

ISTARSKO VELEUČILIŠTE
UNIVERSITÀ ISTRIANA DI SCIENZE APPLICATE
Stručni studij politehnike

Izrada snimača podataka, obrada i vizualizacija
prikupljenih podataka bazirana na principima
slobodnog i otvorednog koda

Završni rad

Kristijan Cetina
JMBAG: 2424011721
kcetina@iv.hr

Pula, 31. kolovoza 2019.

Sažetak

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Ključne riječi *riječ, dva, tri . . .*

Kolegij: Elektronika

Mentorica: Sanja Grbac Babić, mag. računarstva, v.predavač

Abstract

Abstract in English

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Keywords: *word, two, three . . .*

Posveta

Zahvala

Zahvala svima koji zaslužuju

"A good scientist is a person with original ideas. A good engineer is a person who makes a design that works with as few original ideas as possible. There are no prima donnas in engineering" - Freeman Dyson

Izjava o samostalnosti izrade završnog rada

Izjavljujem da sam završni rad na temu *Izrada snimača podataka, obrada i vizualizacija prikupljenih podataka bazirano na principima slobodnog i otvorenog koda* samostalno izradio uz pomoć mentorice Sanje Grbac Babić mag. računarstva, koristeći navedenu stručnu literaturu i znanje stečeno tijekom studiranja. Završni rad je pisan u duhu hrvatskog jezika.

Student: Kristijan Cetina

Sadržaj

1	Uvod i opis zadatka	1
1.1	Opis i definicija problema	1
1.2	Cilj i svrha rada	1
1.3	Hipoteza rada	2
1.4	Metode rada	2
1.5	Struktura rada	2
2	Opis korištenih tehnologija	3
2.1	Slobodan i otvoreni kod	3
2.2	Arduino platforma	4
2.3	Jupyter Notebook	5
2.4	NumPy	6
2.5	Git	7
3	Prikupljanje podataka - hardware	8
3.1	GPS logging shield	9
3.2	Prikupljanje podataka o temperaturi	11
3.2.1	Softwareski filter	11
3.2.2	Hardwareski filter	11
3.3	Prikupljanje GPS podataka	14
3.4	Spremanje podataka na memorijsku karticu	15
4	Obrada podataka - software	16
5	Zaključak	19
	Literatura	20
A	Programski kod na Arduino mikroračunalu	21
B	Analiza podataka kretanja vozila	27

Popis slika

2.1	Usporedba performansi Pythona i NumPy biblioteke	6
3.1	Izgled korištenog hardwareskog sklopa	8
3.2	Shema spoja TMP36 senzora	9
3.3	Shema Adafruit GPS Logger Shield	10
3.4	Vrijednosti senzora bez filtriranja	12
3.5	Vrijednosti senzora sa softwareskim filtriranja	12
3.6	Vrijednosti senzora primjenom kombinacije Sw i Hw filtera . .	13
4.1	Izrađeni graf kretanja vozila	17

Poglavlje 1

Uvod i opis zadatka

1.1 Opis i definicija problema

1.2 Cilj i svrha rada

Cilj ovog rada je bio izraditi jednostavan snimač podataka (*datalogger*) koji će spremati GPS podatke zajedno s podacima prikupljenim sa instaliranih senzora za kasniju analizu. Izrađeni uređaj je namjenjen kao snimač podataka u kompleksnijem sklopu koji je koji se može koristiti kada god postoji potreba za loggiranje podataka. Uređaj je namjenjen da zadovolji široki spektar potreba koje se mogu javiti bilo u industriji npr. prilikom praćenja pošiljki ili pak prilikom skupljanja podataka u istraživačke svrhe kako bi se razumio širi problem.

Sklop je baziran na Arduino platformi koja omogućava lak razvoj prototipova uz široku dostupnost gotovih dodatnih modula (*shields*) koji se jednostavno spajaju na bazno mikroračunalo.

Prikupljeni podaci se spremaju na SD karticu na uređaju u datoteku za kasniju obradu i analizu. Prikupljeni podaci se uz pomoć programskog jezika Python i dodatnih modula za statističku i numeričku analizu kao što su Pandas i Matplotlib obrađuju kroz sučelje interaktivne bilježnice Jupyter Notebook. Pristup obrade putem interaktivne bilježnice uz korištenje raznih tipova ćelija kao što su *Code Cells*, *Markdown Cells* i *Raw Cells* omogućava lakšu vizualizaciju i pregled samog rada koji je pogodan za kasnije dijeljenje svim zainteresiranim stranama koji žele pregledati ili nastaviti rad na analizi.

1.3 Hipoteza rada

1.4 Metode rada

1.5 Struktura rada

Kompletan Git repozitorij ovog rada javno je dostupan na <https://github.com/KristijanCetina/BachelorThesis>

Poglavlje 2

Opis korištenih tehnologija

2.1 Slobodan i otvoreni kod

Izraz otvoreni kod *open source* odnosi se na nešto što ljudi mogu slobodno mijenjati i dijeliti jer je dizajn javno dostupan[1]. Izraz je nastao u kontekstu razvoja računalnog softwarea dok se danas odnosi na pristup radu bio on software, hardware ili bilo kakav drugi tip projekta. Spomenimo kako postoje razne licence pod kojima se objavljuju open source radovi, a u praksi se razlikuju u načinu na koji izmjene i izvorni rad mora biti distribuiran svim ostalim zainteresiranim stranama.

Razlozi i prednosti primjene open source pristupa projektima su višestruke, a neke od njih su:

- Kontrola proizvoda
- Učenje i trening
- Sigurnost
- Stabilnost

Kontrola proizvoda: Kada je izvorni kod i ostala dokumentacija nekog proizvoda otvorena onda se može pogledati kako točno radi taj proizvod i na koji način je izgrađen. Na taj način svaki korisnik može imati kontrolu nad onime što koristi jer ne postoji koncept crne kutije (*BlackBox concept*) te omogućava korisniku da uz dostupni kod i sheme popravi ili unaprijedi proizvod. Zapišimo se koliko puta smo se osobno susreli sa situacijom kada zbog kvara nekog uređaja smo bili primorani posjetiti i platiti ovlaštenog servisera koji ima specijalni alat za diagnostiku i popravke?

Učenje i trening: Uvidom u otvorenu dokumentaciju možemo vidjeti kako je neki stručnjak riješio određeni problem te to rješenje u potpunosti ili modificirano može se primijeniti na vlastiti problem. Otvorena dokumentacija

omogućava proučavanje rješenja određenih problema i na taj način se skraćuje vrijeme i pojeftinjuje razvoj novih proizvoda koji imaju slične zahtjeve. Znanstvenici objavljuju svoja otkrića kako bi ih drugi iza njih mogli koristiti. Inženjeri u svakodnevnom radu ne izvode i dokazuju npr Ohmov ili Newtonove zakone već ih samo primjenjuju.

Sigurnost: Proučavanje objavljene dokumentacije nekog projekta drugi stručnjaci iz područja mogu uvidjeti neke propuste koje autori zbog kompleksnosti proizvoda ili drugih razloga nisu primjetili te dojaviti autorima grešku kao bi se ista mogla ispraviti. Neke greške se mogu pojaviti samo u iznimno malom broju slučajeva ili kada se poslože veliki broj faktora te nije realno očekivati da se prilikom testiranja proizvoda simulira svaki mogući scenarij korištenja. Zainteresirane strane mogu dodatno testirati proizvod u specifičnim uvjetima i na taj način otkriti inače skrivenu grešku u proizvodu te nakon otklanjanja greške sam proizvod postaje sigurniji.

Stabilnost: Mnogi proizvodi se koriste za vrlo bitne aspekte rada nekog većeg sustava te njihova zamjena iziskuje velike promjene i investicije, a ponekada nije niti moguća. Korištenjem proizvoda otvorenog koda i dokumentacije omogućava se korištenje nastavak podrške i korištenja tog proizvoda i nakon eventualnog nestanka kompanije koja je napravila proizvoda te isti više nije dobaljiv od proizvođača. Ako se koriste open source proizvodi moguće je samostalno rekreirati proizvod ukoliko se ukaže takva potreba.

2.2 Arduino platforma

Arduino je elektronička platforma otvorenog koda¹ baziran na hardwareu i softwareu koje je lako za koristiti. Arduino platforma obuhvaća mikrokontrolerske pločice bazirane na AVR arhitekturi s integriranim digitalnim, analognim ulazima i izlazima kao i PWM² izlazima. Platforma omogućava lagano spajanje dodatnih vanjskih uređaja kao što su razni senzori, releji, servo i motori putem dodatnog upravljačkog modula i ostale elektroničke i elektromehaničke komponente. Sheme svih mikrokontrolera objavljene pod Creative Commons³ licencom i javno dostupne svim zainteresiranim stranimama.

