An Overtone Based Algorithm Unifying Counterpoint and Harmonics

Guido Kramann[1]

Fachhochschule Brandenburg
kramann@fh-brandenburg.de

Abstract. In this paper an algorithm is introduced which allows to estimate a chord's degree of consonance as well as its possibilities of progression regardless of the number of tones it consists of and regardless of the underlying pitch space (e.g. tempered or microtonal).
This claim could be achieved by introducing an overtone based point of view to single tones of the examined chords and – more detailed – by not only taking into account which overtone a single tone could have but also of which other tone it could be an overtone itself. In addition the well known tolerant and logarithmic character of human perception of tone pitch was taken into account.

Keywords: computational musicology, music production and composition tools, linear counterpoint, pitch spaces, algorithmic composition

1 Introduction

In musicology there is an ongoing discussion about physical foundation of harmonics [5]. One of the most substantial arguments against a physical foundation based on overtones is that overtones do not explain why a minor third sounds consonant to a human. In any case overtones represent a link between matter and humans as they are tonal representants of the matters' eigenvalues. So they are the best candidates for a physical foundation of music we know and due to this fact often meet with theoretical works about sound, music perception and music production, e.g. [6], [8], [3]. Although the focus of interest of this article does not lie in this discussion, for the here presented overtone based algorithm which will be introduced in chapter 4 it is of high relevance. Particularly it should be mentioned at the beginning that overtones are used in the algorithm not as part of a tone but as potential supplements to a pitch. In addition this algorithm also takes into account the well known tolerant and logarithmic (Weber-Fechner law) character of human perception of tone pitch.[1] This means that physical are

[1] A difference of about three cent is given to unisono voices to achieve a more vivid sound e.g. in accordions. So two pitches with up to three cent difference will be accepted by humans as the "same" tone. Steps in octaves are recognized as linear progression. Direct evidence for the tolerant and logarithmic character of human acoustic perception is given by the tempered pitch space: Equidistant tone steps in this system correspond with equal factors in their representation as frequencies.

considered as well as physiological aspects. By doing this it is possible to obtain e.g. quantitative results for the degree of consonance of intervals which agree with classical music theory comprising minor thirds as will be seen in chapter 5 (Adjustment and verification of tonalCoincidence Algorithm). What is more the same algorithm can also be used to estimate possibilities of chord progression by making use of an idea from a nearly forgotten work "Linearer Kontrapunkt" by Ernst Kurth [4]. Finally the here mentioned implicit algorithmic formulation for horizontal and vertical composition rules can lead to more unifying and compact formulations of composing algorithms and gives a better chance to expand its application area than explicit formulated rules do. This can be seen as an alternative approach to systems of rule based music computation which are highly bound to the context respective music genre they where developed for – especially if they are very complex, e.g. [2] [1]. A showcase for an algorithmic composition program using the here introduced algorithm is given in chapter 6 (Doing Algorithmic Composition using the tonalCoincidence Algorithm). But before all this it is nescessary to introduce some basic definitions, transformation formulas and scales on which the algorithm is based on in the following chapters 2 and 3:

2 Basic Formulas

Using a MIDI pitch would comply with the demand of beeing logarithmic regarding to a representation using frequencies. This is well introduced but the smallest entity here is about one half tone and this is indeed very rough. Also well introduced is it to divide one half tone in 100 steps where one step is called cent. Here instead of defining a pitch by its MIDI number and an additional amount of cents both is drawn together in a scale further called "midicent" [mc]. The way how transformations between frequencies f [Hz], MIDI pitches mp [midi] and midicent pitches mcp [mc] can be done can be seen in the formulas 1 to 6. Following these definitions a MIDI pitch of 69midi corresponds to a frequency of 440Hz and the midicent pitch of the same tone is 6900mc, see also table 2.

