Audio: [smart emission]

Legenda bij de geluidsniveaus, indeling in 5 niveaus op Viewers Heron en SmartApp:

De geluidsniveaus zijn met 5 kleuren aangegeven op de Heron viewer.

Niveau - dB(A) Range - voorbeeldgeluidsbron per niveau

1. 20-35 dB(A) - Stil

2. 35-50 dB(A) - Rustige woonwijk

3. 50-65 dB(A) - Drukke woonwijk

4. 65-80 dB(A) - Bij Spoor

5. 80-90 dB(A) - Rockband

Kleuren bij niveaus:

* Niveau 1 is groen,
* Niveau 2 is blauw,
* Niveau 3 is geel,
* Niveau 4 is oranje,
* Niveau 5 is rood.

Uitleg:

toelichting indeling en software berekening

(info Peter van der Voorn en Just van den Broecke, samenvatting Linda Carton.)

[0: vanaf 0 tot 20 dB(A), Jose sensor meet deze geluidsdruk niet] dB(A) waarde 20: fluister stil
[1: vanaf 20 tot 35 dB(A)] dB(A) waarde 35: rustige woonwijk in een stad
[2: vanaf 35 tot 50 dB(A)] dB(A) waarde 50: drukke woonwijk in een stad
[3: vanaf 50 tot 65 dB(A)] dB(A) waarde 65: wonen op korte afstand van het spoor
[4: vanaf 65 tot 80 dB(A)] dB(A) waarde 80: live optreden van een band aan het einde van het publieksdeel. Praten is mogelijk.
[5: vanaf 80 tot 90/95 dB(A)] dB(A) waarde 95: live optreden van een band midden op een plein. Praten is onmogelijk.

Toelichting:

------ Totaalwaarde van waarden per frequentieband -----------
Peter van der Voorn:

Logaritmisch optellen van de waarden per frequentieband voor het
verkrijgen van de totaalwaarde:

10^(waarde/10)
En dat voor de waarden van alle frequenties en bij elkaar tellen.
Daar de log van en x10

Normaal tellen wij op van 31,5 Hz tot 8 kHz. In totaal 9 oktaafanden.
31,5  63  125  250  500  1000  2000  4000 en 8000 Hz

Of 27   1/3 oktaafbanden: 25, 31.5, 40, 50, 63, 80, enz

Voor het categoriseren van de meetwaarden kunnen we het beste beginnen
bij de 20 dB(A). De hoog[s]te waarde zal 95 dB(A) zijn. Bijvoorbeeld een vogel van heel dichtbij.
Je kunt dit nu gewoon lineair verdelen in 5 categori[e]ën. Ieder 15 dB. Het
betreft buiten meetwaarden.

20 fluister stil
35 rustige woonwijk in een stad
50 drukke woonwijk in een stad
65 wonen op korte afstand van het spoor
80 live optreden van een band aan het einde van het publieksdeel. Praten
is mogelijk.
95 live optreden van een band midden op een plein. Praten is onmogelijk.

Gr. Peter

---

From: Just van den Broecke
Sent: 01 March 2016 15:39
To: Carton, L.J.; Geurts, Paul PI50; Michel Grothe; Janus Hoeks; Vermeulen, Giel PI70; Antoine van de Cruyssen; matthijs kastelijns; robert citigis; freek thuis; bas de greef
Subject: Nieuwe geluidsberekeningen

Beste Mensen,

Op grond van input die ik van Peter vd Voorn  kreeg (zie onder) heb ik
"geluid" een nieuwe berekening gegeven en de categorisering (levels)
aangepast:

http://smartemission.nl/heron

Voor SmartApp maakt dit niets uit. Voor degenen die de  berekeningen
willen zien:
https://github.com/Geonovum/smartemission/blob/master/etl/python/sensorconverters.py
(250-294 en 179-204).

Natuurlijk is nog niet het laatste gezegd en geimplementeerd over geluid
maar de waarden lijken nu veel realistischer.

kind regards / met vriendelijke groet,

--Just

=======CODE documented in GitHub =============

Software code ETL proces, converter informatie Audio data van Jose sensoren op GitHub

Github:

https://github.com/Geonovum/smartemission/blob/a693bbf930045e9c7df1b24b97f964d17071f37d/etl/sensorconverters.py

path: smartemission / etl / sensorconverters.py

a693bbf 30 June 2016

@justb4 justb4 Refiner, very raw version with DB and progress support

2 contributors @justb4 @pietermarsman

478 lines (396 sloc) 15.9 KB

Code:

Lines

1 import time

2 from calendar import timegm

3 from datetime import datetime, tzinfo, timedelta

4 import re

5 import math

6 from stetl.util import Util

7

8 log = Util.get\_log("SensorConverters")

9

10 # According to CityGIS the units are defined as follows. ::

11 #

12 # S.TemperatureUnit milliKelvin

13 # S.TemperatureAmbient milliKelvin

14 # S.Humidity %mRH

15 # S.LightsensorTop Lux

16 # S.LightsensorBottom Lux

17 # S.Barometer Pascal

18 # S.Altimeter Meter

19 # S.CO ppb

20 # S.NO2 ppb

21 # S.AcceleroX 2 ~ +2G (0x200 = midscale)

22 # S.AcceleroY 2 ~ +2G (0x200 = midscale)

23 # S.AcceleroZ 2 ~ +2G (0x200 = midscale)

24 # S.LightsensorRed Lux

25 # S.LightsensorGreen Lux

26 # S.LightsensorBlue Lux

27 # S.RGBColor 8 bit R, 8 bit G, 8 bit B

28 # S.BottomSwitches ?

