?isySUR

it's easy!

Manual

Published January 15, 2015

Contents

1 Introduction to isySUR 5

1.1 Quick start 5
1.1.1 isySUR in command line version 5
1.1.2 isySUR in window version 5

1.2 What is isySUR? 6

1.3 Content of isySUR 6

2 Using isySUR 9

2.1 Requirements and installation 9
2.1.1 Requirements 9
2.1.2 Installation 10

2.2 Input and output 10
2.2.1 Space usage rules 10
2.2.2 Configuration file 11
2.2.3 Keyhole Markup Language 11

2.3 Command line and GUI version 11
2.3.1 Command line version 11
Parameters 12
2.3.2 GUI version 12
Parameters 12
The menu 12
KML list 13
2.3.3 Using android 13

3 Technical and implementation details 15

3.1 Development 15
3.1.1 Data usage 15

3.1.2 Data structure 16
3.2 Algorithm **17**
3.3 Evaluation **18**
3.4 Further Work **20**

4 Added SURs 21

Quick start
isySUR in command line version
isySUR in window version
What is isySUR?
Content of isySUR

1. Introduction to isySUR

Welcome to isySUR (pronounce isy as ['i:zi]), a tool helping you to find the area of application of a space usage rule (SUR). For starting the command line version please see section 1.1.1. For using the window version please read section 1.1.2. You can also run the window version on your android device. This is described in section 2.3.3. For more details about the program see section 1.2.

1.1 Quick start

First decide if you like to use the command line version of isySUR or if you prefer a graphical user interface. Please proceed with section 1.1.1 or section 1.1.2 depending on your choice.

For installation instructions see section 2.1. For additional parameters see section 2.3.1.

For information about input and output file types see section 2.2.

1.1.1 isySUR in command line version

For the command line version type

```
$ python run_isySUR.py cli [-h] [-c CONFIG] input output
```

where `input` is the path to the input file containing the SURs which areas of influence should be computed and `output` is the path for the resulting area(s) of influence. If the path points to a file, only one KML file will be created, containing all calculated areas. If the path points to a directory a KML file for each SUR will be created as well as one containing all areas.

1.1.2 isySUR in window version

For the window version just type

```
$ python run_isySUR.py gui [-h] [-c CONFIG]
```

In Menu you find the possibility to load a SUR input file. After calculating you can save the area(s) of application.

Figure 1.1: isySUR in GUI version

1.2 What is isySUR?

isySUR is a program that tries to calculate the intuitive area of application of a space usage rule by its coordinates and data from OpenStreetMap[1]. It was developed as part of the informatiCup, a programming competition for students organized by the German Informatics Society (Gesellschaft für Informatik e.V.). Our team is called ISY and consists of Adriana-Victoria Dreyer, Jacqueline Hemminghaus, Jan Pöppel and Thorsten Schodde, four students of the master study course 'Intelligent Systems' at Bielefeld University.

Using isySUR is very easy. You give an input file with the coordinates of the SURs and the intended area of application is computed. Optionally, you can add a configuration file that classifies SURs in those that are applicable only indoor, those that are applicable only outdoor or those that are applicable indoor as well as outdoor (both) to get a better result. In the GUI version the calculated areas are shown on a map and you can choose which ones you want to be displayed. They can be saved into a file to reload them whenever you want to have a look at them.

1.3 Content of isySUR

In the archive you can find the following data (bold printed files will be installed with the install command):

- data (directory) - examples for space usage rules[2]
- doc (directory) - documentation directory
  - manual.pdf - this manual

    add pydoc
- **isySUR** (directory) - the source files for the isySUR Python package
  - **gui** (directory) - source files for GUI version

[1] `http://www.openstreetmap.org`
[2] see section 4

  - tests (directory) - isySUR was developed using test driven development. In this directory you will find the test files.
- testData (directory) - data for tests and test data sets
  - dataOnlyForTests (directory) - Some of the test files need data that is stored in this directory
  - Data.txt - a possible input file
- README.txt
- **run_isySUR.py** - the Python script to run isySUR
- run_tests.py - script that runs all or some tests in isySUR/tests/, used for development
- setup.py - the setup script to install isySUR
- surConfig.cfg - a configuration file that can optionally be used for computation[3]

[3] see section 2.3

Requirements and installation
Requirements
Installation
Input and output
Space usage rules
Configuration file
Keyhole Markup Language
Command line and GUI version
Command line version
GUI version
Using android

2. Using isySUR

In this chapter the usage of isySUR is described in detail. First you can find installation instructions in section 2.1, detailed usage instruction for both program versions are following in section 2.3.

