BUSII
Final Report

Sonja Biedermann,
01402891

14. Januar 2019

1 Objective

Our objective was to implement a tool that could be used like something akin to a debugger for the machining process data. It should be possible to step through time and see which tools were used when, where the tool was in space, and the readings of the various sensors (e.g. load, torque).

Further, it should be possible to label sections of time where something unusual happened, and these labels should be able to be exported for later use with e.g. ML algorithms or other analysis tools.

2 Prototype

Our initial low-fidelity prototype is shown in Figures 1a and 1b. Figure 1c shows the main view of the high-fidelity prototype we implemented.

The first view of the low-fidelity prototype (Figure 1a) was scrapped in favor of the more compact representation of the process tree shown in Figure 2a. This was done because it became evident that rendering a tree in SVG would not scale to hundreds of processes.

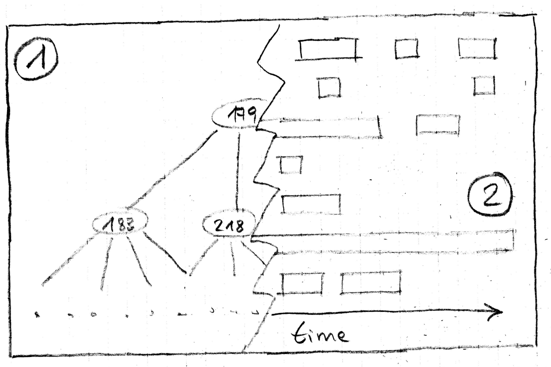
Otherwise, the final prototype is a pretty faithful realization of the second view of the low-fidelity prototype.

3 Functionality

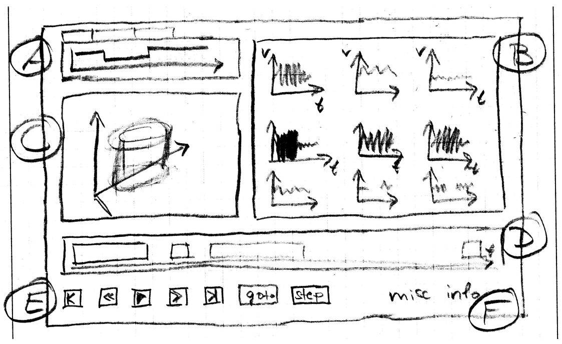
3.1 Description of Views

The most important views as far as interactivity is concerned are the activities chart (marked D in Figure 1c) and the stepping controls directly below it, as well as the selection and settings dialog which can be opened by clicking the buttons at the bottom right.

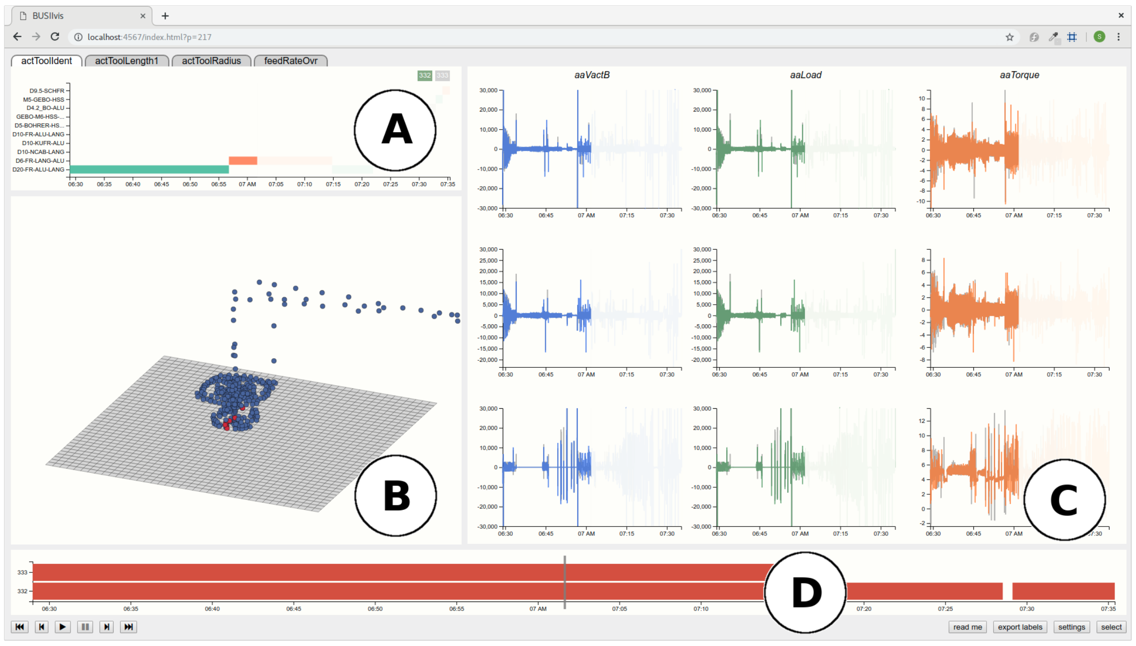
The activities chart displays the time segments in which machining data was recorded without interruptions of more than 15 seconds. It can be clicked to update the current time step, or the vertical bar can be dragged to scan along the time axis and update all



