TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Tanel Lentso 206856IACB

GUI ja automatiseeritud aegridade klassifikatsioon multimodaalsete allvee andurite jaoks

Bakalaureusetöö

Juhendaja: Jeffrey Andrew Tuhtan
Vanemdotsent

Tallinn 2023

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Tanel Lentso

09.04.2023

Annotatsioon

Selle töö peamiseks eesmärgiks oli luua kasutajaliides allveeanduritest saadud andmete kuvamiseks graafilisel vormil ja nende andmete töötlemine ning eksportimine *.CSV* faililaiendi kujul *Python*i programmeerimiskeeles.

Kõige esimene samm oli luua eraldi koodilõik, mis võtaks anduri binaar algandmed .TXT failist ja dekrüpteeriks need loetavaks tekstiks .CSV faili ning samuti kasutades faili andmeid koostaks see graafiku .PDF faili.

Edasine töö toimus kõik graafilise liidese arendamises, mis siis võttis eelnimetatud .CSV failist kõik võimalikud andmed ning kasutades rõhu, aja ja kiirenduse andmepunkte, loos see graafiku kus siis x-teljel oli ajahetk sekundites ning y-teljel rõhk millibaarides. Teine suurim funktsioon on huvipakkuvates piirkondades uute punktide lisamine ning nende eksportimine uute .CSV faili, kus siis on kõik võimalikud andmed graafikule paigutatud ajapunktidest või näiteks saab eksportida kõik andmed kahe sellise punkti vahel. Selliseid eksporditud lõike saab kokku panna üheks suureks grupiks eraldi kausta ja selle grupi programmiga avades saab luua *confidence graph*i, kus on kujutatud peale originaal faili andmete ka kõikidest failidest saadud iga ajapunkti rõhu keskmised ning seda ümbritseb standardhälvega arvutatud rõhu ülem ja madal piirid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 18 leheküljel, 2 peatükki ja 19 joonist.

Abstract

GUI and automated time series classification for multimodal underwater sensors

The aim of this work was to create an interactive graphical user interface in the Python programming language for graphing and working on data that came from underwater sensors. This user interface was needed to make the work of researchers working on this data easier as prior to this all the needed graphing was done using slow *Matlab* scripts with no user interface and with Microsoft's Excel program.

The first thing I worked on was a separate script that could take the sensors' raw binary data from .TXT files and convert them to formatted .CSV files and use the file's data to make a graph into a .PDF file. After that I was only working on the main program with which you could graph and work on the previously made .CSV files. I slowly added more and more functionalities to the program starting with just graphing the data to a plot on which the x-axis was for time in seconds and the y-axis for pressure in millibars.

After that I made it possible to create new custom points that could have any name and placed onto the graph. This made it possible for the researchers to mark regions of interest and with another functionality they could export all the data from the time on which the points were placed to a new .CSV file for further analysis by them. It is also possible to have the graph be cropped between two custom points and then all the raw data between those two points could be exported for later use in confidence graphs.

Confidence graphing was what I worked on last. The user can choose any folder they want and from that all the .CSV files that are in the correct format are used to create a 95% confidence graph. They create a way to easily find mistakes in the sensors' data and also to compare two different types of sensors used. Data came from sensors that were in cylindrical shaped objects, but there was also data that was gotten with sensors on live fish. The confidence graphs make it easy to then compare these two.

The thesis is in Estonian and contains 18 pages of text, 2 chapters, and 19 figures.

Lühendite ja mõistete sõnastik

GUI	graafiline kasutajaliides
Confidence Graph	graafik, mis näitab kui suure tõenäosusega tegelik väärtus esineb seatud vahemikus
Sprint	ajaliselt piiratud periood määratud töömahu täitmiseks
Python	programmeerimiskeel
Süstimispunkt	etteantud töö puhul on see punkt graafikul, kus rõhk hakkab üle teatud piiri tõusma. Inglise keeles *injection point*
Kõvakodeering	fikseeritud andmed või parameetrid, mida ilma koodi muutmata ei saa muuta

Sisukord

Jooniste loetelu

1 Sissejuhatus

Töö projekti kallal algas juhendajaga läbi rääkimisega, et mis vajab tegemist ja kuidas. Otsustasime töötada agiilselt ehk olid 2-nädalased *sprindid*, mille järel tegime uue koosoleku tehtud töö ette näitamiseks tagasiside jaoks ning uute ülesannete saamiseks.

Ülesanded, mis algselt anti, olid kirjutada kaks väiksemat koodi lõiku *Python*is kahe erineva sensori tüübi binaarfailide dekrüpteerimiseks ja eksportimiseks vormindatud .CSV faili ning ka .PDF fail, kus oli kaks graafikut dekrüpteeritud andmetest. Peale neid tuli ainult töötada peaprogrammi kallal, mis siis algse plaaniga pidi võimaldama järgnevaid tegevusi - .CSV failide avamine, failidest saadud andmete järgi rõhu ja kiirenduse suurusjärgu näitamine graafikul, automaatne süstimispunkti leidmine graafikul ja selle märgistamine, graafiku kärpimine kasutaja poolt soovitud kohtades ning lõpuks veel lokaalne andmebaas soovitud graafikute kiireks salvestamiseks ja hiljem edasiseks töötlemiseks hoidmine.

Alustasin siis kahe skripti kirjutamisega, mis siis pidid võtma allveeandurite binaar .TXT failid ja dekrüpteerima loetavasse .CSV faili. Juhendaja andis abiks mulle *MatLab* skripti, milles kood juba tegigi seda, mis mul vaja oli teha *Python*is. Probleem juhendaja antud skriptiga oli see, et see korraga suutis ainult ühte faili korraga töödelda ja see võttis kaua aega selle tegemiseks (võis kuni minut aega minna ühe faili jaoks). Selle skripti seest sain kuigi näha, et missugusel kujul andmeid hoiti binaarfailis, ehk et mitu täisarvu on või mitu ujukoma arvud ja nende järjekorda. Lõpuks valmisid kaks skripti, mis siis suutsid terve kausta faile korraga töödelda ja iga fail võttis ainult paar sekundit.

Pärast neid kahte skripti oli kogu mu aeg suunatud põhiprogrammi kallal töötamisega. Esimese asjana töötasin lihtsa akna valmistamisega, kus siis oli koht graafiku jaoks ja nupp faili valimiseks. Sealt edasi kohe lisasin funktsiooni manuaalselt seada, et mis vahemikus graafikut näidatakse, näiteks kui andmete pikkus on 100 sekundit, siis said valida et näeks ainult 30 kuni 80 sekundi vahemikus graafikut. See funktsioon sai kuigi

hiljem eemaldatud ja asendatud natuke teise süsteemiga, aga kärpimise funktsioonid jäid samaks. Sealt edasi sai tehtud üks tähtsamaid funktsioone, mis on siis automaatne süstimispunkti tuvastus. Seda sai tehtud, nii et võtsin ette kõik rõhu andmed ja võrdlesin hoitavat rõhku sellele järgneva rõhuga. Kui kahe andme väärtuste vahe oli piisavalt suur, siis sai hoitava koha ajapunkti väärtus kasutajaliideses märgitud ning samuti tekkis punkt graafikule.

Niimoodi samm sammu haaval said kõik põhifunktsioonid täidetud ning veel ka vahepealsed väiksemad tööd, mis aitasid programmi korralikult kokku tuua. Viimased kaks nädalat projekti koostamises läks koodi puhastamiseks, kasutute funktsioonide kustutamiseks ja paari väiksema asja lisamiseks ning muutmiseks.

2 Skriptide kirjutamine

Alustuseks juhendaja andis mulle *Matlab*i skriptid, mis tegid seda tööd mis mul *Python*is oli vaja kirjutada ning kust lugesin välja, milline minu kood peaks umbes välja nägema. Kõige tähtsam, mis skriptist välja lugesin, oli see kuidas andmed olid hoitud ja mille kaudu sain erinevate arvu tüüpide järjekorra ning arvutasin välja mitu bitti üks rida pikk on [1]. Kasutades neid andmeid hakkasin pihta esialgse koodi kirjutamisega. Alustuseks võttis kood ainult ühe faili sisse ja printis konsooli dekrüpteeritud andmed, et ma saaks kontrollida nende õigsust.

2.1 CSV failide genereerimine

Pärast dekrüpteerimise osa tuli edasi failist saadud andmed salvestada .CSV faili. Enne salvestamist tuli lisada salvestatavasse faili kõige esimesele reale andmete tüübid tekstina (nagu aeg ja rõhud), mis olid mulle juhendaja poolt ette antud näidisfailides. Andmed ise olid kõik komaga eraldatud, et Exceli programm saaks kõik andmed eraldi oma tulpa panna loetavuse tõstmiseks. Jooniselt 1 on näha mõned read sellest, milline .CSV fail välja nägi. Salvestamiseks sain abi internetist leitud juhendiga [2].