Arduino pločice su relativno povoljne u usporedbi s ostalim platformama i kao takve omogućavaju pristupačnije učenje svim zainteresiranima. Ponekad je ponekad malo spretnosti s lemlicom dok se često mogu slagati moduli na prototipnoj pločici bez lemljenja sa izradom spojeva putem spojnih žica.

¹<https://www.arduino.cc/en/Guide/Introduction>

²Pulse Width Modulation - Pulsno širinska modulacija

³<https://creativecommons.org/>

Jednostavno korisničko sučelje (IDE⁴) za izradu korisničkih programa (*sketch*) je jednostavno za korištenje za početnike dok istovremeno omogućava izradu vrlo kompleksnih programa iskusnim korisnicima. IDE je kompatibilan s većinom danas raspostranjenih operacijskih sustava (GNU/Linux, MacOS i Windows). Programski jezik za izradu programa je baziran na C/C++ te omogućava daljnje proširivanje kroz C++ biblioteke ili koristiti AVR-C programski jezik.

2.3 Jupyter Notebook

U ovom radu za obradu i prikazivanje podataka korišten je programski jezik Python⁵ uz dodatke NumPy⁶, Pandas⁷ i Matplotlib⁸. NumPy i Pandas omogućavaju lakšu manipulaciju podacima dok Matplotlib omogućava izradu kvaitetnih grafova s velikom mogućnošću prilagodbe raznim željama i potrebama. Sve zajedno je implementirano kroz sustav *interaktivne bilježnice* Jupyter⁹ koja omogućava brzu i jednostavnu obradu podataka kao i njeno dijeljenje sa svim zainteresiranim stranama. Jupyter notebook je web aplikacija otvorenog koda koja se može izvršavati na lokalnom računalu ili koristeći resurse računalstva u oblaku. Podržava razne programske jezike poput Julia, Ruby, R, C++ i mnoge druge te u ovom radu korišten Python. Jupyter notebook omogućava kreiranje i djeljenje dokumenata koji sadržavaju izvršivi programski kod, jednadžbe, grafove i vizualizacije te popratni tekst u jednoj cijelini koju trenutno drugim načinima nije moguće ili je vrlo nepregledno za postići. Područja primjene su najčešće obrada i transformacija podataka, numeričke analize, statistički modeli, vizualizacija podataka, strojno učenje i još mnogo toga.

Alati tipa Excel gdje korištene formule su skrivene iza podataka u ćelijama te se greške lako podkraudu i još lakše ostanu nezamječene. Istraživanja su pokazala značaju količinu grešaka u Excel proračunskim tablicama koje su u dnevnoj upotrebi diljem organizacija, od kojih neke su imale i značajne negativne financijske implikacije[2]. Iako je istraživanje starijeg datuma jedan od glavnih razloga grešaka (skrivenne formule) je i dalje prisutan te je realno za očekivati kako se greške i dalje događaju, a sa sve većom upotrebom proračunskih tablica za očekivati je kako broj istih s greškama raste.

Primjenom interaktivnih alata poput Jupyter notebooka koji imaju vidljivo prikazane formule koje koriste za proračun i kod kod za manipulaciju podataka lakše se mogu uočiti greške unutar istih te se mogu ispraviti. Primjenom metodologije testiranja koja je poznata u industriji raznova softwa-

⁴Integrated development environment

⁵<https://www.python.org/>

⁶<https://numpy.org/>

⁷<https://pandas.pydata.org/>

⁸<https://matplotlib.org/>

⁹<https://jupyter.org/>

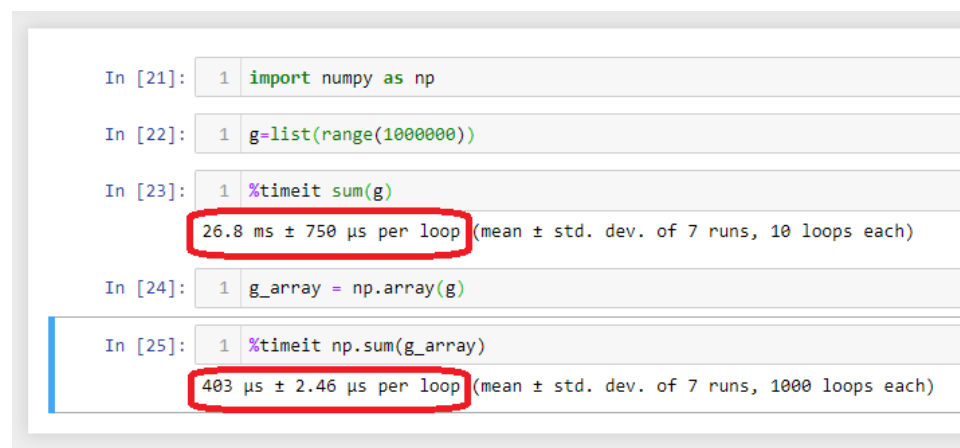
rea greške se mogu dodatno smanjiti. Jedan od poznatijih projekata analize velike količine podataka je zasigurno detekcija gravitacijskih valova nastalih spajanjem dvaju crnih rupa koristeći LIGO teleskop (*Laser Interferometer Gravitational-Wave Observatory*)¹⁰

2.4 NumPy

NumPy je fundamentalni paket za numeričku analizu koristeći Python. Između ostalih funkcija sadrži

- snažan alat za rad s N-dimenzionalnim poljima
- alat za integraciju C/C++ i Fortran programskog koda
- korisne alate za algebarske operacije, Fourierovu analizu i ostale numeričke mogućnosti

