ISTARSKO VELEUČILIŠTE
UNIVERSITÀ ISTRIANA DI SCIENZE APPLICATE
Stručni studij politehnike

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

Završni rad

Kristijan Cetina
JMBAG: 2424011721
kcetina@iv.hr

Pula, 14. rujna 2019.

**Sažetak**

Prikupljanje, obrada i vizualizacija podataka nalazi se na svakom koraku našega života. U ovom radu prikazan je postupak izrade snimača podataka koristeći Arduino platformu te obrada podataka koristeći isključivo alate otvorenog i slobodnog koda. Hardwareska platforma trenutno koristi GPS prijemnik i temperaturni senzor te se u budućnosti planira povećanje broja senzora. Podatci koji se skupljaju na uređaju spremaju se lokalno na memorijsku karticu ne bi li se kasnije analizirali na računalu pri čemu se koristi programski jezik Python unutar Jupyter notebook okruženja uz pomoć Pandas, NumPy i Matplotlib biblioteka. Izrađeni alat za analizu radi na svim popularnim operacijskim sustavima danas u upotrebi.

U duhu otvorenog koda i dobre prakse znanstvene zajednice ovaj rad je napisan koristeći LaTeX sustav te je kompletan rad *hostan* na GitHub-u i slobodano dostupan svim zainteresiranim stranama koji žele nastaviti rad na ovoj temi.

**Ključne riječi** *Arduino, Python, Matplotlib, open-source, obrada podataka*

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

**Sommario**

La raccolta, l'elaborazione e la visualizzazione dei dati è presente in ogni aspetto della nostra vita. In questa tesi è documentata la creazione di uno strumento per la raccolta dati creato su piattaforma Arduino, e l'elaborazione dei dati raccolti usando esclusivamente strumenti *Open Source*. La piattaforma hardware attuale usa un ricevitore GPS ed un sensore di temperatura mentre per future evoluzioni è in programma l'incremento del numero di sensori. I dati raccolti dallo strumento vengono immagazzinati sulla scheda di memoria per poi essere analizzati sul computer. Per l'analisi dei dati viene usato il linguaggio di programmazione Python sull' ambiente di programmazione Jupyter notebook, con l'ausilio di biblioteche Pandas, NumPy e Matplotlib. Lo strumento qui descritto è compatibile con tutti i sistemi operativi piu popolari oggi in uso.

Nello spirito della filosofia Open Source e della cultura scientifica questa tesi e stata scritta usando il sistema LaTeX ed è disponibile liberamente su GitHub per tutti coloro che fossero interessati a continuare il lavoro fatto su questo tema.

**Parole chiave:** *Arduino, Python, Matplotlib, open-source, analisi dati*

**Abstract**

Data acquisition, processing, and visualization are at every step of our lives. This paper describes how to record data using an Arduino and analyze them using free and open-source (FOSS) tools exclusively. The hardware platform is currently using a GPS receiver and a temperature sensor with the planned increase in the number of sensors in the future. The data collected on the devices is stored locally to a memory card and subsequently analyzed on a computer. The Python programming language within the Jupyter notebook environment was used for analysis with the help of the Pandas, NumPy and Matplotlib libraries. The created analysis tool works on all today popular operating systems.

In the spirit of the open-source and good practice of the scientific community, this paper was written using the LaTeX system and a complete project is hosted on GitHub freely available to all interested parties who wish to work on these topic.

**Keywords:** *Arduino, Python, Matplotlib, open-source, data analysis*

*Mojoj mami Vlasti*

# Zahvala

Zahvaljujem svojoj mentorici Sanji Grbac Babić, mag. računarstva, višoj predavačici na izdvojenom vremenu i podršci, kako na izradi ovog rada, tako i tijekom cijelog studiranja na Politehnici. Svojm smirenim pristupom uvelike je olakšala proces učenja i rada, bilo da se radilo o stručnim tehničkim pitanjima ili ostalim izazovima s kojima se studenti susreću.

Zahvaljujem se i svojim timskim kolegama, Stjepanu Grginu i Igoru Mrkiću, s kojima sam od samog početka sudjelovao na svim timskim zadatcima i njihovoj pomoći pri individualnom radu. Team *One* rocks.

Zahvaljujem se i svim ostalim profesorima i djelatnicima Politehnike koja je tokom studija narasla i preimenovana u Istarsko Veleučilište - Università Istriana Di scienze applicate na nesebičnoj potpori kada je god to bilo potrebno. Vi činite ovu ustanovu ono što ona je.

Naposljetku veliko hvala mojoj obitelji na potpori i razumijevanju tijekom mojeg ponovnog studiranja. Draga mama, iako više nisi s nama znam da bi bila sretna. Neizmjerno Ti hvala na svemu.

> "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 bazirana 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 pisan je u duhu hrvatskoga jezika.

Student: Kristijan Cetina

_____

# Sadržaj

# Popis slika

# Poglavlje 1

# Uvod i opis zadatka

Tema ovog rada proizašla je iz autorove želje za proučavanjem tematike te kao gorljivim poklonikom metode učenja kroz praktičan rad i primjenu stečenog znanja i iskustva na rješavanje realnog problema. Dodatno, autorova je želja implementirati stečena znanja i razvijeni hardware i software u budućim projektima koje ima na umu. Koristeći dostupna i otvorena znanja značajno je lakše razvijati projekte, a otvarajući rad ostalim zainteresiranom stranama, omogućava se kolaboracija na usavršavanju projekta ili stvaranju baze za druge, kompeksnije projekte.

Arduino platforma odabrana je za korištenje u hardwareskom djelu rada jer ista predstavlja relativno jednostavan i povoljan pristup te uz veliki izbor dodatnih gotovih pločica za proširivanje mogućnosti (*shields*) s otvorenom dokumentacijom logičan je izbor.

Na softwareskoj strani izbor je pao na programski jezik Python te dodatne biblioteke Matplotlib i NumPy koje su isto tako otvorenog koda i besplatne za korištenje i dovoljno jednostavne za učenje. Dodajmo tome popularnost među akademskom i znanstvenom zajednicom isti se nameću kao logičan izbor.

## 1.1 Opis i definicija problema

Pri proučavanju bilo kojeg inženjerskog ili znanstvenog problema potrebno je prikupiti dovoljnu količinu podataka te iste analizirati i na reprezentativan način prikazati u pokušaju razumjevanja problema. GPS je globalno dostupan sustav putem kojim se mogu prikupiti dovoljno precizni podaci za civilnu primjenu te je format podataka standardiziran čime je omogućeno korištenje i standardnih metoda analize podataka. Ovisno o promatranom problemu, potrebni su razni dodatni podatci koje pružaju razni senzori. Temperatura je jedan o njih te je u ovom radu korišten klasični TMP36 temperaturni senzor i obrađeni podaci koji su skupljeni putem navedenog senzora.

## 1.2 Cilj i svrha rada

Cilj ovog rada bio je izraditi jednostavni snimač podataka (*datalogger*) koji će spremati GPS podatke zajedno s podatcima prikupljenima s instaliranih senzora za kasniju analizu. Izrađeni uređaj namjenjen je kao snimač podataka u kompleksnijem sklopu koji se može koristiti kad god postoji potreba za loggiranje podataka. Uređaj je namijenjen zadovoljavanju širokog spektra potreba koje se mogu javiti bilo u industriji npr. prilikom praćenja pošiljki ili prilikom skupljanja podataka u istraživačke svrhe kako bi se razumio širi problem.