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Time [s]	PL [hPa]	TL [C]	PC [hPa]	TC [C]	PR [hPa]	TR [C]	AX [m/s2]	AY [m/s2]	AZ [m/s2]	RX [rad/s]	RY [rad/s]	RZ [rad/s]	CSM	CSA	CSR	CSTOT
2	0	1000.02	24.8	999.9	25	1000	24.45	5.34	-2.26	6.93	-0.25333	-0.45333	-0.40889	0	0	0	0
3	0.01	999.93	24.8	999.94	25	1000.07	24.45	5.34	-2.33	6.91	-0.29111	-0.34667	-0.49333	0	0	0	0
4	0.02	1000.03	24.8	1000.02	25	999.92	24.46	5.38	-2.35	6.8	-0.15333	-0.14222	-0.4	0	0	0	0
5	0.03	1000.12	24.8	999.91	25	1000.11	24.46	5.51	-2.33	6.8	-0.06889	-0.49778	-0.40222	0	0	0	0
6	0.041	999.9	24.8	999.93	25	1000.07	24.46	5.74	-2.37	6.87	-0.26667	-0.32889	-0.44	0	0	0	0
7	0.051	999.89	24.8	999.99	25	999.98	24.46	5.78	-2.47	6.82	-0.11111	-0.88889	-0.56222	0	0	0	0
8	0.061	999.89	24.8	999.97	25	1000.07	24.46	5.88	-2.43	6.87	-0.03333	-0.46222	-0.39778	0	0	0	0
9	0.072	1000.03	24.8	999.91	25	1000.05	24.46	5.88	-2.35	6.99	-0.19556	-0.23556	-0.49556	0	0	0	0
10	0.082	999.92	24.8	999.99	25	1000.11	24.46	5.74	-2.35	7.08	-0.15111	-0.27556	-0.26667	0	0	0	0
11	0.092	1000.02	24.8	1000.06	25	1000.06	24.46	5.67	-2.45	7.05	-0.23556	-0.23333	-0.40444	0	0	0	0
12	0.103	1000.14	24.8	999.93	25	1000.11	24.46	5.72	-2.45	6.99	-0.20889	-0.28222	-0.32	0	0	0	0
13	0.113	999.99	24.8	999.88	25	1000.13	24.46	5.8	-2.47	6.82	-0.27111	-0.32444	-0.25556	0	0	0	0
14	0.123	1000.01	24.8	999.98	25	1000.07	24.46	5.92	-2.6	6.78	-0.29111	-0.30667	-0.32222	0	0	0	0

Joonis 1. Eksporditud .CSV fail

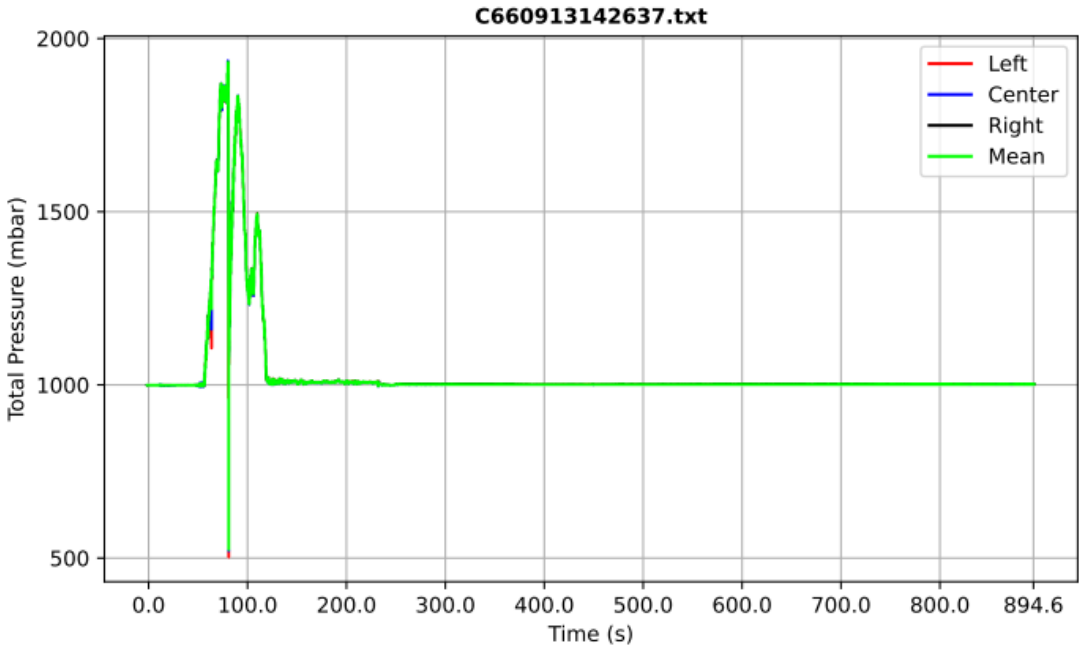
2.2 Terve kausta töötlemine ja kaustade loomine

Järgmiseks tegin võimalikuks lisada terve kausta töötlemiseks korraga ühe faili asemel. Kood otsis üles kõik .TXT failid kaustast ning hakkas neid ühekaupa töötlema. Kui mõne failiga pidi ole probleem, et see ei ole õiges formaadis dekrüpteerimiseks, siis kood läks sellest lihtsalt üle ja andis konsoolis teada tõrkest. Samuti kood genereeris uued kaustad

failide hoiustamiseks - “CSV”, “PDF” ja “TXT”, kuhu siis vastavalt failid viidi pärast valmistamist. “TXT” kausta liigutati algsed failid ning “PDF” kaust oli selles ajahetkes veel tühi.

2.3 PDF failide genereerimine

Viimase asjana tuli veel lisada .PDF faili genereerimine, kus siis oli kaks graafikut - rõhu ja kiirenduse suurusjärgu graafik. See oli ka kõige raskem osa tööst, kuna mul polnud varasemat kogemust graafikute genereerimisega ega .PDF-ga töötamisega. Graafikute genereerimiseks kasutasin *Python*i *matplotlib* teeki ja alustasin rõhu graafikuga esimesena töötamist. Faili andmetes oli kolm erinevat rõhku ja need oli kõik vaja graafikule lisada ning ka nende keskmised igas ajapunktis. X- ja y-teljel oli vastavalt aeg sekundites ning rõhk millibaarides. Selle graafiku tiitel oli ka originaal faili nimi. Joonisel 2 on näha üks näide, kuidas päris andmetega rõhu graafik saaks välja näha.

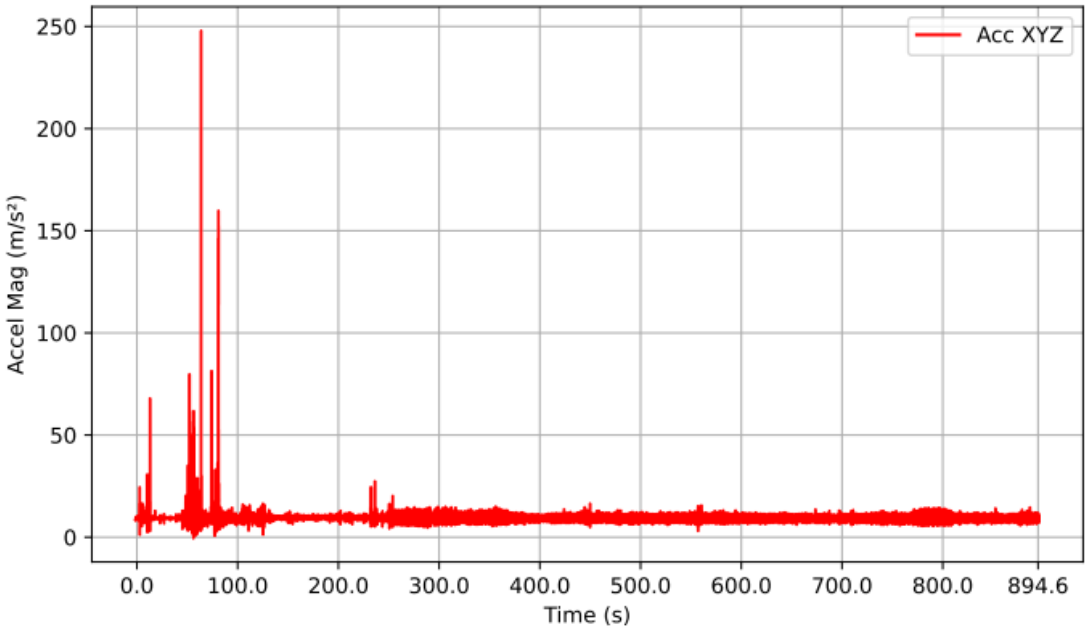


Joonis 2. Rõhu graafik .PDF failis

Teise graafiku jaoks oli samuti andmetes kolm kiirenduse arvu ja pidin neid kasutades arvutama kiirenduse suurusjärgu ja seda graafikul näitamine. Suurusjärgu arvutamiseks ma võtsin kõigi kolme arvu ruudud, liitsin need kokku ja sellest arvust võtsin ruutjuure

igas ajapunktis. Selle jaoks on Valem 1. Graafik istus esimese graafiku all ning x- ja y-teljed olid vastavalt aeg sekundites ning kiirenduse suurusjärk meeter ruutsekundi kohta. Joonises 3 on päris andmetega koostatud näide kiirenduse suurusjärgu graafikust .PDF failis.

