilidades de desarrollo de aplicaciones de geoproceso y la generación de nueva información relevante para los sistemas de geolocalización y por tanto a una mejora de la información de soporte a la población.

Para finalizar, se ha podido comprobar que aunque desde algunos sectores se pone en entredicho la calidad de los datos de OpenStreetMap que son generados por miles de voluntarios y se acostumbra a considerar que son menos fiables que los productos cartográficos profesionales, podemos encontrar estudios que comparan la calidad de OpenStreetMap respecto de otras cartografías institucionales y que muestran que los datos a pesar de tener sus carencias no son nada despreciables y en algunos casos resultan ser muy útiles como por ejemplo por las aplicaciones de cálculo de rutas de navegación. Además hay que tener en cuenta la gran cantidad de información de que dispone esta base de datos, alimentada principalmente por conocedores locales del

ámbito cartográfico donde contribuyen. Este trabajo de campo exhaustivo es el que permite usar OSM como fuente de enriquecimiento semántico en otras bases topográficas.

## 8.3 Problemas encontrados

Durante el desarrollo del proyecto se han encontrado varios problemas, algunos de los cuales han repercutido significativamente en el resultado final del proyecto.
Los principales problemas encontrados han sido:

- **Riesgos materializados.** En el capítulo de *Análisis de Riesgos* del Plan de Trabajo se describieron varios riesgos, de los cuales dos de ellos se materializaron a lo largo del proyecto teniendo consecuencias negativas en el resultado. Para minimizar los riesgos en el producto final se optó por bajar la calidad de algún entregable (PAC 2) al quitarle tiempo de trabajo en su elaboración.

- **Ejecución del geoproceso del plugin.** Aunque la ejecución de los geoprocesos en QGIS es rápida, en estados iniciales del código Python implementado, debido a errores en algunos bucles *for* o a paso de parámetros indebidos, se generaban ejecuciones en bucle con errores que alargaban los tiempos de espera de manera inaceptable y alargaban los test, por lo que el tiempo de dedicación en el desarrollo del código del plugin tuvo que ser intensificado.

## 8.4 Líneas de trabajo futuras

El producto desarrollado para este proyecto ha sido realizado teniendo en cuenta el cumplimiento de los requisitos mínimos de funcionamiento exigidos en los objetivos del tema propuesto por el equipo docente del área. El tiempo utilizado para ello ha sido marcado por la duración del semestre que ha constado de 15 semanas de desarrollo, y la planificación de tareas en el diagrama de Gantt que ha sido expresamente organizada para encajar en este *timing*.
Dicho esto, el producto es susceptible de seguir muchos caminos de mejora, como mejoras gráficas, optimización de código o posibilidades infinitas de realización de diversas acciones. Debido a que no están al alcance de este proyecto así como la dificultad inherente que comportaría y la limitación de tiempo marcadas, no ha sido posible realizar algunas de ellas.

El proyecto ha conseguido cumplir con los objetivos propuestos inicialmente, aunque se consideraría trabajo futuro las siguientes líneas de trabajo:

- Desarrollo de una interfície gráfica mejorada en la que poder seleccionar visualmente a partir de una lista, las capas de PUNTOS y EDIFICIOS que van a realizar el matching vectorial, entre otras opciones. Esto evitaría tener que renombrar las capas manualmente.

- Posibilidad de realizar matching vectorial entre otras formas geométricas como líneas (carreteras, calles, etc.) y con otro predicado geométrico (interseca, cruza, toca, etc.).

- Optimización del código. El código de este producto es muy sencillo y realiza las acciones solicitadas básicas aunque, al haberse realizado la estructura con un generador de plugins, sus líneas de código son susceptibles de ser optimizadas.

# 9. Glosario

**API:** del inglés *Application Programming Interface*, la interfaz de programación de aplicaciones, es un conjunto de subrutinas, funciones y procedimientos para ser utilizados por otro software como una capa de abstracción.

**Applet:** tipo de programa específico incrustado en un navegador y creado mediante el lenguaje de programación Java.

**Batch:** proceso de ejecución secuencial por lotes.

**BT:** base de datos topográfica.

**Extrusión:** alzamiento de una capa vectorial con la información de altitud.

**fTools:** antiguas herramientas vectoriales de versiones anteriores de QGIS.

**GDAL:** Geospatial Data Abstraction Library (también conocida como GDAL/OGR) es una biblioteca de software para la lectura y escritura de formatos de datos geoespaciales.