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

Prikupljeni podatci spremaju se na SD karticu uređaja u datoteku za kasniju obradu i analazu. 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 čelija 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

Hipoteza ovog rada je da primjena pristupa otvorenog i slobodnog koda (računanog koda, metode obrade podataka, algoritama i shema hardwareskog sklopa) omogućava izradu funkcionalnog sustav uz prihvatljive troškove i skromne resurse.

## 1.4 Metode rada

Tijekom izrade ovoga rada korištene su različite znanstveno-istraživačke metode od kojih je svaka najprikladnija postavljenom izazovu, a one su:

- Istraživačka metoda – za stjecanje uvida u zadane okvire zadatka
- Metoda logičke analize i sinteze – za prikupljanje podataka iz literature
- Deskriptivna metoda – za izradu uvodnog i završnog dijela projektnog zadatka
- Eksperimentalna metoda - u potrazi za optimalnim rješenjima za zadani dio problema

## 1.5 Struktura rada

Struktura ovoga rada podjeljena je u logičke cjeline. Nakon uvoda i objašnjavanja rada, u poglavlju 2 opisan je korišten pristup i primjenjene tehnologije kao i dano objašnjenje zašto je ista upotrebljena.

U poglavlju 3 opisan je izrađeni hardwareski elektronički sklop korišten za prikupljanje obrađenih podataka. U dodatku A priložen je izvorni kod koji se izvršava na Arduino mikroračunalu.

Poglavlje 4 opisuje postupak izrade analize prikupljenih podataka. Kompletna analiza nalazi se u dodatku B.

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 kakav drugi tip projekta. Važno je napomenuti kako postoje razne licence pod kojima se objavljuju open soure radovi, a u praksi se razlikuju u načinu na koji izmjenjeni i izvorni rad mora biti distribuiran svim ostalim zainteresiranim stranama.

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

- Kontorla proizvoda
- Učenje i trening
- Sigurnost
- Stabilnost

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

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

dokumentacija omogućava proučavanje rješenja određenih problema skraćujući vrijeme i pojeftinjuje razvoj novih proizvoda koji imaju slične zahtjeve. Znanstvenici objavljuju svoja otkrića ne bili ih mogli koristiti. Inženjeri u svakodnevnom radu ne izvode i dokazuju npr. Ohmov ili Newtonove zakone, već ih samo primjenjuju.

**Sigurnost:** Proučavanjem objavljene dokumentacije projekta drugi stručnjaci iz toga područja mogu uvidjeti neke propuste koje autori zbog kompleksnosti proizvoda ili drugih razloga nisu primjetili te dojaviti autorima pogrešku ne bili se ista mogla ispraviti. Neke pogreške mogu se pojaviti u iznimno malom broju slučaja ili kada se posloži 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 pogrešku u proizvodu čijim otklanjanjem proizvod postaje sigurnijim.

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

## 2.2 Arduino platforma

Arduino je elektronička platforma otvorenog koda[1] temeljena na hardwareu i softwareu koji je lako za koristiti. Arduino platforma obuhvaća mikrokontrolerske pločice temeljene na AVR arhitekturi s integriranim digitalnim, alalognim ulazima i izlazima kao i PWM[2] izlazima. Platforma omogućuje jednostavno spajanje dodatnih vanjskih uređaja poput raznih senzora, releja, serva i motora putem dodatnih upravljačkih modula te ostale elektroničkih i elektromehaničkih komponenti. Sheme svih mikrokontrolera objavljene su pod Creative Commons[3] licencom te javno dostupne svim zainteresiranim stranama.

Adruino pločice relativno su povoljne u usporedbi s ostalim platformama i kao takve omogućavaju pristupačnije učenje svim zainteresiranima. Potrebno je ponekad malo spretnosti s lemilicom, iako se

[1] `https://www.arduino.cc/en/Guide/Introduction`
[2] Pulse Width Modulation - Pulsno širinska modulacija
[3] `https://creativecommons.org/`

često mogu slagati moduli na prototipnoj pločici bez lemljenja sa izradom spojeva putem spojnih žica.

Jednostavno korisničko sučelje (IDE[4]) za izradu korisničkih programa (*sketch*) je jednostavno za korištenje početnicima dok istovremeno iskusnim korisnicima omogućava izradu vrlo kompleksnih programa. IDE je kompatibilan s većinom danas raspostranjenih operacijskih sustava (GNU/Linux, MacOS i Windows). Programski jezik za izradu programa je temeljen na C/C++ te omogućava daljnje proširivanje kroz C++ biblioteke ili korištenje AVR-C programskog jezika.

## 2.3 Jupyter Notebook

U ovom radu za obradu i prikazivanje podataka korišten je programski jezik Python[5] uz dodatke NumPy[6], Pandas[7] i Matplotlib[8]. NumPy i Pandas omogućuju lakšu manipulaciju podacima dok Matplotlib omogućuje izradu kvalitetnih grafova s velikom mogućnošću prilagodbe raznim željama i potrebama. Sve zajedno implementirano je kroz sustav *interaktivne bilježnice* Jupyter notebook[9] koja omogućuje 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 omogućuje kreiranje i dijeljenje dokumenata koji sadrže izvršivi programski kod, jednadžbe, grafove i vizualizacije te popratni tekst u jednoj cijelini koju trenutno drugim načinima nije moguće ili je vrlo kompleksno za postići. Područja primjene su najčešće obrada i transformacija podataka, numeričke analize, statistički modeli, vizualizacija podataka, strojno učenje i još mnogo toga.

U alatima tipa Excel, gdje su korištene formule skrivene iza podataka u ćelijama te se greške lako podkradu i još lakše ostanu nezamječene. Istraživanja su pokazala značajnu količinu grešaka u Excel proračunskim tablicama koje su u dnevnoj uporabi diljem organizacija, od kojih neke su imale i značajne negativne financijke implikacije[2]. Iako je istraživenje starijeg datuma jedan od glavnih razloga pogrešaka (skrivene formule) je i dalje prisutan te je realno za očekivati kako se pogreške i dalje događaju, a sve većom uporabom proračunskih tablica za očekivati je kako broj istih s greškama raste.

[4] Integrated development environment - Integrirano razvojno okruženje
[5] `https://www.python.org/`
[6] `https://numpy.org/`
[7] `https://pandas.pydata.org/`
[8] `https://matplotlib.org/`
[9] `https://jupyter.org/`

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

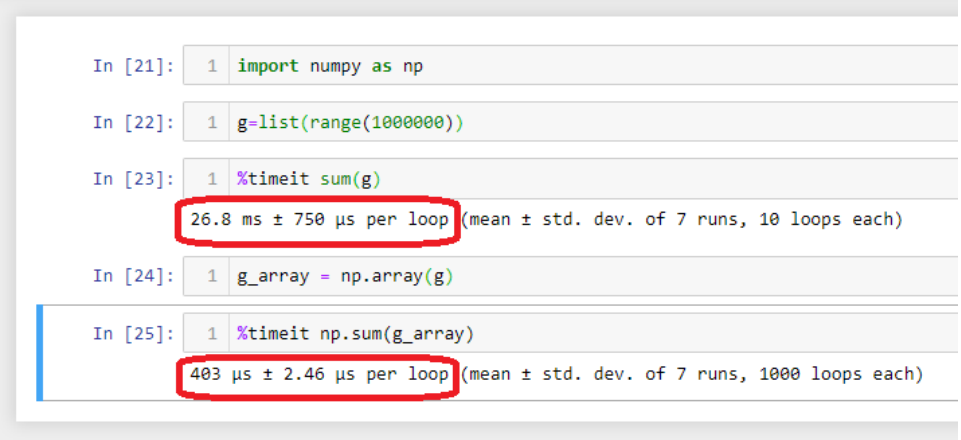
- snažan alat za rad s N-dimenzionalnim poljima
- alat za intergraciju C/C++ i Fortran programskog koda
- korisne alate za algebarske operacije, Fourierovu analizu i ostale numeričke mogućnosti

Osim što značajno olakšava numeričku 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 potrebnog za izvršenje zadatka.

