Beam Physics Note
TRI-BN-22-09
April 12, 2022

Element Command Automated Summary Tool

Olivier Shelbaya

TRIUMF

Abstract: Parsing through the ISAC activities log, which records timestamped I/O controller commands to the ISAC accelerator and beamline optics, it is possible to visualize tuning interventions on the apparatus, by plotting the number of commands per section against time. Additionally, discretization of the 24 h day into timebins allows for a method for automatically estimating tuning time. This tool, `ECAS`, is meant to aid in establishing a more robust quantitative metric for past, present and future tuning overhead times at TRIUMF-ISAC.

1 Introduction

This report documents software that has been developed to visualize and quantify machine tuning time at TRIUMF-ISAC. It is clear that the term *tuning* itself can potentially be interpreted in different ways, for example ”tuning to experiment” versus ”tuning for development” or training. It is acknowledged that the nature of the tuning intervention is highly dependent upon the scheduled intended use of beam at that time. For instance, tuning done for practice over an 8 hour shift may not be carried out with the same efficiency as when it is being tuned to an experiment, with a scheduled starting time. This work does not distinguish between causes of tuning and instead takes a more fundamental approach: counting the number of commands, per section, per time. The advantage of this approach is that it is robust, the disadvantage is that the aforementioned sub-structures (beam delivery vs training vs development) are averaged together and in a sense, lost.

2 Quantification of Tuning

Another issue is the relative ambiguity of defining *tuning time*. As far as the linac is concerned, it is only being tuned when commands are sent which alter its optics or other support infrastructure (vacuum, water cooling, etc..) However, from an operational standpoint, tuning time is a measure of how long a human operator must dedicate during their shift to the task of intervening with the control system to accomplish a goal, such as getting beam to experiment. Unfortunately, the quantification of human-time invested is not straightforward to reconstruct from the logs. The only programatically logged information is that found in the in/out controllers (IOCs) logfiles, located at [1], which list the process variable (PV), time, and other information such as initial and final values of the command. As an example:

```
csbioc:43563 Fri May 21 09:50:26 2021 21-May-21 09:50:19 isacepgate gateway CSB:Q14:POS:VOL.VAL new=984.1 old=934.1
```

shows a recorded command on quadrupole CSB:Q14, set from 934.1 V to 984.1 V, with the command’s time logged along with information on its source. Using `python`, the logfile’s contents to a `list[]`. In parallel, `xml2optr` has been used to create a nonsense `TRANSOPTR` file containing all of the ISAC optics, done by creating a mock beampath in the `acc/` database which includes all ISAC files. This creates a file `data.dat` with over 350 elements, each of which includes its full PV address. The `accpy` package allows for the easy reading of `data.dat` to a `datadat` class, which contains dictionary-key pairs of all these PVs. It is straightforward to then parse through the IOC logfiles and record the number and time of each command, for each distinct PV, for each beamline or accelerator section. This is represented in Figure 1.