(a) Low-fidelity prototype view 1

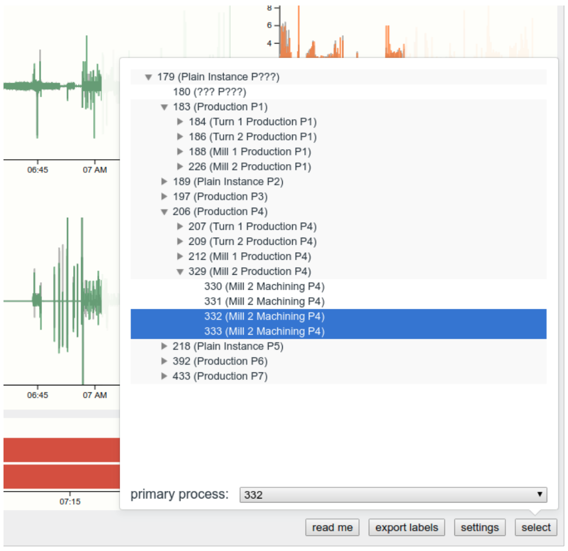


(b) Low-fidelity prototype view 2



(c) High-fidelity prototype

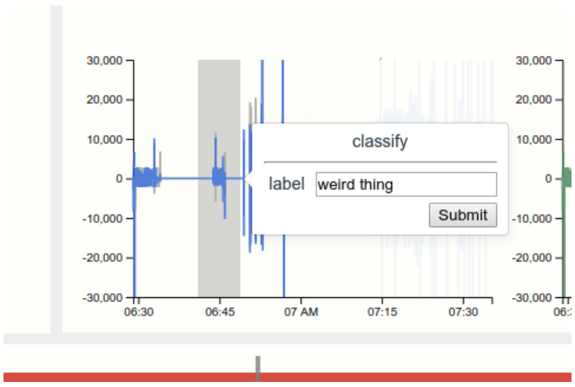
Abbildung 1: Comparison of prototypes



- ▼ 179 (Plain Instance P???)
  - 180 (??? P???)
  - ▼ 183 (Production P1)
    - ▶ 184 (Turn 1 Production P1)
    - ▶ 186 (Turn 2 Production P1)
    - ▶ 188 (Mill 1 Production P1)
    - ▶ 226 (Mill 2 Production P1)
  - ▶ 189 (Plain Instance P2)
  - ▶ 197 (Production P3)
  - ▼ 206 (Production P4)
    - ▶ 207 (Turn 1 Production P4)
    - ▶ 209 (Turn 2 Production P4)
    - ▶ 212 (Mill 1 Production P4)
    - ▼ 329 (Mill 2 Production P4)
      - 330 (Mill 2 Machining P4)
      - 331 (Mill 2 Machining P4)
      - 332 (Mill 2 Machining P4)
      - 333 (Mill 2 Machining P4)
  - ▶ 218 (Plain Instance P5)
  - ▶ 392 (Production P6)
  - ▶ 433 (Production P7)

primary process: 332

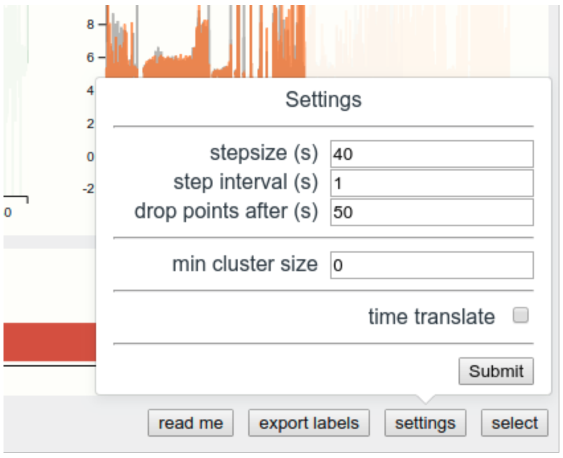
(a) Selecting processes from the tree



classify

label: weird thing

(b) Labeling a time segment



Settings

stepsize (s): 40
step interval (s): 1
drop points after (s): 50
min cluster size: 0
time translate: ☐

(c) Settings

Abbildung 2: UI elements

views correspondingly. If multiple charts are selected, they will be displayed one below the other.