Na slici 2.1 prikazana je razlika u brzini izvršavaja 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 isti zadatak iznosi $403\mu s \pm 2.46\mu s$[11]. Vidljiva je značajna razlika u potrebnom vremenu za izvršenje zadatka, a iskustva industrije[3] pokazuju još veću razliku pri kompleksnijim zadatcima.

[10] `https://www.gw-openscience.org/tutorials/`

[11] Rezultati mogu varirati u zavisnosti o korišenom računalu




Slika 2.1: Usporedba performanski Pythona i NumPy biblioteke

## 2.5 Git

Git[12] je distribuirani sustav za verzioniranje koda i ostalog rada kojeg želimo dijeliti sa suradnicima. Git svojim jednostavnim i brzim granama omogućuje lakši razvoj proizvoda kao i ispitivanje mogućnosti i funkcija. Kada se želi ispitati neka funkcionalnost bez ugrožavanja dosadašnjeg rada 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 razvijene za proizvod. Kako je Git lagan za resurse može se kreirati vrlo veliki broj grana za razne potrebe bez značajnog utjecaja na performanse razvojnog računala ili potrošnje spremišnog prostora.

S obzirom na distibuiranu 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 dijeljenje i suradnja na projektima, 2008. godine je pokrenut GitHub - centralno mjesto za usluge poslužitelja[13] (*hosting*) putem kojeg je moguće pratiti životni ciklus i povijest projekta. Svatko može pronaći projekt koji ga zanima te, ukoliko ima dovoljno vremena i znanja, može pridonjeti njegovom razvoju. Brojne kompanije koriste

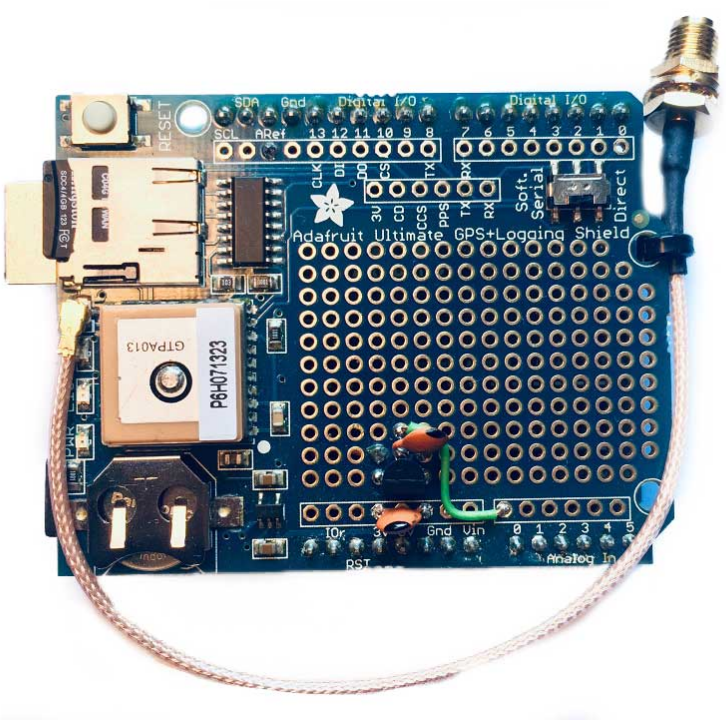
[12] https://git-scm.com/
[13] https://github.com/features

GitHub kako bi podjelile svoje projekte. Podatak od travnja 2019. godine kaže kako više od 2.1 miljuna 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 edukaciju posebice mladih (i onih koji se tako osjećaju), a njihov GitHub sadrži više od 1100 repozitorija[14]. Upravo je njihov GPS Logger Shield korišten u ovom projektu, a dostupnost dokumentacije i podrška jedan je od glavnih razloga zašto je odlučeno korištenje upravo ovog proizvoda.

[14] https://github.com/adafruit/

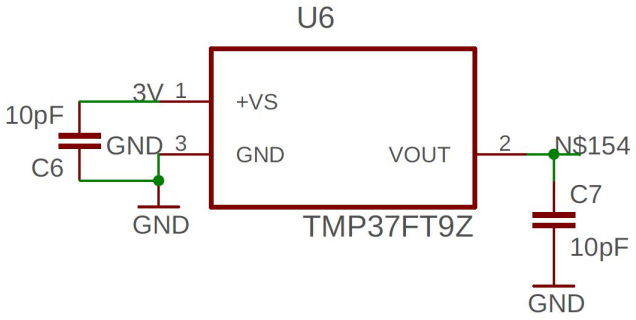
# Poglavlje 3

# Prikupljanje podataka - hardware




Slika 3.1: Izgled korištenog hardwareskog sklopa

U prilogu A prikazan je cjeloviti izvorni kod koji se izvšvana na Arduino mikrokontroleru.




