SIBYL

(SeIsmic monitoring and vulnera**B**ilit**Y** framework for civi**L** protection)

Agreement number: ECHO/SUB/2014/695550

Deliverable DB2: Software platform including processing tools with related manual

Version 1.0 February 2016

Project start date: 01.01.2015 End date: 31.12.2016

Coordinator: Prof. Dr. Stefano Parolai
Centre for Early Warning Systems
Helmholtz Centre Potsdam GFZ German Research Centre for Geosciences, Potsdam, Germany

Contents

Introduction

A set of tools for the processing and analyses of remote sensing imagery and optimising the in situ data collection activities have been expanded upon and developed within the SIBYL project.

A virtual platform, composed by different modules tightly interconnected, is currently being developed by GFZ and in this document a brief overview of the available tools composing the platform is provided.

The platform, named REM (Rapid Environmental Mapping) aims at providing a modular and efficient solution for Civil Protection (CP) agencies and other risk practitioners concerned with natural hazards. The development of the REM platform stems from the observation that in real-world risk-assessment applications, a critical role is played by the exposure and vulnerability model. A poor (incomplete, sparse, uncertain) knowledge of the geographical area exposed to natural hazards such as earthquakes or floods will certainly result in uncertain estimates of the potential consequences.

The development of exposure models traditionally entails the collection of a significant amount of data in the field, with engineering procedures that have been developed for single (usually critical) structures. Since these procedures do not scale well with the geographical extent of a risk-assessment application, the resulting burden (both economical and in terms of resources) might hinder the whole process.

It is therefore of paramount importance to ensure the provision of methodologies, technologies and tools able to streamline the exposure modelling process while ensuring a satisfactory level of accuracy and timeliness.

In the first section of this document the main structure of the REM platform will be described and discussed. Since a critical component of the platform is the database, a section is devoted to describing and discussing its structure and features. The components developed or upgraded within the SIBYL projects will then be presented in the subsequent sections. In the last section an outlook of the research and development currently in progress to further advance the REM platform is provided and discussed. A reading list is also provided for the interested reader.

The REM Platform

In the following, the main structure of the REM platform is introduced, and the current components are described. The platform is continuously evolving while new functionalities are added.

Main components

The main components of REM platform are depicted in Figure 1. The core of the system is represented by the database, which hosts the assets data (geometry, attributes and qualifiers) and the collected images.

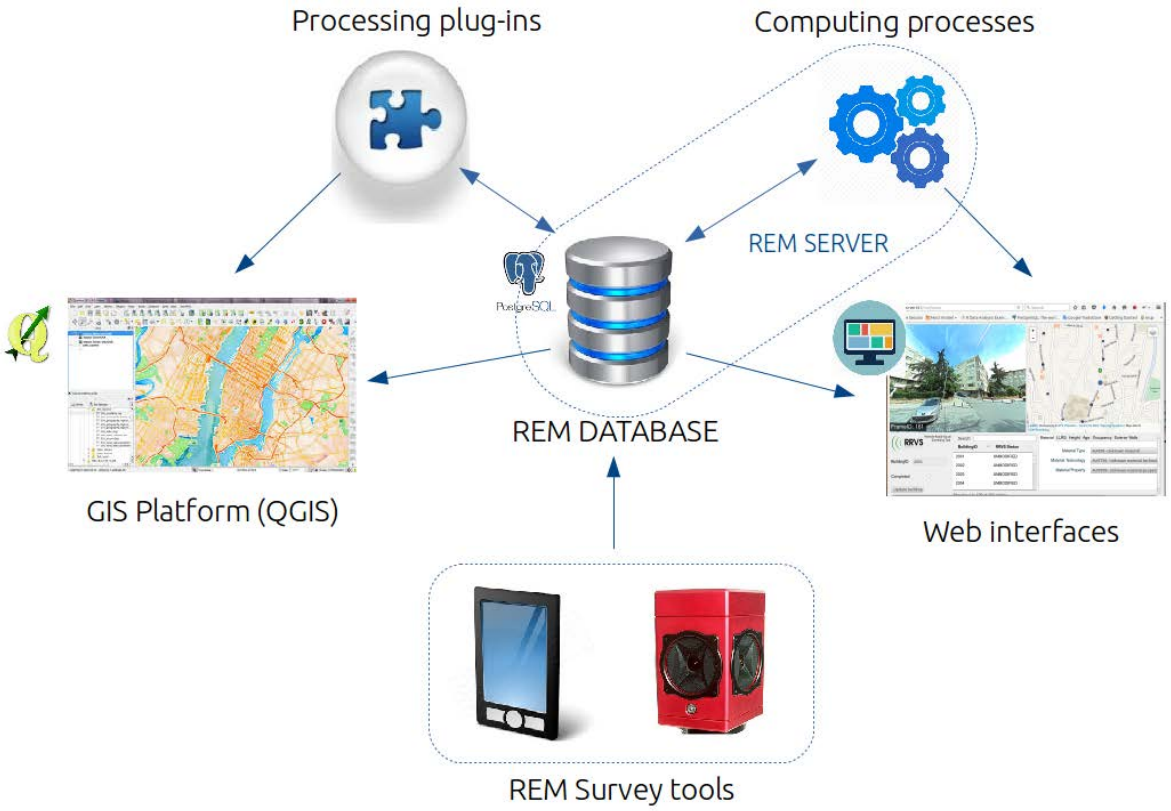

Figure 1 The main components of the REM (Rapid Environmental Monitoring) platform.

REM Database

The database model has been designed to serve the needs for efficient storage and management of data represented at different spatial scales and changing over time. It forms the basis for the implementation of a multi-resolution sampling framework.

The implementation is based on the free and open-source, unix-based and object-relational database management system PostgreSQL[1]. It supports most Structured Query Language (SQL) constructs, including sub selects, transactions and user-defined types and functions as well as many standard data types including date / time types. The latest database schema, shown in Figure 2, can be downloaded directly from the git source:

https://github.com/GFZ-Centre-for-Early-Warning/REM_DBschema

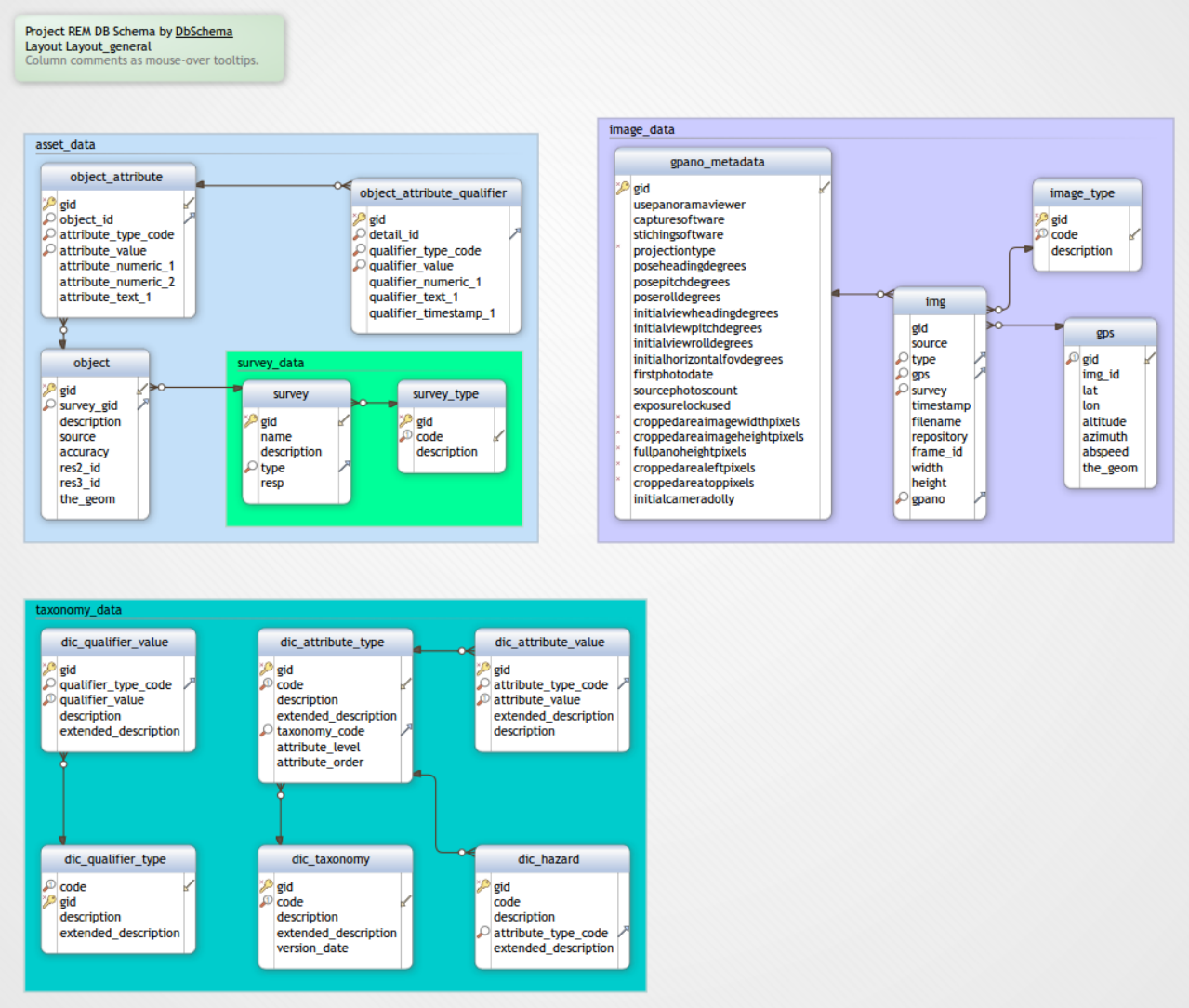

Figure 2 REM database schema. The three table groups refer to the main entities in the database: taxonomy, assets and images.