$$\text{AccMag} = \sqrt{\text{Acc1}^2 + \text{Acc2}^2 + \text{Acc3}^2} \quad (1)$$



Joonis 3. Kiirenduse suurusjärgu graafik .PDF failis

2.4 Teine skript ja nende kokkuvõte

Veel tuli ka kirjutada teine skript, mis tegi täpselt sama tööd. Peaaegu terve kood oli esimesega täpselt sama ja ainukesed vahed olid selles, et ühes oli rohkem andmeid millega töödelda ja nende järjekord oli natuke erinev. Selle jaoks pidin uuesti arvutama bittide suuruse teise skripti jaoks ja graafikute genereerimiseks muutma, et mis kohalt see peaks andmed leidma, aga see oli kõik väike töö. Lõpuks valmis kaks skripti, mis töötlesid kiiresti terve kausta faile ja panid need uutesse kaustadesse ilusti istuma. Sellega oli mu esimene sprint läbi ja töö sai ette näidatud, kus sain tagasisidet paari asja kohta mis olid ebavajalikud (näiteks et sai valida mis kaustas töö tehakse - piisas sellest et töö toimub samas kaustas kus skript on) ning pärast mõne väiksema asja muudatust koodis polnud rohkem nende skriptide kallal enam midagi vaja teha ja läksin edasi peaprogrammi kirjutamisega.

3 Kasutajaliidesega peaprogramm

Lõpetanud skriptide kirjutamisega hakkasin pihta kasutajaliidese kirjutamisega. Esialgu tahtsin valmis saada akna kus istub graafik ja kus oleks nupp, mis laseks valida faili graafikul näitamiseks.

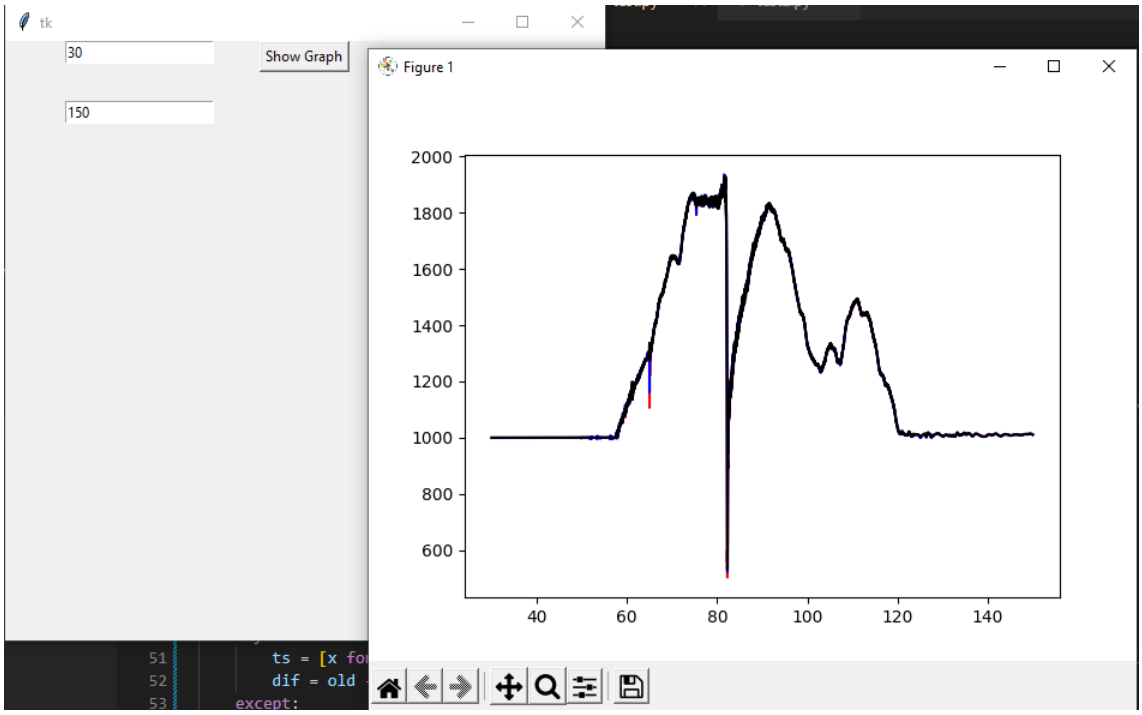
3.1 Esialgne .CSV lugemine ja graafiku näitamine

Alustuseks kirjutasin valmis baaskoodi, mis loeks .CSV failist välja rõhu ja aja andmed ning kasutades neid teeks graafiku sarnaselt nagu oli Joonis *2* näidatud, aga ilma *mean* väärtuseta. .CSV lugemiseks kasutasin *Python*i *csv* teeki ning piiritlejaks (*delimiter*) oli koma, mis siis lasi kõik andmed oma lünka massiivis panna. Faili valimist ise veel polnud ja see oli kõvakodeeritud sisse, et mis faili kood kasutab.

3.1.1 Kasutajaliidesega alustamine ja aja vahemiku valik graafikus

Kui mul graafik oli olemas, siis alustasin lihtsama kasutajaliidese arendusega. Selleks kasutasin *Python*i *tkinter* teeki, mis on väga populaarne oma kasutajasõbralikkuse ja lihtsuse pärast. Alustuseks tegin väikse akna, kus oli ainult "*Show Graph*" nupp, mis siis andis käsu lugeda kõvakodeeritud faili ning seda eraldi aknas graafikul näidata.

Edasi ajavahemiku valikuks lisasin kasutajaliidesele veel kaks tekstikasti, mis siis olid vastavalt esialgse ja viimase aja seadmiseks sekundites graafiku jaoks. Näiteks kui andmed olid 800 sekundit pikad, siis sellega said valida ajavahemiku 30 ja 150 sekundi vahel. Joonisel 4 ongi näha täpselt sama näidet ajavahemiku valikust ning milline see koos kasutajaliidesega välja nägi. Selle jaoks kood võttis numbrilised väärtused tekstikastidest ja eemaldas kõik rõhu ja aja andmed, mis olid väiksemad kui kasti kirjutatud esialgne aeg ja vastupidi eemaldas kõik andmed pärast seatud lõppaega. See funktsionaalsus aitas lihtsasti uurida ja salvestada pildi kujul huvipakkuv piirkond graafikult.