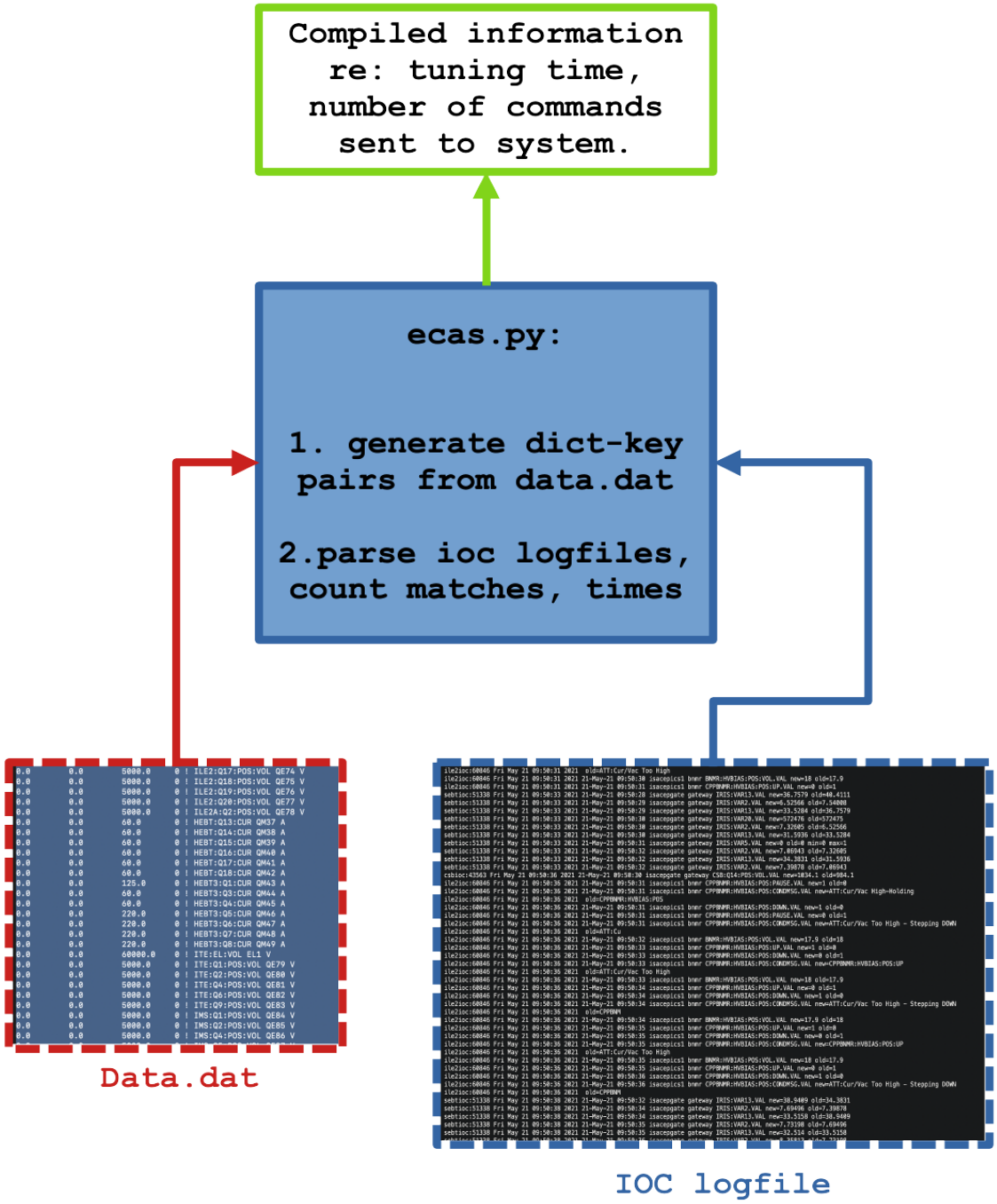

Figure 1: Representation of the structure of operations performed by `ecas.py`, using procedurally generated `data.dat` files with `xml2optr`, which include lists of PVs associated to each optics setpoint. This populates a dictionary-key pair for each element, which is then populated with recorded commands in the IOC logfile, along with the time of the recorded commands.

It is then a matter of specifying the date range, which is handled using the `python` package `pandas`:

```
start_input="2021-10-01"
end_input="2021-10-31"
[...]
daterange=pd.date_range(start=start_input,end=end_input)
[...]
for date in daterange.strftime("%Y%m%d"):
[loop through daterange, obtain IOC files, parse data.dat entries]
```

For each date in the user specified range, the IOC logfile is saved locally and read into a `list[]`:

```
url='http://isacwserv.triumf.ca/onlylocal/isacdata/log/'+str(date)+'/iocLog.isac'
r = requests.get(url,allow_redirects=False)
todayfile='ioclog-'+str(date)+'.dat'
open(todayfile,'wb').write(r.content)
print(url,tunefile)
with open(todayfile) as fileIn:
    rawData = [line.split() for line in fileIn]
```

The list `rawData` contains each line of the logfile. A loop then cycles through `data.dat`'s elements (PV names) and the lines of `rawData`:

```
datadat = DataDat(file=optr_dir+"data.dat")

#cycle through datadat
intervention={}
time={}
#intervention: a dict with keys pv_name which counts total number of key hits in logfile
#time: a dict with keys pv_name which logs an array containing the times of intervention
#for each device
for element in datadat['elements']:
    pv_name_full = element['name']
    pv_name=":".join(pv_name_full.split(":",2)[:2])

    for line in rawData:
        #print('line:',line)
        try: #crashed on empty lines, add try/except - skip/don't count if line empty..
            time_val=str(line[4])

            for item in line:
                if(pv_name in item):
```

```
                if pv_name not in intervention:
                    intervention[pv_name] = 1
                    time[pv_name] = [time_val]
                else:
                    intervention[pv_name] +=1
                    time[pv_name].append(time_val)

            else:
                if pv_name not in intervention:
                    intervention[pv_name] = 0
                    time[pv_name] = [0]
                else:
                    intervention[pv_name]+= 0
    except:
        #print("skipping empty line:",todayfile)
        pass
```