For spatial functions, the free and open-source spatial database extension PostGIS[2] is applied. It adds support for geographic objects to the PostgreSQL object-relational database. PostGIS follows the OpenGIS “Simple Features Specification for SQL” and since version 2.0 supports both vector and raster objects and related spatial query capabilities. Also, topological models are

[1] http://www.postgresql.org/

[2] http://postgis.net/

supported to handle objects with shared boundaries. To deal with the multi-representation of spatial objects, a bottom-up approach is followed where datasets of different resolution are linked by using additional attributes, which identify the corresponding objects in the lower levels of detail. Two cases of multi-resolution datasets can be distinguished and are supported by the database prototype. First, datasets at different resolutions are generated independently and need to be linked in the database by matching procedures. Second, lower resolution representations are derived from higher resolution data by generalization functions and their link is established by the generalization process.

The database model has been designed to be able to integrate and manage various kinds of indicators that follow possibly different standard taxonomies which are, moreover, likely to depend on the type of hazard and/or on the considered phase of the disaster management cycle (pre-disaster vulnerability, post-disaster recovery and reconstruction). In its current implementation, the model could be validated against the GEM Building Taxonomy. However, minor adjustments were needed to the prototype model in order to fit the final list of indicators to be agreed upon in the next phase of the project.

The database model is structured into three schemas. The *asset* schema contains the main description of the database objects, their spatial reference and representations at different resolutions. Besides the spatial properties, the assets are characterized by attributes and qualifier values which are defined within the taxonomy schema.

Assets taxonomy

The database model is intended to integrate and manage various kinds of structural and non-structural features of the exposed assets which are relevant for describing and monitoring exposure and vulnerability to earthquakes (and other natural hazards). The specific indicators should follow standard existing taxonomies, which are likely to depend on the type of hazard and/or on the considered phase of the disaster management cycle (pre-disaster vulnerability, post-disaster recovery and reconstruction).

The project SIBYL focuses on building structures that are mostly used for residential purposes. As a basis, the earthquake-focused taxonomy originally developed with the GEM (Global Earthquake Model[3]) has been proposed.

The GEM Taxonomy has been designed to describe hundreds of building typologies with a global scope, so it is well positioned to be further extended to other hazards, and can be organized as a series of expandable tables, which contain various attributes referring to specific building characteristics. The specific states of the attributes are given by the values which are represented by unique IDs in alphanumeric format associated with the corresponding text descriptions. The taxonomy allows one to describe buildings at different levels of detail, where the number of attributes or the depth of information to be captured depends on the aim of the study, available data sources, and the type of data collection approach. More details and a full list of attributes and associated values included in the GEM Building Taxonomy can be found in Brzev et al. (2012).

[3] http://www.globalquakemodel.org/

Taxonomy attributes are organized in a dictionary table (taxonomy.dic_attribute_type) within the relational database. Each attribute type is assigned a unique ID in alphanumeric format and is linked to a textual description. Optional columns to specify the attribute level (in case of multi-level taxonomies such as the GEM Taxonomy) and attribute order (in case a specific output string needs to be generated from a set of attributes for non-database applications, this column provides an indication about the order in which the attributes need to be assembled) are provided. Categorical attribute values (taxonomy.dic_attribute_value) are stored in a separate dictionary table and are also defined by unique IDs and linked with a textual description. Numeric or textual attribute values are inserted for the single object primitives directly in the table object.main_detail. Each attribute type needs to be associated with one or more hazard types (taxonomy.hazard) and be linked to a specific taxonomy (taxonomy.dic_taxonomy) in order to clearly categorize the attributes and to identify their source and application.

In addition to attribute types and values, object qualifiers are also defined within the taxonomy schema of the database prototype. Qualifiers in the context of this work refer to additional descriptors of object primitives that are independent of the type of hazard and the application. They are more closely linked to uncertainty treatment and knowledge life-cycle management. Potential qualifiers to be supported could include accuracy, precision, quality or valid time. Qualifier types (taxonomy.dic_qualifier_type) are defined by unique IDs and linked with a textual description. The specific states of the qualifier types are given by values which are stored in a separate dictionary table (taxonomy.dic_qualifier_value) in case of categorical variables. Numeric or textual qualifier values are inserted for the single object primitives directly in the table object.main_detail_qualifier.

The current attributes and the respective values are listed in Appendix A.

Information life-cycle support

In order to allow for a detailed description of the temporal variability within objects, a bi-temporal representation of time with *transaction time* and *valid time* has been implemented at the primitive spatial object level. Valid time is considered as an object qualifier and is therefore specified for each record by qualifier timestamps. Valid time refers to real-world timestamps and therefore, unlike the transaction time, it needs to be set for each record (i.e., each entry in the database) by the user or the application that inserts or updates the records as a consequence of a real world change. Therefore, even continuous real world changes are detected in discrete steps in the database. This is, however, in line with the envisaged remote sensing application for which the database is intended to, because remote sensing analysis itself is dependent on the (stepwise) acquisition time of satellite images over an area of interest. Moreover, the satellites have specific revisit periods or temporal resolutions which depend on their orbit, sensor specifications and number of satellites (in case of a satellite constellation) and which practically define the maximum level of temporal granularity. In order to define the lifespan of real world objects, two valid timestamps are set in the object.main_detail_qualifier table: “valid from” (qualifier_timestamp_1) and “valid to” (qualifier_timestamp_2). “Valid from” is defined after insert of a record and “valid to” is defined before delete of a record.

To be able to keep track of the history of the real world objects, a live-history approach is followed. This means that deleted or changed records are archived in the history schema along with additional information about the corresponding database transaction. A generic trigger function history.if_modified_func() was defined which archives the records and logs transaction time and optional additional information about changes to selected tables and attributes. The function is based onPostgreSQL audit trigger[4] and was modified according to the requirements of this project. The logging of transactions can be done at a statement level or a row level. Control is for each logged table separately and columns to be logged can be specified individually. Row values are recorded as hstore fields instead of text, which allows for more sophisticated queries on the history and reduces query complexity and storage space. The logged information includes the following:

- Schema name, table name, table OID, transaction ID: identifiers for the changed table and transaction.
- Transaction user: session user name.
- Transaction time: current timestamp to log the start time of the transaction.
- Transaction query: optional the query text can be logged.
- Transaction type: INSERT, UPDATE, DELETE, TRUNCATE.
- Old record: row value before the change or after in case of INSERT.
- New record: new values of the changed columns in case of UPDATE.
- Changed fields: field affected by the change.

The combination of a bi-temporal data model with a live-history approach allows for the straight forward recovery of former states of the database at defined transaction or valid times and for sophisticated temporal queries. A set of functions have been implemented to exemplify the querying of transaction and valid time, temporal properties and temporal relationships. Preliminary temporal query functions include the following.

- ttime_inside(ttime_from, ttime_to): This function selects from history.logged_actions the records that have been modified in the logged tables at some timestamp inside the defined transaction time range. For each record the latest version within the defined transaction time range is selected.
- ttime_equal(ttime): This function selects from history.logged_actions the records that have been modified in the logged tables at a timestamp that equals the defined transaction time.
- ttime_gethistory(): This function selects from history.logged_actions all records that have been modified in the logged tables. It provides with the transaction time history.
- vtime_inside(vtime_from, vtime_to): This function selects from history.logged_actions the records that have been subject to a real world change at some timestamp inside the defined valid time range. For each record the latest version within the defined valid time range is selected.

[4] http://wiki.postgresql.org/wiki/Audit_trigger_91plus

- vtime_intersect(vtime_from, vtime_to): This function selects from history.logged_actions the records that have been subject to a real world change at some timestamp that intersects with the defined valid time range. For each record the latest version within the defined valid time range is selected.
- vtime_equal(ttime_from, ttime_to): This function selects from history.logged_actions the records that have a valid time range that equals the defined valid time range.
- vtime_gethistory(): This function selects from history.logged_actions all records that have been subject to a real world change. For each record the latest version at each valid time is selected. It provides with the valid time history.

Main operational tools

Survey planning

During the survey planning phase the following processing tools can be used:

1. REM SATEX tool. Provided as a QGIS plugin, this tool allows the user to obtain a preliminary Land Use/Land Cover (LULC) stratification of a wide geographic area by supervised classification of LANDSAT imagery;
2. REM routing tool. This tool allows to the user generate optimized routes based on a set of sampling points, possibly generated according to the stratification produced by the SATEX tool.

REM_SATEX tool

This plugin provides two algorithms for the processing of one or multiple Landsat scenes within a region of interest with the aim of undertaking the Landuse/Landcoverage classification streamlining of all required processing steps to perform a libsvm/orfeo toolbox[5] (OTB) pixel based classification. Please refer to the SIBYL deliverables DB1 and DB3 for further technical information on the plugin.