Osim što značajno olakšava numerički i statističku analizu skupa podataka NumPy zbog svoje strukture i reprezentacije polja podataka omogućava značajno poboljšanje performansi obrade podataka. Kako bi demonstrirali razliku u performansama između čistog Pythona i NumPy biblioteke možemo napraviti jednostavan eksperiment koji se sastoji od sumiranja elemenata u polju veličine 10^6 elemenata i mjeriti vrijeme potrebno za izvršavanje tog zadatka.



```
In [21]: 1 import numpy as np

In [22]: 1 g=list(range(1000000))

In [23]: 1 %timeit sum(g)
26.8 ms ± 750 μs per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [24]: 1 g_array = np.array(g)

In [25]: 1 %timeit np.sum(g_array)
403 μs ± 2.46 μs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Slika 2.1: Usporedba performansi Pythona i NumPy biblioteke

Na slici 2.1 prikazana je razlika u brzini izvršavanja operacija sumiranja zadanog polja elemenata. Vrijeme potrebno za sumiranje elemenata koristeći samo Python iznosi $26.8ms \pm 750\mu s$ dok koristeći NumPy vrijeme za

¹⁰<https://www.gw-openscience.org/tutorials/>

isti zadatak iznosi samo $403\mu s \pm 2.46\mu s$ ¹¹. Vidljiva je značajna razlika u potrebnom vremenu za izvršavanje zadatka, a iskustva industrije[3] pokazuju još veću razliku kod kompleksnijih zadataka.

2.5 Git

Git¹² je distribuirani sustav za verzioniranje koda i ostalog rada kojeg želimo djeliti sa suradnicima. Git sa svojim jednostavnim i brzim granama omogućava lakši razvoj proizvoda kao i ispitivanje mogućnosti i funkcija. Kada se želi ispitati neka funkcionalnost bez da se ugrozi ono što do sada radi kako trebe nema potrebe kopirati cijeli projekt u novi folder i onda u njemu testirati već se jednostavno kreira nova grana u kojoj se radi razvoj i kada smo sigurni da sve radi kako želimo onda se ta grana ujedini s glavnom granom projekta koja prihvati dodatne funkcionalnosti koje su razvijene za proizvod. Kako je Git lagan za resurse onda se može kreirati vrlo veliki broj grana za razne potrebe bez značajnog utjecaja na performanse razvojnog računala ili potrošnje spremiškog prostora.

S obzirom na distribuiranu narav Gita svaki suradnik koji radi na projektu ima svoju kopiju na kojoj radi te nije vezan za neki server i stalnu komunikaciju s ostatkom tima već je ista potrebna samo kada se povlače i šalju učinjene promjene.

Git je nastao 2005 godine za potrebe razvoja Linux jezgre i od tada je poprimio mnoge simpatije unutar inženjerske zajednice koja ga koristi kako bi zajednički razvijala projekte.

Kako bi se olakšalo djeljenje i suradnja na projektima 2008. godine je pokrenut GitHub - centralno mjesto za usluge poslužitelja¹³ (*hosting*) putem kojeg je moguće pratiti životni ciklus i povijet projekta. Svatko može pronaći projekt koji ga zanima te ukoliko ima dovoljno vremena i znanja može i pridonijeti njegovom razvoju. Brojne kompanije koriste GitHub kao bi podjelile svoje projekte. Podatak od travnja 2019. godine kaže kako više od 2.1 milijuna kompanija i organizacija koristi GitHub. Jedna od njih je i Adafruit - kompanija koja proizvodi elektroničke dodatke za Arduino i druge platforme i fokusirana je na edukacija mladih (i onih koji se tako osjećaju), a njihov GitHub sadrži više od 1100 repozitorija¹⁴. Upravo je njihov GPS Logger Shield korišten u ovom projektu, a dostupnos dokumentacije i dostupna podrška je jedan od glavnih razloga zašto je odlučeno koristiti upravo taj proizvod.

¹¹Rezultati mogu varirati u zavisnosti o korištenom računalu

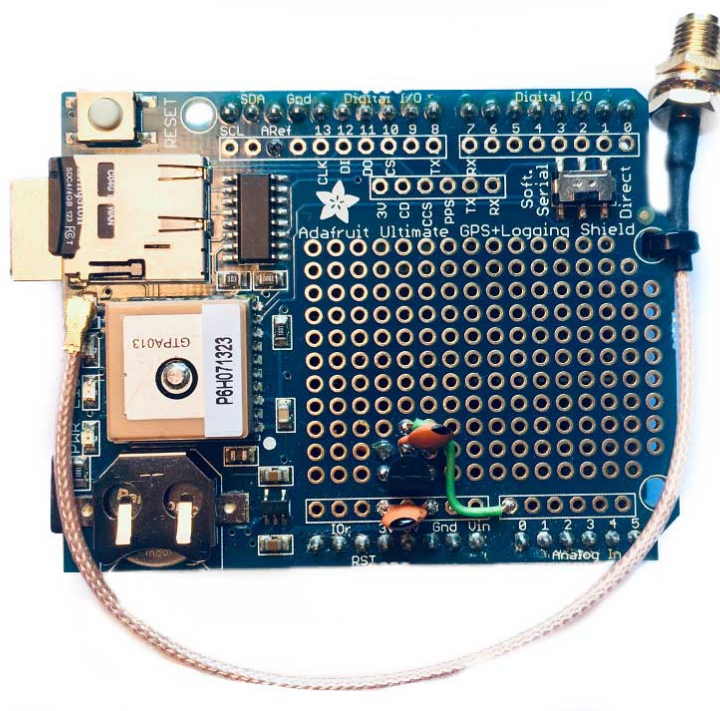
¹²<https://git-scm.com/>

¹³<https://github.com/features>

¹⁴<https://github.com/adafruit/>

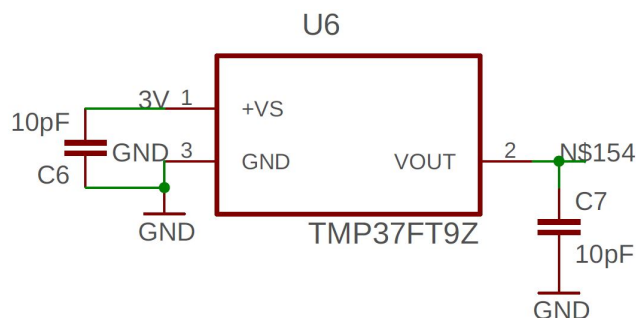
Poglavlje 3

Prikupljanje podataka - hardware



Slika 3.1: Izgled korištenog hardwareskog sklopa

U dodatku A je prikazan kompletan izvorni kod koji se izvršava na Arduino mikrokontroleru.



Slika 3.2: Shema spoja TMP36 senzora

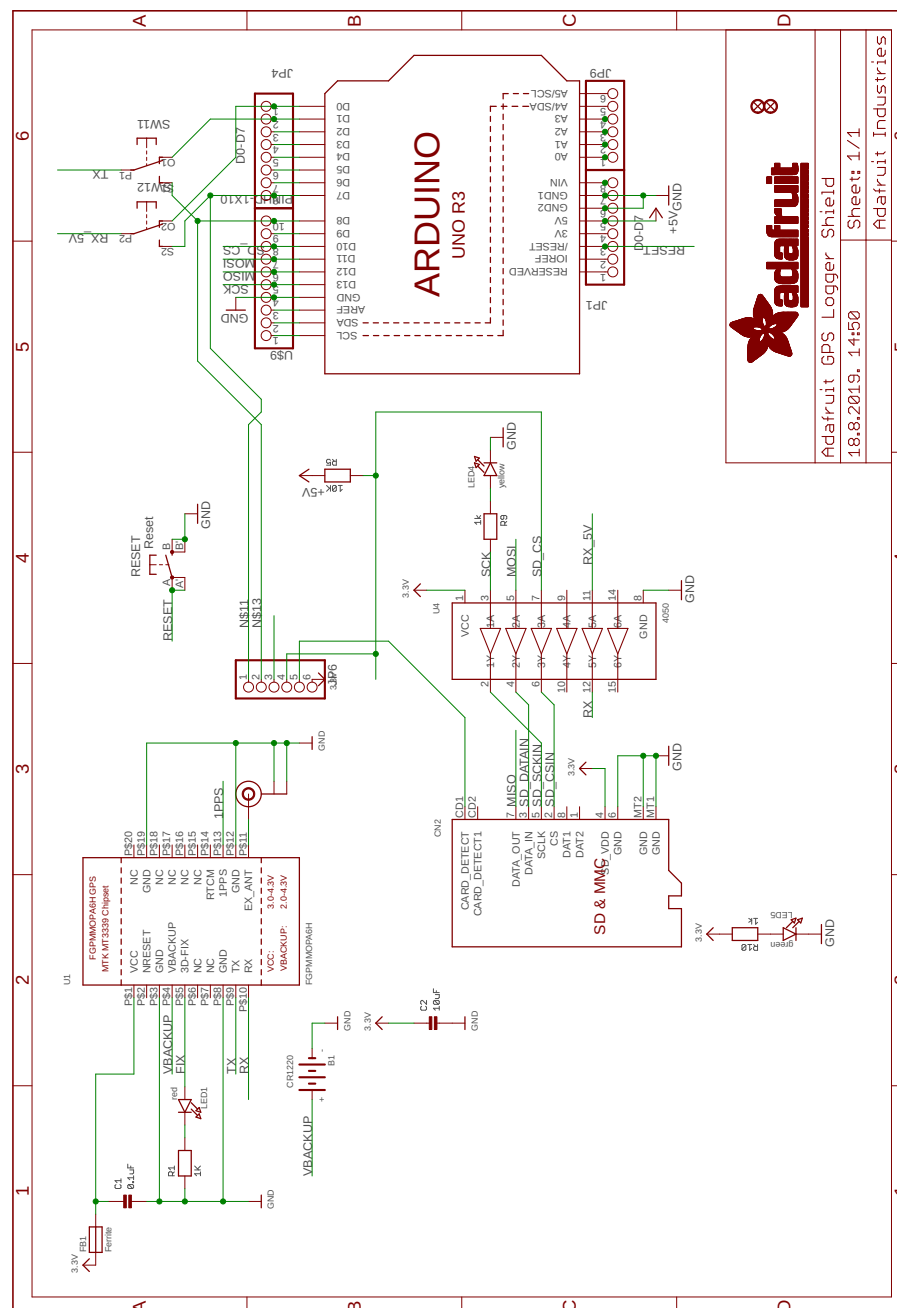
3.1 GPS logging shield

Na shemi 3.3 nalazi se shema gotovog elektroničkog sklopa kako dolazi iz tvornice¹². Na samoj tiskanoj pločici postoji tkz. prototipno područje za dodavanje vanjskih elemenata čiji je raster 2.54 mm koji odgovara standardu *true-hole* elemenata. Na to područje je dodan temperaturni senzor TMP36³ zajedno s dodatnim pasivnim elementima koji služe kao filter smetnji koje se javljaju u radu zbog okoline. Shema spoja je prikazana na slici 3.2.

¹Kompletna dokumentacija dostupna je na <https://learn.adafruit.com/adafruit-ultimate-gps-logger-shield?view=all>

²GitHub repozitorij korištene verzije dostupan na <https://github.com/adafruit/Adafruit-GPS-Logger-Shield-PCB>

³Datasheet dostupan na https://github.com/KristijanCetina/BachelorThesis/blob/master/resources/TMP35_36_37.pdf



Slika 3.3: Shema Adafruit GPS Logger Shield

3.2 Prikupljanje podataka o temperaturi

Kako svaki elektronički sklop ima definirani raspon radne temperature bitno je znati u kojim uvjetima isti se nalazi. Ukoliko je temperatura previsoka može se uključiti aktivno hlađenje ili ako se unaprijed zna da će se sklop nalaziti pod povišenom radnom temperaturom onda se može konstruirati adekvatan sustav hlađenja. Isto vrijedi za prenisku temperaturu. Prema ranije spomenutoj shemi 3.2 dodatn je temperaturni senzor koji mjeri radnu temperaturu okoline uređaja. Pri testiranju ova vrsta senzora se pokazala veoma pouzdana, uz minimalno samozagrijavanje koje bi utjecalo na točnost mjerene veličine ali je isto tako pokazala vrlo brze promjene izlazne vrijednosti koja može biti do vanjskih smetnji. Kako bi se otklonio taj problem primjenjena su dva rješenja. Prvi je hardwareski fiter - kondenzatori koji je prikazan na shemi 3.2, a drugi je softwareski fiter. Tvornički podaci o izlaznom naponu šuma mogu se pronaći u datasheetu uređaja, slika 20. Na slici 3.4 prikazane su izlazne vrijednosti senzora bez ikakvog filtriranja i obrade. frekvencija uzorkovanja je 10Hz (10 očitavanja u sekundi) Svakako nije realno za očekivati da se temperatura mjenja sukladno očitanim vrijednostima.

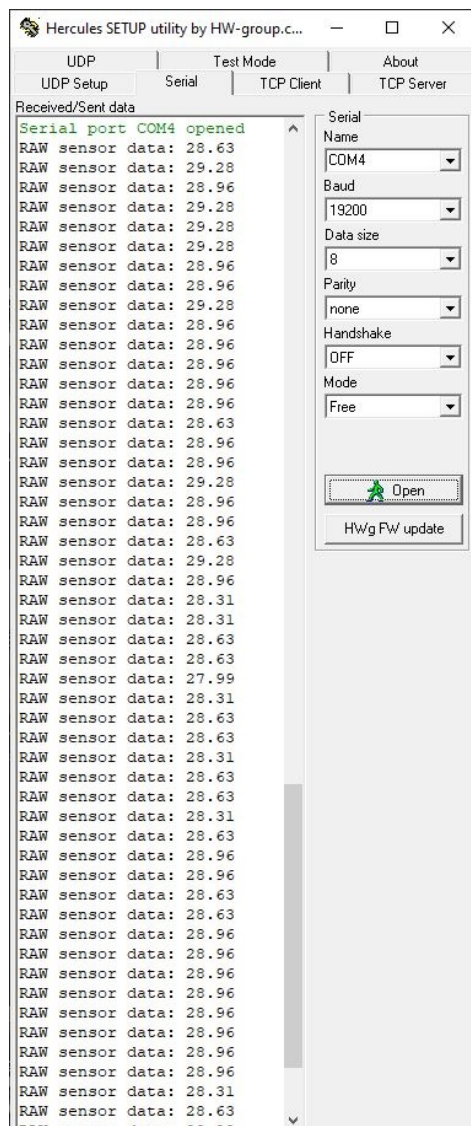
3.2.1 Softwareski filter

Softwareski filter radi na principu da očitava 10 vrijedosti sa senzora te ih sprema u polje. Potom ih sortira po veličini i uzima medijan⁴ vrijednost kao točnu temperaturu. Na taj način se postiže da se eliminiraju sve vrlo visoke i vrlo niske vrijednosti koje se mogu pojaviti zbog šuma u signalu. Vrijednosti se čitaju svakih 100 ms te uz računanje na bazi 10 vrijednosti daje frekvenciju od 1 očitavanja u sekundi koja odgovara i frekvenciji uzorkovanja podataka sa GPS senzora. Prilikom testiranja utvrđeno je da veći broj uzoraka ne doprinosi kvaliteti izmjerenih vrijednosti, a pro manjem broju uzoraka može se potkrasti poneka nerealna vrijednosti. Kako je očekivano vrijeme promjene temperature značajno duže od 1 sekunde onda su prihvaćene navedene vrijednosti i metoda filtriranja. Na slici 3.5 prikazane su izlazne vrijednosti senzora nakon primjene opisanog softwareskog filtera. Primjeti se značajno manje skokova od nečega što se može smatrati stvarna vrijednost.

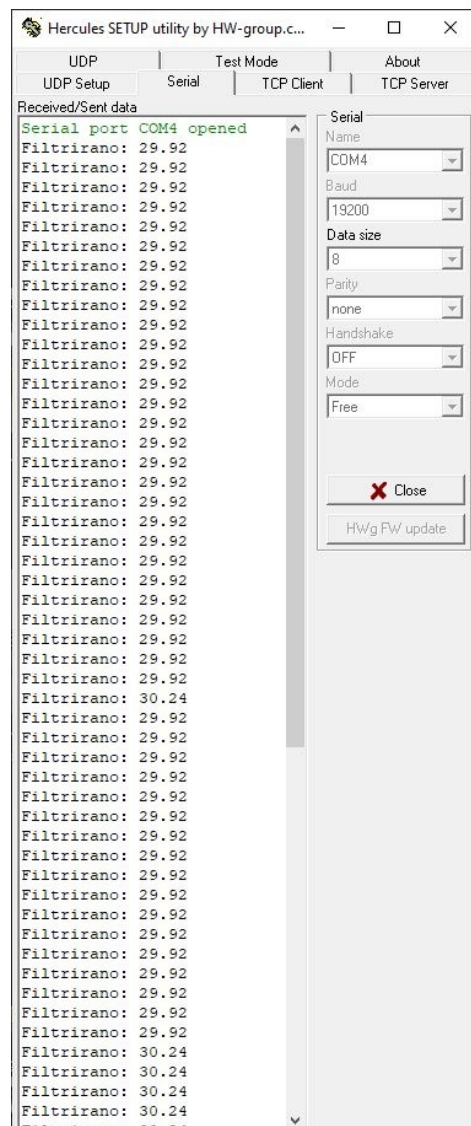
3.2.2 Hardwareski filter

Filter je jednostavna mreža keramičkih kondenzatora vrijednosti $10pF$ koji su spojeni što bliže senzoru između izvoda za napajanje i izlaza senzora prema točki nultog potencijala (*GND*, *masa*) kako bi apsorbirali eventualne smetnje. Iako je softwareski fiter u nekim situacijama dovoljno dobar ovo

⁴Medijan (mediana, centralna vrijednost) je pojam iz statistike koji određuje sredinu distribucije. Pola vrijednosti skupa (distribucije) nalazi se iznad mediane, a pola ispod



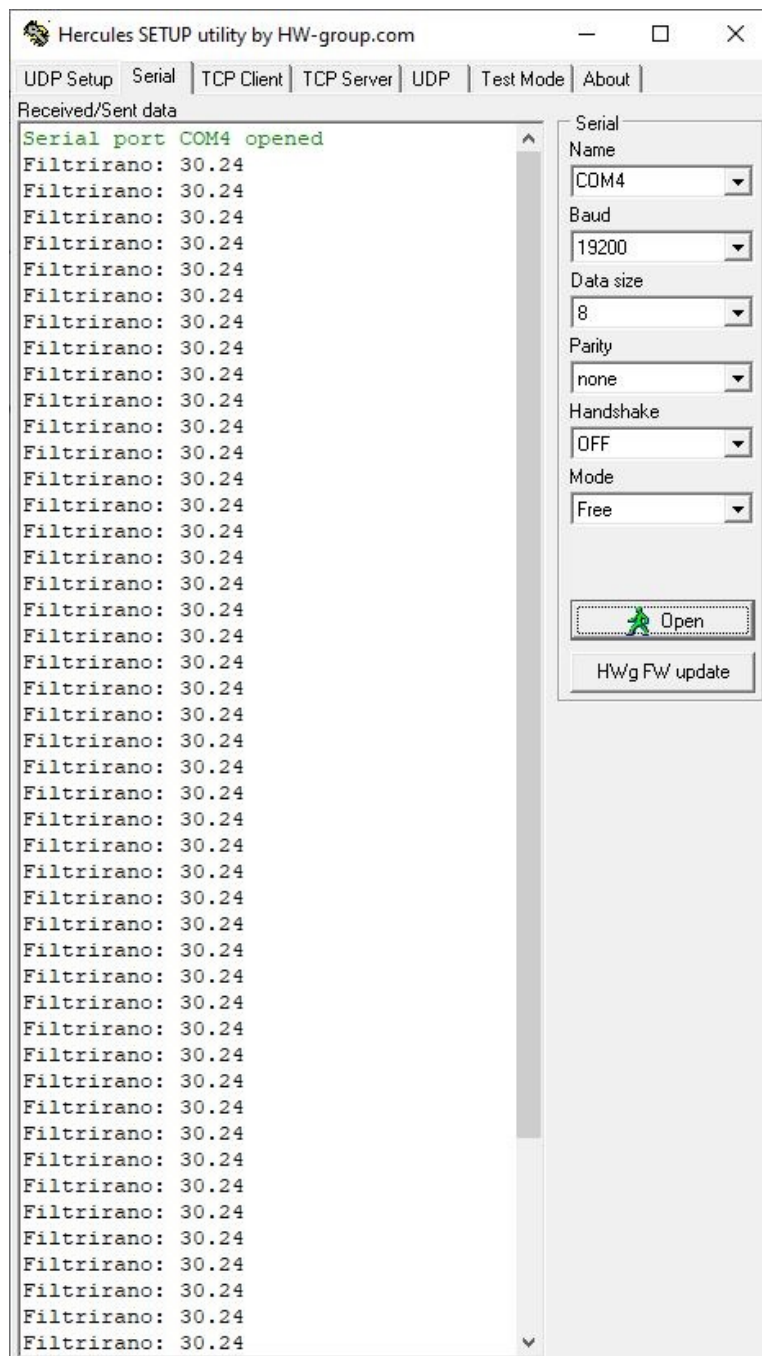
Slika 3.4: Vrijednosti senzora bez filtriranja



Slika 3.5: Vrijednosti senzora sa softwareskim filtriranjem

jednostavno i jeftino rješenje daje dodatan sloj filtriranja koji za posljedicu ima vrlo glatko očitavanje temperature bez skokova u vrijednostima.

Primjenom kombinacije softwareskog i hardwareskog filtriranja postignuta je vrlo zadovoljavajuća karakteristika dobivenih stabilnih vrijednosti bez nereálnih skokova i s vrlo glotkom tranzicijom kod grijanja ili hlađenja sklopa. Dobivene vrijednosti su prikazane na slici 3.6.



Slika 3.6: Vrijednosti senzora primjenom kombinacije Sw i Hw filtera

3.3 Prikupljanje GPS podataka

GPS⁵ je javni sustav u vlasništvu vlade SAD-a⁶ za globalno pozicioniranje baziran na satelitima s atomskim satovima koji odašilju vrlo točno i precizno trenutno vrijeme te su sinkronizirani s zemaljskim satovima. Bilo kakva odstupanja se korrigiraju na dnevnoj bazi. Prijemnik prima signal sa satelita te izračunava točnu poziciju baziranu na poznatoj poziciji satelita i razlikama u primljenim vremenima od svakog satelita. Minimalno su potrebna 3 satelita za dobiti koordinate i 4 satelita za dobiti poziciju o nadmorskoj visini prijemnika.

U ovom radu korišten je GPS chip MTK3339⁷ integriran na prije spomenuti Adafruit Ultimate GPS Logger Shield.

Kao koristan izlaz prijemnik daje NMEA⁸ rečenicu. Ovisno o potrebnim podacima mogu se koristiti razne rečenice, a u ovoj primjeni je korištena \$GPRMC⁹ koja daje minimalne potrebne podatke, a među kojima su vrijeme (UTC) i datum, pozicija i brzina. Primjer \$GPRMC rečenice je