Joonis 4. Esialgne kasutajaliides koos ajavahemiku valikuga

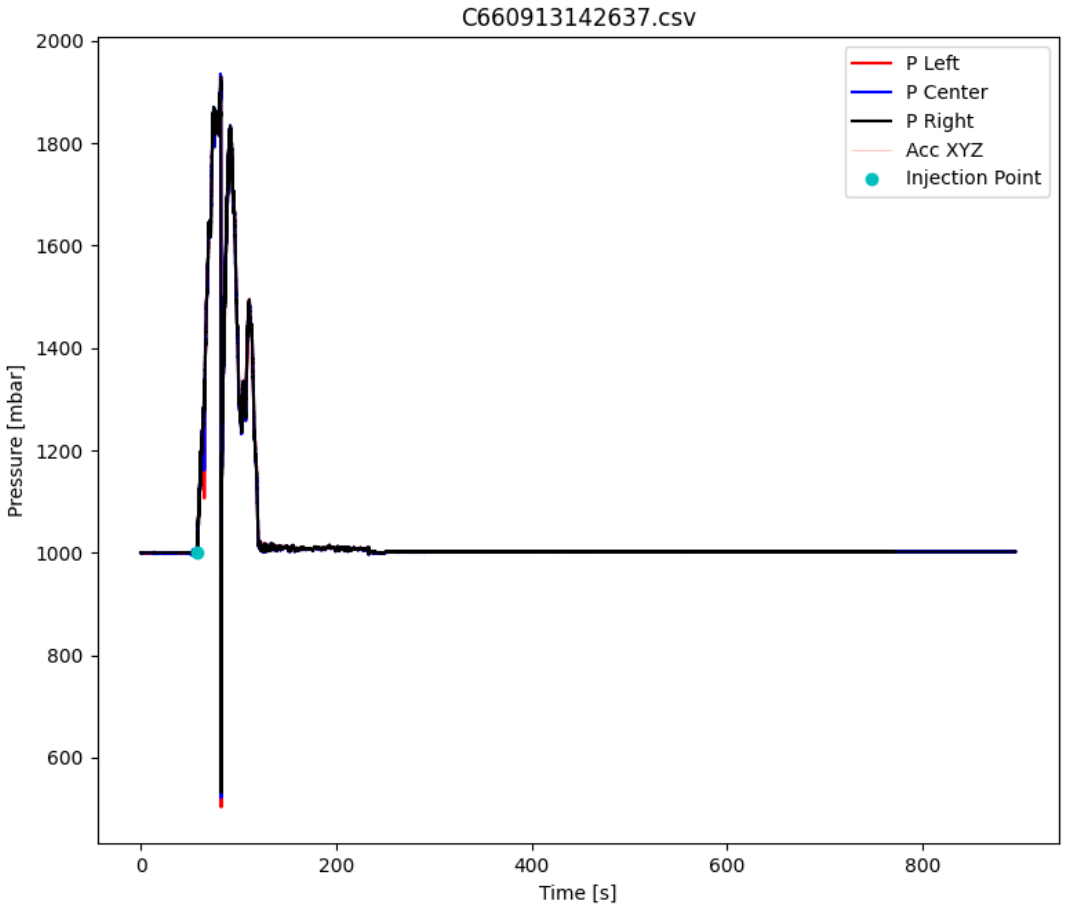
Selline ajavahemiku manuaalne valimine sai paar kuud hiljem eemaldatud koodist, kuna võimaldasin lisada *custom* punkte graafikule. Nende punktide vahelt sai graafikut täpselt samamoodi kärpida selles piirkonnas kust sooviti, baaskood mis tegeles sellega, et andmed saaks kärbitud õiges vahemikus oli sama nagu siin tehtud ja kaotasin ainult ära käsitsi kirjutamise kohad.

3.1.2 Automaatne süstimispunkti tuvastus graafikul

Üks tähtsamatest asjadest, mis mu programm pidi võimaldama, oli automaatne süstimispunkti tuvastus rõhu andmetes. Andmetega, millega töötasin, olid tulnud allveeanduritelt, mis läbisid hüdrojaama turbiine ja see hetk kui andur on turbiini läbimas tekitab rõhus suuri muutusi. Süstimispunkt selles kontekstis on hetk, kus rõhk hakkab järsult tõusma ehk punkt kus andur sisenes turbiini.

Selle leidmiseks kood läheb ühe kaupa läbi kõik rõhu andmepunktid ning võrdleb käes hoitavat andmepunkti sellega järgnevaga, et kas rõhu muutus on piisavalt suur või mitte. See millist rõhu muutust otsitakse, saab kasutaja liuguriga (*slider*) kasutajaliideses valida 1-10 millibaar vahemikus. Kui kood leiab ajahetke, kus kahe rõhu andmepunkti vahe on

piisavalt suur, siis see lõpetab edasise otsimise ja paneb kirja selle ajahetke kus see leiti. See leitud ajahetk ongi vajalik süstimispunkt, mis märgitakse ka ilusti graafikule. Samuti kasutajaliideses on kirjutatud ajahetk kust see leiti. Joonisel 5 on näha, kuidas süstimispunkt sai markeeritud graafikule sinise täpina.



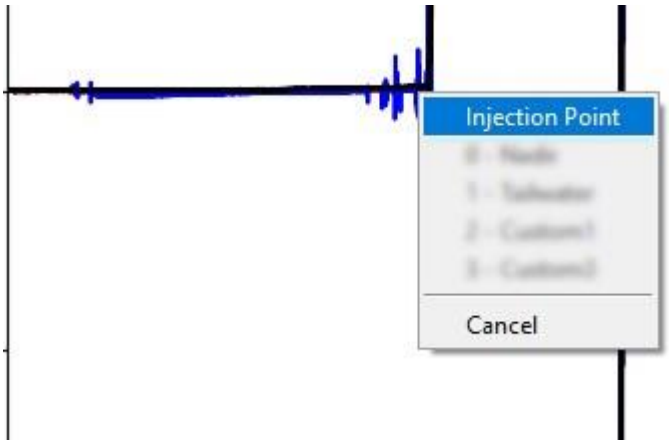
Joonis 5. Automaatselt leitud süstimispunkti markeering graafikul

3.1.3 Manuaalne süstimispunkti leidmine graafikul

Alati ei pruugi automaatne leidmine leida süstimispunkti õiges kohas, näiteks kui enne seda hetke kuskil rõhk muutus järsult. Sellepärast võimaldasin graafikul manuaalselt seada süstimispunkti panemise. See funktsionaalsus sai lisatud umbes kuu aega pärast automaatse punkti leidmist.

Selle võimaldamiseks ma lisasin graafikule hiire parema klahvi vajutusele konteksti menüü kutsumise [3], kus siis oli nupp nimega "*Injection Point*". Nupu vajutamisel tekkis graafikule punkt samamoodi nagu eelnevalt Joonisel 5 oli näidatud, aga täpselt sinna kus

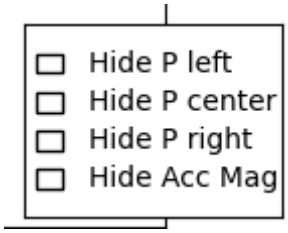
hiir asus parema klahvi vajutuse hetkel. Tänu sellele on nüüd võimalik süstimispunkt lisada täpselt sinna kus vaja, kui peaks juhtuma et automaatne leidmine ei too tulemusi. Joonisel 6 on näha, kuidas see konteksti menüü ka välja nägi.



Joonis 6. Hiire parema klahvi vajutuse konteksti menüü graafikul

3.1.4 Graafiku elementide peitmine

Võimalik, et alati ei taheta näha kõigi kolme rõhu andmeid korraga graafikul või kiirenduse suurusjärk on ebavajalik. Sellepärast sain tagasisidet, et võiks saada vajadusel peita ebavajalikke elemente graafikul. Algselt ma tegin kasti nagu on joonisel 7 näha ja see töötas nagu vaja. Vajutades graafiku elementide vasakul oleval kastile sai peita seda vastavat elementi graafikult. Hiljem sai see kuigi asendatud sellise süsteemiga, kus graafiku legendil vajutades mingi elemendi peale peitis see seda, mille peale sai vajutatud [4]. See oli parem sellepärast, et juhendaja ise oli harjunud varem sellega ning siis sai kaotada ära joonises 7 näidatud kasti, mis istus natuke graafiku peal ja võis ette jääda.



Joonis 7. Graafiku elementide peitmise kast

3.2 Andmebaasi üles seadmine

Üks funktsionaalsustest, millest alguses mulle teada anti, oli see et graafikuid oleks võimalik salvestada kuidagi andmebaasi, et hiljem saaks kergesti poolelioleva graafiku ette võtta koos juba eelnevalt leitud sisestuspunktide ja teiste *custom* punktidega. Algselt oli plaan teha see MySQL abil, aga kuna sain teada, et andmebaas on ainult lokaalne ja ei pea kuskil võrgus olema, siis kasutasin lõpuks SQLite andmebaasi teeki.

Funktsioonide kirjutamiseks kasutasin Pythoni *sqlite3* teeki ja ühtegi autentimis seadet polnud. Andmebaasi fail oli uues kaustas "*database*" juurkataloogis ning seal asus fail "*data.db*". Selleks et andmebaasi kood hoida eemal kasutajaliidese koodist, siis eraldasin suurem osa selle koodist juurkataloogis asuvasse faili "*db.py*". Osad funktsioonid pidid ikkagi jääma kasutajaliidese koodi, sest mõned funktsioonid vajasid kasutamiseks nii kasutajaliidese, kui ka andmebaasi koodide faile, mis Pythonis tekitab *import loop* veateate. Sellepärast oli lihtsam sellised funktsioonid lõigata "*db.py*" failist ning tuua kasutajaliidese koodi faili.