The Plugin is structured into two modules:

1. Preprocessing
2. Classification

In the **Preprocessing** module (see Figure 3) Landsat scenes located in a directory, e.g., the directory created by extracting from the downloaded zip archive of a Landsat 8 scene, as can be found on EarthExplorer[6], are 1) cropped to the region of interest provided as, e.g., a polygon feature in a vector file and then 2) the separate spectral Bands are stacked and 3) a virtual raster tile is created out of these, i.e., in case the region of interest stretches over more than one Landsat scene, these are virtually mosaicked.

[5] https://www.orfeo-toolbox.org/

[6] http://earthexplorer.usgs.gov/

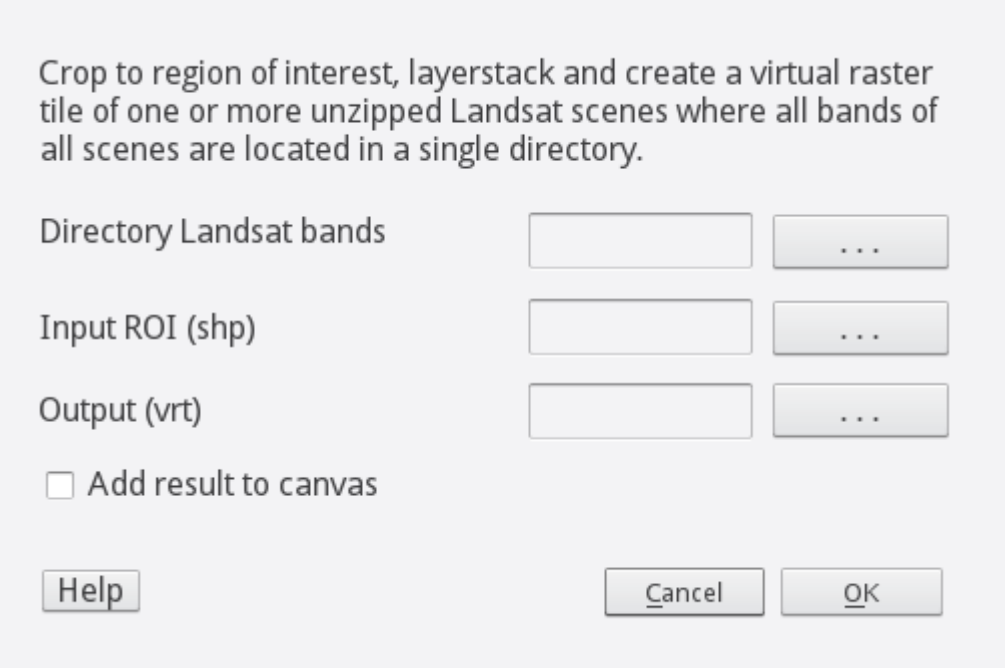

Figure 3 Preprocessing dialog in the SATEX plugin for QGIS.

If present, the panchromatic band 8 (available from Landsat 7 and 8 images) is excluded from the layers. The **Classification** module (see Figure 4) performs a classification of a raster file, e.g., the one resulting from the **Preprocessing** algorithm, by either using a provided trained Support Vector Model (SVM) from OTB or training and testing a SVM on the fly using a provided ground truth testing/training data set. In the case the on-the-fly training/testing is performed, the provided ground truth data is randomly split into a testing (~20%) and a training part (~80%), the latter is then used in the libsvm implementation of OTB to create a SVM. This SVM (or the external SVM) are then used to classify the image. The resulting raster file with class labels is then tested with the testing dataset (all features of the provided vector layer in case an external SVM model was provided) and a confusion matrix is produced. Finally the resulting raster file is sieved (i.e., regions consisting of view pixels are merged to the surrounding). An example of the final stratification obtained is shown in Figure 5.

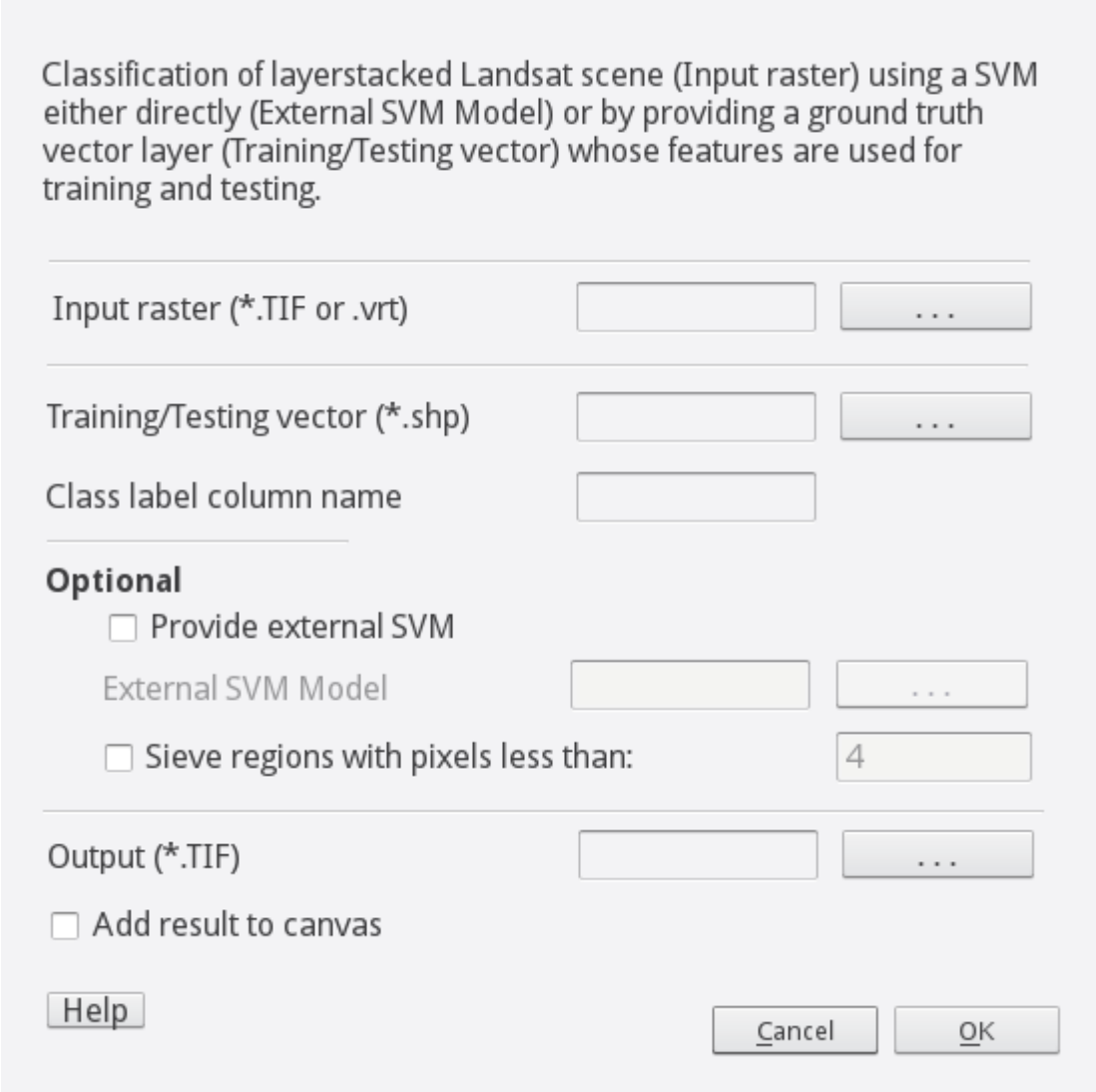

Figure 4 Classification dialog in the SATEX plugin for QGIS.

REM optimized routing tool

The stratification obtained in the preceding section can now be used to generate an adequate sampling distribution on the ground. A set of points is therefore generated, in order to sample the geo-cells of the considered area according to their classification and proportional to their surface coverage. The resulting set of points is shown in Figure 6.

The obtained sampling points can be used as input to the routing engine, together with a topologically corrected street network of the area of interest.

In order to optimize the routing, a number of sample points are randomly selected from the sampling set defined above, and used to select a related set of road segments, while will compose the path of the mobile mapping system. However, it is also important to define in which order to visit the selected road segments in order to make the data collection time- and cost-efficient. Moreover, in-situ data capturing, especially in urban areas, may be influenced by driving restrictions (accessibility, turn-restrictions, one-way streets, etc.) and cost factors (length, time, money, etc.).