2.1 Requirements and installation

isySUR is written in Python so you get a Python script, which starts the main program, and a package with the tools. For running the command line version you do not need much more than basic Python. If you prefer to use a graphical user interface you will have to install some more packages.

2.1.1 Requirements

1. **basic requirements:**
   - Python 2.7
   - requests (HTTP library)
   - internet connection

   To run a Python script you need Python. Python comes with most Linux distributions. For the use on Windows machines you will have to install Python manually[1]. isySUR uses data from OpenStreetMap[2] to calculate the area of application. Therefore an internet connection is required. The data transfer in Python is realised with the requests[3] library. You can easily install it using pip[4], a Python installation tool

   ```
   $ pip install requests
   ```

   or see requests web page[3].

[1] get Python here: `https://www.python.org`
[2] `http://www.openstreetmap.org`
[3] `http://docs.python-requests.org`
[4] `https://pip.pypa.io`

2. **additional requirements for the GUI version:**
   - Kivy
   - concurrent.futures

   For the graphical user interface Kivy is used. Kivy[5] is a cross-platform Python Framework for NUI Development. For installation on Ubuntu we recommend the installation with apt-get as described on their web page. For use on Windows and starting Kivy applications from the command line we recommend to install Kivy and pygame (needed for Kivy) with the unofficial precompiled binaries[6][7]. On MacOSX the easiest way to run Kivy applications, like isySUR, is with the Kivy launcher as mentioned on the Kivy web site[8]. On all OS futures can be installed via pip

   ```
   $ pip install futures
   ```

2.1.2 Installation

You have got an archive of type .tar.gz. Please unpack the archive into a directory of your choosing. Because isySUR is a Python script you do not need to install it. It is also possible to install only the dependencies and run isySUR by typing from within the directory you chose before:

```
$ python run_isySUR.py [parameters]
```

For installing the script and tool package browse the directory you chose and install isySUR with the help of Python's distutils:

```
$ python setup.py install
```

Now you are able to start isySUR any time and anywhere by typing:

```
$ run_isySUR.py [parameters]
```

2.2 Input and output

The input of the space usage rules was specified in the programming task as well as the output in .kml format. For more information about the decision to add a configuration file please see section 3.1.1.

2.2.1 Space usage rules

The SUR file is a plain text file. The first line holds the number of lines in the file, that will be used for constructing SURs, which are to be computed. The following lines represent the rules. A line must always start with the SUR id, followed by the latitude and longitude coordinates of the rule. The last entry is a brief description of the rule. The entries are comma separated. Decimal separator in latitude and longitude is a point. One SUR id can span several consecutive lines in order to add several rules to the same SUR.

[5] http://kivy.org
[6] Kivy: http://www.lfd.uci.edu/%7Egohlke/pythonlibs/#kivy
[7] pygame: http://www.lfd.uci.edu/%7Egohlke/pythonlibs/#pygame
[8] http://kivy.org/docs/installation/installation-macosx.html

Example file:

```
3
0061, 50.9304, 5.34470, smoking="no"
0061, 50.9304, 5.34470, access:dog="no"
0062, 50.9306, 5.34366, smoking="no"
```

2.2.2 Configuration file

The configuration file is a plain text file. In this file one or more blocks starting with '[Indoor]', '[Outdoor]' or '[Both]' are found. After these headlines the names of the rules follow that belong to these categories of area of influence.

Example file:

```
[Indoor]
access:age="21+"
camera="no"
[Outdoor]
fishing="no"
littering="no"
[Both]
access:dog="no"
open_fire="no"
```

These configuration files can be used to adapt the area calculation to better suite local space usage rule customs. For more details about how these categories influence the calculation see section 3.1.1.

2.2.3 Keyhole Markup Language

Output files are in .kml format where KML stands for Keyhole Markup Language. For syntax definition and documentation please visit `https://developers.google.com/kml`. Kml version 2.1 is used for the produced output files.

The GUI version provides also the possibility to read KML files to display the areas. For this KML version 2.1 as well as 2.2 is supported.

2.3 Command line and GUI version

isySUR comes with two versions: One for command line use and another one for users who prefer to use a graphical user interface. For using the command line interface type:

```
$ run_isySUR.py cli [parameters]
```

For using the graphical user interface type:

```
$ run_isySUR.py gui [parameters]
```

Notice that the GUI version has more requirements than the command line version. See 2.1.1 for more details.