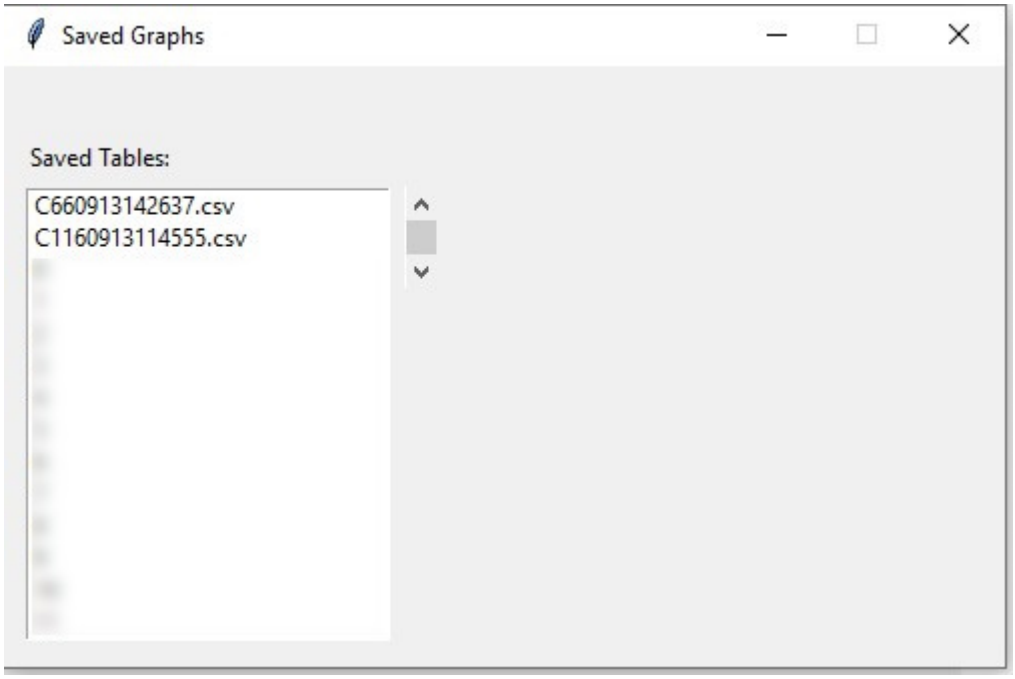
3.2.1 Graafiku andmete salvestamine andmebaasi

Alustasin esmalt tabelite loomisega andmebaasis. Selle jaoks võtsin programmi sisestatud .CSV faili nime ning kasutasin seda kui tabeli nimena andmebaasis, et kergesti ära tunda erinevaid tabeleid. Edasi vaatasin et tabelisse saaks ka loomise ajal tehtud tulbad rõhu ja aja andmete jaoks ning ka süstimispunkti koordinaadid ja stsenaario. Stsenaario on tekst, mida kasutaja sai lisada lisainfona antud faili jaoks kasutajaliideses tekstikastis.

Kui tabel sai loodud, siis edasi esmalt vaatasin, et saaks salvestatud kõik aja ja rõhu andmed andmebaasi. Andmed istusid andmebaasis nii, et ühel real oli ühe ajahetke rõhu väärtused ja selle jaoks lasin siis koodil tsükli teha, kus see ühekaupa sisestas kõik read sisse, kuni kõik andmed said sisestatud.

Nüüd kus baas asjad olid andmebaasi edukalt salvestatud, siis töötasin selle kallal, et neid ka kuidagi sealt kätte saaks ja graafikul näidatud. Selle jaoks lõin uue akna, mis tekib vajutades nuppu "*Show Saved Graphs*". Seal aknas istub loend kõikidest andmebaasi

salvestatud tabelitest ja vajutades ühegi tabeli nime peale kaks korda vasaku hiireklahviga, luges see kõik aja ja rõhu andmed andmebaasist välja ja kutsus välja funktsiooni nende andmete näitamiseks graafikul. Joonisel 8 on näha, milline see aken välja nägi.



Joonis 8. Andmebaasi salvestatud graafikute menüü

Edasi vaatasin, et kui enne andmebaasi salvestamist olid ka süstimispunkti andmed ja stsenaarium olemas, siis need saaks samuti salvestatud oma tulpa. Pärast seda kirjutasin koodi selle jaoks, kui andmebaasis on süstimispunkt olemas, et siis see tekiks ka graafikule samamoodi nagu varasema süstimispunkti koodi puhul, ning et stsenaariumi tekst saaks pandud oma tekstikasti kohta juhul, kui see oli olemas.

3.2.2 Andmebaasi funktsionaalsuse kokkuvõte

Andmebaasi koodi kirjutamine oli kindlasti üks raskemaid asju, mis pidin kirjutama programmi jaoks, sest seal tekkis palju probleeme ka kõige väiksemate asjadega nagu, et missuguseid ülakomasid on teksti ümber kasutatud. Siiski sain kõigest mööda ilma midagi vahele jätmata.

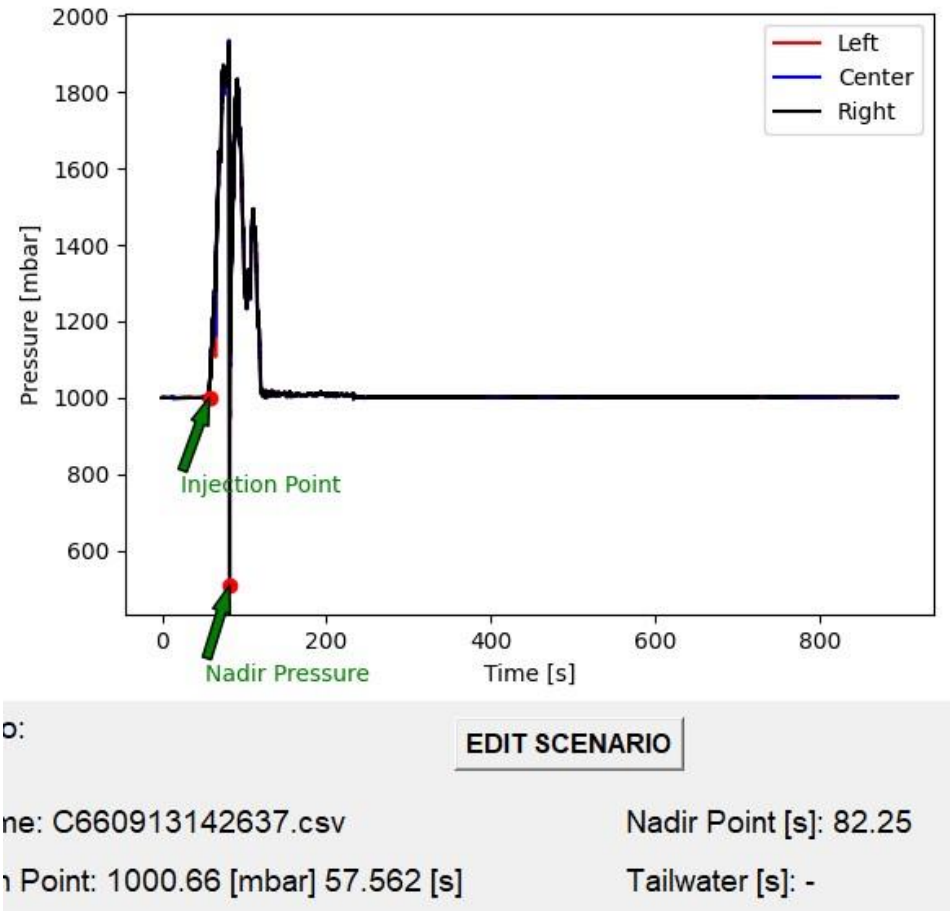
Kuna projekti jooksul tekkis veel teisigi andmeid juurde ja see kuidas andmeid hoiti oli tihtipeale muutumas, siis andmebaasi funktsioonid lõpuks lakkasid töötamast. Kuna selle parandamine oleks nõudnud peaaegu terve andmebaasi koodi ümber kirjutamist, siis küsisin juhendajalt üle, et kas üldse on mõtet enam hoida seda funktsionaalsust alles. Jõudsime arusaamale, et see polnudki nii tähtis kui algselt tundus ja sellepärast sai kõik sellega tehtud töö eemaldatud projektist.

3.3 Graafikule *custom* punktide lisamine

Üks asi, mis oli tahetud programmi jaoks, oli see et saaks lisada graafikule veel teisigi punkte peale süstimispunkti. Rohkem kasutatavate punktide seas olid punktid nimega "*Nadir Pressure*" ning "*Tailwater*", millega ma ka alustasin, et neid saaks graafikule manuaalselt seada.

3.3.1 "*Nadir Pressure*" ja "*Tailwater*" punktide lisamine graafikule

Kuna mul oli eelnevalt tehtud mugav kontekstimenüü parema hiireklahvi vajutuse peale graafikus, siis mõte oli need punktid samuti sinna panna. Seda ma siis tegingi ja mõlema punkti jaoks kirjutasin sarnased funktsioonid, mis andsid kasutajaliideses teada, mis olid nende väärtused ning märkisid punktil graafikul. Joonisel 9 on näha, kuidas graafikul punkti näitamine välja nägi. Ajahetk on nähtav graafikust allpool tekstides „*Nadir Point [s]:* " ja sama *tailwateriga*.