$$\mathrm{midi2frequency(mp)} = \frac{440}{2^{\frac{69}{12}}} 2^{\frac{\mathrm{mp}}{12}} \tag{1}$$

$$\mathrm{frequency2midi(f)} = 12 \log_2(\mathrm{f}\frac{2^{\frac{69}{12}}}{440}) \tag{2}$$

$$\mathrm{midicent2frequency(mcp)} = \frac{440}{2^{\frac{69}{12}}} 2^{\frac{\mathrm{mcp}}{1200}} \tag{3}$$

$$\mathrm{frequency2midicent(f)} = 1200 \log_2(\mathrm{f}\frac{2^{\frac{69}{12}}}{440}) \tag{4}$$

$$\mathrm{midi2midicent(mp)} = 100\mathrm{mp} \tag{5}$$

$$\mathrm{midicent2midi(mcp)} = \frac{\mathrm{mcp}}{100} \tag{6}$$

frequency [Hz]	midi pitch [midi]	midicent pitch [mc]
440	69	6900
660	76	7602
880	81	8100

Table 2: Examples for transformations.

3 Basic Definitions: Overtones and Undertones

Overtones are whole-number multiples of the frequency of a base tone and are well known.

In the context of this work an undertone of a pitch A means a pitch B which could have pitch A as overtone. B is a (virtual) base tone of A.

If there is a tone with the frequency f and a positive integer number m, then the frequency of its m-th overtone is:

$$\mathrm{fo}_m = (\mathrm{m}+1)\mathrm{f} \tag{7}$$

For the same tone its m-th undertone is:

$$\mathrm{fu}_m = \frac{\mathrm{f}}{\mathrm{m}+1} \tag{8}$$

In the midicent representation over- and undertones can be found symmetrically to both sides of the tone they are calculated for and transpositions of this tone result in a linear shift of its over- and undertones. Table 3 and Figure 1 illustrate this by showing three different tones with five over- and undertones in their midicent representation. The corresponding frequencies of these tones are 440Hz (first tone), 660Hz (second tone) and 880Hz (third tone). Listing 3 represents a function to evaluate an array with a midicent pitch in the middle of the array and N undertones on the left and N overtones on the right side and is called *dockingPoints*.

Listing 3: Function "dockingPoints" - Evaluate an array with a midicent pitch in the middle and N under- and overtones

```
dock[ ] = dockingPoints(mcp,N)
  f=midicent2frequency(mcp)
  for i=0:N-1
      dock[i]=frequency2midicent(f/(N+1-i))
  end
  dock[N]=mcp
  for i=0:N-1
      dock[i+N+1]=frequency2midicent(f*(i+2))
  end
  return dock[ ]
end.
```

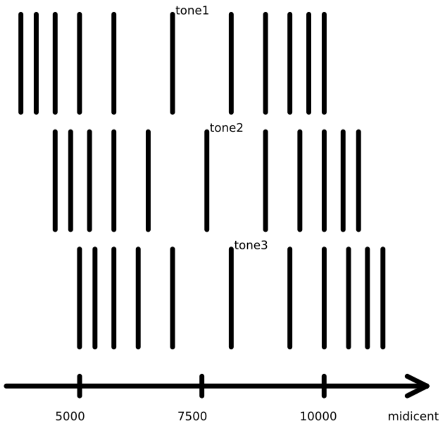

Fig. 1. Over- and undertones in midicent representation.

u5	u4	u3	u2	u1	tone	o1	o2	o3	o4	o5
3798	4114	4500	4998	5700	6900	8100	8802	9300	9686	10002
4500	4816	5202	5700	6402	7602	8802	9504	10002	10388	10704
4998	5314	5700	6198	6900	8100	9300	10002	10500	10886	11202

Table 3: Over- and undertones in midicent representation.

4 Tonal Coincidence Algorithm

The basic concept of this paper to analyse intervals and later also chords is represented in an algorithm called *tonalCoincidence* and is presented in Listing 4-2. For this algorithm overtones and undertones play the role of "virtual docking points" which a tone A exposes to a tone B and vice versa. As variables *tonalCoincidence* needs the midicent pitches of a tone A and its reference tone B. As parameters *tonalCoincidence* needs the number of over- and undertones N to take into account and the range two docking points may differ from each other to still coincide. Whenever the function *tonalCoincidence* is called, it counts the number of docking points of tone A which coincede with those of its reference tone B. The function *dockingPoints* (Listing 3) is called from within *tonalCoincidence* and produces the necessary docking points for a tone B which is called a reference pitch here (mcpref - mdicent pitch reference). A tone A should also coincide to another tone B when it is transposed in octave steps. The function *dockingPointsMult* (Listing 4-1) generates and combines docking points for a tone A and several of its octave transpositions. Last is done upwards and downwards in enough steps that any possible coincidence is taken into account. The function *combine* whithin *dockingPointsMult* combines two arrays and sorts the resulting array. It also takes care that each value in the resulting array only appears once.