Slika 3.2: Shema spoja TMP36 senzora

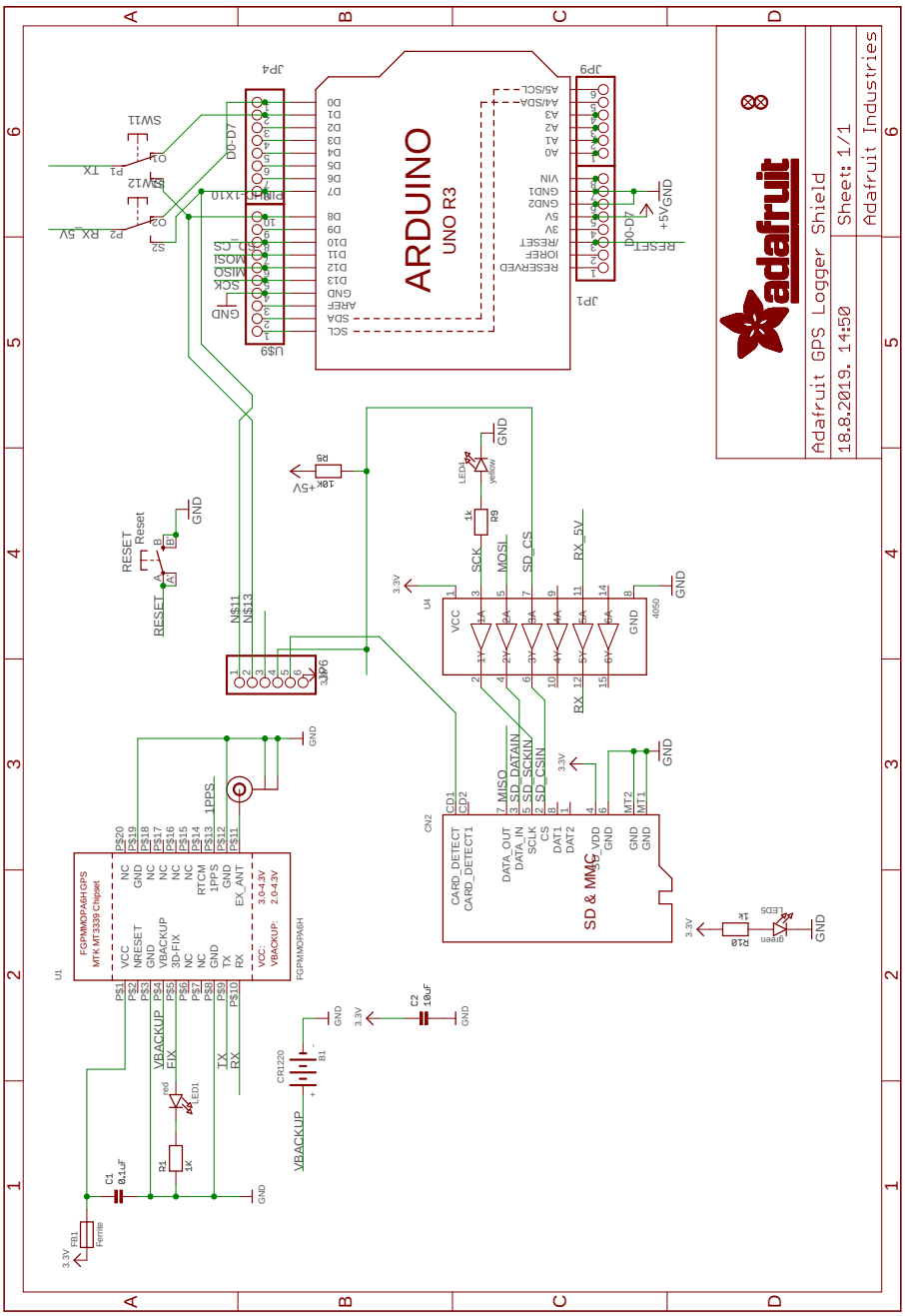
## 3.1 GPS logging shield

Na shemi 3.3 nalazi se shema gotovog elektroničkog sklopa kako dolazi iz tvornice[1][2]. Na samoj tiskanoj pločici postoji takozvano prototipno područje za dodavanje vanjskih elemanata čiji je raster 2.54 mm koji odgovara standardu *true-hole* elemenata. Na to područje je dodan temperaturni senzor TMP36[3] 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.

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

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

[3] Datasheet 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 važno je znati u kojim se uvjetima isti nalazi. Ukoliko je temperatura previsoka, može se uključiti aktivno hlađenje, ili, ako se unaprijed zna da će se sklop nalaziti pod povišenom radnom temperaturom tada se može konstrurati adekvatan sustav hlađenja. Isto vrijedi za prenisku temperaturu. Prema ranije spomenutoj shemi 3.2, dodan je temperaturni senzor koji mjeri radnu temperaturu okoline uređaja. Pri testiranju, ova vrsta senzora pokazala se veoma pouzdana, uz minimalno samozagrijavanje koje bi utjecalo na točnost mjerene veličine, ali je isto tako pokazala vrlo brze promjene izlazne vrijednosti koja može imati uzrok u vanjskim smetnjama. Kako bi se otklonio taj problem primjenjena su dva rješenja. Prvi je hardwareski fiter - kondenzatori koji si prikazani na shemi 3.2, a drugi je softwareski fiter. Tvornički podatci 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 očitanja u sekundi) Svakako nije realno za očekivati da se temperatura mjenja sukladno očitanim vrijednostima.

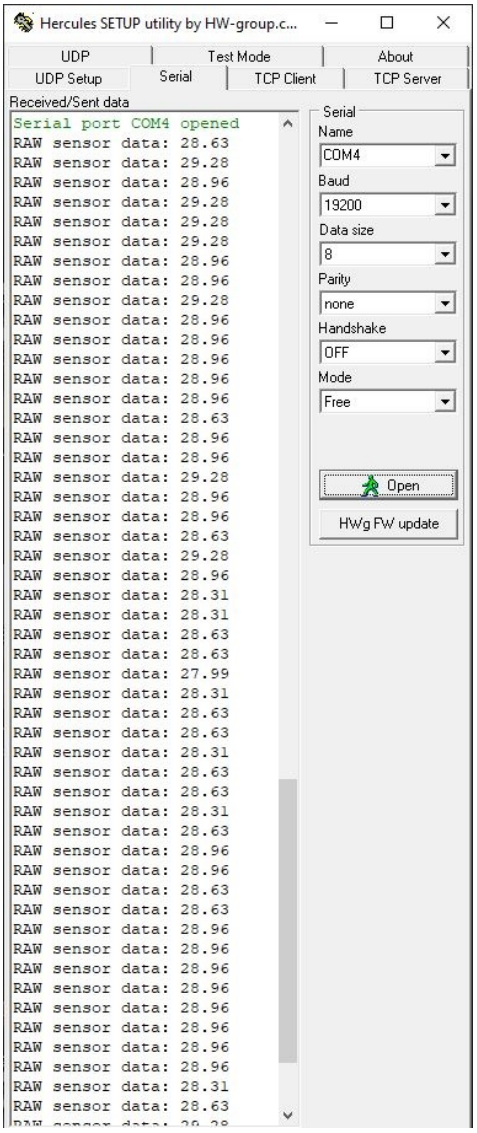
### 3.2.1 Softwareski filter

Softwareski filter radi na principu očitavanja 10 vrijedosti sa senzora te ih sprema u polje. Potom ih sortira po veličini i uzima medijan[4] vrijednost kao točnu temperaturu. Time 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 daju frekvenciju od 1 očitanja u sekundi koja odgovara i frekvenciji uzorkovanja podataka s GPS senzora. Prilikom testiranja utvrđeno je da veći broj uzoraka ne doprinosti kvaliteti izmjerenih vrijednosti, dok se pri manjem broju uzoraka može se potkrasti poneka nerealna vrijenosti. 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 stvarnom vrijendosti.

