HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Darstellung der aktuellen Position in einem Musikstück

Studienarbeit Informatik

von

Christoph Hüsler, Olivia Müller

Betreuer:
Joseph Joller

Rapperswil, 17. Dezember 2014

Zusammenfassung

Die Noten im richtigen Moment umzublättern ist eine schwierige Aufgabe, wenn beide Hände mit dem Instrument beschäftigt sind. Der in dieser Arbeit beschriebene Algorithmus ermöglicht das automatische Ermitteln der aktuellen Position anhand von Mikrofon-Aufnahmen.

Die Analyse wird auf der Grundlage von gesampelten Audio-Daten durchgeführt. Diese werden in kurze Timeslots unterteilt und pro Slot ein Hash-Wert gebildet. Die gleiche Analyse wird auch mit dem vom Mikrofon aufgenommenen Datenstrom gemacht. Die Hashes der Grundlage und des Mikrofoninputs werden nun laufend verglichen. Durch Ermitteln der Position mit der minimalen Abweichung wird die aktuelle Position im Stück bestimmt.

Das Resultat wird in einer digitalen Partitur durch Einfärben der Noten angezeigt. Nach einer kurzen Aufwärm-Phase kann die Position auf einen Takt genau angezeigt werden.

Die Applikation wurde mit Java 8/Swing umgesetzt und verwendet eine leicht erweiterte Version der Bibliothek Zong! zur Darstellung der Noten.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

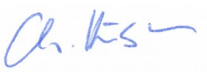
Erklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- das wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Ort, Datum: Rapperswil, 17.12.2014

Name, Unterschrift:



Ch. Hüsler

O. Müller

Inhaltsverzeichnis

1. Danksagung

- Als erstes möchten wir uns bei Herr Joller bedanken, für das Betreuen des Projektes sowie die Stifte und den Zeichenblock.
- Vielen Dank auch an Andreas Wenger, der viel Herzblut in die Zong!-Library steckt und dafür einen excellenten Support bietet.
- Schlussendlich noch ein ganz herzlicher Dank von Olivia an Daniel Meier, der mich in die Tonhalle mitnahm und mich auf die Idee brachte, sowie Boris Wechner, der mich ermutigte diese auch umzusetzten.

2. Aufgabenstellung

2.1. Betreuer

Josef Joller Dozent für Informatik, Hochschule für Technik Rapperswil

2.2. Ausgangslage, Problembeschreibung

Das Umblättern von Noten, kann sich je nach dem wie die Noten gedruckt wurden, als schwierig erweisen. Um dies zu automatisieren wird ein Algorithmus benötigt, welcher die aktuelle Position des Musikers erkennt.

2.3. Aufgabenstellung

Aufgabe dieser Studienarbeit ist es, anhand vorhandener Noten (als Bilddatei vorliegend) und einem einfachen Mikrophoninput zu erkennen, wo im Stück sich der Musiker befindet. Die gefundene Lösung soll anschliessend visualisiert werden. Es sollen möglichts schon Vorhandene Ansätze und Lösungen verwendet und kombiniert werden.

3. Ausgangslage

Die Idee für diese Studienarbeit (SA) entstand während einer Kammermusik-Matinée in der Tonhalle in Zürich. Ein Trio in G-Dur mit einer Cellistin, einem Violonisten und einem Pianisten spielte Debussy.

Hinter dem Pianist sass ein junger Herr, ebenfalls elegant gekleidert wie die Musiker. Er lass die Noten von welchen der Pianist spielte mit und erhob sich jedes mal, wenn der Musiker sich dem Ende der Seite näherte. Er beugte sich vor, hielt mit der einen Hand sein Jaquet zurück und blätterte mit der andern exakt in dem Moment um, in dem die Pianist das Ende der Seite erreicht hatte. Eine solche Hilfsperson wird gemeinhin als "Blattlaus" bezeichnet.