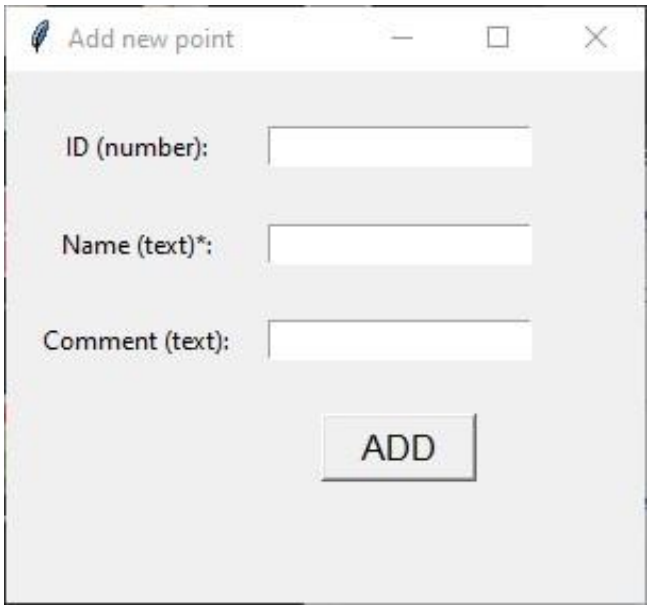
Joonis 9. "*Nadir Pressure*" markeering graafikul ja selle ajahetk kasutajaliideses

3.3.2 *Custom* punktide koodi ümberkirjutamine

Eelnevalt koostatud koodil otseselt midagi väga viga ei olnud, aga selle suurimaks probleemiks oli see et mõlemad punktid "*Nadir*" ning "*Tailwater*" olid kõvakodeeritud. See oli problemaatiline, sest on võimalik et läheb vaja rohkem *custom* punkte, kui ainult neid kahte.

Sellepärast sai kogu vana kood tasapisi kaotatud ja hakkasin mõtlema, kuidas seda punktide asja dünaamiliselt koostada. Selle jaoks ma koostasin süsteemi, kus kasutaja saab kasutajaliidese kaudu koostada uued punktid oma tahetud nime ja numbrilise ID-ga (Joonis 10). Koostatud punktide andmed hoitakse massiivides, mis hoolitsevad selle eest et punkt püsiks graafikul ning, et selle koordinaadid oleksid õiged. Punkti sai graafikule panna läbi parema hiireklahvi vajutusega tekkinud menüüga täpselt nagu süstimispunkti manuaalse seadmisega. Kõikide loodud punktide nägemiseks on kasutajaliideses

punktide nimekiri koos ka ajahetkega sekundites, kus punkt graafikul on, kui punkt oli graafikule pandud (Joonis 11).



Joonis 10. *Custom* punktide loomise menüü

Joonis 11. Nimekiri koostatud *custom* punktidest ja nende ajahetkedest

3.3.3 Kommentaaride lisamine ja konfiguratsiooni fail

Uus süsteem oli palju parem vanemast selle poolest, et see oli dünaamiline ja ka üleüldiselt paremini vormistatud kasutajaliideses lihtsamaks lugemiseks. Juhendaja oli ka väga rahul sellega, aga andis nõuandeid et kuidas seda veel arendada. Üks neist oli kommentaaride lisamine punktidele, sest mõned punktid võivad olla millegi väga spetsiifilise jaoks ja mis niisama nimest lugedes ei pruugi meelde tulla, nii et kommentaar punkti juurde oleks väga hea. Samuti veel ka oli probleem, et igakord kui programm kinni ja lahti teha, siis loodud punktid kadusid ja need tuli uuesti seadistada.

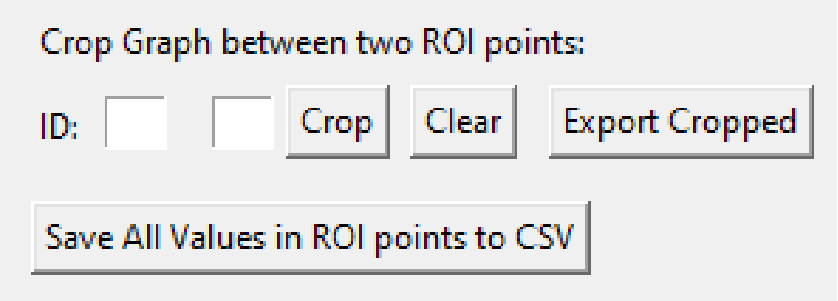
Selle jaoks ma koostasin uue süsteemi, kus mul on fail nimega “points.json”, kus massiivis istuvad punktid kolme väärtusega - “*id*”, “*name*” ja “*comment*”. Joonisel 12 on faili sisu näha. Kui peaprogramm lahti tehti, siis see luges andmed “points.json” failist läbi ja kandis seal olevad punktid nimekirja kasutajaliideses (Joonis 11). Samuti, kui kasutajaliideses koostati uus punkt, siis selle andmed salvestati “points.json” faili. Selle lisamisega oli nüüd võimalik programm rahulikult kinni panna teadmisega, et kui see järgmine kord lahti teha siis kõik punktid on kenasti salvestatud.

Joonis 12. “points.json” ülesehitus

Veel üks positiivne lisa “points.json” failiga on see, et see andis ka võimaluse kustutada ja muuta loodud punkte, sest kasutajaliideses selle jaoks eraldi funktsionaalsust ei olnud. Varem kui sai loodud mõni punkt vale nimega, siis ainuke võimalus parandamiseks oli kõigega uuesti alustamine programmi restartides, aga selle faili olemasolu nüüd leevendas ka seda probleemi.

3.3.4 Graafiku kärpimine ja andmete eksportimine

Paragrahvis 3.1.1 sai kirjutatud sellest, kuidas oli võimalik sisestada kasutajaliideses alguse ja lõpuaeg sekundites graafiku kärpimiseks. Nüüd, kui oli võimalik oma punkte mugavalt graafikule lisada, sai see manuaalne sekundite sisestus eemaldatud ja selle asemel graafiku kärpimine toimub läbi *custom* punktide. Kui graafikule märkida vähemalt kaks punkti, siis on võimalik nende kahe punkti ID-d panna kasutajaliideses kirja ja graafik siis näitab ainult seda, mis on nende kahe punkti vahel. Joonis 13 näitab kärpimise jaoks kasutatavat kasutajaliidese osa.



Joonis 13. *Custom* punktide abil joonise kärpimise seaded

Samuti võimaldas kood kärbitud graafikult kõik toorandmed eksportida .CSV faili nupuga "*Export Cropped*", mida on näha jooniselt 12. See on vajalik, kui sa tahad näiteks ainult 30 sekundit enne ja pärast kõige madalamat rõhu punkti andmetes, mida saab siis veel hiljem kasutada *confidence* graafikute loomiseks. Salvestatud fail kasutab originaalse faile nime, aga see algab sõnaga "*cropped_*" ja lõpus on eraldatuna allkriipsuga olid nende punktide ID-d, mille vahelt said andmed kärbitud.

Viimasena sai ka eksportida kõikide loodud *custom* punktide andmed sellel ajahetkel, kuhu punkt oli graafikul asetatud. Need said samuti eksporditud .CSV laiendiga ja faili nimi algas sõnaga "*export_*" ja lõppes originaal faili nimega. Joonisel 14 on näha, kuidas andmed välja nägid.

	A	B	C	D	E
1	File Name	INJECTION	Nadir	Tailwater	Custom1
2	B080614145037.csv	-	29.9	-	49.97
3	Time [ms]	-	29.898	-	49.974
4	PL [hPa]	-	1499.3	-	1233.7
5	TL [C]	-	25.77	-	24.96
6	PC [hPa]	-	1570.8	-	1234.6
7	TC [C]	-	26.56	-	26.03
8	PR [hPa]	-	1550	-	1234.4
9	TR [C]	-	25.73	-	24.95
10	EX [deg]	-	127.06	-	341.69
11	EY [deg]	-	-62.125	-	3.6875
12	EZ [deg]	-	80.625	-	123.75
13	QW [-]	-	0.32678	-	-0.46564
14	QX [-]	-	0.39954	-	0.85687
15	QY [-]	-	0.54987	-	0.20819
16	QZ [-]	-	-0.65668	-	-0.07501
17	MX [microT]	-	14.75	-	4.6875
18	MY [microT]	-	-7.25	-	18.063

Joonis 14. *Custom* punktide ajahetke andmed eksporditud .CSV faili

3.4 *Confidence* graafikud

95% *Confidence* graafikute lisamine oli viimane suurematest funktsioonidest, mille kallal ma töötasin. Selle jaoks oli hea kood näiteandmetega internetis olemas [5], mis kergendas tööd mitte tuttavas alas väga palju.

Mulle küll ei räägitud kõigest, mille jaoks need graafikud vajalikud on, aga üks nende tähtsamatest kasutustest on kahe erineva sensori grupi andmete võrdlemine. Osad sensorid olid silindrilise objekti sees, mis visati vette ja see läbis hüdrojaama turbiini niimoodi, aga mingi osa neist oli otse kalade küljes. *Confidence* graafik annab siis võimaluse kergesti neid kahte omavahel võrrelda, et näha kuidas päris kala turbiini läbib võrreldes tehisliku objektiga.