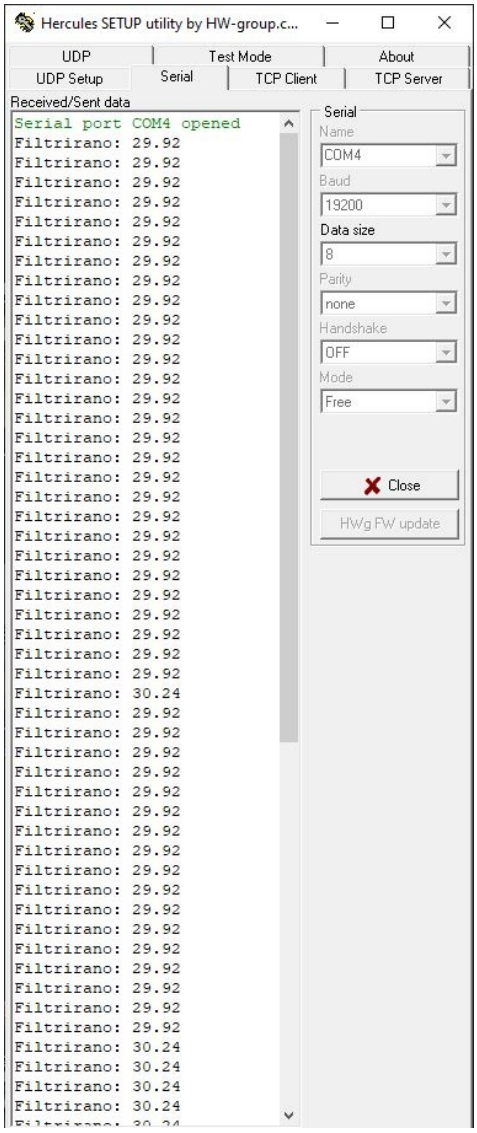
### 3.2.2 Hardwareski filter

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

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



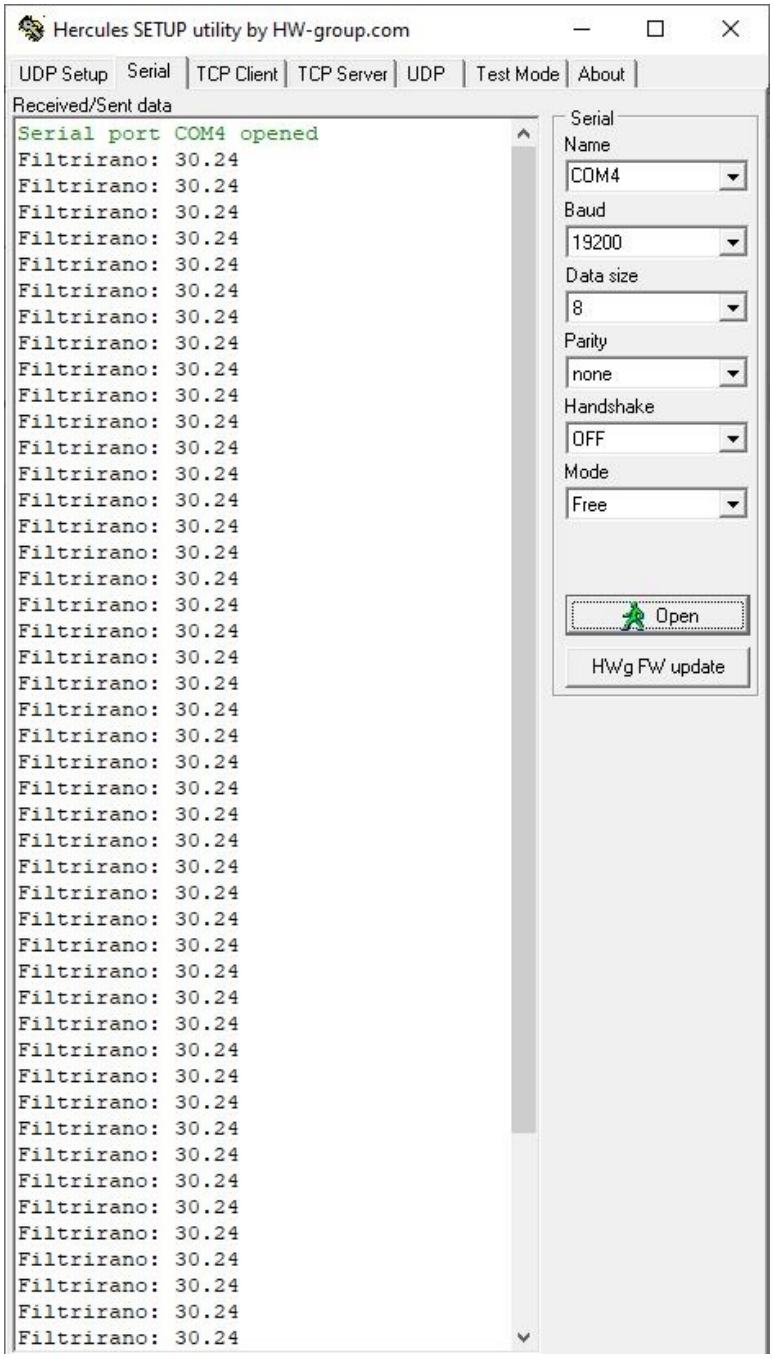

Slika 3.4: Vrijednosti senzora bez filtriranja




Slika 3.5: Vrijednosti senzora sa softwareskim filtriranja

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

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




Slika 3.6: Vrijednosti senzora primjenom kombinacije Sw i Hw filtera

## 3.3 Prikupljanje GPS podataka

GPS[5] je javni sustav u vlasništvu vlade SAD-a[6] koji služi globalnom pozicioniranju temeljen na satelitima s atomskim satovima koji odašilju vrlo točno i precizno trenutno vrijeme te su sinkronizirani sa zemaljskim satovima. Bilo kakva odstupanja se korigiraju na dnevnoj bazi. Prijemnik prima signal satelita te izračunava točnu poziciju temeljenu 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 MTK3339[7] integriran na prije spomenuti Adafruit Ultimate GPS Logger Shield.

Kao koristan izlaz prijemnik daje NMEA[8] rečenicu. Ovisno o potrebnim podatcima, mogu se koristiti razne rečenice, a u ovoj primjeni korištena je $GPRMC[9] koja daje minimalne potrebne podatke, a među kojima su vrijeme (UTC) i datum, trenutna 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:

| | |
|---|---|
| $GPRMC | Oznaka rečenice |
| 053005.000 | UTC vrijeme (7:30:05 lokalno) |
| A | Oznaka valjanosti, A = OK, V = warning |
| 4457.8784,N | Zemljopisna širina |
| 01356.1351,E | Zemljopins dužina |
| 36.41 | Brzina u čvorovima ($\approx 67 km/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,
- je li oznaka valjanosti *A* što znači da uređaj ima prijem s dovoljnog broj satelita da se primljenim podatcima može vjerovati.

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

[5] Global Positioning System - Sustav globalnog pozicioniranja

[6] `https://www.gps.gov`

[7] `https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMOPA6C-Datasheet-V0A-Preliminary.pdf`

[8] `https://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 .csv [10] formatu pogodnim za kasniju obradu 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 $^{\circ}C$ 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 GPSLOG$XX$.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 ($\approx$ 36000 zapisa) snimljenih podataka iznosi otprilike 2.7 Mb.

[10] Comma Separated Values

# Poglavlje 4

# Obrada podataka - software

Kako je već spomenuto u uvodu 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.

## 4.1 Prikaz podataka vozila u kretanju

Kao prvi primjer obrade podataka izrađena je analiza kretanja vozila i vizualizirani su prikupljeni podatci. U dodatku B prikazana je cjelokupna analiza podataka te demonstriran završni proizvod koji može biti dalje distribuiran u printanoj verziji ili preferirano u digitalnom obliku koji onda omogućava daljni rad na istome.

Koristeći razne tipove ćelija unutar Jupyter notebooka, stvorena je prikazana analiza. Tip ćelije *Markdown* podržava formatiranje i sintaksu `.md` Markdown datoteke[1], a koje se koriste za opisni dio analize kako bi čitatelj znao o čemu se radi bez potrebe proučavanja programskog koda.

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

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

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

[1] `https://www.markdownguide.org/`

```
import numpy as np
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
```

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

Nakon toga potrebno je navesti koja se datoteka s podatcima 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 podatci.