Bisher gibt es wenige automatisierte Lösungen für automatisches Umblättern und noch weniger davon sind brauchbar. Das Ziel dieser Arbeit war, eine bessere Lösung für eine automatische "Blattlaus" zu finden.

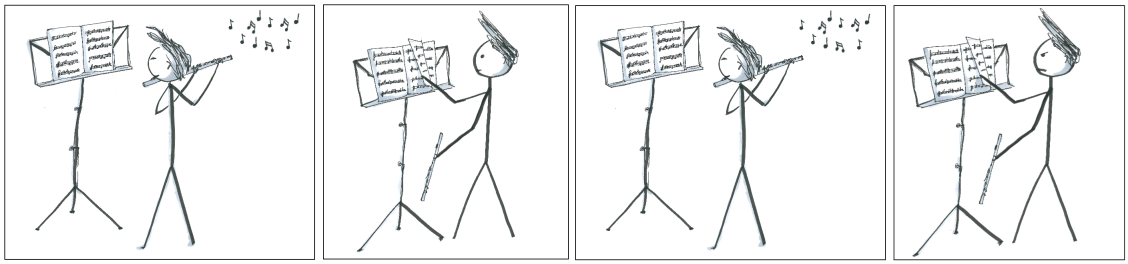


Abbildung 3.1.: Umblättern von Hand

4. Lösungskonzept

4.1. Grundlage

Die Grundidee für den Algorithmus stammt aus einem Paper [1], welches beschreibt wie Musikstücke anhand von kurzen Audio-Aufnahmen identifiziert werden können. Dazu wird eine kurze Aufnahme erstellt und in eine Serie von Hash-Werten umgewandelt. Eine Suche in einer Datenbank mit Musik-Stücken liefert dann Informationen zu den besten Treffern.

Die Bildung der Hashes geschieht wie folgt:

1. Die Aufnahme wird in kurze, ca. 100ms lange Abschnitte unterteilt
2. Die einzelnen Abschnitte werden per Fast Fourier Transformation (FFT) in den Frequenz-Raum transformiert
3. In 3 Frequenz-Bereichen, z.B. 40-80Hz, 80-120Hz sowie 120-160Hz, werden die Frequenzen mit der maximalen Amplitude selektiert.
4. Der Hash wird nun wie folgt gebildet:

$$hash(a, b, c) = (a \ << \ 16) \mid (b \ << \ 8) \mid c$$

   wobei << der Shift Left- und | der Bitwise-OR-Operator sind.

Die Suche in der Datenbank verlangt keine perfekten Matches, sondern lediglich eine ähnliche Abfolge von Werten. Dies sowie die Art der Hash-Bildung resultieren in erstaunlicher Resilienz gegenüber schlechter Mikrofon-Qualität, Hintergrundgeräuschen, etc.

4.2. Alternativen

Als Alternative zum beschriebenen, FFT-basierten Ansatz prüften wir eine auf MIDI-Input basierende Version.

Es bestehen bereits Produkte, welche Mikrofon-Input in MIDI-Sequenzen umwandeln. Diese kommen jeweils mit starken Einschränkungen (nur ein Instrument, nicht alle Instrumente werden unterstützt, etc.) Da wir bereits eine uns als sinnvoll erscheinende Idee für den FFT-Ansatz hatten, entschieden wir uns, die MIDI-basierte Version nicht weiter zu verfolgen.

4.3. Anpassung

Anstatt herauszufinden, welches Musikstück gespielt wird, soll nun die Position innerhalb eines bekannten Stückes gefunden werden. Wir entschieden uns, die Noten als Grundlage für den Vergleich zu verwenden. Die Erkennung von Musiknoten (Optical Music Recognition, OMR) kann nur als nicht-trivial bezeichnet werden, daher gehen wir davon aus, dass dieser Schritt bereits durchgeführt wurde [1] und die Noten bereits in digitaler Form vorliegen.