```
$GPRMC,053005.000,A,4457.8784,N,01356.1351,E,36.41,124.90,310719,,A*58
```

pri čemu je:

\$GPRMC	Oznaka rečenice
053005.000	UTC vrijeme (7:30:05 lokalno)
A	Oznaka valjanosti, A = OK, V = warning
4457.8784,N	Zemljopisna širina
01356.1351,E	Zemljopisna dužina
36.41	Brzina u čvorovima ($\approx 67km/h$)
124.90	Smjer kretanja
310719	Datum (31. srpnja 2019.)
A*58	Checksum (kontrolni broj)

Prilikom provjere primljenih podataka obavezno se provjerava

- da li primljeni checksum odgovara izračunanom za datu rečenicu kako bi se izbjegle pogreške u komunikaciji,
- da li je oznaka valjanosti A što znači da uređaj ima prijem s dovoljnog broj satelita da se može vjerovati primljenim podacima.

Provjeru valjanosti i checksuma odrađuje biblioteka koja je dostupna za Arduino platformu zajedno s ostalom dokumentacijom uređaja te nije bilo potrebno pisati poseban kod koji će to raditi.

⁵Global Positioning System - Sustav globalnog pozicioniranja

⁶<https://www.gps.gov>

⁷<https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPM0PA6C-Datasheet-VOA-Preliminary.pdf>

⁸https://www.nmea.org/content/STANDARDS/NMEA_0183_Standard

⁹<http://aprs.gids.nl/nmea/>

3.4 Spremanje podataka na memorijsku karticu

Na korištenom Adafruit Ultimate GPS Logger Shieldu postoji utor za microSD memorijsku karticu koja se koristi za zapisivanje prikupljenih podataka kako bi se isti mogli kasnije obraditi i prikazati. Sustav skupljene podatke sprema na memorijsku karticu u .csv¹⁰ formatu koji je pogodan za kasniju obradu bilo putem Excel programskog alata ili drugih alata za obradu i vizualizaciju podataka. Svaki red predstavlja jedan zapis, a u odnosu na ranije prikazanu \$GPRMC rečenicu na kraju je dodan i podatak o trenutnoj temperaturi u °C koja je očitana sa senzora opisanog u poglavlju 3.2. Frekvencija zapisivanja podatka je postavljena na 1 zapis u sekundi. Datoteka se automatski kreira prilikom uključivanja sklopa ako je SD kartica prisutna. Ime datoteke je GPSLOGXX.csv pri čemu je XX broj koji počinje od 00 i uvećava je za 1 kod svakog pokretanja. Testiranje je pokazalo da veličina datoteke s 10 sati (≈ 36000 zapisa) snimljenih podataka iznosi otprilike 2.7 Mb.

¹⁰Comma Separated Values

Poglavlje 4

Obrada podataka - software

Kako je već spomenuto u uvodi i opisu korištenih tehnologija analiza podataka izrađena je koristeći Python programski jezik unutar Jupyter Notebook interaktivne bilježnice uz dodatak Numpy biblioteke za numeričku analizu i Matplotlib biblioteke za izradu grafova.

Kao prvi primjer obrade podataka izrađena je analiza kretanja vozila i vizualizirani prikupljeni podaci. U dodatku B prikazana je kompletna analiza podataka te demonstriran završni proizvod koji može biti dalje distribuiran bilo u printanoj verziji ili preferirano u digitalnom obliku koji onda omogućava daljnji njad na istome.

Koristeći razne tipove ćelija unutar Jupyter notebooka stvorena je prikazana analiza. Tip ćelije *Markdown* podržava formatiranje i snitaksu `.md` Markdown datoteke¹, a one se koriste za opisni dio analize kako bi čitatelj znao o čemu se radi bez potrebe za proučavanjem programskog koda.

Tip ćelije *Code* omogućava unošenje i izvršavanje programskog koda koji će napraviti neku željenu radnju. Ćelije se ogu izvršavati jedno po jedna ili sve ćelije u sekvencijalnom nizu.

Na kraju postoje i *Output* ćelije koje prikazuju izlazni rezultat izvršene *code* ćelije.

Kako bi mogli koristiti navedene dodatne module u Python ih je prvo potrebno uvesti