2.3.1 Command line version

Usage of the command line version:

```
$ run_isySUR.py cli [-h] [-c CONFIG] input output
```

Parameters

The parameters of `run_isySUR.py cli` (bold arguments are required):

- **input** - Path to the input file containing the SURs. See 2.2 for the required format.
- **output** - Path where to put the output file(s). If this path points to a file, a single output .kml-file containing all computed areas is created. If it points to a directory, one .kml-file for each SUR is created in this directory as well as one .kml-file containing all computed areas.
- -c CONFIG (–config CONFIG) - With this parameter an optional config file with SUR classifications (indoor, outdoor, both) is used for computation. For the format of the file see 2.2.
- -h - If this parameter is given, a short help message containing all parameters is shown.

2.3.2 GUI version

add information with config - first safe, afterwards it will be applied

Usage of the GUI version:

```
$ run_isySUR.py gui [-h] [-c CONFIG]
```

If you do not want to give parameters you can also start the GUI version with

```
$ run_isySUR.py
```

or a simple double click on the file (when .py-files are connected to python). When the GUI cannot be loaded, the command line version is started.

When the GUI version could not be started, e.g. because of missing requirements, you can proceed with the command line version. You will be asked for an input file and an output path, to run the program. If you do not want to use this version you can type `'exit'` to exit the program.

Parameters

The parameters of `run_isySUR.py gui`:

- -c CONFIG (–config CONFIG) - With this parameter an optional config file with SUR classifications (indoor, outdoor, both) is used for computation. For the format of the file see 2.2.
- -h - If this parameter is given, a short help message containing all parameters is shown.

The menu

On the left side you can find the menu that holds the functionality you can use. In the menu you will find the following possibilities:

- **Load SUR** - Choose the file with the coordinates of your SURs. See section 2.2 for more information about the format. The area of application of the SURs will be calculated. An information text will inform you about the ID of the current SUR. When the KML list is open you can see that areas are added. Calculated areas are displayed on the map as well as red markers at the SUR's coordinates. After a calculation is finished you can start another one. The new areas will be added to the ones already in the KML list.
- **Load KML** - Choose some KML file you have already saved during the last usage. Or maybe you made your own KML file with another program. See section 2.2 for more information about the format. The areas contained in your file are displayed on the map and an entry for this file is added to those already in the KML list.
- **Save KML** - All KMLs that are currently selected in the KML list are saved to the file or folder that you can choose here.

Figure 2.1: GUI with open menu and KML list

- **SUR Position** - The red SUR markers can be hidden and shown with this button.
- **Config** - With this button you can open the configuration window. If you used the `-config` parameter there is already a configuration file loaded. You can load another configuration file, add or remove some rules and save your personal configuration file. With your configuration you say if you prefer to find an area for a SUR indoor or outdoor or if you think that kind of rule can be applicable indoor as well as outdoor. The following computations will use the new configuration.

KML list

On the right side of the window, there is the KML list. This list tells you which areas you have calculated or loaded. You can scroll the list if it is too long for the window with the mouse wheel or by dragging it with the mouse. When you click on an entry which area is not in view the map will jump to that area. When you click on an entry which area is already in view you can hide it. Hidden areas are not saved into a KML file. With another click you can make an area visible again.

2.3.3 Using android

Of course calculating SURs at home is nice but in general you would like to know the area of application a space usage rule standing in front of it. This is possible on your android device (with internet connection). In the archive you can find the isySUR.apk that you can install on your device. Just copy it on the device e.g. via USB and enable the device option to install from unknown sources. Browsing to the .apk and clicking on it starts the installation. After that you will find isySUR in your apps and can use it exactly like the GUI version described in section 2.3.2.

name of file corre

Development
Data usage
Data structure
Algorithm
Evaluation
Further Work

3. Technical and implementation details

In this chapter you will gain an insight into the development and learn some implementation details. The implemented algorithm to calculate the intended areas of application is explained in section 3.2. After that an evaluation of the achieved results follows in section 3.3.

Code documentation -> pydoc?

3.1 Development

For this project we decided to use Python. First of all, we all wanted to challenge and expand upon our Python knowledge. Furthermore Python is mostly OS independent and can even be ported to android for portable usage which seems to be a good feature here.