Unser Ansatz ist, den oben beschriebenen Prozess zu verwenden. Anstatt jedoch nach dem Hash in einer Datenbank von Stücken zu suchen, verwenden wir die Sequenz der letzten k Hashes,

[1] z.B. mithilfe von Audiveris (`https://audiveris.kenai.com/`)

um die aktuelle Position festzustellen. Wenn Abweichungen festgestellt werden wird die Position von neuem berechnet.

Ein echtes Instrument wird leicht anders klingen sowie in einem anderen Tempo als unser MIDI-Output sein. Der Unterschied im Klang wird vom oben beschriebenen Algorithmus abgefangen, Tempo-Unterschiede stellen jedoch ein Problem dar.

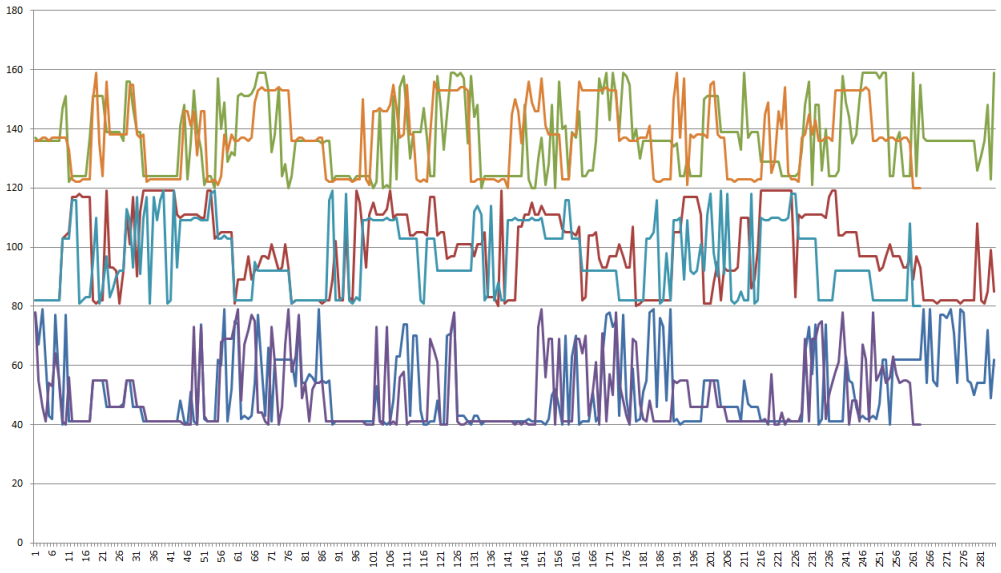


Abbildung 4.1.: Bereiche 40-80Hz, 60-100Hz, 80-120Hz

In Abbildung 4.1 wurde mit zwei unterschiedlich schnellen Aufnahemen getestet. Es ist ersichtlich, dass die Frequenzen der Aufnahme und die Frequenzen des Originals zu Beginn des Stückes relativ gut aufeinander passen. Gegen Ende jedoch kann wegen der Zeitverzögerung keine Ähnlichkeit mehr gefunden werden.

Da laufend neue Samples mit dem ganzen Stück verglichen werden, werden geringe Unterschiede im Tempo korrigiert. Ist der Tempounterschied jedoch zu gross, werden an der gleichen Stelle, in Aufnahme und Original, unterschiedliche Hash's gebildet und das Matching wird ungenauer.

5. Implementation

Als ersten Schritt implementierten wir den ersten Teil des in Abschnitt 4.1 beschriebenen Algorithmus. Die folgenden Bilder sind am Beispiel des bekannte Liedes “Alle Vögel sind schon da”.