Listing 4-1: Function "dockingPointsMult" - Evaluate more docking points by doing transpositions in octave steps

```
dockmult[ ] = dockingPointsMult(mcp,sref[ ])
  N=(size(sref[ ])-1)/2
  dockmult[ ] = dockingPoints(mcp,N)
  mcpOct=mcp+1200
  while frequency2Midicent(midicent2Frequency(mcpOct)/N)<=max(sref[ ]) do
      dockmult[ ] = combine( dockmult[ ] , dockingPoints(mcpOct,N) )
      mcpOct=mcpOct+1200
  end
  mcpOct=mcp-1200
  while frequency2Midicent(midicent2Frequency(mcpOct)*N)>=min(sref[ ]) do
      dockmult[ ] = combine( dockmult[ ] , dockingPoints(mcpOct,N) )
      mcpOct=mcpOct-1200
  end
  return dockmult[ ]
end.
```

Listing 4-2: Algorithm "tonalCoincidence" - Coincidence of a midicent pitch with a reference pitch

```
count = tonalCoincidence(mcp,mcpref,TOLERANCE=24,N=5)
  count=0
  sref[ ]=dockingPoints(mcpref,N)
  s=dockingPointsMult(mcp,sref[ ])
  foreach s[ ] as v
      foreach sref[ ] as vref
          if v+TOLERANCE>=vref AND v-TOLERANCE<=vref THEN count=count+1
      end
  end
  return count
end.
```

5 Adjustment and Verification of Tonal Coincidence Algorithm

As can be seen in the *tonalCoincidence* algorithm (Listing 4-2), the first parameter **TOLERANCE** is set to 24 and the second one **N** is set to 5 by default. There is no physical or physiological reason for this at the moment, but it is parametrized like this to achieve suitable results when compared to a subset of states from classical music theory.

A verification for the algorithm is done inasmuch as it is shown that there will not appear inconsistences for this parameter set.

First Test: Intervals. As first test consonant and dissonant intervals are built and opposed to their tonal coincidence value. For symmetry reasons – it is not determinable which pitch is reference – *tonalCoincidence* is called twice where pitch and reference are interchanged in the second call and both results are added then. This variant will be called *tonalSymmetricCoincidence*. Its implementation can be seen in Listing 5-1. The test can be done e.g. with C' (60midi) as base tone or with any other tone without changes in the results. The results of this test can be seen in Table 5: Consonant intervals have values smaller and dissonant ones have values bigger than five in the *tonalSymmetricCoincidence* column. Octaves have the highest *tonalSymmetricCoincidence* value. Minor thirds obtained a much smaller value than major thirds but are still distinguishable from dissonant intervals.

Listing 5-1: Degree of consonance for intervals

```
count = tonalSymmetricCoincidence(mcpA,mcpB,TOLERANCE=24,N=5)
  return tonalCoincidence(mcpA,mcpB,TOLERANCE,N) ...
   ... + tonalCoincidence(mcpB,mcpA,TOLERANCE,N)
end.
```

Interval name	Half tone steps	*tonalSymmetricCoincidence*
Perfect unison	0	22
Minor second	1	4
Major second	2	4
Minor third	3	6
Major third	4	12
Perfect fourth	5	14
Tritone	6	0
Perfect fith	7	14
Minor sixth	8	12
Major sixth	9	6
Minor seventh	10	4
Major seventh	11	4
Perfect unison + 1 Perfect octave	12	22
Minor second + 1 Perfect octave	13	4
Major second + 1 Perfect octave	14	4
Minor third + 1 Perfect octave	15	6
Major third + 1 Perfect octave	16	12
Perfect fourth + 1 Perfect octave	17	14
Tritone + 1 Perfect octave	18	0
Perfect fith + 1 Perfect octave	19	14
Minor sixth + 1 Perfect octave	20	12
Major sixth + 1 Perfect octave	21	6
Minor seventh + 1 Perfect octave	22	4
Major seventh + 1 Perfect octave	23	4
Perfect unison + 2 Perfect octaves	24	22

Table 5: Tests with intervals.