```
import pandas as pd
import matplotlib.pyplot as plt
from numpy import genfromtxt, arange, sin, pi
from matplotlib import style
from matplotlib import dates as mpl_dates
import numpy as np
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
```

¹<https://www.markdownguide.org/>

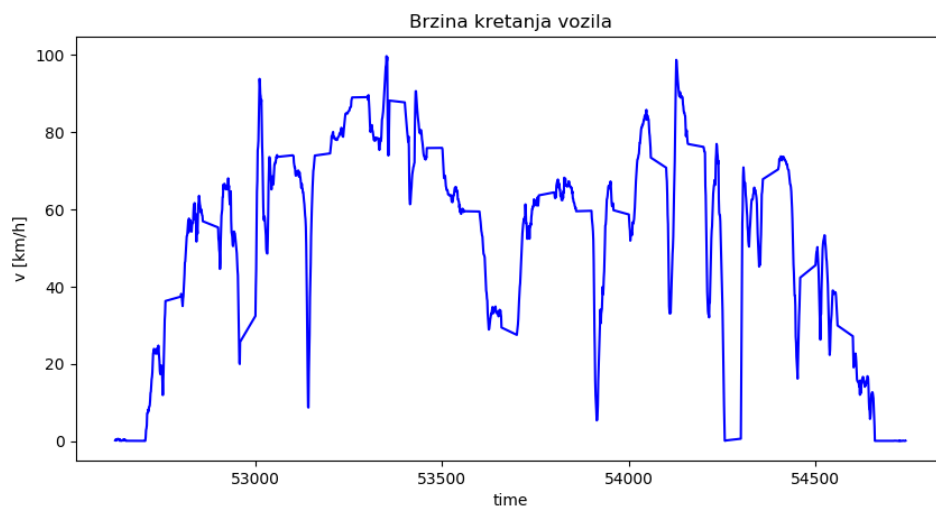
Konvencija je da se NumPy biblioteka skraćeno imenuje `np` kao bi se olakšalo kasnije korištenje u radu. Isto vrijedi za ostale često korištene biblioteke.

Nakon toga je potrebno navesti koja datoteka s podacima se koristi i nazvati kolone podataka