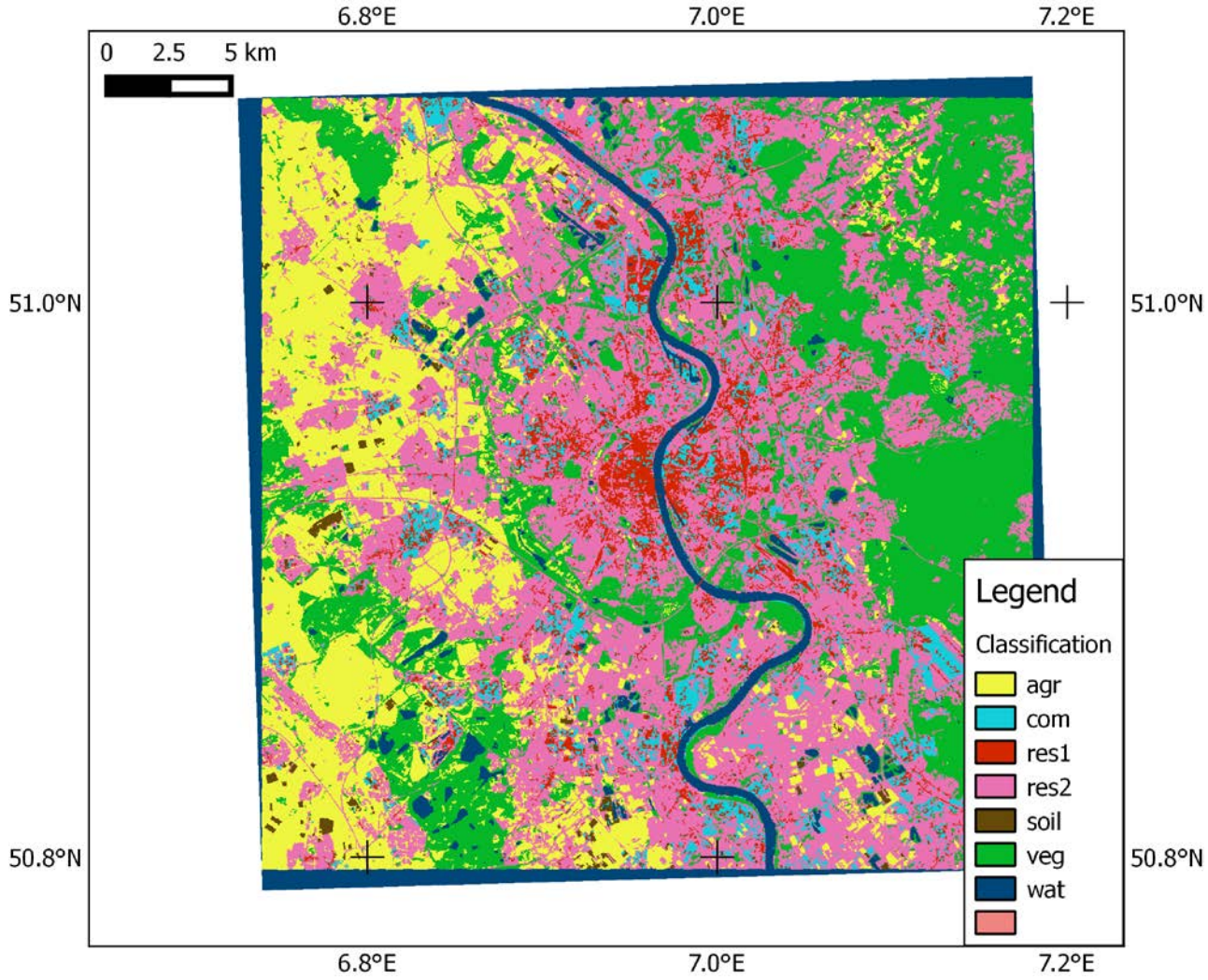

Figure 5 Resulting pixel-based classification of the input Landsat image corresponding to the selected ROI. The pixels are colored according to the specific class estimated by the statistical learning machine. Example is of the city of Cologne, Germany.

The main steps that are involved in tackling the aforementioned challenges include:

- Ordering the sample points based on a predefined cost function, especially the start and end points of the planned route.
- Finding the route through all the ordered stops that minimizes the cost function while considering the restrictions imposed by the road network.

As input data for the routing operation, a road network dataset must be provided. Such information is often available from qualified institutional sources, although simpler alternatives such as OpenStreetMap[7] (OSM), as for example shown in Figure 7, can also be used.

The data needs to be topologically corrected and are used to create a routable geometric network with defined cost-factors for travelling along street segments. The cost-factor used within a standard routing operation is the length of a street segment. Additional cost-factors and

[7] http://www.openstreetmap.org

restrictions, such as street quality, turn restrictions or traffic information, can be added to the network if available for an area of interest.

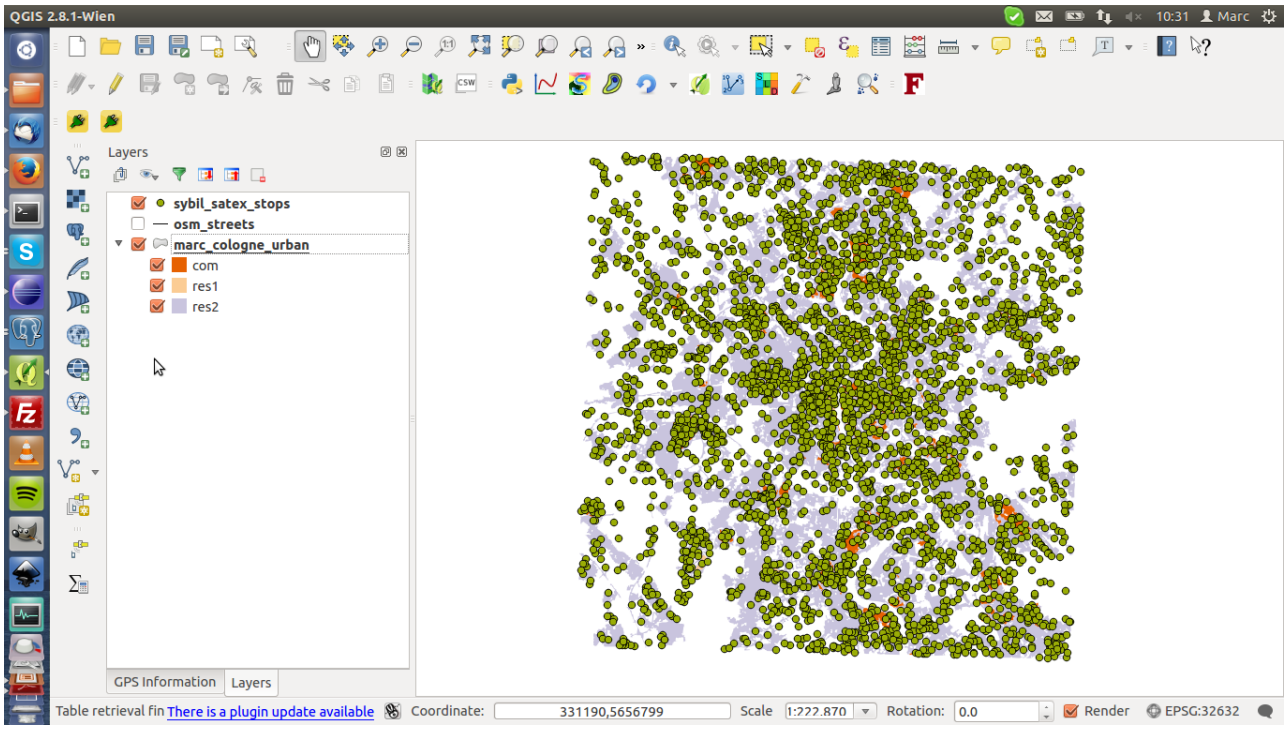

Figure 6 Generation of stratified sampling distribution, with proportional allocation, according to the considered classes. The example is the city of Cologne, Germany (see Figure 5).

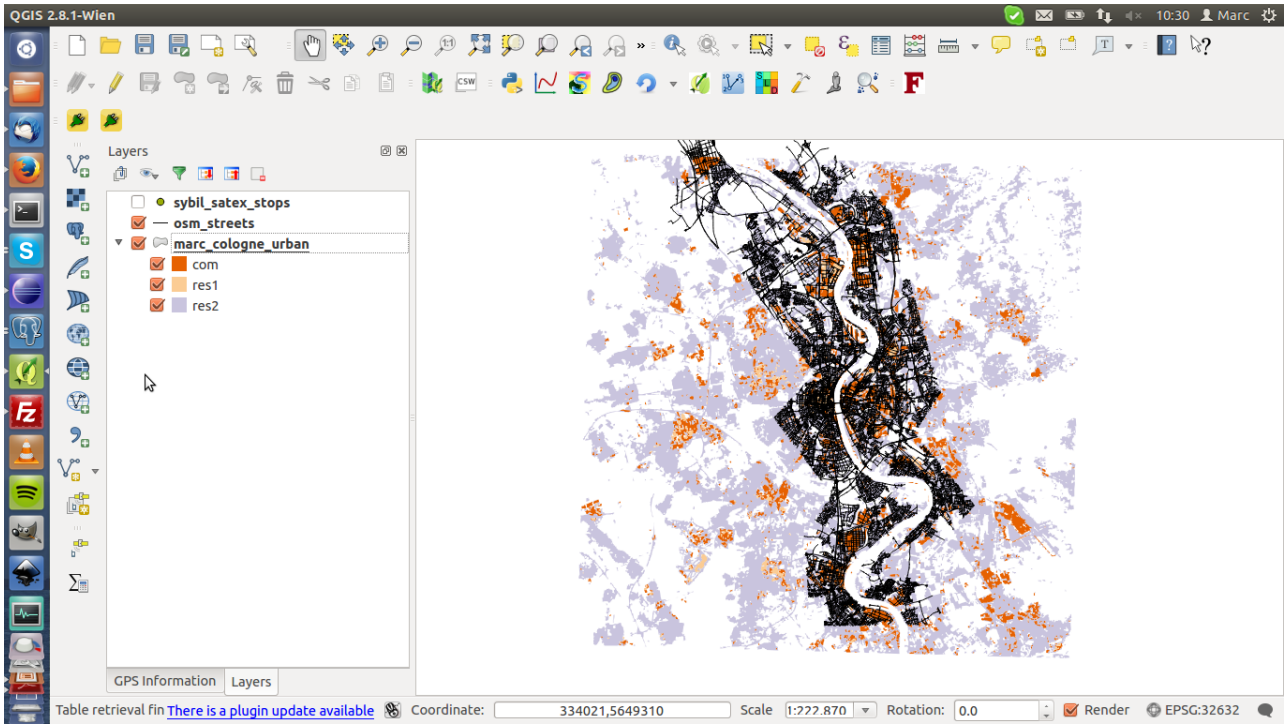

Figure 7 Road network obtained from OpenStreetMap (OSM) for the city of Cologne, Germany.

