Construction sites SonVis Algorithm documentation

Svoronos - Kanavas Iason

June 27, 2022

Contents

1 Algorithm architecture

The overall algorithm is structured as:

1. Interface
2. Sub-processes
3. data processing algorithm
   (a) outlier identification,
   (b) replacement,
   (c) data-frame creation and manipulation,
   (d) min-max extraction script
   (e) datetime re-sample function
4. on-run functions (includes matrix printing functions)
5. IPC inter-process communication (& connection setup functions)
6. Sonification algorithm
   (a) processing functions
   (b) data receiver – handler
   (c) mappings
   (d) map to scale frequency mapping patch
      i. Map to Lydian scale
      ii. Map to Beat
   (e) event receiver - actions
   (f) synthesisers

2 Dependencies

SuperCollider (sclang has to be in $PATH)

Python & Python packages:

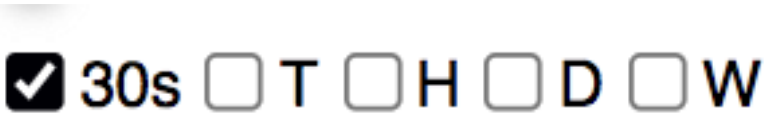
from future import $\text{print}_{\text{function}}$ import datetime as dt import panel as pn import param # for FormatDateRangeSlider import subprocess # to run SC import threading # stop iteration cycle when kill button pressed #from pythonosc import $\text{udp}_{\text{client}}$ from bokeh.models import Slider, CheckboxGroup, CustomJS, Label import os import numpy as np import pandas as pd import time import datetime from dateutil import parser from pythonosc import $\text{udp}_{\text{client}}$ import subprocess import numpy as np import pandas as pd import matplotlib.pyplot as plt import random import sys from bokeh.plotting import figure from pythonosc.$\text{osc}_{\text{server}}$ import AsyncIOOSCUDPServer from pythonosc.dispatcher import Dispatcher import asyncio

3 Interface

The interface consists of certain visual elements. The datetime range slider and text input (for the time) are used for date and time accordingly. Using these the user can specify a desired datetime period in the data to sonify and visualise.

The **start** button is used so that the datetime period is extracted and then sent to be sonified and visualised, while the **killall** button can interrupt/stop every running procedure in relation to the sonification and visualisation process anytime.

Below, there is a checkbox where the user can select the period of the values in the data to re-sample.



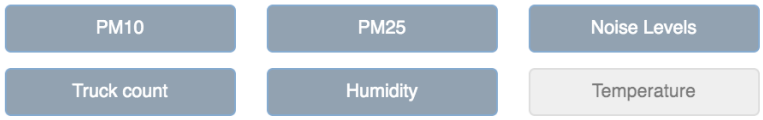
The original collection period is (every) 30 seconds. The re-sample options are, per minute (T), per 30 minutes (30M), per hour (H), per week (W), per month (M). In the re-sample function (see below), the data values are derived in every period with selecting the max value when doing the frequency conversion to highlight peaks in the data. For example, if we want to re-sample with 1 minute re-sample frequency (T). We have initially:

timestamp,	data
2021-08-01 00:00:00,	2
2021-08-01 00:00:30,	0
2021-08-01 00:01:00,	4

Re-sample result: The extracted value for the first minute will be "4" because is the max value observed for this minute.

Then, there is a another slider that controls the data iteration frequency and it has a range from 1 to 1000 values per second.