**GIS:** véase SIG.

**ICGC:** siglas del *Institut Cartogràfic i Geològic de Catalunya*.

**Información semántica:** información con significado, valor o interpretación en mapas.

**Matching:** del inglés que se encaja, empareja o corresponde.

**Mercator:** tipo de proyección cartográfica ideada por Gerardus Mercator en 1569, para elaborar mapas de la superficie terrestre.

**MET:** modelo de elevación de terreno.

**Notepad++:** editor de texto y de código fuente libre con soporte para varios lenguajes de programación.

**Open source:** código abierto o de libre uso.

**Ortofotomapa:** documento cartográfico que consiste en una fotografía aérea vertical o una imagen de satélite que ha sido rectificada geométricamente.

**OSM:** OpenStreetMap. Proyecto colaborativo para crear mapas libres y editables. Los mapas se crean utilizando información geográfica capturada por usuarios y otras fuentes libres.

**Plugin:** pequeño subprograma añadido a otro programa para complementarlo o mejorarlo.

**Plugin Builder:** plugin constructor de plugins en Python para QGIS.

**PoI:** del inglés, *Point of Interest* o Punto de interés con información semántica.

**PYQGIS:** Python aplicado a QGIS.

**Python:** lenguaje de programación interpretado multiparadigma cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

**QGIS:** Quantum GIS. Sistema de Información Geográfica (SIG) de código libre para plataformas GNU/Linux, Unix, Mac OS, Microsoft Windows y Android.

**QT Creator:** Entorno de desarrollo multiplataforma programado para el desarrollo de aplicaciones con Interfaces Gráficas de Usuario (GUI por sus siglas en inglés).

**Shapefile:** formato vectorial de almacenamiento digital donde se guarda la localización de los elementos geográficos y los atributos asociados a ellos.

**SIG:** Sistema de Información Geográfica.

**Spatial Join:** Unión de atributos por localización.

**SRC:** Sistema de Referencia de Coordenadas**.**

**VGI:** del inglés, *Volunteered geographic information* es el aprovechamiento de herramientas para crear, ensamblar y diseminar datos geográficos provistos voluntariamente por individuos.

# 10. Bibliografía

## *Material UOC*

**[8] Sistemas de Información geográfica y geotelemática**. Material docente de la UOC.
PID_00216147

**Treballs i Projectes Finals de Carrera**. Sistemes d'Informació Geogràfica de la Biblioteca Virtual de la UOC. http://openaccess.uoc.edu/webapps/

## *Enlaces electrónicos*

**[1] Wikipedia: QGIS**.
[en línea]. https://es.wikipedia.org/wiki/QGIS.
[fecha de consulta: 11/03/2018].
[en línea]. https://www.qgis.org/es/site/
[fecha de consulta: 18/03]

**[2] Wikipedia: GDAL**.
[en línea]. https://es.wikipedia.org/wiki/GDAL
[fecha de consulta: 18/03]

**[3] Wikipedia: Sistema de Información Geográfica**.
[en línea]. http://es.wikipedia.org/wiki/Sistema_de_Información_Geográfica.
[fecha de consulta: 10/03/2018].

**[4] Mapping GIS**
[en línea]. https://mappinggis.com/
[fecha de consulta: 19/03]
[en línea]. https://mappinggis.com/2013/07/transformar-datos-openstreetmap-a-gis/
[fecha de consulta: 24/03]

**[5] Institut Cartogràfic i Geològic de Catalunya**.
[en línea]. http://www.icgc.cat/
[fecha de consulta: 10/03/2018].

**[6] Wikipedia: OpenStreetMap**.
[en línea]. https://es.wikipedia.org/wiki/OpenStreetMap
[fecha de consulta: 11/03/2018].