This project was created using test driven development on the unittest layer. This does not only allow us to keep track of unwanted side effects of new functionality, but also helps assigning work load to the different members, because it can be easier to write a failing unittest with a bit of information, that is assigned to someone, than it is to explain all the requirements for some new functionality.

isySUR was created in two steps: First the command line version was created and the first versions of the algorithm were implemented. While the algorithm was improved and reworked, it was decided to add a GUI because a lightweight tool for displaying the calculated areas was missing. Furthermore, it helped in improving the algorithm by enabling rapid testing of changes. On top of that, not all people like to use a command line tool.

For the GUI the Kivy framework is used. We base the display of the OSM map on mapview from the Kivy Garden[1] which we adapted to suit our needs.

At the beginning of the project the idea of an app came up and with the Kivy framework for the GUI the foundations for this was laid. With buildozer[2] a nice tool for packaging an apk file from our project was found.

3.1.1 Data usage

The competition rules state that a SUR file (see section 2.2 for details about the format) as well as photos of the real signs belonging to the space usage rules are to be the only inputs to the

[1] see `https://github.com/kivy-garden/garden.mapview`
[2] see `https://github.com/kivy/buildozer`

program. The only other allowed source of information is OpenStreetMap.

We deliberately do not use the images in our analysis. This is because they do not give us reliable information about the interesting polygons. The variety of possible signs, including totally unknown ones, (and their placement in/on/around the intended area) makes it infeasible to perform robust image recognition or classification. Furthermore the only relevant information we think one could extract from images, refers to an inside/outside classification of the rule. However, even this classification would not be reliable and robust enough in the general use case.

One might consider using images and compute vision techniques in the future, when the Google StreetView recordings cover virtually everything and these images are available to this program as well, but until then we do not believe we can extract useful information from the images.

We do allow the usage of the rule types/names (e.g. access:21="no") to some minimal extend. We allow configuration files (see 2.2.2 for details about the format) that classify the rules to be applicable indoor, outdoor or indoor as well as outdoor (both). We use these classifications to narrow down the possible OSM elements around the target coordinates. Unknown rules will always get the "both" classification so that we do not ignore relevant information in those cases. We hope to improve our results, by complying with local space usage rule customs, and reduce computational costs by doing so. However, in a general context most rules are applicable indoor as well as outdoors, which means that these classifications will most likely only be useful in individual cases.

Although the organizers of the competition explicitly stated, that we should not only use existing OSM elements for the computation of the space usage rules' intended areas, we mainly focus on existing OSM elements. We do so because we believe that humans naturally split up space into areas with different meanings which often correlate to the intuitive areas of application of space usage rules. The OSM community thrives to include as much meaningful information as possible and is constantly adding new information. We therefore believe that, considering the uncertainty and scarcity of the available positional data, selecting reasonable existing OSM elements (which are constantly being improved) is a good approximation. Furthermore, any computation of new polygons in areas where there is no OSM data available would result in arbitrary polygons that may or may not be intuitive to the human. We also do not believe in merging multiple existing OSM elements that already mark meaningful areas on their own, since we do not have enough information to reliably distinguish all the different scenarios where one could merge, for example buildings.

In order to reduce the amount of data that we need to parse and search for reasonable elements, we filter out a lot of tags that we do not want to get at all: Since we are looking or areas or polygons, we do not request any ways or relations that have tags containing route*, highway, associatedStreet or TMC, since these kind of elements do not provide us with polygon information. We furthermore do not request any boundary tags, since we do not believe that we should mark entire political or administrative areas as the intended areas of SURs. A last tag that we are ignoring, is building:part. According to the OSM wiki, any building part that is tagged with building:part should be contained in another element, that covers the entire building, which we are more interested in.

clear that this is card

3.1.2 Data structure

In order to be able to work freely with all the data that we require when computing suitable areas for SURs we have introduced our own data structure.

- sur: A wrapper object for the SURs. It enables us to handle the file only once at the beginning.
- osmData: Wrapper for the OSM elements Node, Way and Relation. We convert the xml

data we receive from OSM to these objects, so that we do not have to work on the xml for computation.

- kmlData: Wrapper objects for a KML and included placemarks. These will be created by our algorithm and can be saved to a .kml file.

Furthermore, our data structure splits our program into nice modular chunks that can be reused independently in other project if desired.