These dicts then allow for the generation of plots using `gnuplot` for robustness. Together with the package `pylatex`, `ecas.py` is capable of automatically producing `.pdf` formatted reports showing plots of the daily tuning interventions, color coded by section, shown in Figure 2. The plot shows the recorded commands on the optics elements at ISAC during the 24 hour day `2021-10-21`, showing optics setpoint changes commanded at a variety of sections in the ISAC beamline and accelerator system.

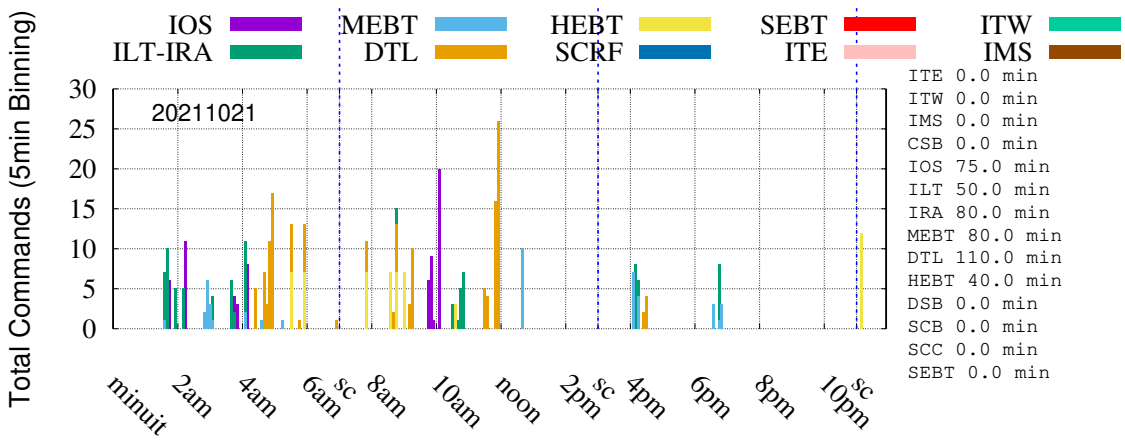

Figure 2: `ECAS` compiled tuning summary for `2021-10-21`.

3 Cumulative Tuning Time

The tuning time is quantified by discretizing each 24 hour calendar day into a fixed number of timebins, 288 in the case of Figure 2, working out to 5 min. intervals. The tuning time is computed as the number of non-empty timebins in a fixed calendar day. This granularity is intended to account for that fact that changes to the optics is procedurally done by operators, while following procedures that call for other actions such as tune documentation or safety checks.

Thus, the counting methodology draws no distinction between 100 commands sent in a 4 minute interval, versus a single command sent at one time, both will count as "five minutes of tuning", if the bin count is set to 288. Figure 3 shows the tallied total number of optics setpoint change commands recorded during `2021-06`, over a 30 day period. This is useful in visualizing the relative share of optics setpoint changes commanded in different sections, a proxy for both their complexity and their subscription.