Kuna see oli natuke raskem töö, kui esialgselt arvasin, siis esimese päev selle kallal läks puhtalt õppimisele. Kasutades internetist leitud koodi [5] ma uurisin, kuidas see töötab eraldi failis ning kui hakkasin koodist aru saama, siis oli plaan asendada näidisandmed tasapisi oma andmetega.

Sisestatavad andmed selle funktsionaalsuse puhul on terve kaust *.CSV* faile, mille jaoks ma siis pidin kirjutama uue funktsionaalsuse, et kasutaja saaks kasutajaliideses valida kausta. Kood luges kaustast *.CSV* failid läbi ning kõigi faili andmed läksid ühte *pandas* teegi massiivi. Peaprogrammi rõhu graafiku jaoks sai kasutatud tavalist *Python*i massiivi ja nii oli ka plaanis teha algselt siin, kuna ma olin sellega tuttavam, aga *confidence* graafiku näidiskoodi vaadates sain aru, et vajalikke arvutusi on lihtsam teha *pandas* teegi massiividega, sest see suutis mitme faili andmetega ühes massiivis palju kergemini kõik arvutused teha, mis vajasid mitme faili andmeid korraga.

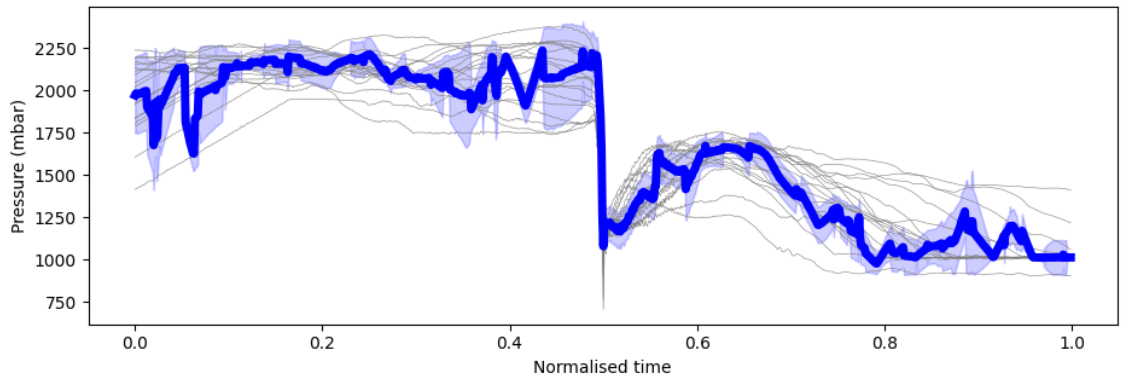
3.4.1 Graafikute arvutused

Graafikute koostamiseks oli mul vaja rõhu ja kiirenduse andmeid massiivist. Nendega ma arvutasin keskmise, mediaan ja standardhälve väärtused igas ajapunktis andmetes. Selle jaoks ma valemeid ei kasutanud, vaid *Python*i *pandas* teegi käske "*mean*", "*median*" ja "*std*". *Confidence* jaoks oli vaja ka arvutada madal ning ülempiirid ja seda sai vastavalt lahutades ja liites standardhälve väärtuseid keskmistest.

Viimaseks oli veel vaja ka aeg normaliseerida nullist üheni, mille jaoks ma tegin *numpy.arange* käsklusega massiivi nullist üheni, kus kasutaja saab valida resolutsiooni ehk sammu iga järgneva arvu vahel kasutajaliideses tekstikasti abil. Kasutades seda uut massiivi ma *resample*-isin vajalikud andmed uute aja väärtustega. Algselt ma kasutasin *scipy.fit_transform* käsklust aja normaliseerimiseks, aga kuna sellel polnud võimalik sammu täpsustada kasutaja poolt, siis sai tehtud kirjutatud *numpy.arange* käsklusega.

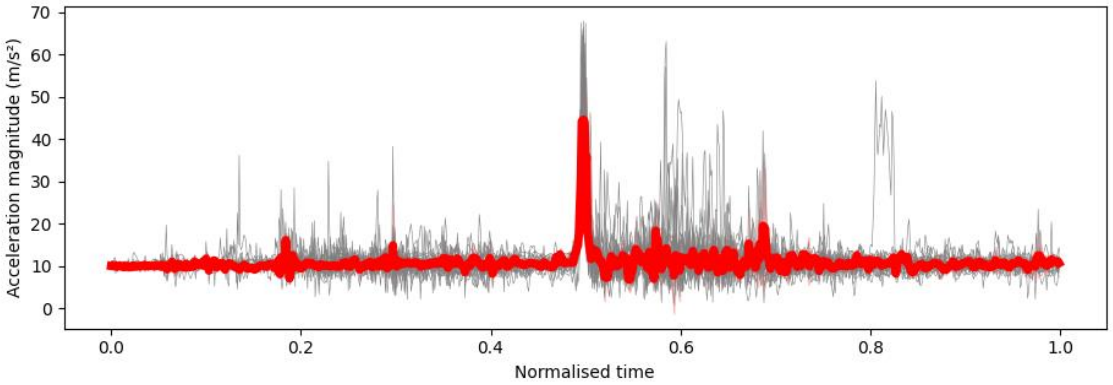
3.4.2 Graafikute loomine

Nüüd kus suurem osa tööst oli ära tehtud, oli vaja ainult veel graafik luua. Koodi poolest see töötas samamoodi nagu tavaline rõhu graafik, ainult teiste andmetega. Joonisel 15 on näha, kuidas rõhu graafik välja nägi. Halli joontega on iga faili algandmed, paks tumesinine joon on arvutatud keskmine ja seda ümbritsev helesinine ala on *confidence* vahemik arvutatud keskmise ja standardhälve abil. Selleks, et mediaan väärtuseid näha on kasutajaliideses nupp, mis graafikus vahetab, et kas näidatakse keskmist või mediaan väärtust paksu tumesinise joonega.



Joonis 15. Rõhu *confidence* graafik

Teine *confidence* graafik oli kiirenduse suurusjärgu jaoks. Samamoodi sai arvutatud igas ajapunktis kiirenduse suurusjärk nagu eelnevalt valem 1 näitas. Kõik edasised arvutused olid samad ja graafiku peal näidatavad elemendid on ka samad, kus ainuke vahe on selles et rõhu asemel kiirenduse andmed ja kasutatakse punast värvi. Joonisel 16 on näha, milliselt kiirenduse suurusjärgu *confidence* graafik välja nägi. Samamoodi sai nupuvajutusega kasutajaliideses vahetada, et kas on näha keskmist või mediaan väärtust.

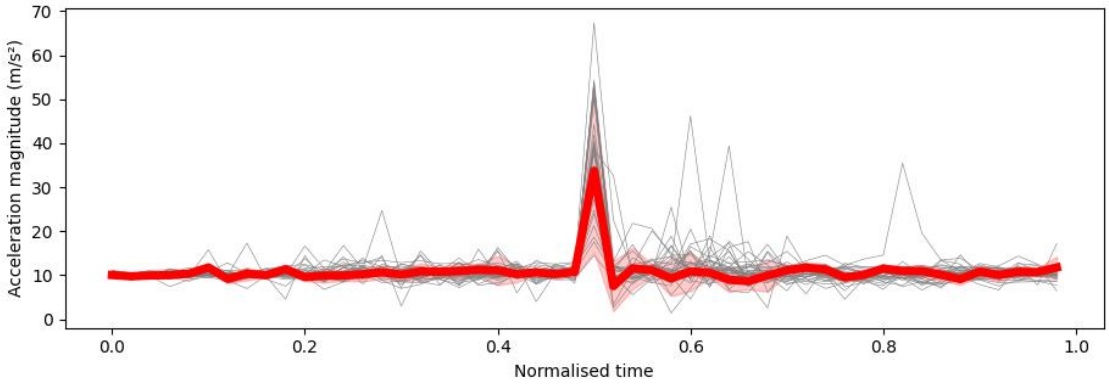


Joonis 16. Kiirenduse suurusjärgu *confidence* graafik

3.4.3 Graafiku algandmete eksportimine

Kui aeg normaliseerida kasutaja poolse sammuga, siis algandmete kuju samuti muutub. Vaikimisi on üks samm nullist üheni vahemikus 1/1000 ja seda oli see ka joonistel 15 ja 16. Joonis 17 näitab kuigi, kuidas näevad algandmed (hallide joontega) välja, kui samm oleks 1/50. Mõlemat kiirenduse suurusjärgu graafikut vaadates on kerge märgata, kuidas algandmed on muutunud.