3.2 Algorithm

pseudocode

rework this section

The actual algorithm is a combination of query selection and finding appropriate OSM elements within the returned OSM data. The OSM data is requested in an adaptive bounding box around the coordinates. This bounding box starts with a size of roughly 60m and increases until at least one way element was returned. We already filter out any unwanted elements as mentioned in section 3.1.1, which allows us to use such a relatively big bounding box without running out of memory when parsing the xml. If the rule classification of the current SUR, given by the optional configuration file, labels this space usage rule to be applicable only indoor, we also explicitly request buildings.

After we have received and parsed the OSM data around the given SUR coordinates, we search for the closest elements. If we know that the SUR is only applicable indoor, we only search for relations and ways containing the building tag. Otherwise we search for any relation and way. Since relations are only usable to us, if they contain some sort of polygon information, and the distance of a relations is equal to the distance of it's closest member, we discard any relations that are further away than the closest ways we find. In order to reduce the risk of finding wrong polygons in areas where there are no detailed OSM elements, we search for ways containing the tag landuse, if the SUR coordinates are not inside the closest way and the closest way is further away than an experimentally set threshold of roughly 16m. We use the closest landuse instead of the previously found ways if the SUR coordinates are inside the landuse. This allows us, for example, to find the residential area for SURs whose intended area is a building that is not present in the OSM data.

Since multiple elements can have the same distance, the list of closest objects is further analysed in order to find the polygon we intend to use.

If we have found relations we search it for way members that are labelled as outer because we are interested in the outline of a polygon. These way members are added to a list of possible solutions.

Any ways we found will also be added to the list of possible solutions unless they are tagged as landuse. Landuse ways are analysed further: If the way should be used to mark a residential area, we search for buildings within the outline of the landuse. This is done because most buildings are inside the landuse instead of on the border, which prevented us from finding them when searching for the closest ways before. If the closest buildings from within the landuse are closer than the experimentally determined distance of roughly 8m we use the ways marking these buildings instead of the landuse. If we do not find a suitable building we keep the landuse but store that we are uncertain about the quality of such an result, which will be visualized by using a different colour (blue) for the polygon in the kml file. This uncertain colour is also used for other landuse ways such as industrial and commercial, since we do not think that the SUR's area of application is intended to cover such a big area.

If we still have multiple possible ways after filtering through the list of closest elements, we continue to filter out ways that seem less likely. First we check if one of the possible ways marks

a building or shop. If this is the case and the rule classification indicates that the SUR can be applicable indoor, we filter out anything that is not a building because in these cases we assume that it is more likely that the intended area for the SUR covers a building.

From the remaining possible ways we use a final discriminating feature that favours smaller areas over larger ones. The decision for this was made based on the observation that in a lot of cases where ways were equally likely up to this point, the smaller one appears to be intuitively more likely. Instead of actually calculating the area of an arbitrarily formed polygon we use the average distance of all the polygon vertices to the SUR coordinates as the discriminating feature.

The last remaining way and its reference nodes are finally used to construct a placemark together with the appropriate style (“certain”, “uncertain”) which is added to a kml and returned.

3.3 Evaluation

Figure 3.1: Left: found area for example SUR 0038; Right: given truth file of SUR 0038

The quality of our results greatly depends on how detailed the OSM data is for the requested area. As mentioned in the sections 3.1.1 and 3.2 we mainly focus on the coordinates of a given SUR to compute its intended area of application. Therefore, our results are good for areas where the OSM data is detailed and less optimal when the OSM data is sparse.

A good example for this is the number 0038 of the provided test dataset, that is shown in figure 3.1: The intended area is a building which is however not mapped in OpenStreetMap yet.

Figure 3.2: Left: found area for example SUR 0023; Right: given truth file of SUR 0023

Since there is no information about the intended polygon area in the OSM data, we cannot build the polygon. In this case we find the entire residential block and mark it with a different colour to show our uncertainty.

Another example for insufficient information is that the received data does not include parts of a buildings. This is the reason we only find the whole building and not the intended part of the truth files.

Once the OSM data gets updated further we would identify the correct area as well.

In other cases (e.g. figure 3.2) the provided coordinate positions would indicate completely different areas than the ones that were intended. When just judging by the OSM data and the coordinates, we would usually rate our results to be more accurate then the provided truth areas in such cases. We therefore would not usually rate these results poorly.

In a few cases (e.g. figure 3.3, the test data intends a combination of multiple connected buildings where we apply the rule only to one building. Since in these cases, the buildings are different, we do not think it is justified to extend the area of application to the other buildings as well.