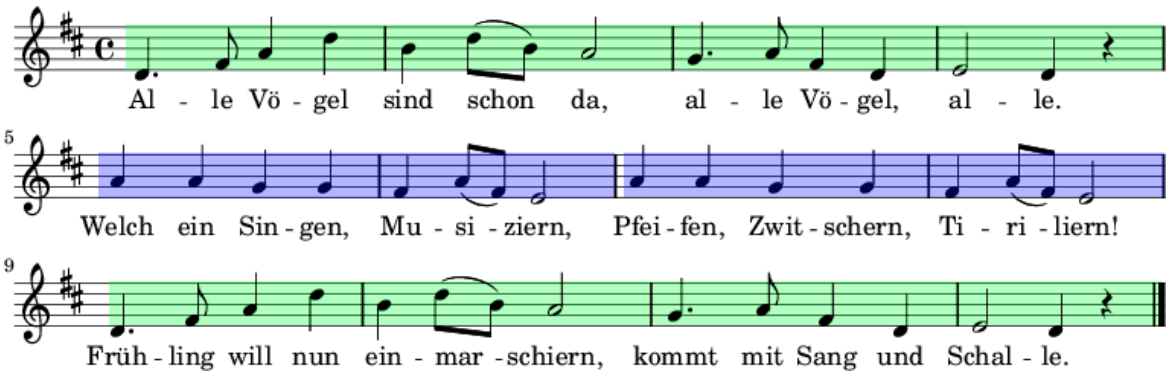


Abbildung 5.1.: Alle Vögel sind schon da

Die eingefärbten Bereiche kennzeichnen Wiederholungen des selben Themas.

5.1. Auswahl der Bereiche

Um herauszufinden, welche Bereiche am hilfreichsten zur Positionsfindung sind führten wir einige Experimente durch.

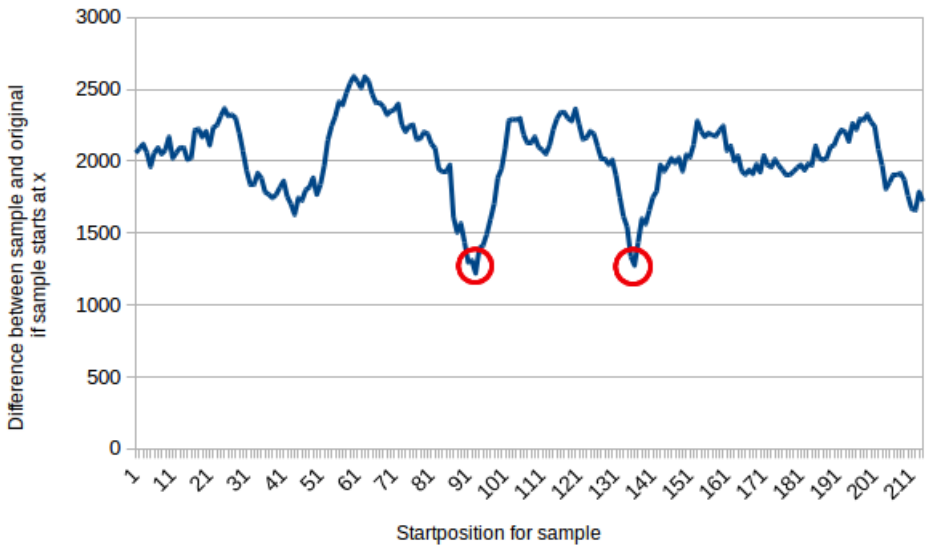


Abbildung 5.2.: Differenz zwischen Sample und Original wenn das Sample an Position x platziert wird. Bereiche 40-80Hz, 80-120Hz, 120-160Hz.

Um diese Graphen zu erstellen wählten wir ein Sample aus einer Harfen-Aufname des Lieds

ausgewählt (60 Abschnitte, ab Position 90). Danach wurde der oben beschriebene Algorithmus auf das Sample sowie das ganze Stück (konvertiert von einer MIDI-Version) wiederholt mit verschiedenen Parametern angewendet und die Differenz zwischen Sample und Original berechnet. Die Minima im Graphen sind die Match-Kandidaten. Zu beachten ist an dieser Stelle, dass das Test-Stück aus mehreren wiederholten Teilen besteht.