Second Test: Microtones. A fifth does not sound as nice as it should sound on a piano because here the frequency ratio is not exactly 3:2==1.5 but is $2^{(7/12)}$==1.498... The second interval can be represented as interval C-G with 6000mc and 6700mc. The nearest integer representation of the first interval would be 6000mc and 6702mc. Both values obtained by calling *tonalSymmetricCoincidence* with these intervals remain at 14. This estimation is too rough but at least it does not produce inconsistent results. To distinguish between interval one and interval two or in any other case where microtonal differences play a role the parameter **TOLERANCE** could be decreased until one of both results differs from the other. At a **TOLERANCE** smaller or equal to 1 the perfect fith "wins" (interval one) and remains at 14 whereas the imperfect one (interval two) drops down to 0. Listing 5-2 shows an implementation (*tonalSymmetricCoincidenceMicro*) which quantifies also microtonal intervals. It is assumed here that it does not make any sense to set the parameter **TOLERANCE** to a value higher than 99.

Listing 5-2: Degree of consonance for microtone intervals

```
count = tonalSymmetricCoincidenceMicro(mcpA,mcpB,TOLERANCE=24,N=5)
  c = tonalSymmetricCoincidence(mcpA,mcpB,TOLERANCE,N)
  micro = 0
  TOLERANCE = TOLERANCE - 1
  while TOLERANCE>=0 AND c==tonalSymmetricCoincidence(mcpA,mcpB,TOLERANCE,N)
      micro = micro + 1
      TOLERANCE = TOLERANCE - 1
  end
  return c*100 + micro
end.
```

Third Test: Consonance of Chords. Now the evaluation of a degree of consonance will be expanded to handle also chords consisting of two, three, four and more tones. To achieve compatibility between chords of different numbers of tones the following is done: Each tone of the examined chord is taken and its pairwise coincidence value to the remaining tones of the same chord is calculated. From the obtained values the minimum is taken. The degree of consonance of a chord then is defined as the average of these minimums (Listing 5-3). To achieve a wide applicability also for microtones *tonalSymmetricCoincidenceMicro* is called for the pairwise calculations within *chordConsonance*. Some results can be seen in Table 5. The relative results for chords with the same number of tones is consistent, but in one case a dissonant chord (C-F-G) obtains a degree of consonance in a range which is obtained for a consonant chord/interval with a lower numbers of tones (C-Es). This is obviously not nice, but it is not easy to decide if it is wrong at this moment as it is not determinable to which extent a fuller sound – more tones in a chord – compensates dissonances.

Listing 5-3: Degree of consonance for chords

```
count = chordConsonance(chord[ ],TOLERANCE=24,N=5)
  c = 0
  m = size(chord[ ])
  for i=0:m-1
    minimum=tonalSymmetricCoincidenceMicro(chord[i],chord[i],TOLERANCE,N)
    for k=0:m-1
      test = tonalSymmetricCoincidenceMicro(chord[i],chord[k],TOLERANCE,N)
      if i!=k AND test<=minimum
        minimum =  test
      end
    end
    c = c + minimum
  end
  return c/m
end.
```

chord [mc]	cons. / diss.	name	*chordConsonance*
6000 6700	consonant	C-G (Fifth)	1422
6000 6702	consonant	C-G (Natural fifth	1424
6000 6400	consonant	C-E (Major third)	1210
6000 6300	consonant	C-Es (Minor third)	608
6000 6400 6700	consonant	C-E-G (C-Major)	808
5500 6000 6400	consonant	G-C-E (C5-Major)	808
5200 5500 6000	consonant	E-G-C (C3-Major)	808
6000 6300 6700	consonant	C-Es-G (C-Minor)	808
6000 6500 6700	dissonant	C-F-G (C45)	754
6000 6500 6700	consonant	C-F-A (F5-Major)	808
6000 6400 6700 7000	dissonant	C-E-G-Hb (C^7)	269
6000 6400 6700 7200	consonant	C-E-G-C (C-Major)	909
6000 6400 6702 7200	consonant	C-Major with natural fifth	910

Table 5: Tests with chords.