**[7] QGIS Plugin fTools.**
[en línea]. https://docs.qgis.org/2.8/es/docs/user_manual/plugins/plugins_ftools.html.
[fecha de consulta: 11/03/2018].
[en línea]. https://www.cursosgis.com/como-anadir-plugins-en-qgis/
[fecha de consulta: 23/03]

**[9] Wiki OSM: Etiquetas**.
[en línea]. https://wiki.openstreetmap.org/wiki/ES:Etiquetas
[en línea]. [fecha de consulta: 20/03]

**[10] Planet OSM**
[en línea]. https://planet.openstreetmap.org
[fecha de consulta: 20/03]

**[11] Wikipedia: Shapefile.**
[en línea]. https://es.wikipedia.org/wiki/Shapefile
[fecha de consulta: 22/03]

**[12] Wikipedia: WKT.**
[en línea]. https://es.wikipedia.org/wiki/Well_Known_Text
[fecha de consulta: 23/03]

**[13] Blog José Guerrero.**
[en línea]. https://joseguerreroa.wordpress.com/2015/02/26/creando-un-plugin-simple-con-el-plugin-builder-de-qgis-en-windows/
[fecha de consulta: 10/04]

**[14] Blog SIG & Territoires.**
[en línea]. https://goo.gl/TD3sVL
[fecha de consulta: 10/05]

**[15] Blog d'Anita Graser.**
[en línea]. https://goo.gl/Gbfm7o
[fecha de consulta: 10/05]

## *Otras publicaciones*

**[16] *Sherman, Gary (2014): The Pyqgis Programmer's Guide***
Editorial: Locate Press

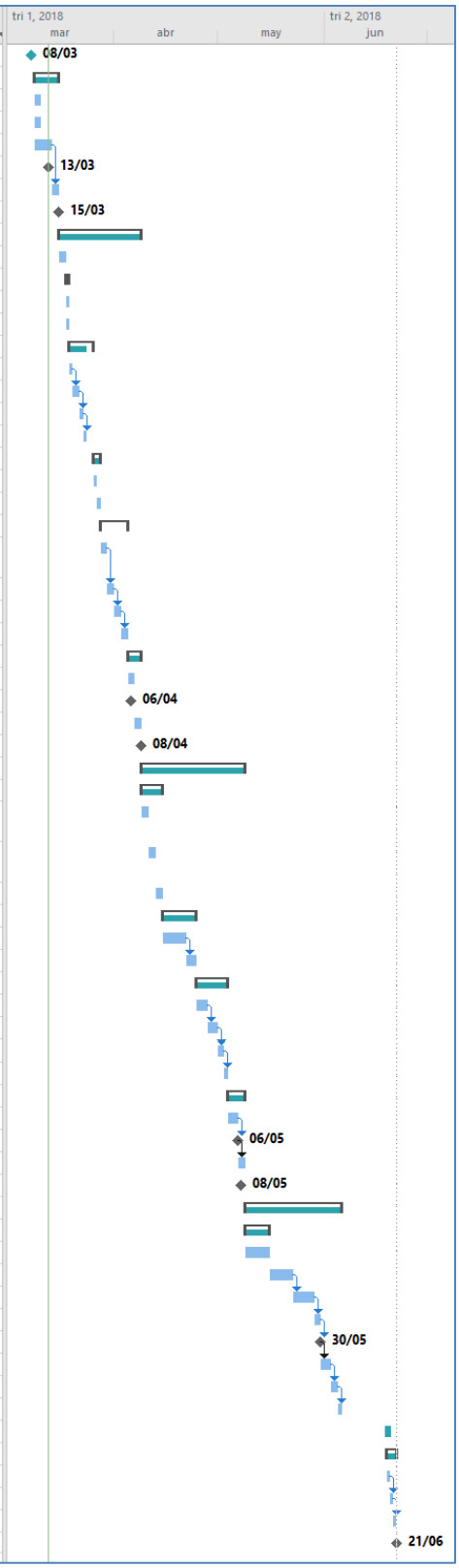
**[17] QGIS Project (2018): *PyQGIS developer cookbook***
*Release* 2.18