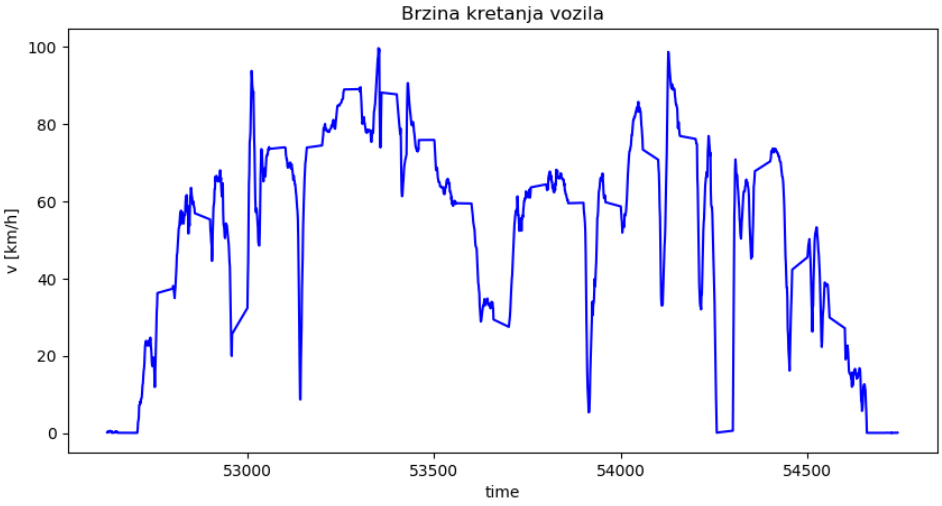
Nakon uvodnih radnji, spremni smo započeti s analizom podataka i vizualizcijom onih koje nas zanimaju. U ovisnosti o željenim rezultatima ovisit će i potrebne operacije koje je potrebno poduzeti. Po nekad su podatci već u takvom formatu da se mogu odmah koristiti, dok je ponekada potrebno napraviti veću ili manju manipulaciju nad njima kako bi bili pogodni za korištenje. U svakom slučaju, vrlo je važno poznavati strukturu seta podataka koji se koristi kako bi se izbjegle pogreške zbog njegovog nepoznavanja ili korištenja krive mjerne jedinice. Godine 1999., NASA je izgubila 125 milijuna vrijedan *Mars Climate Orbiter* zbog previda pretvorbe jedinica[4] što samo pokazuje koliko je to realna zamka u koju i 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 $km/s$.

Obzirom da je uvijek korisno vizualizirati kretanje veličine na grafu tako je i u ovom radu napravljen graf promjene brzine u vremenu prikazan na 4.1, što je koristeći Matplotlib vrlo jednostavno.

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

Pri čemu *data['Time']* predstavlja apcisu grafa, a *data['Speed']* predstavlja ordinatu prikazanog grafa. *'b-'* je parametar kojim je definirana linija - linija plave boje. Svaki graf treba biti propisno obilježen pa su tako 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 tijekom promatranog razdoblja.




Slika 4.1: Izrađeni graf kretanja vozila

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

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

Zanima li nas prosječna temperatura tokom promatranog razdoblja, tada 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.

## 4.2 Prikaz podataka lifta

Analiza prikazana u dodatku C vizualizira podatke prikupljene mobilnim telefonom koristeći aplikaciju *phyphox*[2]. Korišten je modul koji očitava podatke s akcelerometra u sve 3 osi koordinatnog sustava. Mobilni uređaj bio je prislonjen na bočni panel lifta i pokušano je postaviti ga da stoj okomito na smjer kretanja sa stražnjom stranom priljubljenim na panel

[2] `https://www.phyphox.org`

kako bi se dobili čim točniji podatci bez unošenja pogreške zbog pomaknutih osi mobitela i lifta. Izmjerene vrijednosti predstavljaju put od prizemlja do 5. kata zgrade na adresi Koparska 58, Pula.

Podatci su zatim spremljeni u .csv datoteku te je ista prenesena na računalo. Datoteka je formatirana na način da ima imena vrijednosti u zaglavlju te su korišteni zadani nazivi, a polja odvojena znakom ";".

# Poglavlje 5

# Zaključak

U ovom radu prikazan je postupak izrade uređaja za prikupljanje podataka kao i analiza prikupljenih podataka. Sve je urađeno primjenom komponenti s otvorenom i besplatnom dokumentacijom te softwarea koji je isto tako slobodan i besplatan za korištenje, čime je potvrđena hipoteza postavljena u uvodu ovog rada.

Sve ovo moguće je zahvaljujući popularnosti koju su korišteni sustavi postigli zbog svojeg *open soucre* pristupa radu te su zbog svoje raširenosti i prihvaćenosti, čime su omogućili razvoj zajednice koja podupire takav pristup. Upravo je ta zajednica autor mnogih vodiča, knjiga, kratki demo primjera i ostalih dragocjenih resursa koji omogućavaju savladavanje temetike i ulazak u svijet elektronike i razvoja softwarea čak i relativnim početnicima koji imaju volje za učenje.

# 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.).

[11] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, *et al.*, "Jupyter notebooks-a publishing format for reproducible computational workflows.," in *ELPUB*, pp. 87–90, 2016.

[12] F. Perez, B. E. Granger, and J. D. Hunter, "Python: an ecosystem for scientific computing," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 13–21, 2010.

[13] H. Koepke, "Why python rocks for research," *Hacker Monthly*, vol. 8, 2011.

[14] J. Hunter and D. Dale, "The matplotlib user's guide," *Matplotlib 0.90. 0 user's guide*, 2007.

[15] I. Idris, *NumPy Cookbook*. Packt Publishing Ltd, 2012.

[16] M. Borkin, K. Gajos, A. Peters, D. Mitsouras, S. Melchionna, F. Rybicki, C. Feldman, and H. Pfister, "Evaluation of artery visualizations for heart disease diagnosis," *IEEE transactions on visualization and computer graphics*, vol. 17, no. 12, pp. 2479–2488, 2011.

# Dodatak A

# Programski kod na Arduino mikroračunalu

```
#include <SPI.h>
#include <Arduino.h>
#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>
#include <SD.h>
#include <avr/sleep.h>
#include <Wire.h>         // this #include still required because the RTClib depends on it
#include "RTClib.h"

int voltage;
int Temperatura;
int TempSenzor = A0;

#define aref_voltage 3.3
float temp[10] = { 0 };

float i = 0;
int n = 0;
float median = 0;

SoftwareSerial mySerial(8, 7);
Adafruit_GPS GPS(&mySerial);

// Set GPSECHO to 'false' to turn off echoing the GPS data to the Serial console
// Set to 'true' if you want to debug and listen to the raw GPS sentences
#define GPSECHO  false
/* set to true to only log to SD when GPS has a fix, for debugging, keep it false */
#define LOG_FIXONLY true

// this keeps track of whether we're using the interrupt
// off by default!
#ifndef ESP8266 // Sadly not on ESP8266
boolean usingInterrupt = false;
#endif

// Set the pins used
#define chipSelect 10
#define ledPin 13

File logfile;

RTC_DS1307 RTC;  // define the Real Time Clock object
RTC_Millis rtc;

 char timestamp[30];
// call back for file timestamps
void dateTime(uint16_t* date, uint16_t* time) {
 DateTime now = RTC.now();
 sprintf(timestamp, "%02d:%02d:%02d %2d/%2d/%2d \n", now.hour(),now.minute(),now.second(),now.month(),now.day(),now.year()-2000);
 Serial.println("yy");
 Serial.println(timestamp);
 // 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
       dropping chars
102    // also spit it out
103    Serial.begin(115200);
104    Serial.println("\r\nUltimate GPSlogger Shield");
105    pinMode(ledPin, OUTPUT);
106    pinMode(TempSenzor, INPUT); //postavi izvod TempSenzor (A0) kao ulazni
107    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
        data) including altitude
141   //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
142   // uncomment this line to turn on only the "minimum recommended" data
143   GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
144   // Set the update rate
145   GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);   // 100 millihertz (once every
        10 seconds), 1Hz or 5Hz update rate
146   // Turn off updates on antenna status, if the firmware permits it
147   GPS.sendCommand(PGCMD_NOANTENNA);
148   // the nice thing about this code is you can have a timer0 interrupt go off
149   // every 1 millisecond, and read data from the GPS for you. that makes the
150   // loop code a heck of a lot easier!
151 #ifndef ESP8266 // Not on ESP8266
152   useInterrupt(true);
153 #endif
154   Serial.println("Ready!");
155 }
156
157 // Interrupt is called once a millisecond, looks for any new GPS data, and
      stores it
158 #ifndef ESP8266 // Not on ESP8266
159 ISR(TIMER0_COMPA_vect) {
160   char c = GPS.read();
```

```
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 ↵
        to false
232       return;  // we can fail to parse a sentence in which case we should just ↵
           wait for another
233
234     // Sentence parsed!
235     Serial.println("OK");
236     if (LOG_FIXONLY && !GPS.fix) {
237       Serial.print("No Fix");
238       return;
239     }
240
241     float voltage = analogRead(TempSenzor) * 3.3;  //ocitava vrijednosti      ↵
        izvoda (A0)
242     voltage /= 1024.0; //10bit ADC
243     float Temperatura = (voltage - 0.5) * 100;
244     Serial.print("Trenutno: ");
245     Serial.println(Temperatura);
246
247     // Rad. lets log it!
248     Serial.println("Log");
249
250     char tempBuff[5];
251     dtostrf(Temperatura,0,2,tempBuff);
252     uint8_t tempSize = strlen(tempBuff);
253
254     //logfile.flush();
255
256  // ovaj blok kao dela pa pomalo s tim :-)
257         uint8_t stringsize = strlen(stringptr);// + tempSize;
258         if (stringsize != logfile.write((uint8_t *)stringptr,                ↵
             stringsize))    //write the string to the SD file
259         error(4);
260     if (strstr(stringptr, "RMC") || strstr(stringptr, "GGA") )   logfile.flush ↵
        ();
261     logfile.write(tempBuff);
262     Serial.println();
263
264   }
265 }
266
```