Fourth Test: Chords Progression. Finally the idea to calculate also possibilities of chord progression with another variant of the here introduced algorithms gives the key to use the tonal coincidence algorithm also for algorithmic composition.

As mentioned before the idea is based on the work "Linearer Kontrapunkt" by Ernst Kurth [4]. Kurth describes there the phenomenon that in a major chord the major third has a tendency to a movement up to fourth and interpretes it as a kind of horizontal dissonance.

In other words: Following the reasoning of Kurth the tone E in C E G does not sound dissonant in a static way but it has a certain proportion of restlessness in the context with C and G which could be neutralized by a specific chord progression.

It will be shown now that this portion of restlessness can be quantified for any tone of a chord using a method also based on the tonal coincidence algorithm.

By using the here introduced technique of counting coinciding docking points a measurement can be obtained which gives information about how good one tone of a chord fits together with the others in the same chord. To find out this e.g. for E in the context of C and G the docking points of C and G are combined and their coincidence with those of E is calculated. An algorithm named *tonePersistence* to evaluate this can be found in Listing 5-4 and a ”microtone variant” in Listing 5-5.

By doing this for each tone of a chord and also for a subsequent chord (e.g. C-E-G to C-F-A – tonica to subdominante) information can be obtained which can be used as an alternative for classical counterpoint rules.

To understand this it is necessary first to interpret each step of the successive chords as a tone of an individual voice. In the example C-E-G to C-F-A the voice in the middle moves from E to F. As can be seen in Table 5 E has the lowest persistence in C-E-G – it is the third of this chord – and F has the highest persistence in C-F-A – it is the root of this chord. In addition in the same table can be seen that the root has allways the highest and the third the lowest persistence in a chord. In a diminished chord all tones have a relative low persistence and in a chord consisting of octaves all tones have a relative high persistence.

Listing 5-4: Degree of persistence of a tone in a Chord

```
count = tonePersistence(index, chord[ ],TOLERANCE=24,N=5)
  count = 0
  m = size(chord[ ])
  q = 0
  testtone = chord[index]
  for i=0:m-1
      if i!=index
          if q==0
            dockref[ ] = dockingPoints(chord[i],N)
          else
            dockref[ ] = combine(dock , dockingPoints(chord[i],N))
          end
          q=q+1
      end
  end
  dock = dockingPointsMult(testtone,dockref[ ])
  foreach dock[ ] as v
      foreach dockref[ ] as vref
          if v+TOLERANCE>=vref AND v-TOLERANCE<=vref THEN count=count+1
      end
  end
  return count
end.
```

Listing 5-5: Degree of persistence of a microtone in a chord

```
count = tonePersistenceMicro(index, chord[ ],TOLERANCE=24,N=5)
  count = tonePersistence(index, chord[ ],TOLERANCE,N)
  micro = 0
  TOLERANCE = TOLERANCE - 1
  while TOLERANCE>=0 AND count==tonePersistence(index, chord[ ],TOLERANCE,N)
      micro = micro + 1
      TOLERANCE = TOLERANCE - 1
  end
  return count*100 + micro
end.
```

chord [mc]	testtone	description	*tonePersistenceMicro*
6000 6400 6700	6000	Persistence of C in E-G	1510
6000 6400 6700	6400	Persistence of E in C-G	1108
6000 6400 6700	6700	Persistence of G in C-E	1408
6000 6500 6900	6000	Persistence of C in F-A	1408
6000 6500 6900	6500	Persistence of F in C-A	1510
6000 6500 6900	6900	Persistence of A in C-F	1108
5900 6200 6500	5900	Persistence of H in D-F	907
5900 6200 6500	6200	Persistence of D in H-F	1008
5900 6200 6500	6500	Persistence of F in H-D	907
4800 6000 7200	4800	Persistence of c in c'-c"	1924
4800 6000 7200	6000	Persistence of c' in c-c"	1624
4800 6000 7200	7200	Persistence of c" in c-c'	1624

Table 5: Persistence

6 Doing Algorithmic Composition by Counting Coincidences of Over- and Undertones

The algorithms *tonePersistenceMicro* and *chordConsonance* can be used to formulate integral criteria to evaluate the quality of a piece of music. To demonstrate this as a simple showcase an algorithmic composition program will be described to generate a canon with four voices which uses *tonePersistenceMicro* and *chordConsonance*.[2]