The actual routing problem can be reduced to the so-called Travelling Salesman Problem (Abraham and Roddick, 1999, TSP), a well-studied combinatorial optimization problem where a travelling salesman is required to visit all the stops on his list only once in order to minimize the costs. To solve the TSP, a routing engine can be implemented directly on the database (server-side). The routing engine is based on the pgrouting[8] extension to PostgreSQL and implements a set of custom functions for advanced routing operations. The functions include, amongst others, solutions for the TSP under the consideration of custom cost functions and a multiple Dijkstra algorithm to determine the best route through a series of stops while minimizing the cost function. In a first step, the sampling points are filtered and mapped onto the street network to define route stops that should be covered during the field operation. The closest nodes of the street network are selected for each sampling point as route stops using a straight line distance from point-to-point. Only one route stop is selected in case multiple sampling points refer to the same network node. This effectively filters the sampling points based on their accessibility. Once identified, the route stops are fed into the routing engine and the TSP solver is applied, where the cost-factor to be used is defined as an attribute in the street network data. A Dijkstra algorithm (Abraham and Roddick, 1999) is than applied multiple times between the sorted stops in order to calculate the shortest path across all the stops (see Figure 8).

The routing engine can be successfully used to optimize the implementation of the planned survey (that is, the coverage of the sampling units selected according to the chosen sampling design) accounting for different time and cost- constraints which can significantly impact upon the required survey resources. For instance, placing a penalty on the repeated scan of the same street would force the routing engine to enlarge the geographical scope of the survey, adding potentially additional useful observations to the planned ones. Also, highly dynamic parameters, such as, for instance, real-time traffic information, might be considered in the routing phase which could also be conducted *in situ* using a mobile platform. This would also allow the mobile mapping system to adapt to changed environmental conditions without losing the general focus of the survey.

The final routing is shown in Figure 9.The routing engine has been implemented in a free, open-source environment by exploiting the computing capabilities of the postgreSQL/postGIS database solution. The tool is provided as SQL/pgsql code, and as QGIS plugin. The latest version can be downloaded from the GFZ public github repository:

https://github.com/GFZ-Centre-for-Early-Warning/REM_optimized_routing

Further details on the operational application of the plugin can be found in the Appendix B of this document and in SIBYL deliverable DB3.

[8] http://pgrouting.org

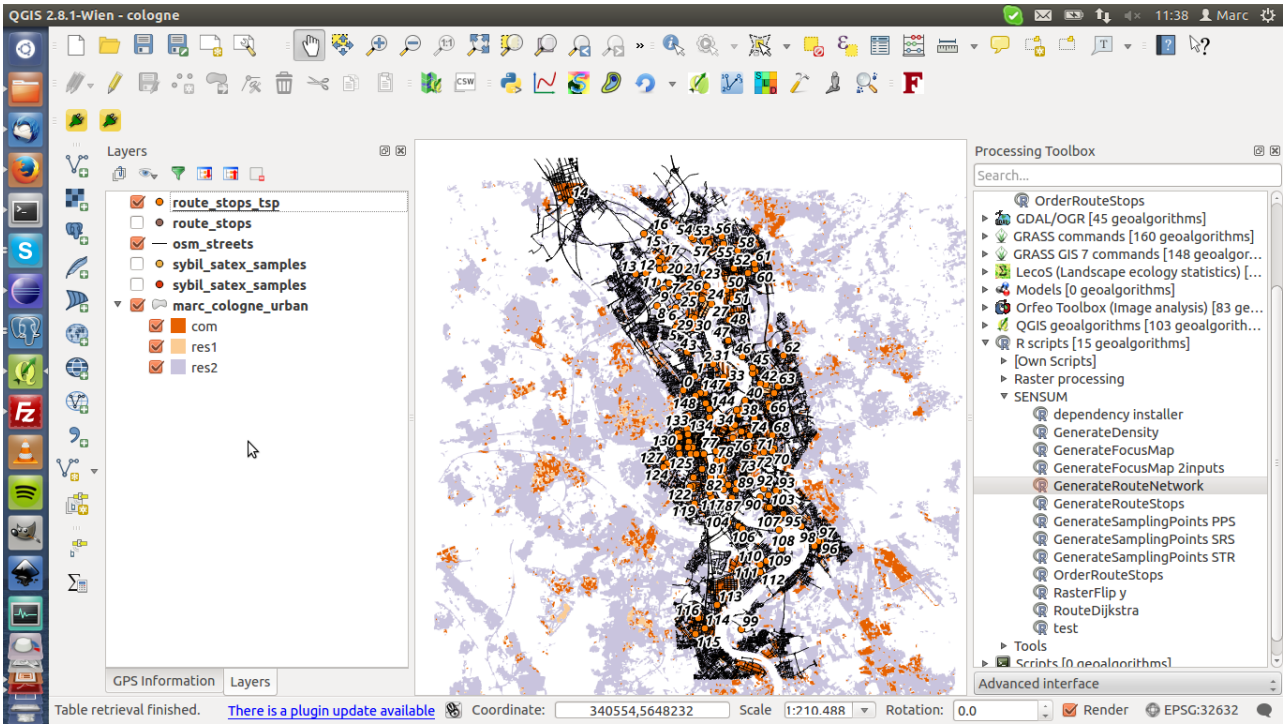

Figure 8 Route stops computed from the sampling set and the available road network.

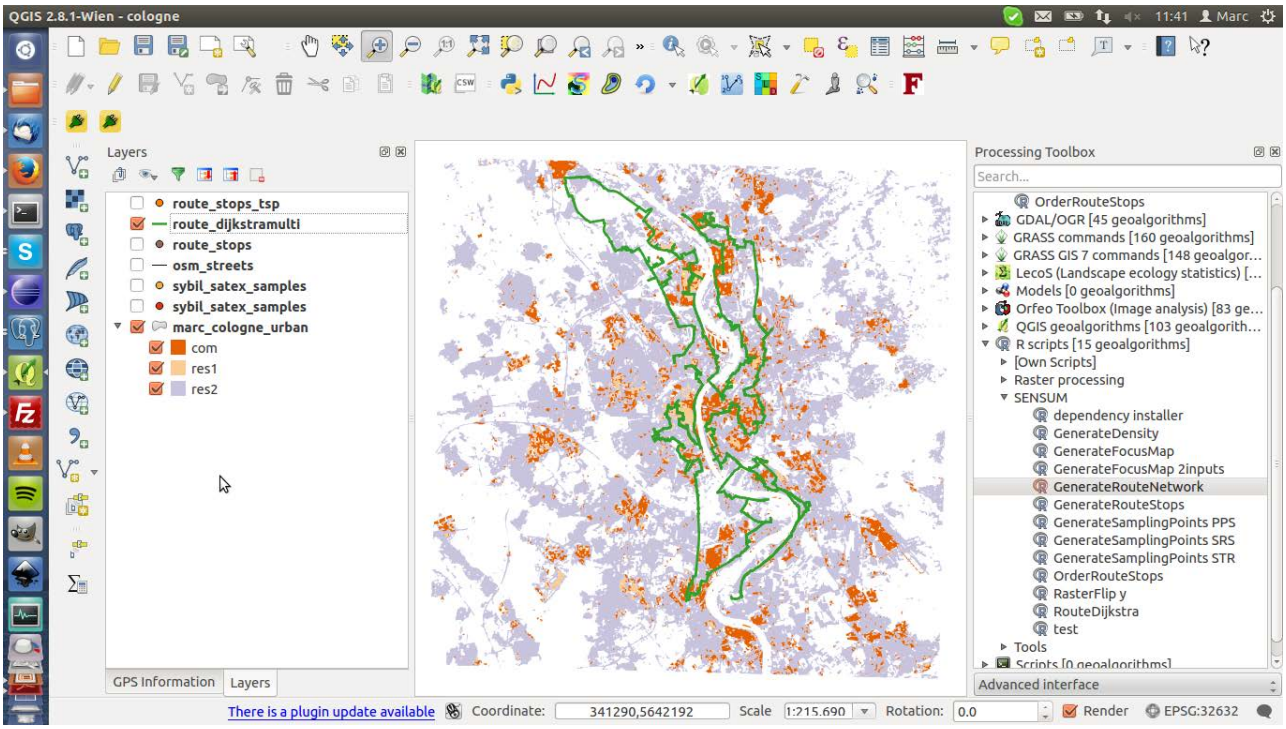

Figure 9 Final optimized route implementing the desired sampling on the ground.