Um zu untersuchen, ob die Grenzen zwischen den einzelnen Bereichen ein Problem darstellen wurden auch Überlappungen verschiedener Grösse untersucht.

Folgende Parameter wurden untersucht:

- Ohne Überlappung, 40-160Hz, 80-200Hz, 200-320Hz, 320-440Hz
- 5Hz Überlappung, 40-150Hz, 80-190Hz
- 10Hz Überlappung, 40-140Hz, 80-180Hz
- 20Hz Überlappung, 40-120Hz, 80-160Hz, 160-240Hz

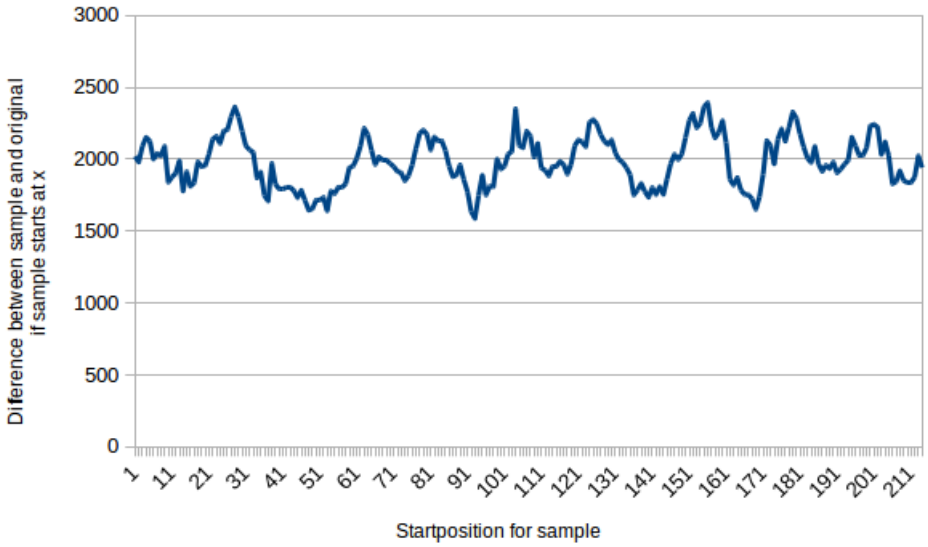


Abbildung 5.3.: Bereiche 40-80Hz, 60-100Hz, 80-120Hz

Die klarsten Resultate waren beim Bereich 40-160Hz ohne Überlappung festzustellen. Im weiteren wird daher dieser Bereich verwendet.

5.2. Sample-Länge zur Positionsbestimmung

Als nächstes wurde getestet, wie viele 100ms-Abschnitte notwendig sind, um eine brauchbare Genauigkeit bei der Positionsfindung erreichen zu können. Dazu wurden für eine gegebene Sample-Länge je 200 zufällige Samples ausgewählt und ihre Position im Stück gesucht. Anschliessend wurde verglichen, wie genau die Resultate waren.

Länge des Samples	Mittlerer Fehler	Stddev des Fehlers
10	5.19	11.98
20	8.99	8.62
30	8.39	5.15
40	7.74	5.55
50	7.69	5.38

Bei dem von uns verwendeten Tempo entspricht ein Takt einer Länge von 22 Abschnitten. Bereits bei kleinen Sample-Längen kann also eine gute Genauigkeit erreicht werden.

5.3. Serien

Die bisher beschriebene Positionsfindung funktioniert gut für einzelne Samples, kann jedoch noch verbessert werden. Schwachpunkte sind z.B. die Erkennung von Neuanfängen, Erkennung der aktuellen Position wenn Abschnitte mehrfach vorkommen, Ausreisser bei der Erkennung, etc.

Für jedes Sample liefert die Positionsfindung eine Serie von Positionskandidaten - die lokalen Minima im Graphen 5.2. Die erwarteten Positionen von aufeinanderfolgenden Samples unterscheiden sich jeweils um eine Sample-Länge voneinander.