Figure 3.3: Left: found area for example SUR 0016; Right: given truth file of SUR 0016

Of course there are some cases, where we compute areas, that we do not deem to be intuitive areas of application of the given rules. One example would be the number 0056 from the test dataset, that is shown in figure 3.4. The intended area is not present in the osm data and the

Figure 3.4: Left: found area for example SUR 0056; Right: given truth file of SUR 0056

residential area spans such a big area (covering several blocks) that we find a different building on the other side of the road. A more detailed analysis of our actual results when comparing with our SURs and the ones provided by the test data can be found below:

Daten einfügen

give tabular a caption

say sth about missing data

	calculated=truth	calculated intuitively correct	calculation uncertain	calculation unintuitive
given SURs	44,16%	21,12%	13,44%	21,28%
own SURs	75%	0%	25%	0%

Table 3.1: test

Overall we are quite pleased with our results in most cases. Although the organizers of the competition did not want us to find suitable OSM elements that would represent the intended areas, we found that most space usage rules are intended to be applicable to certain meaningful areas, such as buildings, parks, forests, courtyards etc. Such meaningful areas are often explicitly included and labelled in OSM and the community intends to improve the level of detail even further in the future. It is for that reason, that we believe our approach of finding suitable OSM elements around the given coordinates is justified and already provides useful results.

3.4 Further Work

restructure - this looks ugly

Our major focus of future work will be the development of the GUI. Most of all we like to implement a possibility to list up all placemarks of one KML similar to Google Earth, so that the user can jump to every placemark on the map he or she likes and not only to the last one of each KML. Furthermore the user should be able to display the rules that are applied in any polygon when clicking on it. Another improvement of the GUI should be the opportunity to load several KMLs at once, because currently the user can only load one KML file at a time.

Try using the actual area or the area of the bounding box of a polygon instead of the average vertex distance.

For a better usability on mobile devices it should be implemented that a click anywhere on the map adds a new SUR and the program calculates the corresponding area.

To improve the calculation of our KMLs we think about using multiple data sources because OpenStreetMap sometimes has a lack of data information.

In order to improve isySUR and make it available for others, we are going to make our repository public after the competition.

4. Added SURs

For the competition at least ten new space usage rules should be found. Here our 20 found SURs are presented.

isy0001.jpg:
smoking="no"

intuitive area of
isy0001.jpg

isy0002.jpg:
smoking="no",
access:dog="no"

intuitive area of
isy0002.jpg

---

isy0003.jpg:
access:dog="no"

intuitive area of
isy0003.jpg

isy0004.jpg:
smoking="no"

intuitive area of
isy0004.jpg

isy0005.jpg: smoking="no"

intuitive area of isy0005.jpg

isy0006.jpg: smoking="no"

intuitive area of isy0006.jpg

isy0007.jpg: smoking="no"

intuitive area of isy0007.jpg

isy0008.jpg: smoking="no"

intuitive area of isy0008.jpg

isy0009.jpg: smoking="no", access:bicycle="no"

intuitive area of isy0009.jpg

isy0010.jpg: access:bicycle="no"

intuitive area of isy0010.jpg

isy0011.jpg: access="no"

intuitive area of isy0011.jpg

isy0012.jpg: access:pram="no"

intuitive area of isy0012.jpg

isy0013.jpg: smoking="no"

intuitive area of isy0013.jpg

isy0014.jpg: smoking="no", noise="no"

intuitive area of isy0014.jpg

isy0015.jpg: smoking="no", access:dog="no"

intuitive area of isy0015.jpg

isy0016.jpg: access:fire_case="no"

intuitive area of isy0016.jpg

isy0017.jpg: dog_waste="no"

intuitive area of isy0017.jpg

isy0018.jpg: smoking="no"

intuitive area of isy0018.jpg

isy0019.jpg: access:bicycle="no"

intuitive area of isy0019.jpg

isy0020.jpg: access:dog="no", access:skate="no"

intuitive area of isy0020.jpg

Todo list

add pydoc 6
add information with config - first safe, afterwards it will be applied 12
name of file correct? 13
Code documentation -> pydoc? 15
make clear that this is a wildcard 16
pseudocode 17
rework this section 17
Another example for insufficient information is that the received data does not include parts of a buildings. This is the reason we only find the whole building and not the intended part of the truth files. 19
Daten einfügen 20
give tabular a caption 20
say sth about missing data 20
restructure - this looks ugly 20
Try using the actual area or the area of the bounding box of a polygon instead of the average vertex distance. 20