The canon to be generated will be cyclic and the melody it is based on is shifted vertically as well as horizontally for the involved voices. "Cyclic" means that the chord the canon ends with has to fit to the one at the beginning. The horizontal shifts for voice 1, 2, 3, 4 are determined to 24, 0, 12, 36 time periods and the vertical ones are 700,0,-500,-1200 (in midicent). One time period is the smallest entity in the music piece and is identical to the shortest tone – here one eighth note. The algorithm starts with a random melody which will be optimized.

[2] As a canon has a very strict musical architecture which is ruled mainly by harmonic laws there is not needed too much more than the harmonic rules for the realization of the composition program. This is why this musical form was selected.

To be able to do that the melody is copied to a matrix with one row for each voice while taking into account the horizontal and vertical shifts. So the matrix represents the chord progression when all voices have started – because of the cyclic character of the canon overlapping parts of a voice at the end are copied to the beginning. Only this matrix is needed for the harmonical optimization. The optimization criteria are brought in a hierarchical order. The first one is the sum of all values for *chordConsonance* which is applied to each chord in the matrix. The second optimization criterion summates the absolute values of all changes of tone persistence (calculated with *tonePersistenceMicro*). The idea behind this is to make the piece as vivid as possible by having a high amount of interchanges in tone persistence. Hence the function is called *restlessness*), see Listing 6-1.

Listing 6-1: Interchanges in tone persistence in a piece of music

```
count = restlessness(matrix[ ][ ])
    count=0
    foreach chord c in matrix and its successor d
        count = count + abs(tonePersistenceMicro(0,c)-tonePersistenceMicro(0,d))
        count = count + abs(tonePersistenceMicro(1,c)-tonePersistenceMicro(1,d))
        count = count + abs(tonePersistenceMicro(2,c)-tonePersistenceMicro(2,d))
        count = count + abs(tonePersistenceMicro(3,c)-tonePersistenceMicro(3,d))
    end
    return count
end.
```

Two subordinate optimization criteria evaluate for the canon melody something which could be called "self-similarity" of pitches and intervals to obtain a catchy structure for it. Representative for all the canons which could be generated with this program the melody of one of them can be seen in Figure 2.

For a further examination a java source code representing the here described canon composition program as well as a soundfile and the score and its parts of the canon above can be obtained on this website: `http://www.kramann.info/unifying` As proof for the quality of the algorithms used in the canon composition one optimization process was analyzed: On the intermediate data of the optimization process two classical counterpoint rules were applied to find out if the number of classical counterpoint errors has a correlation to the actual computed quality of the canon. As first classical rule the ban of consecutive fiths was taken. As second classical criterion it was determined how often dissonances were not reached and left stepwise. As crossings of voices are allowd in canons the tones of two successive chords had to be sorted first before doing this analysis.

As can be seen in Figure 3 for the examined optimization process the number of consecutive fiths increases while the number of unallowed jumps into dissonances decreases. So it is possible to get influence on these criteria but the actual version of the composition algorithm tends to a kind of "archaic" and respectively "rock music".

Fig. 2. voice of an algorithmic generated canon.

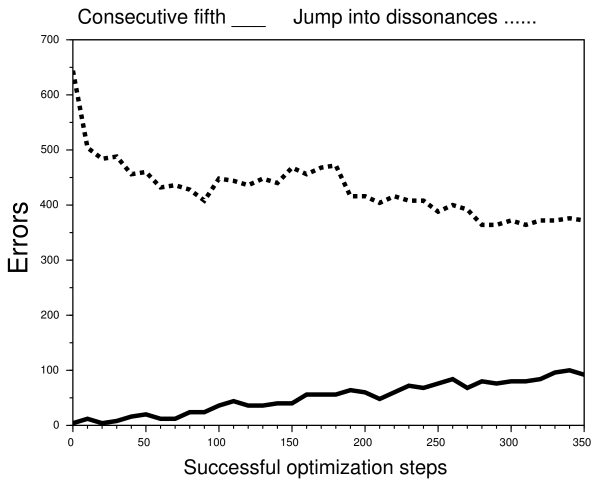

Fig. 3. Harmonics errors.