In situ survey deployment

In order to efficiently carry out the survey as planned using the tools described in the preceding sections, the following solutions have been provided:

1. The GFZ-MOMA (MObile MApping) system. This is a lightweight mobile mapping system, consisting of a high-resolution omnidirectional camera and related acquisition unit and mechanical support, which allows for the rapid acquisition of georeferenced panoramic images that can then be analysed off-line.
2. REM RRVS (Remote Rapid Visual Survey) web platform. This web application allows different operators to access the REM database from remote, analyse the collected omnidirectional images (captured by the MOMA system) and fill in the visible attributes of selected buildings.

The GFZ-MOMA system

The GFZ-MOMA system, as discussed above, is designed for the rapid and efficient collection of in-situ exposure data which could be relevant for different natural and man-made applications (Pittore and Wieland, 2012, Wieland et al., 2014, see Figure 10). The system is composed of a Ladybug3 omnidirectional camera[9] from Point Grey Research Ltd., a data capturing and storage unit, a navigation unit and an external battery pack that supplies the energy for up to 6 hours of autonomous operation.

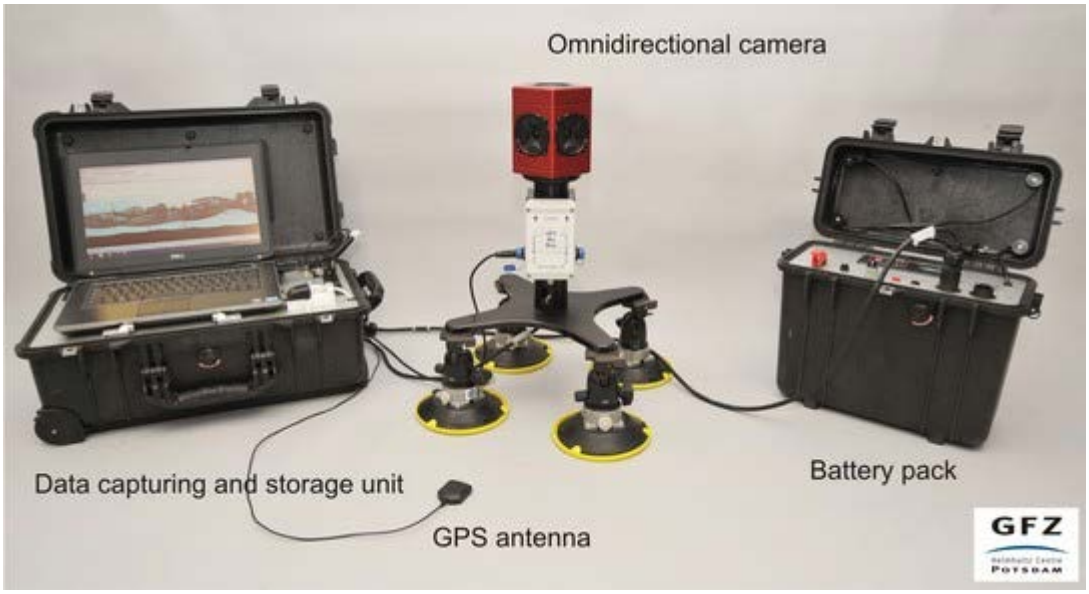

Figure 10 The GFZ-MOMA omnidirectional mobile mapping system, with data capturing and storage unit, omni-directional camera and battery pack.

[9] https://www.ptgrey.com/ladybug3-360-degree-firewire-spherical-camera-systems

The Ladybug3 camera is made up of 6 colour Complementary Metal Oxide Semiconductor (CMOS) sensors that capture concurrent image sequences with an acquisition rate of up to 15 fps (frames per second). The 6 single-camera image streams are synchronized and automatically stitched into an omnidirectional (panoramic) high resolution (5700x2700px) format with JPEG compression. The camera system is operated from inside a vehicle and is mounted on its roof by a simple system comprising a light-weight aluminium frame and 4 high-power suction cups.

The data capturing and storage unit has been designed and developed by GFZ with a specific focus on ease of use and ruggedness for robust outdoor applications, even under rough conditions (e.g., unpaved roads, dust). The main component of the unit is a standard notebook with a 750GB Serial ATA hard drive. A commercial-grade GPS receiver provides geo-localization. An optional Inertial Measurement Unit (IMU) can be used to record additional data about the camera's orientation. The notebook and all the other components are fixed into a rugged hard plastic case. A custom-designed software application captures, synchronizes and saves the different data streams coming from the camera, the GPS and the IMU. The synchronization of the data, within 125 msec, is based on the timer embedded in the ieee-1394B hardware controller. Location is associated with each omnidirectional image by a b-spline interpolation of GPS positioning.

The navigation unit uses geographical information systems (GIS) software as the main component for location tracking and car navigation. Its map interface is able to combine various background maps of the study area and to display pre-calculated sample areas and routes. The position can be tracked and displayed in real time with the GPS live tracking functionality. This allows an operator to not only navigate the car along pre-calculated routes, but also to reschedule the path on-the-fly to cope with unexpected environmental conditions (e.g., traffic jams or road blockages). All necessary software for capturing, storing, processing and visualizing the omnidirectional images recorded by the GFZ-MOMA system is provided with the system itself.

The operation of the system is relatively straightforward, and does not require special skills, nor particular tools or additional devices. The system can be easily mounted on different vehicles, including, for example, cars, vans and vehicles from Civil Protection and fire brigade units.

The REM_RRVS tool

The resulting high-density, high resolution, geo-referenced panoramic images of the urban environment can then be analysed by a skilled operator (for example, a local engineer who is familiar with the specific engineering practices in the region of interest) in order to infer the composition of the building stock and the way it changes according to the geographical location. The Rapid Remote Visual Screening is a modern version of the well-known Rapid Visual Screening methodology (see ATC-13 and FEMA-154[10] methodologies) largely used in the engineering community.

The geographic locations where the images have been captured are stored in a database. A complete solution for visualization, analysis and entry of the observed data is depicted in Figure 11. A desktop operator can efficiently conduct virtual observations of the geographical area of

[10] http://www.fema.gov/media-library/assets/documents/15212

interest, and store the captured attributes of the population in an efficient relational database, for later estimation and processing.

The main task of the RRVS tool is to quickly associate to each imaged building, described by its geographical coordinates or by its footprint in a GIS model, a set of structural and non-structural features included in the particular taxonomy considered. This information can then be used in the analysis phase to estimate the structural typology of the building, and its expected vulnerability with respect to an earthquake or some other natural hazards.

In order to accomplish this, a web-based platform (see Figure 11) has been developed within the project SIBYL for the remote, rapid screening of the buildings. On the top left, an omnidirectional image visualizer allows the user to undertake interactive browsing of the panoramic images. On the top right, an interactive map represents the selected buildings footprints and the available panoramic images, superimposed on an environmental map. In the bottom panel, a list of the buildings composing the specific task is provided, along with the status of the building's description. On the right side, a series of tabs allows the operator to populate the REM database with the structural and non-structural features observed on the buildings. Out of the attributes listed in the REM taxonomy (see Appendix A) only the ones visible from street-view perspective are included in the user interface.

The system can be accessed by remote through its public access point. The users have to provide a user name (previously registered into the system) and a task number. The task number picks up a subset of the buildings to be inspected, which are previously selected from the database according to the specific sampling schema to be realized. Every task is composed by a variable number of buildings to be inspected (e.g. 100 units). The spatial distribution of the buildings composing the task can vary according to the sampling approach.

The use of tasks allows several operators to work in parallel on the same dataset, therefore increasing the flexibility of the system and its potential applications. All elements of the interface are interactive. For instance, a user can click on an image icon in the map panel in order to load the corresponding image in the panoramic visualizer. Clicking on a building´s footprint in the map, the corresponding information will be queried in the database and will be used to populate the taxonomy tabs for reviewing and modification.

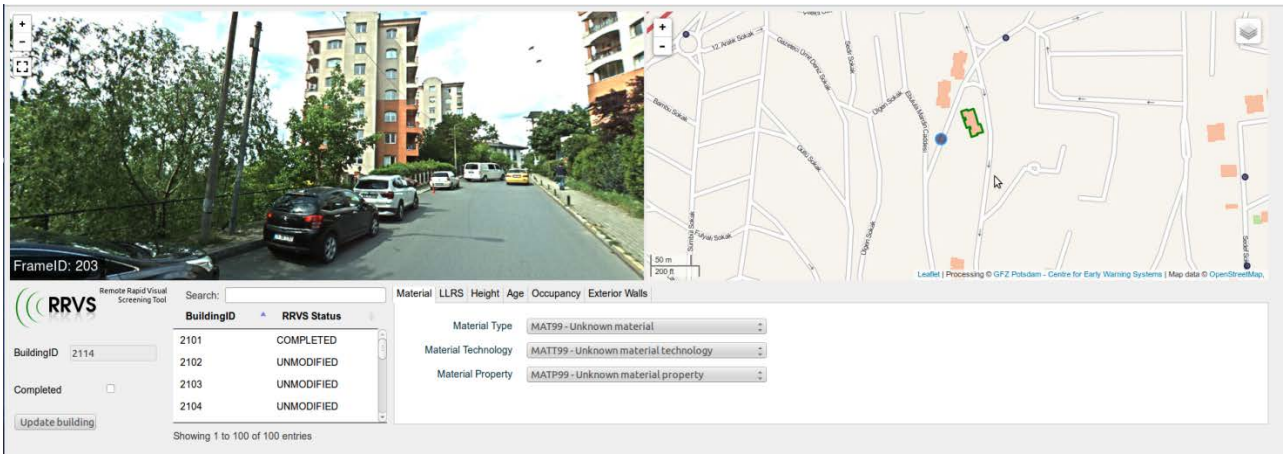

Figure 11 Web-based interface of the RRVS (Remote Rapid Visual Screening) Tool.