# 11. Anexos

## *Anexo 1*

**Calendario completo detallado del proyecto.**

| | Modo de | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|---|
| 1 | | Hito 1: Inicio del curso - Recepción tema TFG | 0 horas | jue 08/03/18 | jue 08/03/18 |
| 2 | | **PAC 1** | **7 días** | **vie 09/03/18** | **jue 15/03/18** |
| 3 | | Búsqueda previa de información Internet y fuentes | 4 horas | vie 09/03/18 | sáb 10/03/18 |
| 4 | | Instalación programa y aprendizaje Microsoft Project | 4 horas | vie 09/03/18 | sáb 10/03/18 |
| 5 | | Elaboración del plan de trabajo | 5 días | vie 09/03/18 | mar 13/03/18 |
| 6 | | Entrega borrador PAC 1 | 0 horas | mar 13/03/18 | mar 13/03/18 |
| 7 | | Revisión del plan de trabajo | 2 días | mié 14/03/18 | jue 15/03/18 |
| 8 | | Entrega PAC 1 | 0 horas | jue 15/03/18 | jue 15/03/18 |
| 9 | | **PAC 2** | **24 días** | **vie 16/03/18** | **dom 08/04/18** |
| 10 | | Repasar apuntes materiales SIG UOC | 2 días | vie 16/03/18 | sáb 17/03/18 |
| 11 | | **Preparació del entorno de trabajo** | **1 día** | **dom 18/03/18** | **dom 18/03/18** |
| 12 | | Instalar QGIS + otro software necesario | 1 día | dom 18/03/18 | dom 18/03/18 |
| 13 | | Analizar novedades de QGIS 3.0 | 1 día | dom 18/03/18 | dom 18/03/18 |
| 14 | | **Descarga de datos** | **7 días** | **lun 19/03/18** | **dom 25/03/18** |
| 15 | | Investigar sobre OSM | 1 día | lun 19/03/18 | lun 19/03/18 |
| 16 | | Familiarización con el diccionario de datos OSM | 2 días | mar 20/03/18 | mié 21/03/18 |
| 17 | | Análisis de la estructura de BBDD OSM y elementos | 1 día | jue 22/03/18 | jue 22/03/18 |
| 18 | | Análisis de los métodos de acceso a la información | 1 día | vie 23/03/18 | vie 23/03/18 |
| 19 | | **Búsqueda fuentes de datos cartográficos** | **2 días** | **lun 26/03/18** | **mar 27/03/18** |
| 20 | | Investigación fuentes de datos abiertas ICGC o OpenData | 1 día | lun 26/03/18 | lun 26/03/18 |
| 21 | | Investigar carga datos OSM | 1 día | mar 27/03/18 | mar 27/03/18 |
| 22 | | **Tratamiento Datos** | **8 días** | **mié 28/03/18** | **mié 04/04/18** |
| 23 | | Investigar cómo comparar datos geométricos con cordenadas | 2 días | mié 28/03/18 | jue 29/03/18 |
| 24 | | Investigar sobre matching vectorial con geometrías | 2 días | vie 30/03/18 | sáb 31/03/18 |
| 25 | | Desarrollar pseudocódigo algoritmo | 2 días | dom 01/04/18 | lun 02/04/18 |
| 26 | | Investigar y aprender sobre Python | 2 días | mar 03/04/18 | mié 04/04/18 |
| 27 | | **Documentación** | **4 días** | **jue 05/04/18** | **dom 08/04/18** |
| 28 | | Informe de la PAC 2 | 6 horas | jue 05/04/18 | vie 06/04/18 |
| 29 | | Entrega borrador PAC 2 | 0 horas | vie 06/04/18 | vie 06/04/18 |
| 30 | | Modificaciones memoria del proyecto | 2 días | sáb 07/04/18 | dom 08/04/18 |
| 31 | | Entrega PAC 2 | 0 horas | dom 08/04/18 | dom 08/04/18 |