Dodatak B

# Analiza podataka kretanja vozila

# AnalizaKretanjaVozila

September 14, 2019

## 1 Analiza podataka kretanja vozila iz .csv filea

Demo kako uz pomoc python programskog jezika i *matplotlib* biblioteke za prikaz grafova

Prvo uvezemo potrbne bibliteke

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
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.

[2]:
```
filename='GPSLOG10.CSV'
#plt.style.use('ggplot')
data=pd.read_csv(filename, header=None, delimiter=',',␣
 ↪names=['Sentence','Time','Validity','Latitude','NS',
                                                        ␣
 ↪'Longitude','EW','Speed','Direction','Date',
                                                        ␣
 ↪'NA1','NA2','Checksum','Temperature'])
```

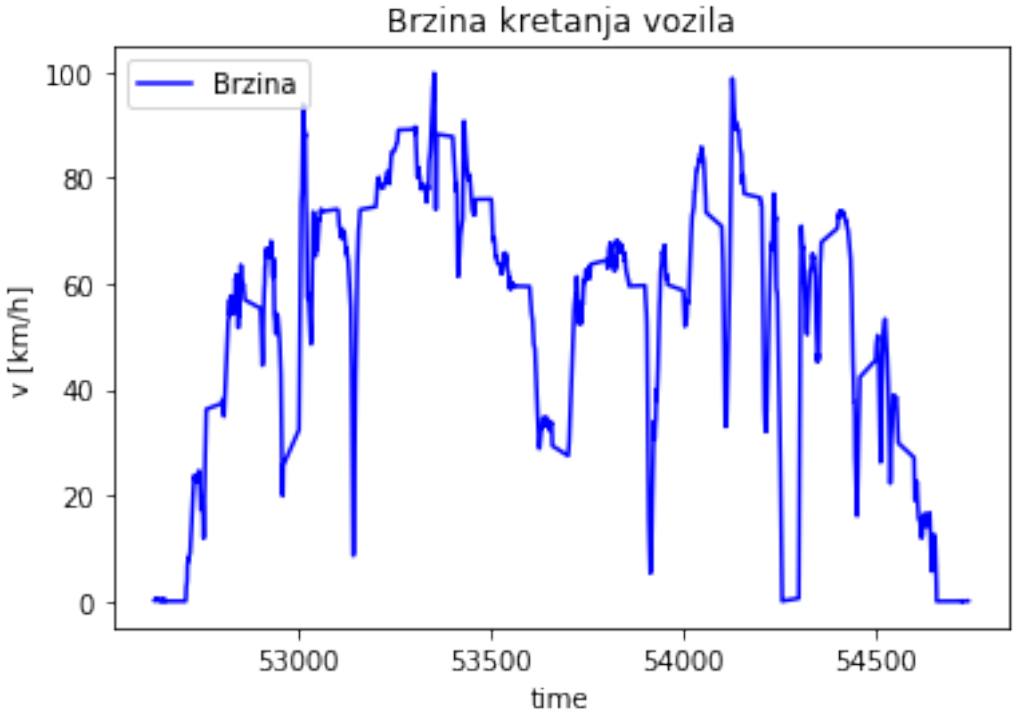
### 1.1 Line plot

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 izvrsiti 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, pojedinacni grafovi se mogu posebno spremiti u visokoj rezoluciji i zeljenom formatu za kasniju upotrebu.

[3]:
```
#otvori graf u novom prozoru
#%matplotlib qt
plt.plot(data['Time'],data['Speed']*1.852, 'b-',label='Brzina')

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)
```

[3]:
```
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:

[4]:
```
print('Maksimalna brzina = ',np.max(data['Speed']*1.852) , 'km/h')
```

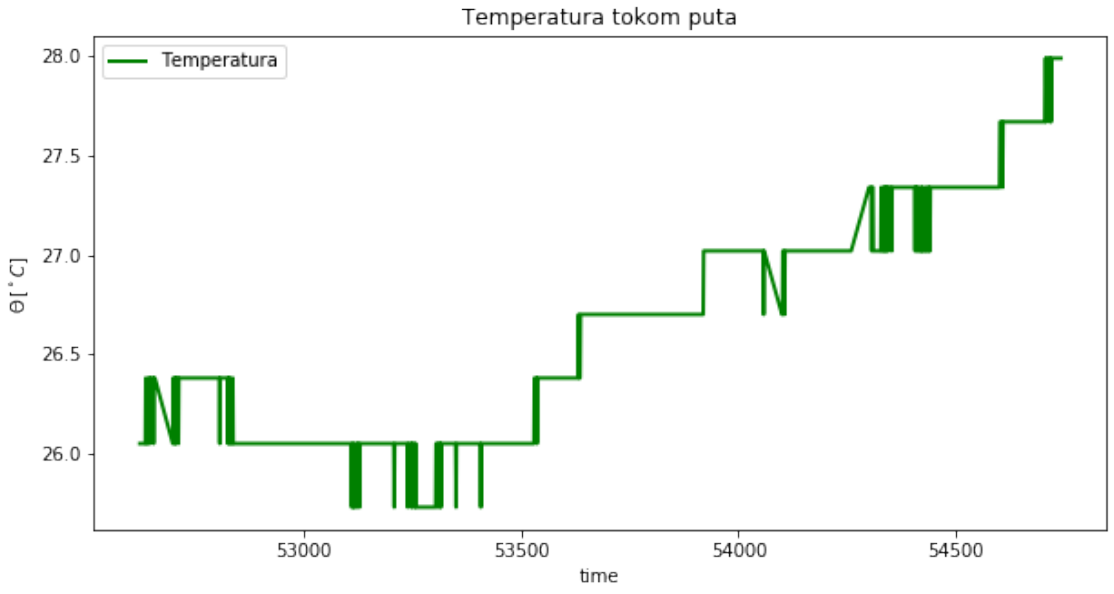
```
Maksimalna brzina =  99.73020000000001 km/h
```

Ako nas zanimaju podaci o temperaturi moguce je cak koristiti i ugradene statisticke funkcije za izracunati zeljene podatke

[8]:
```
# prikazi graf inline
%matplotlib inline
```