Outlook and conclusions

The ongoing development of the REM platform aims to provide an integrated solution to risk practitioners and Civil Protection authorities for the rapid assessment of exposure and vulnerability in complex urban environments. The platform is continually evolving following the efforts of the SIBYL consortium. Among the possible directions of development for the platform, which will also consider feedback from the end-users who are in contact with the project, the following extensions are already in progress:

1. Extending the REM taxonomy to different natural hazards. Currently a taxonomy extension to floods is available as an experimental feature.
2. Extension of the REM taxonomy to post-disaster surveying. This entails the agreement of the consortium as to the most suitable damage indicators to be considered.
3. Implementation of a background process for the estimation of statistical properties of the exposure modelling, resulting from the analysis of the screened buildings.
4. Implementation of an active-learning module which could be used to infer statistical properties of the resulting exposure models, and to guide the sampling, collection and analysis of further data.

The platform and the implemented tools provide a simple, yet sophisticated solution which can be easily adapted to the needs and constraints of civil protection users. A closer interaction between end-users and research community is sought in order to advance the currently developed tools and methodology into fully operational solutions.

Suggested reading

Abraham, T. and Roddick, F.J. (1999) Survey of Spatio-Temporal Databases, *GeoInformatica*, 3(1), 61–99.

Allen, J.F. (1981) An Interval-Based Representation of Temporal Knowledge, in *Proceedings of the International Joint Conference on Artifical Intelligence 1981*, 221–226.

Brzev, S., Scawthorn, C., Charleson, A. and Jaiswal, K. (2012) Interim Overview of GEM Building Taxonomy V2.0.

Koubarakis, M. (2003) *Spatio-Temporal Databases: The CHOROCHRONOS Approach*. Springer.

Langran, G. (1992) *Time in Geographic Information Systems*. Taylor & Francis.

Paredaens, J., Van den Bussche, J. and D. Van Gucht, D. (1994) Towards a theory of spatial database queries (extended abstract), in *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, New York, NY, USA, 1994, pp. 279–288.

Pelekis, N., Theodoulidis, B., Kopanakis, I. and Theodoridis, Y. (2004) Literature review of spatio-temporal database models, *Knowl. Eng. Rev.*, 19(3), 235–274.

Peuquet, D.J. (2001) Making Space for Time: Issues in Space-Time Data Representation, *GeoInformatica*, 5(1), 11–32.

Schneider, A. (2012) Monitoring land cover change in urban and peri-urban areas using dense time stacks of Landsat satellite data and a data mining approach, *Remote Sensing of Environment*, 124, 689–704.

Sellis, T.K. (1999) Research Issues in Spatio-temporal Database Systems, in *Proceedings of the 6th International Symposium on Advances in Spatial Databases*, London, UK, UK, 1999, pp. 5–11.

Sester, M., Sarjakoski, T., Harrie, L., Hampe, M., Koivula, T. Sarjakoski, T., Lehto, L., Birgit, E., Nivala, E.-M. and Stigmar, H. (2004) Real-time generalisation and multiple representation in the GiMoDig mobile service, IST-2000-30090, 2004.

Snodgrass, R.T. (1992) Temporal databases, in *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, A. U. Frank, I. Campari, and U. Formentini, Eds. Springer Berlin Heidelberg, 1992, pp. 22–64.

Stoter, J.E., Morales, J.M., Lemmens, R.L.G., Meijers, B.M., van Oosterom, P.J.M., Quak, C.W., Uitermark, H.T. and L. van den Brink, l. (2008) A Data Model for Multi-scale Topographical Data, in *Headway in Spatial Data Handling*, Ruas, A. and Gold, C. eds. Springer Berlin Heidelberg, 233–254.

Taubenböck, H., Esch, T., Wurm, M., Roth, A. and Dech, S. (2010) Object-based feature extraction using high spatial resolution satellite data of urban areas, *Journal of Spatial Science*, 55(1), 117–132.

Wieland, M., Pittore, M., Parolai, S. and Zschau, J. (2012) Exposure Estimation from Multi-Resolution Optical Satellite Imagery for Seismic Risk Assessment, *ISPRS International Journal of Geo-Information*, 1(3), 69–88.

Appendix A – REM – Taxonomy

In the following, the attributes implemented in the REM taxonomy, along with the respective values are provided. Also listed are the tables related to the hazards the attributes refer to (currently only earthquakes).

The basic taxonomy is based on the taxonomy developed within the GEM (Global Earthquake Model) project. The version 2.0 of the GEM taxonomy has been completed in August 2013, and superseded the previous version v1.0 (Basic Building Taxonomy) from March 2012. The taxonomy was developed by an international team chaired by Charles Scawthorn (USA) and Svetlana Brzev (Canada) with significant contributions from Andrew Charleson and Luke Allen (New Zealand), Marjorie Greene (USA), Kishor Jaiswal (USA) and Vitor Silva (Portugal). The taxonomy was developed in conjunction with other GEM researchers and builds on the knowledge base from the EERI and IAEE World Housing Encyclopedia and the USGS PAGER project. The original GEM Building Taxonomy is accompanied by an electronic Glossary that contains text and graphic illustrations which describe the attributes included in the taxonomy, and can be accessed for further information. These electronic resources can be accessed at the following web pages:

1. Online version of the GEM Building Taxonomy v2.0 in tabular form:

http://www.nexus.globalquakemodel.org/gem-building-taxonomy/overview

2. Online glossary (review individual terms and provide comments):

http://www.nexus.globalquakemodel.org/gem-building-taxonomy/overview/glossary

The taxonomy describes characteristics of an individual building or a class of buildings with similar characteristics, commonly referred to as a building typology, by means of the following 13 attributes: i) direction, ii) material of the lateral load-resisting system, iii) lateral load-resisting system, iv) height, v) date of construction or retrofit, vi) occupancy, vii) building position within a block, viii) shape of the building plan, ix) structural irregularity, x) exterior walls, xi) roof, xii) floor, and xiii) foundation. Each attribute describes a specific building characteristic that can potentially affect seismic performance of an individual building or a building typology.

The description of a single building structure is therefore defined by a single string, a combination of unique IDs for selected attribute values and delimiters (e.g. "/" and "+"). Each attribute has a specific position in the string, specified in the database schema.

Table 1 Table 'dic_attribute_type'

Table 2 Table 'dic_attribute_value'

Table 2 (cont.)

Table 3 Table 'dic_qualifier_type'

Table 4 Table 'dic_qulifier_value'

Table 5 Table 'taxonomy'

Table 6 Table 'dic_hazard'

Appendix B - Routing engine

The routing engine is based on the pgrouting[11] extension to PostgreSQL. The routing engine can be accessed either directly using SQL by calling the respective pgsql functions or by using the QGIS processing scripts. The QGIS processing scripts provide a simple user interface and easy access to the routing functionality, whereas the pgsql functions are meant for users who wish to build their own applications or are just more familiar with SQL. In the following, the pgsql functions implemented to carry out the optimized routing are described. In the subsequent section the QGIS plugin wrapping these functions is also described.

SQL / pgsql functions package

A set of custom functions for advanced routing operations have been implemented on top of the standard pgrouting extension. The functions include the following:

- pgr_dijkstra(varchar, integer, integer): wrapper for a simple Dijkstra function with geometry output.
- pgr_dijkstramulti(varchar, varchar): function to run Dijkstra iteratively on a sequence of nodes
- pgr_makecostmatrix(varchar, varchar, text): function to create a custom cost matrix (e.g. for TSP with street length)
- pgr_createnetwork(varchar): function to create a routable street network
- pgr_createroutestops(varchar, integer): function to create route stops from a set of sample points

Dependencies: PostgreSQL 9.x, PGRouting 2.0, PostGIS 2.0 or higher.

In the following, a typical workflow for a routing operation is exemplified with a set of simple SQL queries that make use of the above mentioned functions. The example assumes that a spatially enabled database exists with pgrouting extension activated and the above mentioned functions enabled. The database has a schema named “routing” were all the routing related input and output tables are stored. Moreover, as input, a street network (here “osm_streets”) and a table that holds the sampling points (here “samplepoints_sp001”) needs to be available in the database.

[11] http://pgrouting.org

```
-----------------------------------------------------------------------------------------------

--1. create a routable streetnetwork

--(note: streets table should at least contain the columns "gid" and "the_geom". Geometry should be cleaned beforehand (e.g. with GRASS v.clean))

-----------------------------------------------------------------------------------------------

SELECT * FROM pgr_createnetwork('routing.osm_streets');
```

This query creates a routable network that can be used with pgrouting from a table that holds streets of an area of interest. The streets can be for example imported to PostGIS from OpenStreetMap. The structure of the table is open to the user. The only mandatory columns are “gid” that holds a unique identifier for each row and “the_geom” that holds the geometry information. The geometry should be of type “polyline”.