5.3.1. Serienfindung

Die verbesserte Positionsfindung sucht nach einer kontinuierlichen Serie von Positionen in den Top-Kandidaten aus den vorhergehenden Schritten. Dazu interpretieren wir die Positionskandidaten als Knoten in einem gerichteten Graphen, wo jeder Kandidat in Schritt i mit jedem Kandidaten in Schritt $i+1$ verbunden ist. Die Kanten sind mit der Abweichung von der erwarteten Position gewichtet. Anschliessend verwenden wir einen von Shortest Path First inspirierten Algorithmus, um den Pfad mit der kleinsten Abweichung zu finden.

Falls bereits eine Serie besteht wird der Algorithmus nur auf die neuen Daten angewendet, mit der bereits bekannten Position als Startpunkt.

5.3.2. Erwartete Position

Auf Basis der ausgewählten Serie wird eine erwartete Position berechnet. Um Ausreisser in den Daten abzufangen wird diese Position aus den letzten 5 Positionen berechnet, wobei k die Länge der Serie und i die Position innerhalb der Serie ist:

$$expectedPosition = avg(match_i.position + (k - i)\ \Delta position))$$

5.3.3. Abweichung

Falls der Musiker abbricht und neu anfängt, z.B. beim Einüben eines neuen Stückes, muss der Algorithmus detektieren, dass die alte Serie nicht mehr gültig ist. Zu diesem Zweck berechnen wir die Abweichung der selektierten Position von der erwarteten Position. Falls die durchschnittliche Abweichung der Serie grösser wird als ein Schwellwert wird die alte Serie verworfen und eine neue gesucht.

Listing 5.1: Beispiel-Output des Serien-Matchers

```
1  nextMatch - Missing data, collecting: [152, 163, 381, 312, 0]
2  nextMatch - Missing data, collecting: [169, 90, 413, 324]
3  nextMatch - Missing data, collecting: [190, 114, 395, 445]
4  nextMatch - Missing data, collecting: [212, 195, 443, 410]
5  nextMatch - Missing data, collecting: [233, 231, 516, 619]
6  findAndUpdateSeries - Retroactively selected [152, 169, 190, 212, 233]
7  nextMatch - MatchInformation [matches=[261, 271, 463, 607], position=261, uncertainty
       =0.1818], expected position: 257
8  nextMatch - MatchInformation [matches=[283, 220, 368, 449], position=283, uncertainty
       =0.1818], expected position: 279
9  nextMatch - MatchInformation [matches=[303, 242, 541, 644], position=303, uncertainty
       =0.0455], expected position: 302
10 nextMatch - MatchInformation [matches=[329, 298, 431, 631], position=329, uncertainty
       =0.2277], expected position: 324
11 nextMatch - MatchInformation [matches=[350, 271, 527, 530], position=350, uncertainty
       =0.0909], expected position: 348
12 nextMatch - MatchInformation [matches=[369, 300, 422, 651], position=369, uncertainty
       =0.0909], expected position: 371
```

```
13 nextMatch - MatchInformation [matches=[393, 352, 533, 550], position=393, uncertainty
       =0.0], expected position: 393
14 nextMatch - MatchInformation [matches=[417, 351, 492, 552], position=417, uncertainty
       =0.0909], expected position: 415
```

5.4. Visualisierung

Um den Algorithus zu veranschaulichen, dient eine kleine Demoapplikation. Grundlage der Visualisierung ist ein MusicXml [1] File der Partitur. Dies kann genau wie die Grundlage für den Vergleich mittels Optical Music Recognition (OMR) aus einem Bild erstellt werden. Die Visualisierung der Position erfolgt durch Einfärbung der Noten im aktuellen Takt und dem vorhandenen MusicXml File.