```
filename='GPSLOG10.CSV'
data=pd.read_csv(filename, header=None, delimiter=',',
names=['Sentence', 'Time', 'Validity', 'Latitue', 'NS', 'Longitude', 'EW',
'Speed', 'Direction', 'Date', 'NA1', 'NA2', 'Checksum', 'Temperature'])
```

Pri čemu je `GPSLOG10.CSV` ime datoteke u kojem su spremljeni podaci.

Nakon uvodnih radnji sada smo spremni započeti s analizom podataka i vizualizcijom onih koje nas zanimaju. U ovisnosti o željenim rezultatima ovisiti će i potrebne operacije koje je potrebno poduzeti. Nekada su podaci već u takvom formatu da se mogu odmah koristiti, dok je ponekada potrebno napraviti veću ili manju manipulaciju nad njima kako bi bili pogodni za korištenje. U svakom slučaju vrlo je bitno poznavati strukturu seta podataka koji se koristi kako bi se izbjegle greške zbog njegovog nepoznavanja ili korištenja krive mjerne jedinice. 1999. godine NASA je izgubila 125 milijuna vrijedan *Mars Climate Orbiter* zbog previda pretvorbe jedinica[4] što samo pokazuje kako to realna zamka u koji u najbolji svjetski stručnjaci mogu upasti. U ovom slučaju brzina je prikazana u čvorovima što odgovara prijeđenom putu od jedne nautičke milje u sat vremena. Ukoliko želimo prikazati brzinu u *km/h* potrebno je izvršiti pretvorbu jedinica. Jedna nautička milja iznosi 1852 *m* što znači da zapisanu brzinu treba pomnožiti s 1.852 kako bi dobili *km/s*.



Slika 4.1: Izrađeni graf kretanja vozila

Kako je uvijek korisno vizualizirati kretanje veličine na grafu tako je i u ovom radu napravljen graf promjene brzine u vremenu prikazan na 4.1. Koristeći Matplotlib je to vrlo jednostavno

```
plt.plot(data['Time'], data['Speed']*1.852, 'b-')
```

Pri čemu *data['Time']* predstavlja apcisu na grafu, a *data['Speed']* predstavlja ordinatu prikazanog grafa. 'b-' je parametar kojim je definirana linija - linija plave boje. Svaki graf treba biti propisno obilježen tako su i na ovom grafu označene vrijednosti i imena osi, prikazane mjerne jedinice i dodana legenda iako je prikazana samo jedna veličina.

Dodatno iz dostupnih podataka može se napraviti graf kretanja temperature koji je prikazan u kompletnoj analizi u dodatku B. Osim podataka koji se mogu prikazati grafički možemo pogledati i ostale statističke podatke kao što su minimalna i maksimalna temperatura tokom promatranog razdoblja.

```
print ('Minimalna temperatura: _', np.min(data['Temperature']), 'C')
print ('Maksimalna temperatura: _', np.max(data['Temperature']), 'C')
```

Ovdje su korištene ugrađene funkcije `min` i `max` NumPy biblioteke koje za parametar uzimaju set podataka temperature.

Ako nas zanima prosječna temperatura tomok promatranog razdoblja onda i za to postoji urađena funkcija koja nam olakšava rad umjesto da zbrajamo sve elemente te zbroj djelimo s brojem elemenata jednostavno možemo koristiti

```
print ('Prosjecna temperatura: _', '
{:.2f}'.format(np.mean(data['Temperature'])), 'C',
' {:.2f}'.format(np.std(data['Temperature'])), 'C')
```

Pri čemu `np.mean` daje prosječnu temperaturu, a `np.std` daje standardnu devijaciju vrijednosti. `' {:.2f}'.format` na opcija formatiranja stringa pomoću koje se prikazuje dva decimalna mjesta.

Poglavlje 5

Zaključak

Ovdje dođe zaključak

Literatura

- [1] R. H. Inc., “What is open source?.” <https://opensource.com/resources/what-open-source>. (3.8.2019.).
- [2] R. R. Panko, “What we know about spreadsheet errors,” *Journal of Organizational and End User Computing (JOEUC)*, vol. 10, no. 2, pp. 15–21, 1998. (3.8.2019.).
- [3] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, “The numpy array: a structure for efficient numerical computation,” *Computing in Science & Engineering*, vol. 13, no. 2, p. 22, 2011.
- [4] B. J. Sauser, R. R. Reilly, and A. J. Shenhar, “Why projects fail? how contingency theory can provide new insights—a comparative analysis of nasa’s mars climate orbiter loss,” *International Journal of Project Management*, vol. 27, no. 7, pp. 665–679, 2009.
- [5] S. Tosi, *Matplotlib for Python developers*. Packt Publishing Ltd, 2009.
- [6] “Matplotlib homepage.” <https://matplotlib.org>. (3.8.2019.).
- [7] “Numpy homepage.” <https://numpy.org/>. (3.8.2019.).
- [8] “Git homepage.” <https://git-scm.com>. (3.8.2019.).
- [9] “Github homepage.” <https://github.com/>. (3.8.2019.).
- [10] “Python homepage.” <https://www.python.org>. (3.8.2019.).

Dodatak A

Programski kod na Arduino
mikroračunalu

```
1 #include <SPI.h>
2 #include <Arduino.h>
3 #include <Adafruit_GPS.h>
4 #include <SoftwareSerial.h>
5 #include <SD.h>
6 #include <avr/sleep.h>
7 #include <Wire.h>          // this #include still required because the RTCLib ↗
    depends on it
8 #include "RTCLib.h"
9
10 int voltage;
11 int Temperatura;
12 int TempSenzor = A0;
13
14 #define aref_voltage 3.3
15 float temp[10] = { 0 };
16
17 float i = 0;
18 int n = 0;
19 float median = 0;
20
21 SoftwareSerial mySerial(8, 7);
22 Adafruit_GPS GPS(&mySerial);
23
24 // Set GPSECHO to 'false' to turn off echoing the GPS data to the Serial ↗
    console
25 // Set to 'true' if you want to debug and listen to the raw GPS sentences
26 #define GPSECHO false
27 /* set to true to only log to SD when GPS has a fix, for debugging, keep it ↗
    false */
28 #define LOG_FIXONLY true
29
30 // this keeps track of whether we're using the interrupt
31 // off by default!
32 #ifndef ESP8266 // Sadly not on ESP8266
33 boolean usingInterrupt = false;
34 #endif
35
36 // Set the pins used
37 #define chipSelect 10
38 #define ledPin 13
39
40 File logfile;
41
42 RTC_DS1307 RTC; // define the Real Time Clock object
43 RTC_Millis rtc;
44
45 char timestamp[30];
46 // call back for file timestamps
47 void dateTime(uint16_t* date, uint16_t* time) {
48     DateTime now = RTC.now();
49     sprintf(timestamp, "%02d:%02d:%02d %2d/%2d/%2d \n", now.hour(),now.minute ↗
        (),now.second(),now.month(),now.day(),now.year()-2000);
50     Serial.println("yy");
51     Serial.println(timestamp);
52     // return date using FAT_DATE macro to format fields
```

```
53  *date = FAT_DATE(now.year(), now.month(), now.day());
54
55  // return time using FAT_TIME macro to format fields
56  *time = FAT_TIME(now.hour(), now.minute(), now.second());
57  }
58
59  // read a Hex value and return the decimal equivalent
60  uint8_t parseHex(char c) {
61      if (c < '0')
62          return 0;
63      if (c <= '9')
64          return c - '0';
65      if (c < 'A')
66          return 0;
67      if (c <= 'F')
68          return (c - 'A')+10;
69  }
70
71  // blink out an error code
72  void error(uint8_t errno) {
73      /*
74      if (SD.errorCode()) {
75          putstring("SD error: ");
76          Serial.print(card.errorCode(), HEX);
77          Serial.print(',');
78          Serial.println(card.errorData(), HEX);
79      }
80      */
81      while(1) {
82          uint8_t i;
83          for (i=0; i<errno; i++) {
84              digitalWrite(ledPin, HIGH);
85              delay(100);
86              digitalWrite(ledPin, LOW);
87              delay(100);
88          }
89          for (i=errno; i<10; i++) {
90              delay(200);
91          }
92      }
93  }
94
95  void setup() {
96      Wire.begin();
97      if (!RTC.begin()) {
98          Serial.println("RTC failed");
99          while(1);
100  };
101  // connect at 115200 so we can read the GPS fast enough and echo without
102  // dropping chars
103  // also spit it out
104  Serial.begin(115200);
105  Serial.println("\r\nUltimate GPSlogger Shield");
106  pinMode(ledPin, OUTPUT);
107  pinMode(TempSenzor, INPUT); //postavi izvod TempSenzor (A0) kao ulazni
108  analogReference(EXTERNAL); // Koristim 3.3 Vref
```