The above query adds additional columns to the streets table and populates them. These columns are needed by pgrouting and include the following:

- *source, target*: Source and target columns that hold the id's of each source and target node for a street segment. These columns define the directivity of the network segments in case a directed network graph analysis is carried out.
- *length*: a default cost attribute that simply holds the length of each street segment in meters. Other useful cost attributes that can be defined separately by the user include, for example, travel time to pass through a street segment.

Note that typically when GIS files are loaded into the database for use with pgrouting, they do not have topology information associated with them. To create a useful topology, the data needs to be “noded”. This means that where two or more roads form an intersection a node should be placed and all the road segments need to be broken at the intersection. This assumes that one can navigate from any of these segments to any other segment via that intersection.

The *graph analysis functions* can be used to help see where there may be topology problems in the data. If there is a need to node the data, there is also a function available in pgrouting called *pgr_nodeNetwork()*. This function splits ALL crossing segments and nodes them. There are some cases where this might NOT be the right thing to do. For example, when there is an overpass and underpass intersection, these should not be noded, but pgr_nodeNetwork() does not know that is the case and will node them, which is not good because then the router will believe it is able to turn off the overpass onto the underpass like it was a flat 2D intersection.

```
-----------------------------------------------------------------------------------------------

--2. create route stops from sample points

--(note: a random subset of the sample points is used. sample points should have same SRID as streetnetwork)

-----------------------------------------------------------------------------------------------

SELECT * FROM pgr_createroutestops('routing.osm_streets_vertices_pgr', 'routing.samplepoints_sp001', 150);
```

This query creates a defined number of (unordered) route stops from a set of sampling points. First, the sampling points are filtered and mapped on to the street network to define route stops that represent nodes of the street network. The closest node of the street network is selected for each sampling point as route stop using a straight line distance from point-to-point. Only one route stop is selected in case multiple sampling points refer to the same network node. This effectively filters the sampling points based on their accessibility. The number of route stops can be defined by the user in case a sub-sampling of the sampling points is desired for the routing. In such case, a random selection of the sampling points is done before mapping the route stops.

```
-----------------------------------------------------------------------------------------------

--3. order route stops using TSP with custom cost attribute

--(note: use route stops id minus one to define start and stop point = index of points in cost matrix)

-----------------------------------------------------------------------------------------------

DROP TABLE IF EXISTS routing.route_stops_tsp;

SELECT seq, a.id+1 as id, b.node as id2, b.the_geom
    INTO routing.route_stops_tsp FROM pgr_tsp(
        (SELECT dmatrix from pgr_makecostmatrix('routing.route_stops',
        'routing.osm_streets', 'length'))::float8[], 0) a
    LEFT JOIN routing.route_stops b
    ON (a.id+1 = b.id);
```

The standard implementation of the TSP solver in pgrouting is based on ordering the points using straight line (euclidean) distance between nodes. This is a fast, but not exact solution that becomes increasingly inaccurate when the street layout is diverging from a regular dense network towards a sparse irregular network. Especially when there are rivers and bridges present, a TSP solution based on euclidean distance becomes inaccurate, and a more exact solution that uses the street network itself to calculate the cost factor is needed.

The above query orders a set of points using the TSP solver with a custom cost matrix, where the costs are calculated on the street network. The cost is defined in a column of the street table and is calculated for each row of the table. The cost is passed on to the function by defining the column

name of the cost attribute (here 'length'). The node and end node of the route can then be passed on to the function. If only a start node is defined, the end node will be chosen closest to the start node in order to force a loop ordering.

```
----------------------------------------------------------------------------------------------

--4. compute shortest path across all ordered stops

----------------------------------------------------------------------------------------------

SELECT * FROM pgr_dijkstramulti('routing.routestops_sp001_tsp', 'routing.osm_streets', 'length');
```

Once the route stops are ordered by the TSP solver, this query runs a Dijkstra algorithm multiple times between any ordered stop and its successor stop to calculate the shortest path across all the stops.

QGIS processing scripts

In QGIS, the *Processing Toolbox* is a geoprocessing environment that can be used to call native and third-party algorithms. The tools are subdivided in scripts and models.

The *scripts* are used to execute a single algorithm or run a batch process based on that algorithm. Each script is an ASCII file containing a *header* and *body*, where the header contains a set of special instructions to automatically generate the graphical user interface of the script itself.

The *models* refer to a graphical processing environment recently introduced in QGIS. This framework allows for combining different algorithms, possibly developed in different environments (R, GRASS, SAGA, OTB, etc.) in a single processing pipeline which is defined by visual blocks. Contrary to the scripts, the models are saved in a binary format and cannot be directly edited.

The routing engine and all the functions mentioned above have been transferred into scripts that can be loaded to the QGIS processing toolbar. The body of the script in this case is composed of R code that calls the according pgsql functions that have been described in the previous section. The R scripts in this case act as simple wrappers for the pgsql functions. The following scripts are available:

- GenerateRouteNetwork (see Figure 12).
- GenerateRouteStops.
- OrderRouteStops.
- RouteDijkstra.

Input parameters for the different scripts are the same as described above for the pgsql functions and can be entered via the user-interface along with the appropriate database connection. Documentation of the scripts and their parameters is given directly within the user interface in the form of a help menu.

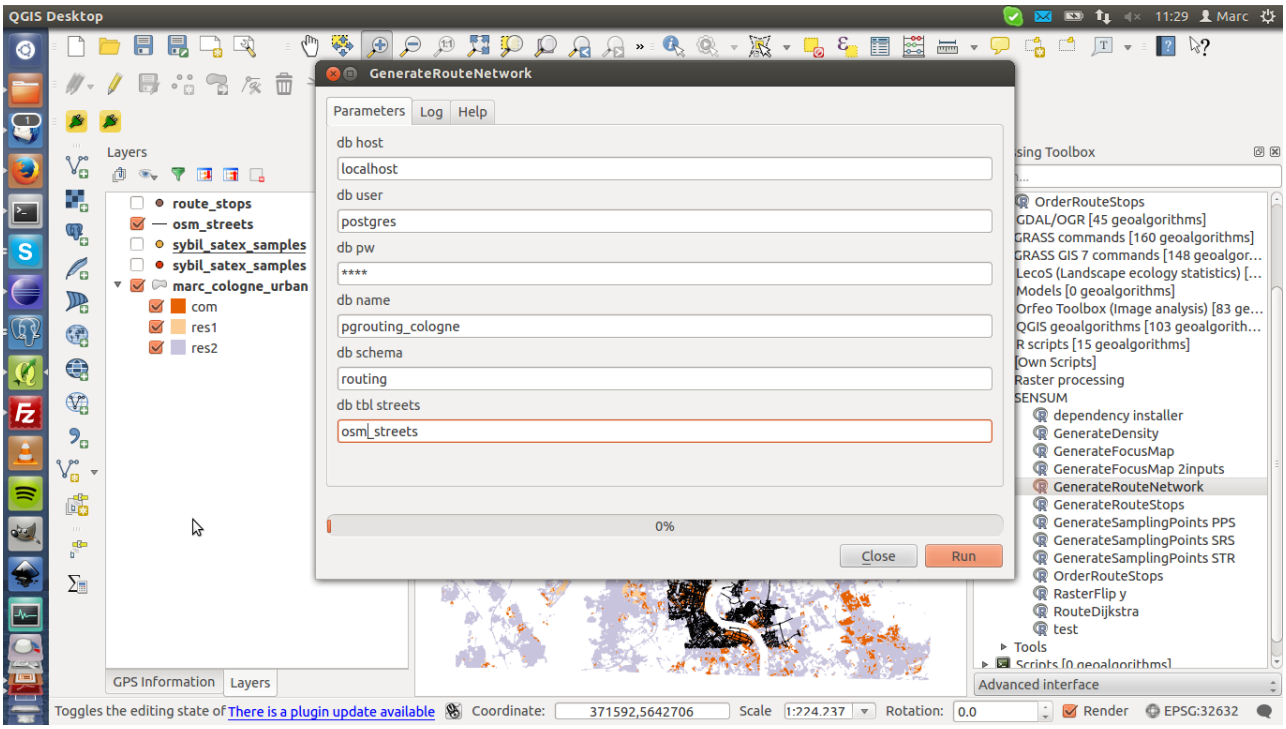

Figure 12 Example dialogue of the GenerateRouteNetwork scripts, that can be accessed via the QGIS processing toolbar (right).

Installation

The scripts and models can be downloaded from the github repository under

https://github.com/GFZ-Centre-for-Early-Warning/REM_optimized_routing

Linux: copy all the files from the folder “rscripts” to the folder “~/.qgis2/processing/rscripts”. The files with extension “.help” contain a short documentation about the individual scripts.

Windows: same procedure, with the related paths (always in the user´s Documents folder).

The processing scripts are available in QGIS under the Processing->Toolbox menu item, which provides a simple graphical interface. When starting QGIS, all available scripts and models are automatically loaded.

In order to run the scripts, a spatially enabled database with pgrouting enabled has to be present in the system. Please note that R scripts have to be explicitly activated in the Processing Toolbox Settings of QGIS.