Finally, there are six buttons where the user can use to turn on/off, the desired data parameters to sonify - visualise



Overall, the button python bokeh elements, trigger osc messages that are sent from python to supercollider in order to control different synths. The ones that utilise this functionality is the **start killall** and the synth on/off buttons (pm10, pm25, temp, humidity, noise levels, truck count). This will be elaborated in the IPC section

4 Sub-processes

On launch, sclang is initialised and runs as a sub-process within the python session. More specifically, the SuperCollider patch for sonification is evaluated using the following command in Python.

```
# run sonification patch
sclang = subprocess.Popen(
    'sclang particleSonification.scd', shell=True,
    stdout=subprocess.PIPE,
    stderr=subprocess.STDOUT)
```

Getting back now to the initialisation python script where a function obtains the IP address of the computer using a shell command and then stores it as a global variable. After that, the OSC client configuration setup uses the variable's value ($\text{udp}_{\text{client}}$ object). The function is defined the following way as well as the OSC setup. This process easily configures OSC intercommunication between python and SuperCollider therefore mistakes and hassle by hard-coding IP addresses or manual configurations are avoided.

```
# get IP address
def getip():
    global ip
    ip = subprocess.Popen(
        'ipconfig getifaddr en0', shell=True,
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT)
    ip, _ = ip.communicate()
```

```
    ip = ip.decode('utf-8')
    ip = ip.strip()
    print(ip)

# Python osc
getip() # run getip function
client = udp_client.SimpleUDPClient(ip, 57120)
```

Note: *this works* ***only for macOs****. Therefore it has to be adjusted for linux or windows.*

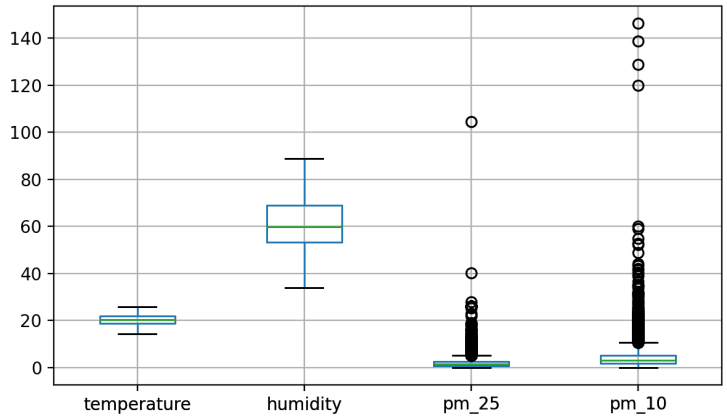
WIN hint:

```
ipconfig | grep IPv4 Address.
```

5 data processing

In this section the data processing will be described. The algorithm is developed in Python. The idea is based on combining and re-constructing the data-sets after the processing results that come out from the derived stats (IQR). SC has also access to the derived data-set (it is written to disk) so that it has access to the min max values for the correct mapping (see 5.4). In this way, it is also possible to re-use the algorithm with different data since the mapping is not hard-coded.

Outlier identification and replacement was deemed necessary since it was observed by using box-plot stats the PM (both 25 and 10) showed extreme values (far from accurate measurements (140~ PM10)) that we would like to exclude.



Code process:

The very first step is that the original data are loaded from the CSV file while the timestamp column is stored in a variable. Then the timestamp column is removed from the data-set to do the processing and then added again in the very end of the procedure.

5.1 outlier identification

Descriptive statistics are applied in the data-set using the 'describe()' method from pandas. That is to calculate percentiles, max, min and mean of every column in the data-set. Then the Q1 and Q3 of PM10 and PM25 are stored in variables. The IQR of both is calculated as well as the max and min threshold. The threshold will be used to identify the outliers. Values that exceed the min and max threshold are the outliers.

```
# calculate IQRange for pm_25 from q1 and q3
iqr_pm25 = pm25_q3-pm25_q1
iqr_pm10 = pm10_q3-pm10_q1

# calculate thresholds from IQR -- acc. skewed distribution
# max_thresh: Q3+1.5IQR
# min_thresh: Q1-1.5IQR
max_thresh_pm_25 = pm25_q3+(1.5*iqr_pm25)
min_thresh_pm_25 = pm25_q1-(1.5*iqr_pm25)
max_thresh_pm_10 = pm10_q3+(1.5*iqr_pm10)
min_thresh_pm_10 = pm10_q1-(1.5*iqr_pm10)
```

```
thresholds = {'min thresh_pm_25': min_thresh_pm_25,
         'max thresh_pm_25': max_thresh_pm_25,
         'min thresh_pm_10': min_thresh_pm_10,
         'max thresh_pm_10': max_thresh_pm_10}
```