The stepping controls below allow to jump to the start or the end of the recorded events as well as (dis)enabling the automatic stepping. One single manual step can also be taken. By default, the stepsize is calculated as $\frac{\text{length of process}}{100}$, i.e. a process is divided into 100 steps. This can be adjusted in the settings.

The 3D scatterplot (marked B) displays the locations of the tools during the process. The data is processed into 600 microclusters by using k-means to deal with the high volume of points in some processes. Unfortunately, SVG does not scale to thousands of elements. If more than one process are selected, the primary process will be shown in blue and the secondary proceses in red, and the secondary points will disappear after a predefined amount of time (by default 10 seconds). The microclusters appear as soon as the earliest point summarized by the cluster would have appeared.

View C shows timeseries data for the X, Y and Z axes (ordered from top to bottom). They are not preprocessed. It might be a good idea to apply PAA (piecewise aggregate approximation) to cut down on the number of points. Data points that would only appear after the current timestep are hidden behind a white overlay.

A brush over the time axis can be triggered by clicking on one of the 9 timeseries charts. Right-clicking on the brush opens a dialog allowing to label the selected segment, see Figure 2c.

Finally, view A displays miscellaneous data, like the identifiers of the tools used, their length and radius, and the feedrate override. Similar to the timeseries view, data occuring after the current timestep are hidden behind a white overlay. The bars of the tool identifier chart can be used to filter the scatterplot below to display only points made by certain tools.

If multiple processes are selected, the buttons on the top right of the tool identifier chart can be used to toggle between the processes for that chart. The button for the primary process is green to distinguish it from the others and to offer a visual cue as to which process is currently selected as the primary.

3.2 Omitted functionality

Some functionality was omitted in the final prototype. Most notably this is the integration of detecting discords using the Matrix Profile algorithm. Although this algorithm is cool and potentially easy to use, it is very unclear whether the results are meaningful—after all a time segment that is distinctly different to all others may not necessarily be anomalous in our case. Such an approach would be more suited for highly regular processes, e.g. a heart beat, where deviations are clearly anomalous.

We instead spent our time implementing the option to display multiple processes at the same time, which was suggested after the presentation of the initial protoype, as this was deemed to be a useful feature (as opposed to integrating MP, whose usefulness is uncertain).

4 Implementation

The implementation is based on the excellent SVG and DOM-manipulation library D3. We used D3-3D for rendering 3D elements (i.e. the 3D scatterplot), which was a bit of a hassle (the library is not very well documented, and if you have never dealt with 3D can be a bit puzzling, and the one single example is confusing).

We have a web server which does some basic tasks such as serving the data and modifying it, e.g. if the user wants the processes to be normalized to the same start time, it correctly alters the timestamps, or filters out clusters that have not enough points assigned. It also stores and aggregates the labelings done by the user.

The frontend itself is organized around some shared state including the current time-step, which tool is currently selected (if any), and the data retrieved from the backend. Changes triggered by the UI result in a D3 dispatcher object being triggered, which relays the data to all views that listen to it. Thus the views are decoupled from each other.

The data is gathered from the logs using a small army of preprocessing scripts. Unfortunately, this takes quite a bit of time (about an hour for all processes). Thankfully it needs to be done only once. This high time requirement stems from data being read often (every script will read it again—combining them would be smart) and some operations being plain costly depending on input size, e.g. clustering many points with k-means.

5 Challenges

The main challenge was how to keep the frontend (relatively) responsive while dealing with large amounts of data. We have made adjustments to this end preemptively, such as microclustering the position data and summarizing multiple data points into broken bars (i.e. activity chart).

Although the current implementation is not extremely fast and responsive, it is very much usable without developing anger issues. The longest wait is for the server to transfer the data (especially if time adjustments need to be made). A further step to reduce data volume would be to use PAA to summarize points of the timeseries data.

A big headache was to implement the possibility of comparing multiple processes after we had already had all views and basic functionality implemented—many assumptions which were made previously went out the window and rewriting almost all of the frontend-size data wrangling was frustrating.

The most cumbersome and time intensive task was definitely data preparation.