| 32 | | **PAC 3** | **30 días** | **lun 09/04/18** | **mar 08/05/18** |
| 33 | | **Algoritmos de matching de geometrías** | **6 días** | **lun 09/04/18** | **sáb 14/04/18** |
| 34 | | Búsqueda de algoritmos sencillos de matching entre geometrías | 2 días | lun 09/04/18 | mar 10/04/18 |
| 35 | | Búsqueda de metodologías para definir el grado de similitud | 2 días | mié 11/04/18 | jue 12/04/18 |
| 36 | | Búsqueda de plugins QGIS similares (como fTools) | 2 días | vie 13/04/18 | sáb 14/04/18 |
| 37 | | **Comienzo desarrollo del algoritmo** | **10 días** | **dom 15/04/18** | **mar 24/04/18** |
| 38 | | Pruebas pseudocódigo algoritmo | 7 días | dom 15/04/18 | sáb 21/04/18 |
| 39 | | Implementación de consultas | 3 días | dom 22/04/18 | mar 24/04/18 |
| 40 | | **Análisis de resultados** | **9 días** | **mié 25/04/18** | **jue 03/05/18** |
| 41 | | Generar modelos | 3 días | mié 25/04/18 | vie 27/04/18 |
| 42 | | Crear diferentes propuestas | 3 días | sáb 28/04/18 | lun 30/04/18 |
| 43 | | Confirmar funcionalidad y viabilidad | 2 días | mar 01/05/18 | mié 02/05/18 |
| 44 | | Conclusiones | 1 día | jue 03/05/18 | jue 03/05/18 |
| 45 | | **Documentación** | **5 días** | **vie 04/05/18** | **mar 08/05/18** |
| 46 | | Informe de la PAC 3 | 3 días | vie 04/05/18 | dom 06/05/18 |
| 47 | | Entrega borrador PAC 3 | 0 horas | dom 06/05/18 | dom 06/05/18 |
| 48 | | Modificaciones memoria del proyecto | 2 días | lun 07/05/18 | mar 08/05/18 |
| 49 | | Entrega PAC 3 | 0 horas | mar 08/05/18 | mar 08/05/18 |
| 50 | | **PAC 4** | **28 días** | **mié 09/05/18** | **mar 05/06/18** |
| 51 | | **Implementar plugin QGIS 3.0 matching entre geometrías** | **7 días** | **mié 09/05/18** | **mar 15/05/18** |
| 52 | | Trasladar pseudocódigo del algoritmo a Python | 7 días | mié 09/05/18 | mar 15/05/18 |
| 53 | | Memoria del proyecto | 7 días | mié 16/05/18 | mar 22/05/18 |
| 54 | | Creación diapositivas Presentación Virtual | 6 días | mié 23/05/18 | lun 28/05/18 |
| 55 | | Grabación Presentación Virtual con audio | 2 días | mar 29/05/18 | mié 30/05/18 |
| 56 | | Entrega borrador PAC 4 | 0 horas | mié 30/05/18 | mié 30/05/18 |
| 57 | | Modificar memoria del proyecto | 3 días | jue 31/05/18 | sáb 02/06/18 |
| 58 | | Modificar presentación virtual | 2 días | dom 03/06/18 | lun 04/06/18 |
| 59 | | Entrega PAC 4 | 1 día | mar 05/06/18 | mar 05/06/18 |
| 60 | | Hito externo 2: Inicio del debate virtual | 1 día | mar 19/06/18 | mar 19/06/18 |
| 61 | | **Debate Virtual** | **3 días** | **mar 19/06/18** | **jue 21/06/18** |
| 62 | | Debat Virtual 1/3 | 1 día | mar 19/06/18 | mar 19/06/18 |
| 63 | | Debat Virtual 2/3 | 1 día | mié 20/06/18 | mié 20/06/18 |
| 64 | | Debat Virtual 3/3 | 1 día | jue 21/06/18 | jue 21/06/18 |
| 65 | | Hito externo 3: Cierre del proyecto | 1 día | jue 21/06/18 | jue 21/06/18 |