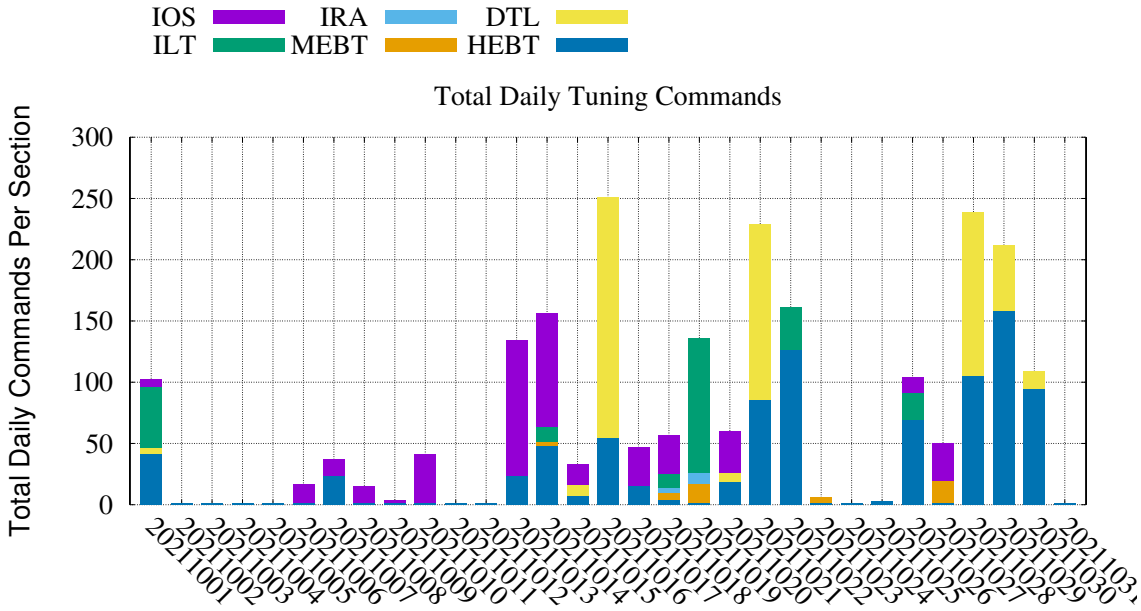

Figure 3: `ECAS` compiled record of total PV optics setpoint changes between `2021-10-01` and `2021-10-31`, color coded by section.

Finally, a stacked area plot is created by creating an output file `combined_times.dat` which contains the date in `YYYYMMDD` format, along with the section-by-section breakdown of total tallied tuning time. `gnuplot` uses this file to produce the output shown in Figure 4.

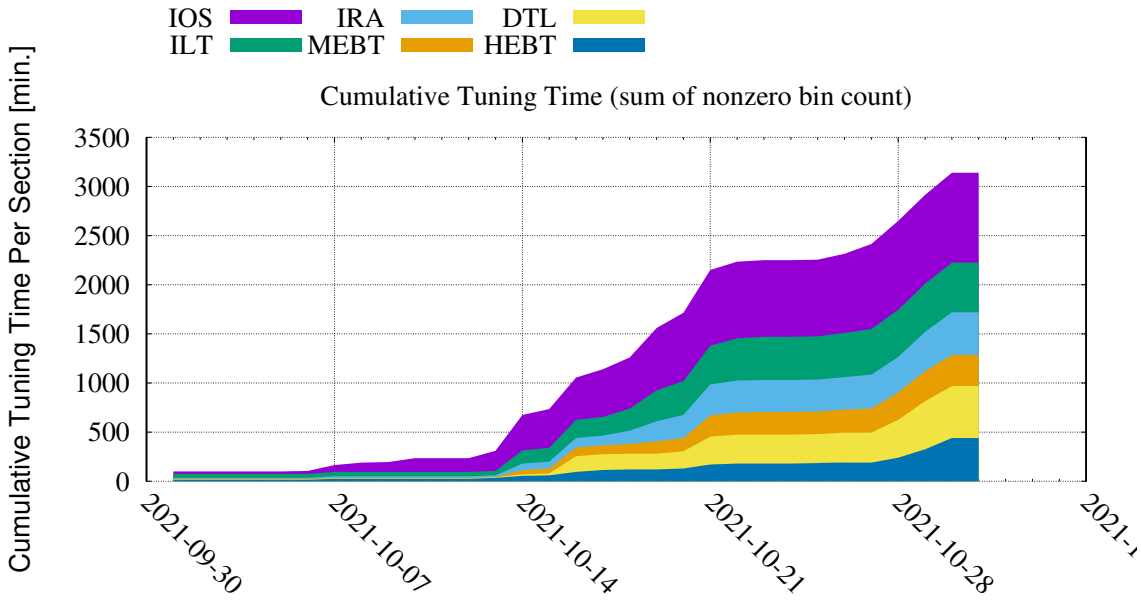

Figure 4: `ECAS` compiled record of cumulative tuning time using 5 min. timebins between `2021-10-01` and `2021-10-31`, color coded by section.

4 Conclusion

The capability presented[2] in this report shows an application of the package `accpy` and `xml2optr` to produce a quantified estimate of tuning times along with a visual display of the recorded tuning interventions. This is presently being used to develop a more robust quantitative approach to analyzing tuning overhead time at TRIUMF-ISAC.

References

[1] isacwserv.triumf.ca log file repository.

[2] gitlab.triumf.ca ECAS-development repository.