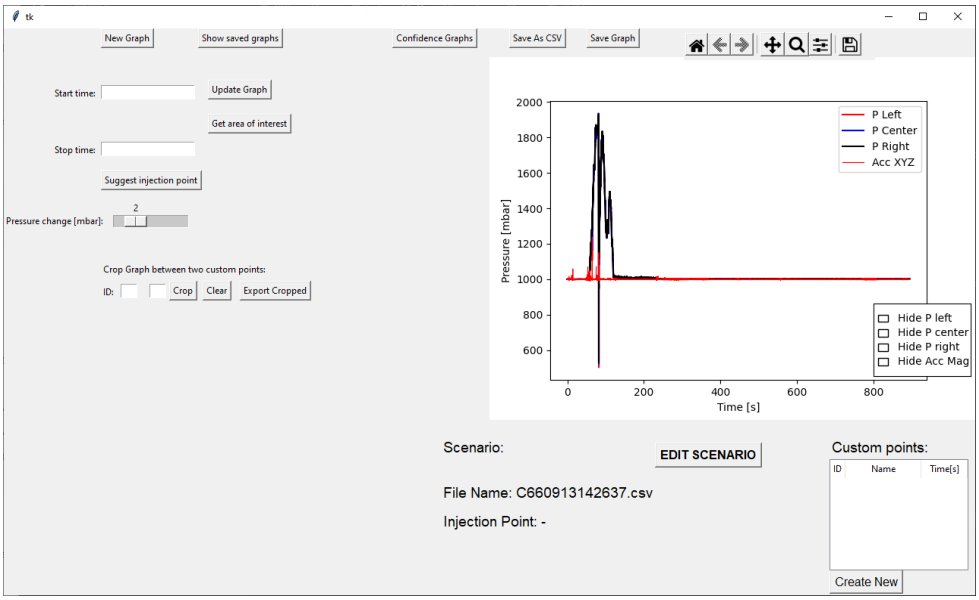
Tagasisidena koosolekus sain veel viimase palve lisada funktsiooni need muundatud algandmed ka eksportida. See oligi viimane funktsioon, mille kallal töötasin ja millega ma võimaldasin need algandmed eksportida *.CSV* failidesse uute kausta nimega "*normalised*". Faile tekkis sama palju, kui esialgses kaustas oli ja nende nimed olid sama, mis originaalid aga failide nimede ees oli "*normalised_*" kergeks tuvastuseks.



Joonis 17. 1/50 sammuga kiirenduse suurusjärgu *confidence* graafik

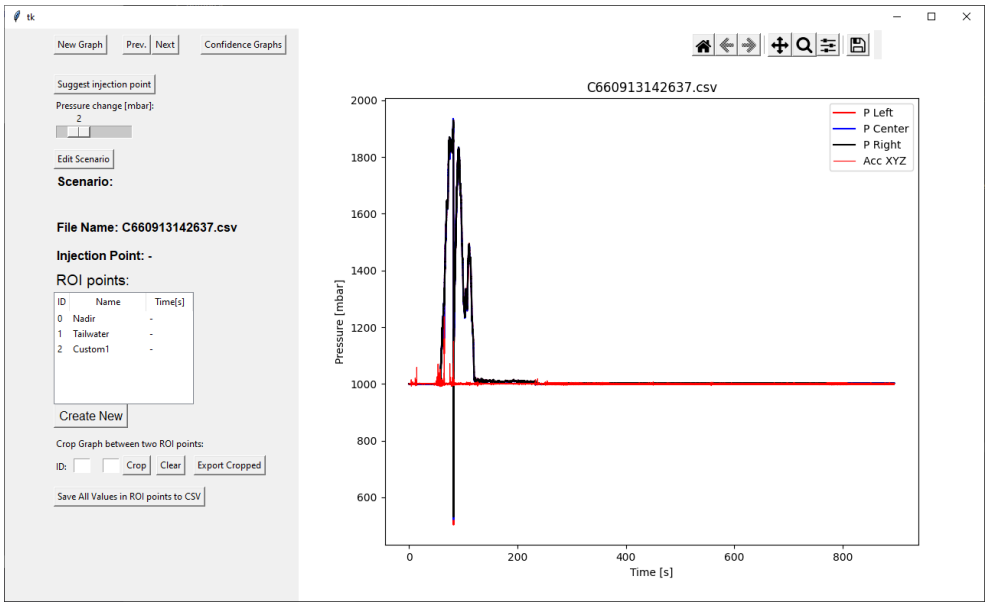
3.5 Kasutajaliidese ümberkujundused

Kasutajaliides oli kolm kujundust kokku. Vähesemate funktsionaalsustega kasutajaliidest on näha Joonis *4*. Selline kujundus oli ainult paar nädalat ja edasi mitu kuud oli kujundus nagu on näha joonisel 18. Seal kõik funktsionaalsuse elemendid olid suhteliselt suvaliselt visatud kuhu mahtusid, kuna ma veel ei teadnud, kui palju asju tekib juurde ja sellepärast ei hakanud suurt ümberkujundust tegema. Alguses oli ainult *start* ja *stop time* funktsionaalsus sellel koos graafikuga ja üleaja tekkis asju rohkem juurde.



Joonis 18. Kasutajaliides suurem osa tööajast

Umbes 3-4 nädalat enne projektiga lõpetamist olid mul kõik peaprogrammi funktsionaalsused peaaegu olemas peale *confidence* graafikute lisamist, mispärast tegin ka suure ümberkujunduse. Juhendajalt sain palve teha selline kujundus, et graafik oleks umbes kaks korda suurem ning et kõik nupud ja kastid asuksid vasakul pool programmi. Jooniselt 19 on näha, milline peaprogramm projekti lõpus välja nägi.



Joonis 19. Viimane kasutajaliidese kujundus

4 Kokkuvõte

Projekti jooksul valmis üks suur programm allveeanduritest tulevatest andmete töötlemiseks graafilisel kujul. Töö algas kõigepealt kahe skripti kirjutamisega, mis dekodeeriks sensoritest tulevad toorandmed ja ekspordiks vajalikku formaati. Peale seda sai jupp jupi haaval üles ehitatud peaprogrammi funktsionaalsust alustades lihtsalt andmete lugemisest graafiku näitamiseks.

Iga kahe nädala tagant oli mul koosolek oma juhendaja, kus ma näitasin ette tehtud töid ja siis tagasiside põhjal tegin muudatusi. Seal ka arutasime, et millele peaks järgmisena keskenduma. Eesmärke sai püstitatud paljusi, näiteks andmebaas, *custom* punktid, ajavahemiku valimine ja *confidence* graafikud, aga kõik oli tehtav ja sai ka lõpp programmi jäetud. Ainult andmebaasi süsteem leidsime, et polnud tegelikult vajalik ja selle funktsionaalsus sai täielikult eemaldatud.

Töö jooksul ise ka õppisin palju Pythoni kohta, sest eelnevalt mul oli ainult ühes aines olnud kokkupuude sellega ja seal ma peamiselt töötasin andmebaasi süsteemi integreerimisega ja uuendamisega. See projekt oli palju laiem ja seepärast sain tutvuda paljuga, mida Python tegelikult pakub, nagu andmete töötlus või kasutajaliidese elementide ehitamine.

Kasutatud kirjandus

[1] Oracle Corporation. “Data Types and Sizes.” *Oracle*, (Kasutatud 5. veebruar 2023) https://docs.oracle.com/cd/E19253-01/817-6223/chp-typeopexpr-2/index.html. (2010)

[2] thisPointer, „How to save Numpy Array to a CSV File using numpy.savetxt() in Python“ Varun, (Kasutatud 6. veebruar 2023) https://thispointer.com/how-to-save-numpy-array-to-a-csv-file-using-numpy-savetxt-in-python/

[3] GeeksforGeeks, „Right Click menu using Tkinter“, Shreyasi Chakraborty (Kasutatud 7. märts 2023) https://www.geeksforgeeks.org/right-click-menu-using-tkinter/#article-meta-div (20.04.2020)

[4] YouTube, „How to Toggle Graphs Visibility By Clicking Legend Label in Matplotlib“, Jie Jenn (Kasutatud 20. aprill 2023) https://www.youtube.com/watch?v=QcRE8dinls4 (2021)

[5] Python Charts “Line Chart with Confidence Interval in Python” Alex, (Kasutatud 6. aprill 2023) https://www.pythoncharts.com/python/line-chart-with-confidence-interval/ (25.05.2022)

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Tanel Lentso

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "GUI ja automatiseeritud aegridade klassifikatsioon multimodaalsete allvee andurite jaoks", mille juhendaja on Jeffrey Andrew Tuhtan

   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktil 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

09.04.2023

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Link tehtud töö *GitHub*-ile

https://github.com/talentTanel/intCSVmainGUI