# Anexo 2

**Código Python de la librería de QGIS *Spatial Join (Unir atributos por localización).***

```
import os

from qgis.PyQt.QtGui import QIcon

from qgis.core import (QgsFields,
                       QgsFeatureSink,
                       QgsFeatureRequest,
                       QgsGeometry,
                       QgsProcessing,
                       QgsProcessingUtils,
                       QgsProcessingParameterBoolean,
                       QgsProcessingParameterFeatureSource,
                       QgsProcessingParameterEnum,
                       QgsProcessingParameterField,
                       QgsProcessingParameterFeatureSink)

from processing.algs.qgis.QgisAlgorithm import QgisAlgorithm
from processing.tools import vector

pluginPath = os.path.split(os.path.split(os.path.dirname(__file__))[0])[0]


class SpatialJoin(QgisAlgorithm):
    INPUT = "INPUT"
    JOIN = "JOIN"
    PREDICATE = "PREDICATE"
    JOIN_FIELDS = "JOIN_FIELDS"
    METHOD = "METHOD"
    DISCARD_NONMATCHING = "DISCARD_NONMATCHING"
    OUTPUT = "OUTPUT"

    def icon(self):
        return QIcon(os.path.join(pluginPath, 'images', 'ftools', 'join_location.png'))

    def group(self):
        return self.tr('Vector general')

    def groupId(self):
        return 'vectorgeneral'

    def __init__(self):
        super().__init__()

    def initAlgorithm(self, config=None):
        self.predicates = (
            ('intersects', self.tr('intersects')),
            ('contains', self.tr('contains')),
            ('equals', self.tr('equals')),
            ('touches', self.tr('touches')),
            ('overlaps', self.tr('overlaps')),
            ('within', self.tr('within')),
            ('crosses', self.tr('crosses')))

        self.reversed_predicates = {'intersects': 'intersects',
                                    'contains': 'within',
                                    'isEqual': 'isEqual',
                                    'touches': 'touches',
                                    'overlaps': 'overlaps',
                                    'within': 'contains',
                                    'crosses': 'crosses'}
```

```
        self.methods = [
            self.tr('Create separate feature for each located feature'),
            self.tr('Take attributes of the first located feature only')
        ]

        self.addParameter(QgsProcessingParameterFeatureSource(self.INPUT,
                                                              self.tr('Input layer'),
                                                              [QgsProcessing.TypeVectorAnyGeometry]))
        self.addParameter(QgsProcessingParameterFeatureSource(self.JOIN,
                                                              self.tr('Join layer'),
                                                              [QgsProcessing.TypeVectorAnyGeometry]))

        predicate = QgsProcessingParameterEnum(self.PREDICATE,
                                               self.tr('Geometric predicate'),
                                               options=[p[1] for p in self.predicates],
                                               allowMultiple=True, defaultValue=[0])
        predicate.setMetadata({
            'widget_wrapper': {
                'class': 'processing.gui.wrappers.EnumWidgetWrapper',
                'useCheckBoxes': True,
                'columns': 2}})
        self.addParameter(predicate)
        self.addParameter(QgsProcessingParameterField(self.JOIN_FIELDS,
                                                      self.tr('Fields to add (leave empty to use all fields)'),
                                                      parentLayerParameterName=self.JOIN,
                                                      allowMultiple=True, optional=True))
        self.addParameter(QgsProcessingParameterEnum(self.METHOD,
                                                     self.tr('Join type'), self.methods))
        self.addParameter(QgsProcessingParameterBoolean(self.DISCARD_NONMATCHING,
                                                        self.tr('Discard records which could not be joined'),
                                                        defaultValue=False))
        self.addParameter(QgsProcessingParameterFeatureSink(self.OUTPUT,
                                                            self.tr('Joined layer')))
    def name(self):
        return 'joinattributesbylocation'

    def displayName(self):
        return self.tr('Join attributes by location')

    def tags(self):
        return self.tr("join,intersects,intersecting,touching,within,contains,overlaps,relation,spatial").split(',')

    def processAlgorithm(self, parameters, context, feedback):
        source = self.parameterAsSource(parameters, self.INPUT, context)
        join_source = self.parameterAsSource(parameters, self.JOIN, context)
        join_fields = self.parameterAsFields(parameters, self.JOIN_FIELDS, context)
        method = self.parameterAsEnum(parameters, self.METHOD, context)
        discard_nomatch = self.parameterAsBool(parameters, self.DISCARD_NONMATCHING, context)

        source_fields = source.fields()
        fields_to_join = QgsFields()
        join_field_indexes = []
        if not join_fields:
            fields_to_join = join_source.fields()
            join_field_indexes = [i for i in range(len(fields_to_join))]
        else:
            for f in join_fields:
                idx = join_source.fields().lookupField(f)
                join_field_indexes.append(idx)
                if idx >= 0:
                    fields_to_join.append(join_source.fields().at(idx))
```

```
        out_fields = QgsProcessingUtils.combineFields(source_fields, fields_to_join)

        (sink, dest_id) = self.parameterAsSink(parameters, self.OUTPUT, context,
                                               out_fields, source.wkbType(), source.sourceCrs())

        # do the join

        # build a list of 'reversed' predicates, because in this function
        # we actually test the reverse of what the user wants (allowing us
        # to prepare geometries and optimise the algorithm)
        predicates = [self.reversed_predicates[self.predicates[i][0]] for i in
                      self.parameterAsEnums(parameters, self.PREDICATE, context)]

        remaining = set()
        if not discard_nomatch:
            remaining = set(source.allFeatureIds())

        added_set = set()

        request = QgsFeatureRequest().setSubsetOfAttributes(join_field_indexes).setDestinationCrs(source.sourceCrs()
        features = join_source.getFeatures(request)
        total = 100.0 / join_source.featureCount() if join_source.featureCount() else 0
        for current, f in enumerate(features):
            if feedback.isCanceled():
                break

            if not f.hasGeometry():
                continue

            bbox = f.geometry().boundingBox()
            engine = None

            request = QgsFeatureRequest().setFilterRect(bbox)
            for test_feat in source.getFeatures(request):
                if feedback.isCanceled():
                    break
                if method == 1 and test_feat.id() in added_set:
                    # already added this feature, and user has opted to only output first match
                    continue

                join_attributes = []
                for a in join_field_indexes:
                    join_attributes.append(f.attributes()[a])

                if engine is None:
                    engine = QgsGeometry.createGeometryEngine(f.geometry().constGet())
                    engine.prepareGeometry()

                for predicate in predicates:
                    if getattr(engine, predicate)(test_feat.geometry().constGet()):
                        added_set.add(test_feat.id())

                        # join attributes and add
                        attributes = test_feat.attributes()
                        attributes.extend(join_attributes)
                        output_feature = test_feat
                        output_feature.setAttributes(attributes)
                        sink.addFeature(output_feature, QgsFeatureSink.FastInsert)
                        break

            feedback.setProgress(int(current * total))

        if not discard_nomatch:
            remaining = remaining.difference(added_set)
            for f in source.getFeatures(QgsFeatureRequest().setFilterFids(list(remaining))):
                if feedback.isCanceled():
                    break
                sink.addFeature(f, QgsFeatureSink.FastInsert)

        return {self.OUTPUT: dest_id}
```