29 # S.O3 ppb

30 # S.CO2 ppb

31 # v3: S.ExternalTemp milliKelvin

32 # v3: S.COResistance Ohm

33 # v3: S.No2Resistance Ohm

34 # v3: S.O3Resistance Ohm

35 # S.AudioMinus5 Octave -5 in dB(A)

36 # S.AudioMinus4 Octave -4 in dB(A)

37 # S.AudioMinus3 Octave -3 in dB(A)

38 # S.AudioMinus2 Octave -2 in dB(A)

39 # S.AudioMinus1 Octave -1 in dB(A)

40 # S.Audio0 Octave 0 in dB(A)

41 # S.AudioPlus1 Octave +1 in dB(A)

42 # S.AudioPlus2 Octave +2 in dB(A)

43 # S.AudioPlus3 Octave +3 in dB(A)

44 # S.AudioPlus4 Octave +4 in dB(A)

45 # S.AudioPlus5 Octave +5 in dB(A)

46 # S.AudioPlus6 Octave +6 in dB(A)

47 # S.AudioPlus7 Octave +7 in dB(A)

48 # S.AudioPlus8 Octave +8 in dB(A)

49 # S.AudioPlus9 Octave +9 in dB(A)

50 # S.AudioPlus10 Octave +10 in dB(A)

51 # S.SatInfo

52 # S.Latitude nibbles: n1:0=East/North, 8=West/South; n2&n3: whole degrees (0-180); n4-n8: degree fraction (max 999999)

53 # S.Longitude nibbles: n1:0=East/North, 8=West/South; n2&n3: whole degrees (0-180); n4-n8: degree fraction (max 999999)

54 #

55 # P.Powerstate Power State

56 # P.BatteryVoltage Battery Voltage (milliVolts)

57 # P.BatteryTemperature Battery Temperature (milliKelvin)

58 # P.BatteryGauge Get Battery Gauge, BFFF = Battery full, 1FFF = Battery fail, 0000 = No Battery Installed

59 # P.MuxStatus Mux Status (0-7=channel,F=inhibited)

60 # P.ErrorStatus Error Status (0=OK)

61 # P.BaseTimer BaseTimer (seconds)

62 # P.SessionUptime Session Uptime (seconds)

63 # P.TotalUptime Total Uptime (minutes)

64 # v3: P.COHeaterMode CO heater mode

65 # P.COHeater Powerstate CO heater (0/1)

66 # P.NO2Heater Powerstate NO2 heater (0/1)

67 # P.O3Heater Powerstate O3 heater (0/1)

68 # v3: P.CO2Heater Powerstate CO2 heater (0/1)

69 # P.UnitSerialnumber Serialnumber of unit

70 # P.TemporarilyEnableDebugLeds Debug leds (0/1)

71 # P.TemporarilyEnableBaseTimer Enable BaseTime (0/1)

72 # P.ControllerReset WIFI reset

73 # P.FirmwareUpdate Firmware update, reboot to bootloader

74 #

75 # Unknown at this moment (decimal):

76 # P.11

77 # P.16

78 # P.17

79 # P.18

80

81

82 # Conversion functions for raw values from Josene sensors

…

296 # From https://www.teachengineering.org/view\_activity.php?url=collection/nyu\_/activities/nyu\_noise/nyu\_noise\_activity1.xml

297 # level dB(A)

298 # 1 0-20 zero to quiet room

299 # 2 20-40 up to average residence

300 # 3 40-80 up to noisy class, alarm clock, police whistle

301 # 4 80-90 truck with muffler

302 # 5 90-up severe: pneumatic drill, artillery,

303 #

304 # Peter vd Voorn:

305 # Voor het categoriseren van de meetwaarden kunnen we het beste beginnen bij de 20 dB(A).

306 # De hoogte waarde zal 95 dB(A) zijn. Bijvoorbeeld een vogel van heel dichtbij.

307 # Je kunt dit nu gewoon lineair verdelen in 5 categorieen. Ieder 15 dB. Het betreft buiten meetwaarden.

308 # 20 fluister stil

309 # 35 rustige woonwijk in een stad

310 # 50 drukke woonwijk in een stad

311 # 65 wonen op korte afstand van het spoor

312 # 80 live optreden van een band aan het einde van het publieksdeel. Praten is mogelijk.