5.2 replacement

Values for PM10 and PM25 that exceeded min and max threshold derived from the IQR calculation will be NaN-ed and then replaced with randomly selected samples from the same column in the data-set. This outlier replacement process takes place for PM10, PM25 and noise levels. The replacement function also prints how many values were replaced.

```
def replaceOutliers(col,minimum_thres,maximum_thres):
    for i in [col]: # replace outliers with nan value
        min = minimum_thres
        max = maximum_thres
        df.loc[df[i] < min, i] = np.nan  # if value is < min_thresh_pm25: nan it
        df.loc[df[i] > max, i] = np.nan  # if value is > max_thresh_pm25: nan it
        df.loc[df[i] == 0, i] = 0.1  # if zero: replace it with 0.1 (smallest val)
        print( # print how many null values are in the specified column
            'sum of null replaced values',
            df[col].isnull().sum())
        global des_col
        des_col = [col] # specify column

df = df.apply( # replace NaN values from random samples same column
    lambda x: np.where(x.isnull(), x.dropna().sample(len(x), replace=True), x))
```

5.3 data-frame creation and manipulation

As mentioned the df is first loaded from the CSV file, while the timestamp column is removed and stored in a variable. This was done to easily process the data-set without interfering with the datetime object (timestamp column). After that the 5.1 takes place. That results to a new data-frame and then the timestamp column is added (insert method).

```
# insert timestamp column
df.insert(0, "timestamp", timestamp, True)
```

Afterwards, the noise level data are loaded and stored in a variable. The last (cat_{24}) column was used. This column is added to the data-frame that

contains everything. While in the next step the replaceOutliers function is applied to the noise levels column as well ('db'). The threshold that was used aimed to exclude one outlier (5.444976) that was identified by rendering a boxplot from the column data.

Then the $\text{truck}_{\text{count}}$ data are inserted to the main data-frame after the appropriate data processing that is related to the ';' delimiter character splitting. This was done using the pandas data-frame loading process.

```
trucks_df = pd.read_csv(  # read truck data file
    "./fake_passage_time.csv",
    delimiter=';')
```

After that the main data-frame is written to disk within a certain directory path in the current working directory environment. That would be the `./df_out` directory. Later the data-frame is registered to a global variable for easier access.

5.4 min-max extraction script

The min-max python script was used for use within SuperCollider. Its purpose is to run the min and max basic python methods to certain columns. These values will be used for the paramenter mapping. It returns the min and max value of the specified column. These are stored in a dictionary. More information can be found at 8.

It takes 2 arguments, these are:

1. data-file that the min max values will be extracted
2. column in the data-set

It runs from the terminal with the following command.

```
python minmax.py data-file column
```

In SuperCollider this command will run using the "unixCmdGetStdOutLines" method. It will return the values as "string" in the SuperCollider environment.

5.5 datetime re-sample function

This function was implemented to re-sample the processed data-frame. In the non-resampled one the collection frequency is 30s. So, every 30 seconds a new value is stored for all parameters.

While this can of course result to precise estimations regarding events in the data it might not be very convenient if someone would like to quickly listen longer time periods. For example, with an iteration frequency of 1000 values per second it takes 86.4 seconds time to listen to one day. This was thought as a limitation and that's why this function was implemented to create down-sampled versions of the main data-set. Speeding up the iteration frequency was not an option because of computing power limitations.

The frequencies in the re-sampling process that was selected are:

1. 30S (non re-sampled)
2. T / 1 minute
3. H / 1 Hour
4. D / 1 Day
5. W / 1 Week

The re-sample function is accessed by the checkbox on the interface 3. In every re-sample period the maximum value observed is stored. For example, the non re-sampled data-set has:

00:00:00	0
00:00:30	1
00:01:00	4

If the re-sampling frequency is T (1 min) the 4 value will be stored in this cycle.