```
108
109 // make sure that the default chip select pin is set to
110 // output, even if you don't use it:
111 pinMode(10, OUTPUT);
112
113 if (!SD.begin(chipSelect)) {
114     Serial.println("Card init. failed!");
115     error(2);
116 }
117 char filename[15];
118 strcpy(filename, "GPSLOG00.csv");
119 for (uint8_t i = 0; i < 100; i++) {
120     filename[6] = '0' + i/10;
121     filename[7] = '0' + i%10;
122     // create if does not exist, do not open existing, write, sync after write
123     if (! SD.exists(filename)) {
124         break;
125     }
126 }
127
128 logfile = SD.open(filename, FILE_WRITE);
129 if( ! logfile ) {
130     Serial.print("Couldnt create ");
131     Serial.println(filename);
132     error(3);
133 }
134 Serial.print("Writing to ");
135 Serial.println(filename);
136
137 // connect to the GPS at the desired rate
138 GPS.begin(9600);
139
140 // uncomment this line to turn on RMC (recommended minimum) and GGA (fix
141 // data) including altitude
142 //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
143 // uncomment this line to turn on only the "minimum recommended" data
144 GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMONLY);
145 // Set the update rate
146 GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ); // 100 millihertz (once every
147 // 10 seconds), 1Hz or 5Hz update rate
148 // Turn off updates on antenna status, if the firmware permits it
149 GPS.sendCommand(PGCMD_NOANTENNA);
150 // the nice thing about this code is you can have a timer0 interrupt go off
151 // every 1 millisecond, and read data from the GPS for you. that makes the
152 // loop code a heck of a lot easier!
153 #ifndef ESP8266 // Not on ESP8266
154     useInterrupt(true);
155 #endif
156 Serial.println("Ready!");
157 }
158
159 // Interrupt is called once a millisecond, looks for any new GPS data, and
160 // stores it
161 #ifndef ESP8266 // Not on ESP8266
162 ISR(TIMER0_COMPA_vect) {
163     char c = GPS.read();
164 }
```



```
161 // if you want to debug, this is a good time to do it!
162 #ifdef UDR0
163     if (GPSECHO)
164         if (c) UDR0 = c;
165         // writing direct to UDR0 is much much faster than Serial.print
166         // but only one character can be written at a time.
167 #endif
168 }
169
170 void useInterrupt(boolean v) {
171     if (v) {
172         // Timer0 is already used for millis() - we'll just interrupt somewhere
173         // in the middle and call the "Compare A" function above
174         OCR0A = 0xAF;
175         TIMSK0 |= _BV(OCIE0A);
176         usingInterrupt = true;
177     }
178     else {
179         // do not call the interrupt function COMPA anymore
180         TIMSK0 &= ~_BV(OCIE0A);
181         usingInterrupt = false;
182     }
183 }
184 #endif // ESP8266
185
186 // function to sort the array in ascending order
187 void Array_sort(float *array, int n)
188 {
189     // declare some local variables
190     int i = 0, j = 0, temp = 0;
191     for (i = 0; i < n; i++)
192     {
193         for (j = 0; j < n - 1; j++)
194         {
195             if (array[j] > array[j + 1])
196             {
197                 temp = array[j];
198                 array[j] = array[j + 1];
199                 array[j + 1] = temp;
200             }
201         }
202     }
203 }
204
205 float Find_median(float array[], int n)
206 {
207     float median = 0;
208     // if number of elements are even
209     if (n % 2 == 0)
210         median = (array[(n - 1) / 2] + array[n / 2]) / 2.0;
211     // if number of elements are odd
212     else
213         median = array[n / 2];
214     return median;
215 }
216
```

```
217 void loop(){
218   DateTime now = rtc.now();
219   if (! usingInterrupt) {
220     // read data from the GPS in the 'main loop'
221     char c = GPS.read();
222     // if you want to debug, this is a good time to do it!
223     if (GPSECHO)
224       if (c) Serial.print(c);
225   }
226
227   // if a sentence is received, we can check the checksum, parse it...
228   if (GPS.newNMEAReceived()) {
229     char *stringptr = GPS.lastNMEA();
230
231     if (!GPS.parse(stringptr)) // this also sets the newNMEAReceived() flag ↗
232       return; // we can fail to parse a sentence in which case we should just ↗
233       wait for another
234
235     // Sentence parsed!
236     Serial.println("OK");
237     if (LOG_FIXONLY && !GPS.fix) {
238       Serial.print("No Fix");
239       return;
240     }
241
242     float voltage = analogRead(TempSenzor) * 3.3; //ocitava vrijednosti ↗
243     // izvoda (A0)
244     voltage /= 1024.0; //10bit ADC
245     float Temperatura = (voltage - 0.5) * 100;
246     Serial.print("Trenutno: ");
247     Serial.println(Temperatura);
248
249     // Rad. lets log it!
250     Serial.println("Log");
251
252     char tempBuff[5];
253     dtostrf(Temperatura,0,2,tempBuff);
254     uint8_t tempSize = strlen(tempBuff);
255
256     //logfile.flush();
257
258     // ovaj blok kao dela pa pomalo s tim :-)
259     uint8_t stringsize = strlen(stringptr); // + tempSize;
260     if (stringsize != logfile.write((uint8_t *)stringptr, ↗
261       stringsize)) //write the string to the SD file
262       error(4);
263     if (strstr(stringptr, "RMC") || strstr(stringptr, "GGA") ) logfile.flush ↗
264     ();
265     logfile.write(tempBuff);
266     Serial.println();
267   }
268 }
```

Dodatak B

Analiza podataka kretanja vozila

Analiza Kretanja Vozila

August 29, 2019

1 Plot grafa podataka iz .csv filea

Demo kako uz pomoc pythona i *matplotlib* biblioteke za prikaz grafova

Prvo uvezemo potrebne biblioteke

```
[7]: import pandas as pd
import matplotlib.pyplot as plt
from numpy import genfromtxt, arange, sin, pi
from matplotlib import style
from matplotlib import dates as mpl_dates
import numpy as np
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
```

Unese se ime datoteke s podacima i mapiraju se polja sukladno zapisanome.

U ovom primjeru podaci su razdvojeni s znakom ',' ali cesti je slucaj kada su podaci odvojeni nekim drugim znakom te se to treba posebno naznaciti kako bi program znao granice izmedu polja.

```
[8]: filename='GPSLOG10.CSV'
#plt.style.use('ggplot')
data=pd.read_csv(filename, header=None, delimiter=',',
    →names=['Sentence', 'Time', 'Validity', 'Latitue', 'NS', 'Longitude', 'EW', 'Speed',
    →'Direction', 'Date', 'NA1', 'NA2', 'Checksum', 'Temperature'])
```

Sada smo spremni za prikazati prikupljene podatke.

Prvo mozemo prikazati jednostavan s/t graf - brzinu u vremenu. Kako je brzina zapisana u cvorovima, a mi je zelimo prikazati u km/h potrebno izvrstiti konverziju. 1 nauticna milja odgovara 1.852 km.

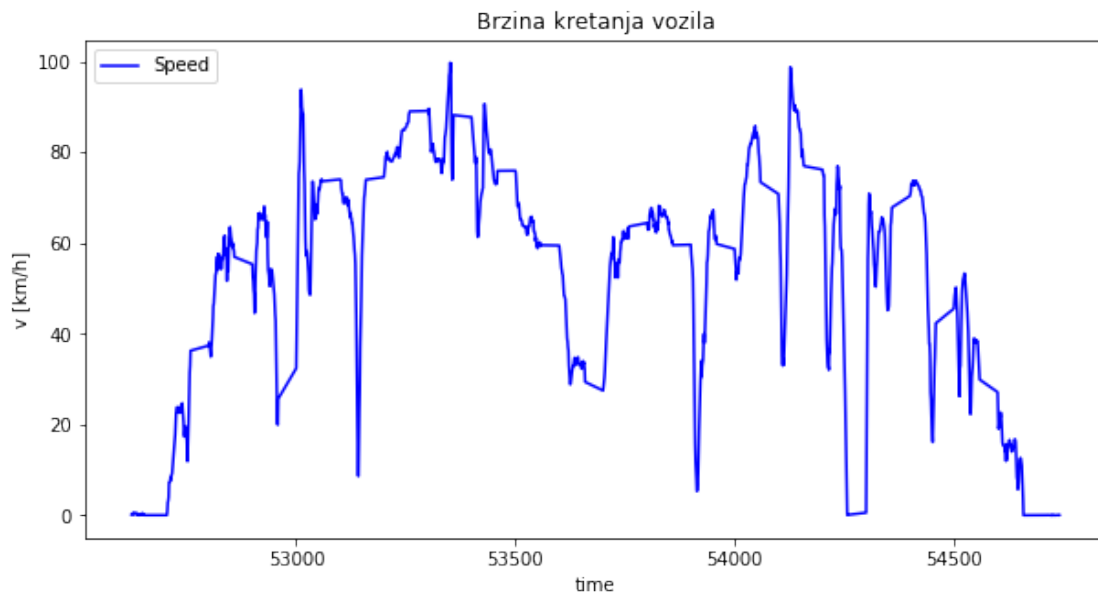
Svaki graf treba imati oznacene osi. S komandom plt.xlabel i ylabel oznacili smo osi grafa i analogno tome imenovan je i graf kako bi citatelj znao to graf predstavlja. Naravno, pojedinačni grafovi se mogu posebno spremiti u visokoj rezoluciji i zeljenom formatu za kasniju upotrebu.