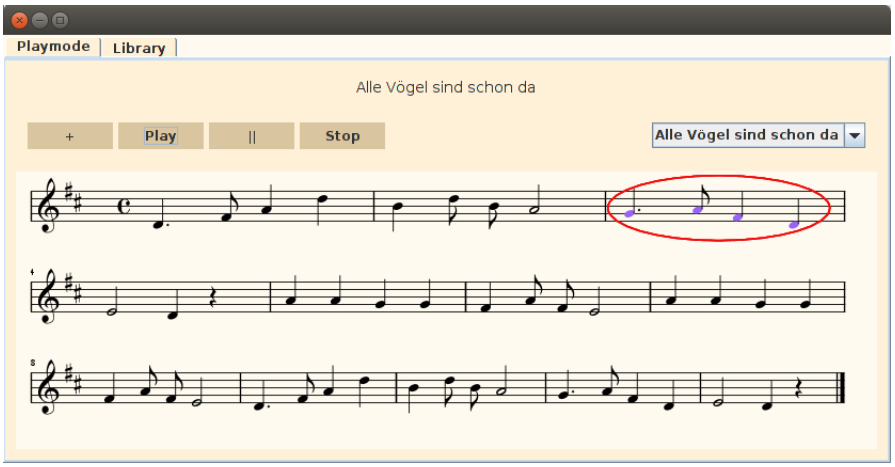


Abbildung 5.4.: Applikation mit markierter Position

Zur Darstellung der Noten verwenden wir die Library Zong! und Java/Swing. Da Zong! zwar bereits Support für eingefärbte Elemente hatte, die Farb-Information aber noch nicht aus dem MusicXML herauslas, mussten wir eine kleine Anpassung vornehmen.

[1] MusicXml ist ein beschreibendes Datenformat zum Austausch von Musiknoten (Wikiepedia)

6. Ergebnisse

Der Algorithmus funktioniert mit guter Genauigkeit unter folgenden Bedingungen:

- Die Tempi von Vergleichs-Stück und Mikrofon-Input unterscheiden sich nicht zu stark. Dies kann behoben werden, indem das Tempo beim Sampling der Midi/MusicXML-Daten angepasst wird.
- Wir erwarten, dass der Algorithmus bei einem bis wenigen Instrumenten besser funktioniert, als bei vielen. Insbesondere gehen wir davon aus, dass Orchester-Situationen zu unbrauchbaren Ergebnissen führen, haben aber keine Tests dazu durchgeführt.

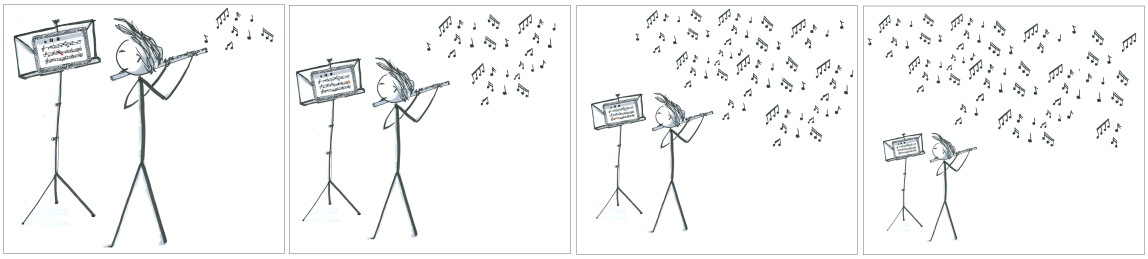


Abbildung 6.1.: Musizieren am Tablet

A. Software

Audiveris `https://audiveris.kenai.com/`

Optical Music Recognition Software, analysiert ein Bild/Foto der Noten und liefert MusicXML

JTransforms `https://sites.google.com/site/piotrwendykier/software/jtransforms`

FFT-Implementation

MuseScore `http://musescore.org/`

MusicXML-Bearbeitung, MIDI-Sampling

MigLayout `http://www.miglayout.com/`

LayoutManager für Swing und andere Frameworks

Zong `http://www.zong-music.com/`

MusicXML-Darstellung

Literaturverzeichnis

[1] Avery Wang et al. An industrial strength audio search algorithm., 2003.