7 Concluding Remarks and Further Work

In this paper a new implicite algorithmic formulation for horizontal and vertical composition rules was presented and its verification was done as well as its applicability was shown. As this work focuses on algorithmic composition it was important to find a system which is able to deal with any sound structure. Actually a verification was only done by comparisons to a small selection of rules taken from classical music theory. Works like [7] – here harmonic and inharmonic sounds were fed to ear models – could give a better proof for formulas and algorithms which evaluate e.g. a degree of consonance for chords. Unfortunatly in [7] only pitches with additional overtones were examined as representants of harmonic sounds. On the other hand doing algorithmic composition requires to objectify the compositional process and its rules in a much higher degree than a reliable but patchy music theory can offer, which is oriented more on analytical than on generative demands. So the specific aim of this work can also be seen in giving an approach to a generic base for the classical compositional rules to enable e.g. more compact optimization criteria in accordant composition algorithms.

As interactive application for the presented algorithms it is planned to use them in masks made of concrete which contain embedded systems with audio sensors (Figure 4). They will be mounted on the campus of the Brandenburg University of Applied Sciences and will respond to sound and whistling of pedestrians nearby and to each other with accompanying music and sounds composed in realtime.

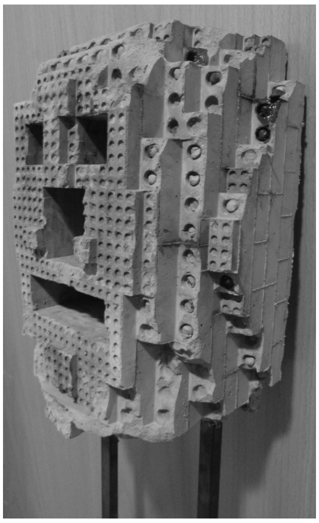

Fig. 4. Sounding masks project.

References

1. Acevedo, A.G.: Fugue Composition with Counterpoint Melody Generation Using Genetic Algorithms. In: Wiil, U.K. (ed.) Computer Music Modeling and Retrieval. LNCS, vol. 3310, pp. 96–106. Springer, Heidelberg (2004)
2. Dixon, S., Mauch, M., Anglade, A.: Probabilistic and Logic-Based Modelling of Harmony. In: Ystad, S., Aramaki, M., Kronland-Martinet, R., Jensen, K. (eds.) Exploring Music Content. LNCS, vol. 6684, pp. 1–19. Springer, Heidelberg (2011)
3. Horner, A.: Evolution in Digital Audio Technology. In: Miranda, E.R., Biles, J.A. (eds.) Evolutionary Computer Music, pp. 52–73. Springer, London (2007)
4. Kurth, E.: Grundlagen des Linearen Kontrapunkts - Bachs melodische Polyphonie: Die unterdominantische Entwicklungstendenz der Dur-Tonalitt, pp. 77–80. Krompholz, Bern (1916)
5. Ploeger, R.: Studien zur systematischen Musiktheorie: Zum Problem Monismus - Dualismus, pp. 69–107. Eutin, Nordstedt (2002)
6. Saranti, A., Eckel, G., Pirro, D.: Quantum Harmonic Oscillator Sonification. In: Ystad, S., Aramaki, M., Kronland-Martinet, R., Jensen, K. (eds.) Auditory Display. LNCS, vol. 5954, pp. 184–201. Springer, Heidelberg (2009)
7. Schneider, A., Frieler, K.: Perception of Harmonic and Inharmonic Sounds: Results from Ear Models. In: Ystad, S., Kronland-Martinet, R., Jensen, K. (eds.) Computer Music Modeling and Retrieval. LNCS, vol. 5493, pp. 18–44. Springer, Heidelberg (2008)
8. Sciabica, J-F., Bezat, M-C., Roussaire, V., Kronland-Martinet, R., Ystad, S.: Towards Timbre Modeling of Sounds Inside Accelerating Cars. In: Ystad, S., Aramaki, M., Kronland-Martinet, R., Jensen, K. (eds.) Auditory Display. LNCS, vol. 5954, pp. 377–391. Springer, Heidelberg (2009)