```
[9]: #otvori graf u novom prozoru
#%matplotlib qt
#plt.plot(data['Time'],data['Speed']*1.852, 'b-')
plt.plot(data['Time'],data['Speed']*1.852, 'b-')

plt.legend(loc='upper left')
```

```
plt.xlabel ('time')
plt.ylabel ('v [km/h]')
plt.title('Brzina kretanja vozila')
#plt.savefig('GrafKretanjaBrzineVozila.png',format='png', bbox_inches='tight',
→dpi=100)
```

[9]: Text(0.5, 1.0, 'Brzina kretanja vozila')



Dodatno se mogu izracunati i pogledati razni podaci koje nas zanimaju.

Ako npr. zelimo znati koja je bila maksimalna brzina kojom se vozilo kretalo to se moze vidjeti na sljedeci nacin:

```
[10]: print('Maksimalna brzina = ',np.max(data['Speed']*1.852) , 'km/h')
```

Maksimalna brzina = 99.73020000000001 km/h

Ako nas zanimaju podaci o temperaturi moguće je čak koristiti i ugrađene statističke funkcije za izracunati zeljene podatke

```
[11]: # prikazi graf inline
%matplotlib inline
plt.rcParams["figure.figsize"] = (10,5)
plt.plot(data['Time'],data['Temperature'], 'g', linewidth=2)

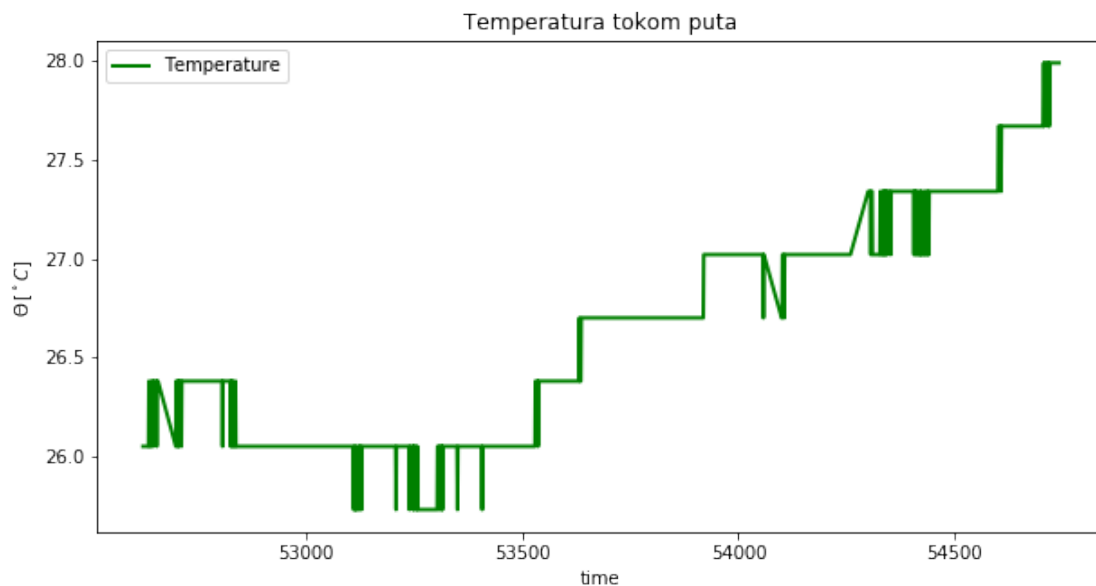
plt.legend(loc='upper left')
plt.xlabel ('time')
plt.ylabel ('$\\Theta \\, , [^\\circ C]$')
plt.title('Temperatura tokom puta')
```

```

print ('Minimalna temperatura: ', np.min(data['Temperature']), 'řC')
print ('Maximalna temperatura: ', np.max(data['Temperature']), 'řC')
print ('Razlika temperature: ', np.max(data['Temperature']) - np.
    ↳min(data['Temperature']), 'řC')
print ('Razlika temperature: ', '{:.2f}'.format(np.ptp(data['Temperature'])), 'řC')
print ('Prosječna temperatura: ', '{:.2f}'.format(np.
    ↳mean(data['Temperature'])), 'řC š ',
    '{:.2f}'.format(np.std(data['Temperature'])), 'řC')

```

Minimalna temperatura: 25.73 řC
 Maximalna temperatura: 27.99 řC
 Razlika temperature: 2.2599999999999998 řC
 Razlika temperature: 2.26 řC
 Prosječna temperatura: 26.64 řC š 0.57 řC



Ako elimo plotati ove dvije veliine na istom grafu da vizualno utvrdimo postojanje korelacije moemo kreirati subplot

```

[27]: fig, ax1 = plt.subplots()
      plt.title('brzina i temperatura tokom puta')

      l1,=ax1.plot(data['Time'],data['Speed']*1.852, 'b-', label='Speed')
      ax1.set_xlabel('time')
      ax1.set_ylabel('v [km/h]', color='b')
      ax1.tick_params('y', colors='b')

```

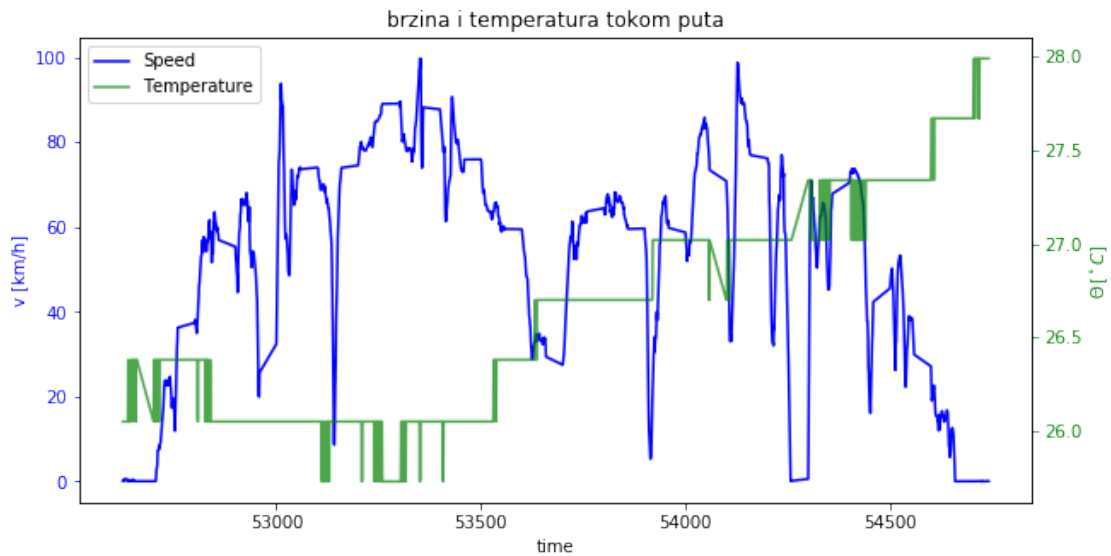
```

ax2 = ax1.twinx()
l2,=ax2.plot(data['Time'],data['Temperature'], 'g', alpha=0.7,
→label='Temperature')
ax2.set_ylabel('$\Theta$, [^\circ C]$', color='g')
ax2.tick_params('y', colors='g')

plt.legend([l1, l2],["Speed", "Temperature"])

```

[27]: <matplotlib.legend.Legend at 0x2027ec20f98>



Uvidom u graf ne mozemo vizualno utvrditi postoje zavisnosti jedve velicine o drugoj, ali se moze primjetiti trend porasta temperature s vremenom. Koji je tocan uzrok tome treba dodatno istraziti sto nije predmet ovog rada.