# *Anexo 3*

**Sección *run* del código Python implementado "*matching vectorial*".**

```
187     def run(self):
188         """Run method that performs all the real work"""
189         # show the dialog
190         self.dlg.show()
191         # Run the dialog event loop
192         result = self.dlg.exec_()
193         # Comprueba si se ha pulsado Aceptar (OK)
194         if result == 1:
195
196             # Hace un listado de todas las capas cargadas en QGIS
197             layers = QgsProject.instance().mapLayers().values()
198
199             # Busca cual es la capa con el nombre EDIFICIOS...
200             for layer in layers:
201                 if layer.name() == "EDIFICIOS":
202                     # ... averigua la ruta en el disco duro de esta capa ... y
203                     myfilepath_EDIFICIOS = layer.dataProvider().dataSourceUri()
204                     # ... deduce el nombre de la ruta hasta la carpeta que la contiene y el nombre del
                        fichero shape
205                     (myDirectory_EDIFICIOS,nameFile_EDIFICIOS) = os.path.split(myfilepath_EDIFICIOS)
206
207             # Lo mismo que el bloque anterior de código pero para la capa PUNTOS (PoI)
208             for layer in layers:
209                 if layer.name() == "PUNTOS":
210                     myfilepath_PUNTOS = layer.dataProvider().dataSourceUri()
211                     (myDirectory_PUNTOS,nameFile_PUNTOS) = os.path.split(myfilepath_PUNTOS)
212
213             # Ruta donde se guardará la capa resultante de la union de atributos por localizacion
                Capa_matching_vectorial (en formato shapefile).
214             OUTPUT = str(myDirectory_EDIFICIOS) + '/' + 'Capa_matching_vectorial.shp'
215
216             # Parametros del geoproceso de union de atributos por localizacion.
217                 #INPUT: Capa de entrada (puntos).
218                 #JOIN: Capa que hará el matching (edificios).
219                 #PREDICATE: [5]= Predicado geométrico "dentro".
220                 #DISCARD_NONMATCHING [True]= Descartará los que no cumplan el predicado geométrico.
221             parameters = {'INPUT':myfilepath_PUNTOS,'JOIN': myfilepath_EDIFICIOS,'PREDICATE':[5],
                'JOIN_FIELDS':[],'METHOD':0,'DISCARD_NONMATCHING':True,'OUTPUT':OUTPUT}
222             feedback = QgsProcessingFeedback()
223
224             # Ejecución del geoproceso y carga en el proyecto de la capa resultante
                Capa_matchimp_vectorial
225             processing.runAndLoadResults("qgis:joinattributesbylocation", parameters, feedback =
                feedback)
226
227             # Dar el estilo de simbolización deseado (como se almacenará la capa en QGIS y en el disco
                duro).
228             STYLE = str(myDirectory_EDIFICIOS) + '/' + 'Capa_matching_vectorial.qml'
229             Capa_unida = QgsProject.instance().mapLayersByName('Capa unida')[0]
230             Capa_unida.loadNamedStyle(STYLE)
231             pass
```