313 # 95 live optreden van een band midden op een plein. Praten is onmogelijk.

314 def calc\_audio\_level(db):

315 levels = [20, 35, 50, 65, 80, 95]

316 level\_num = 1

317 for i in range(0, len(levels)):

318 if db > levels[i]:

319 level\_num = i + 1

320

321 return level\_num

322

323 def convert\_noise\_level(value, json\_obj, name):

324 return calc\_audio\_level(value)

325

326 # Converts audio var and populates virtual max value vars

327 # NB not used: now taking average of max values, see convert\_audio\_avg()

328 def convert\_audio\_max(value, json\_obj, name):

329 # For each audio observation:

330 # decode into 3 bands (0,1,2)

331 # determine max of these bands

332 # determine if this is greater than current t\_audiomax

333 # determine audio\_level (1-5) from current t\_audiomax

334

335 # Extract values for bands 0-2

336 bands = [value & 255, (value >> 8) & 255, (value >> 16) & 255]

337

338 band\_max = 0

339 band\_num = 0

340 for i in range(0, len(bands)):

341 if band\_max < bands[i] < 255:

342 band\_max = bands[i]

343 band\_num = i

344

345 if band\_max == 0:

346 return None

347

348 # Assign band max value to virtual sensors if these are not yet present

349 # or band\_max value greater than current max

350 if 't\_audiomax' not in json\_obj or band\_max > json\_obj['t\_audiomax']:

351 json\_obj['t\_audiomax'] = band\_max

352 # Determine octave nr from var name

353 json\_obj['t\_audiomax\_octave'] = int(re.findall(r'\d+', name)[0])

354 json\_obj['t\_audiomax\_octband'] = band\_num

355 json\_obj['t\_audiolevel'] = calc\_audio\_level(band\_max)

356

357 return band\_max

358

359

360 # Converts audio var and populates average

361 # Logaritmisch optellen van de waarden per frequentieband voor het verkrijgen van de totaalwaarde:

362 #

363 # 10^(waarde/10)

364 # En dat voor de waarden van alle frequenties en bij elkaar tellen.

365 # Daar de log van en x10

366 #

367 # Normaal tellen wij op van 31,5 Hz tot 8 kHz. In totaal 9 oktaafanden. 31,5 63 125 250 500 1000 2000 4000 en 8000 Hz

368 #

369 # Of 27 1/3 oktaafbanden: 25, 31.5, 40, 50, 63, 80, enz

370 def convert\_audio\_avg(value, json\_obj, name):

371 # For each audio observation:

372 # decode into 3 bands (0,1,2)

373 # determine average of these bands

374 # determine overall average of all average bands

375

376 # Extract values for bands 0-2

377 bands = [float(value & 255), float((value >> 8) & 255), float((value >> 16) & 255)]

378

379 # determine average of these 3 bands

380 band\_avg = 0

381 band\_cnt = 0

382 for i in range(0, len(bands)):

383 band\_val = bands[i]

384 # outliers

385 if band\_val < 1 or band\_val > 150:

386 continue

387 band\_cnt += 1

388

389 # convert band value Decibel to Bel and then get "real" value (power 10)

390 band\_avg += math.pow(10, band\_val / 10)

391 # print '%s : band[%d]=%f band\_avg=%f' %(name, i, bands[i], band\_avg)

392

393 if band\_cnt == 0:

394 return None

395

396 # Take average of "real" values and convert back to Bel via log10 and Decibel via \*10

397 band\_avg = math.log10(band\_avg / float(band\_cnt)) \* 10.0

398

399 # print '%s : avg=%d' %(name, band\_avg)

400

401 if band\_avg < 1 or band\_avg > 150:

402 return None

403

404 # Initialize average value to first average calc

405 if 'v\_audioavg' not in json\_obj:

406 json\_obj['v\_audioavg'] = band\_avg

407 json\_obj['v\_audioavg\_total'] = math.pow(10, band\_avg / 10)

408 json\_obj['v\_audioavg\_cnt'] = 1

409 else:

410 json\_obj['v\_audioavg\_cnt'] += 1

411 json\_obj['v\_audioavg\_total'] += math.pow(10, band\_avg / 10)

412 json\_obj['v\_audioavg'] = int(

413 round(math.log10(json\_obj['v\_audioavg\_total'] / json\_obj['v\_audioavg\_cnt']) \* 10.0))

414

415 # Determine octave nr from var name

416 json\_obj['v\_audiolevel'] = calc\_audio\_level(json\_obj['v\_audioavg'])

417 # print 'Unit %s - %s band\_db=%f avg\_db=%d level=%d' % (json\_obj['p\_unitserialnumber'], name, band\_avg, json\_obj['v\_audioavg'], json\_obj['v\_audiolevel'] )

418 return band\_avg

…