00:01:00	4

Overall, the re-sample function is base on the resample() method in combination with certain conditional tasks so that the correct checkbox element corresponds to the according resample function parameters. Technically, it is actually divided into two functions. The first does the conditional argument setting (feeds the correct arguments to the other function) while the other does the actual re-sampling and writes it to disk (CSV).

In the iteration process ... **TO BE DONE & REPORTED** (Write only re-sampling function related info, next section is: on-run functions)

6 on-run functions

7 IPC inter-process communication (includes connection setup functions)

The IPC is based on the OSC protocol and its aim is to interconnect Python and SueprCollider. It is based on sending individual messages from the Python process triggered by the Python interface elements.

In the OSC configuration there are 3 different OSC addresses that allow communication between the two software. These are:

1. | main address for the iteration process. SC received the data values (OSCdef). Py → SC
2. | controls the synths, acts like an ON/OFF switch. Triggered by the 6 synth ON/OFF buttons. Py → SC
3. | This acts like main ON/OFF switch for the all synths, it is triggered by the START and Killall buttons. Py → SC
4. | Initialisation address, activates the interaction elements on the interface when (or if) the SC responds. This configuration uses the 1234 port instead. SC → Py

The initial configuration is related to obtaining the IP address of the computer automatically to avoid manual configurations. This was implemented with the following functions in Python and SuperCollider.

Python

```
# get IP address
def getip():
    global ip
    ip = subprocess.Popen(
        'ipconfig getifaddr en0', shell=True,
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT)
    ip, _ = ip.communicate()
    ip = ip.decode('utf-8')
    ip = ip.strip()
    print(ip)
```

```
# Python osc
getip() # run getip function
client = udp_client.SimpleUDPClient(ip, 57120)
```

To simplify, this function runs

```
'ipconfig getifaddr en0'
```

in the terminal and extracts the current ip address, stores it in a variable and uses it for the OSC UDP client setup.

SuperCollider

On the SC side now the configuration is implemented in the following way

```
~ip = ("ipconfig getifaddr en0").unixCmdGetStdOutLines[0]; // get ip
n = NetAddr(~ip, 1234); // set netaddress
n.sendMsg('/startup/',1); // send to python that everything is loaded to enable buttons
("python communication established").postln;
```

At this point where the above command runs, an OSC message is sent to python using the '*startup*' address. In the OSC server function (receiver) in Python a task activates the interaction elements on the interface. This process can be found in the `../src/oscServerPython.py` file.

The communication using the 1 address is the most important one since concerns the iteration process. The data values in every iteration cycle (row by row) are sent to SuperCollider as shown below.

```
client.send_message("/pysc", datetime_selection.iloc[i])
```

8 Sonification algorithm

8.1 processing functions

The sonification processing functions are required for the correct mapping of the values. The aim here is to create a dictionary that contains the min and max values observed in the data for each column. The first function (stored in the ~minmax variable) will read and extract the min and max values from the raw data file. It utilises a python script (`../src/minmax.py`) and the unixCmdGetStdOutLines SuperCollider method. The unixCmdGetStdOut-Lines will "execute a UNIX command asynchronously using the standard shell (sh)..". In this case, that would be the evaluation of the minmax.py

script along with two arguments, the raw data file path and the desired column. This takes place in the second function that actually evaluates the previous one and stores the values. The python script evaluation returns the min and max values while these are being stored in a list for every column (temperature, humidity, noise levels, particles). Then these values are stored again in a dictionary to be used for the mapping later. The truck count data min max range were used manually (range: 0 - 9).

8.2 data receiver – handler

The data receiver from the SuperCollider side is an OSCdef that uses the "/pysc" address. The data that are being send are in array type. The elements of the array are stored in variables (temperature, humidity, pm25, pm10 and numTrucks). Every time an incoming message is received all data values are being extracted and mapped linearly to a 0.0 - 1.0 scale for further mapping convenience. Then these values were mapped to the appropriate ones either linearly or exponentially by utilising the min max values from the dictionary (see 8.1).

8.3 mappings

There is also information on the "documentation" page on the interface.

- The pm10 and pm25 values were mapped exponentially to a range of 261 - 523 Hz and 1044Hz - 2092Hz for the "warning" sounds and then mapped to the Lydian scale (see 8.4). For the "warning" sounds the values were mapped to pan values (-1) - (-0.5) for the pm10 and 1 - 0.5 for the pm25. Also, the values were mapped to the ranges 1 - 5 sec, 0.1 - 0.4 (multiplier), 0.0 - 1.0 and for release time, amplitude (warnings) and amplitude of Gendy accordingly.

- The humidity values were linearly mapped to a range of 1.5 to 3.0 for the "t60" reverb parameter and from 0.5 to 1.5 for the "wet" reverb parameter as well.

- The noise levels were linearly mapped to the depth of the modulator on a range from 0.0 to 5.0. The values were also exponentially mapped to a range from 6000 Hz to 18000 Hz of a low pass filter. They were linearly mapped to frequency values from 120Hz to 520Hz for the main oscillator frequency and as multiplier values for the modulation oscillator mix. Pulse wave → linear 0.0 - 1.0, Sawtooth-Tri → inverted linear 1.0 - 0.0.

- The temperature values were not used in the sonification.

The mapping takes place by utilising a dictionary for the synth parameters. The new values are registered to the dictionary keys and then mapped as synth parameters.

8.4 map to scale frequency mapping patch

8.4.1 Map to Lydian scale

To use a musical scale for the frequency mapping a function was built (~mapToScale). The incoming frequency value is subtracted from each element of a list containing the exact frequencies of the Lydian scale while an ABS is applied to the new list. After that the smallest value is returned. In this way the frequency of the Lydian scale that was the closest to the incoming one will be selected.

This function was used for the particle and the truck count data.

8.4.2 Map to Beat

The exact same logic was followed for the Beat mapping as well (noise values). In this case an array (~beatList) containing rhythmic values was used.

```
~beatList = [0,2,4,6,8,10,12,14,16,18,20];
```

8.5 event receiver - actions

Certain actions on the interface trigger OSC messages that were being sent to SuperCollider. This event receiver is based on two OSCdefs (defined in the `../src/parameterReceiver.scd` file). One is for turning ON & OFF the appropriate synths according to the active on/off buttons on the interface. The second one is responding to OSC messages from the "Start" & "Killall" buttons. This will start or stop & mute everything that is currently playing when the buttons are pressed.

8.6 synthesisers

All synthesisers are defined in the `../src/synthDefs.scd` file, except the one that plays back the truck values `../src/trucks.scd`

The general idea is based on a having a master bus mixing all synths to stereo. The buses were first declared and then used as the "bus" argument in the Out.ar class. Whenever a synth button on the interface is activated

or deactivated the appropriate "gate" parameter in the master synth is set to 1 and 0 accordingly. First the particles and particle warnings were mixed together and on the derived signal the noise level and truck count synths were added.

This signal is being sent to the reverb that is controlled by the humidity values. In the output a Limiter was used to prevent signal clipping.

A detailed list of all synthesisers is the following:

main output	
pm10	x.pm10synth
pm25	x.pm25synth
warnings pm 10	pm10
warnings pm 25	pm25
noise levels	
truck count	(in `../src/trucks.scd`
humidity	controls busOut reverb parameters

9 Line graphs

The Line graphs were implemented using "line" class from Bokeh. The line plots show a re-sampled version of the extensive data-set for the sake of speed and efficiency. The data-set is too cumbersome/bulky for efficient line plot rendering in the browser. The data-set was re-sampled with a period of 5 minutes. In that period the maximum value observed is plotted as a point in the graph.

The pm10 values and pm25 are in the same graph where pm10 are represented with the blue colour while the pm25 with green. The "olivedrab" colour was selected for the noise values, "indigo" for the humidity and "olive" for the truck count.

It was essential to create a moving box annotation that shows the current date-time that the user listens but also the area in the line graph that is played already. Transparency is used for the played back area.

The line graphs share common zoom. So if the user zooms in or out in any graph the rest will respond to the action.