```
plt.rcParams["figure.figsize"] = (10,5)
plt.plot(data['Time'],data['Temperature'], 'g', linewidth=2,label='Temperatura')

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 ('Prosjecna temperatura: ', '{:.2f}'.format(np.
 ↪mean(data['Temperature'])), 'řC ś ',
       '{:.2f}'.format(np.std(data['Temperature'])), 'řC')
```

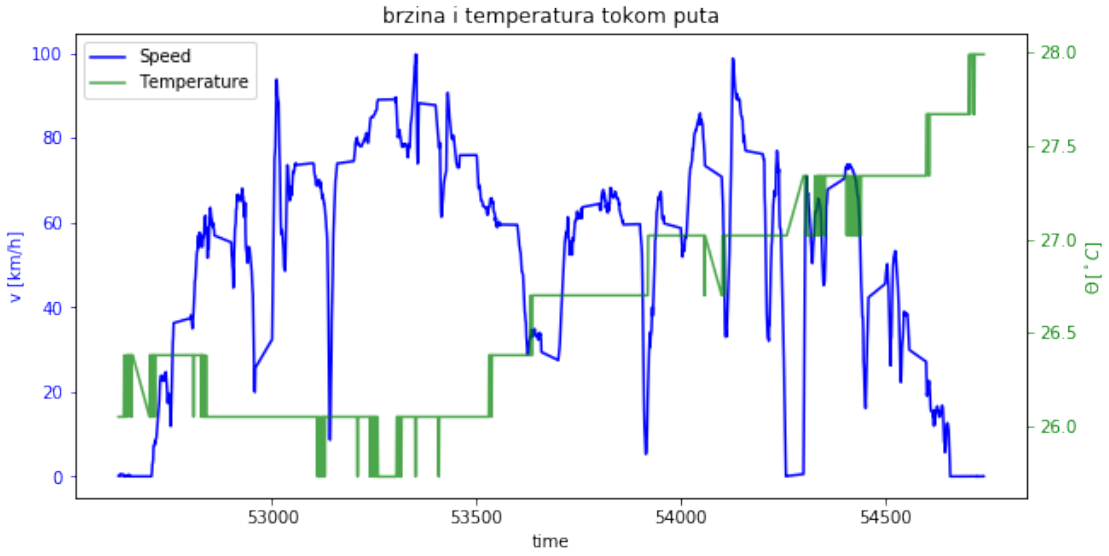
```
Minimalna temperatura:  25.73 řC
Maximalna temperatura:  27.99 řC
Razlika temperature:  2.259999999999998 řC
Razlika temperature:  2.26 řC
Prosjecna temperatura:  26.64 řC ś  0.57 řC
```




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

[6]:
```
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"])
```

[6]:
```
<matplotlib.legend.Legend at 0x1146096d8>
```




Uvidom u graf ne mozemo vizualno utvdriti 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.

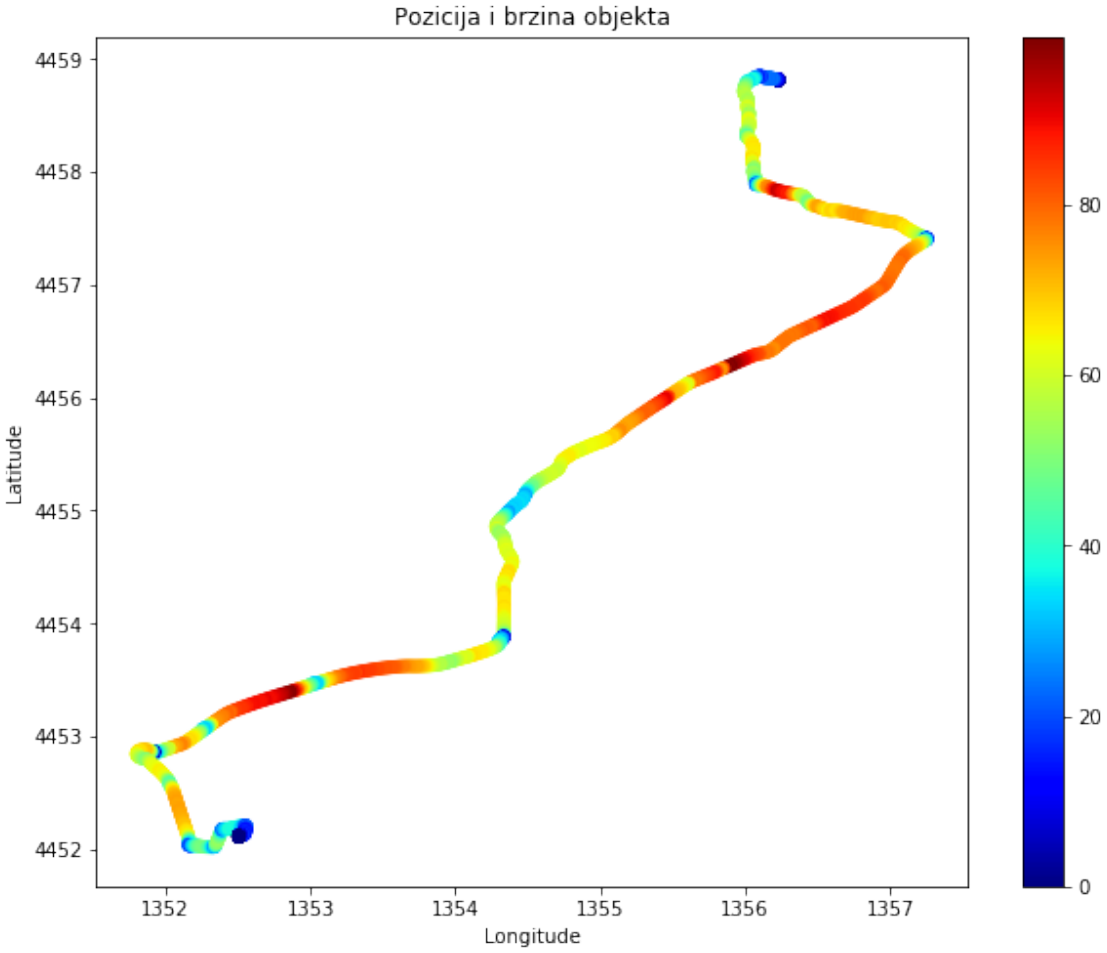
## 1.2 Scatter plot

Koristeci funkciju *scatter* koji za razliku od *line* ne spaja susjedne toke linijom moemo prikazati kretanje vozila u koordinatnom sustavu.

Dodavanjem vrijednosti parametru *c* (color) moemo prikazati i dodatnu dimenziju, a to je u ovom sluaju brzina kretanja prikazana putem obojene skale.

[7]:
```
plt.rcParams["figure.figsize"] = (10,8)
fig, ax = plt.subplots()

p=ax.scatter(data['Longitude'],data['Latitude'],c=data['Speed']*1.
 ↪852,marker='o',cmap='jet')
#RdYlGn, jet, viridis // _r - rev
fig.colorbar(p)

plt.xlabel ('Longitude')
plt.ylabel ('Latitude')
plt.title('Pozicija i brzina objekta')
```

[7]:
```
Text(0.5, 1.0, 'Pozicija i brzina objekta')
```



Dodatak C

# Vizualizacija podataka kretanja lifta

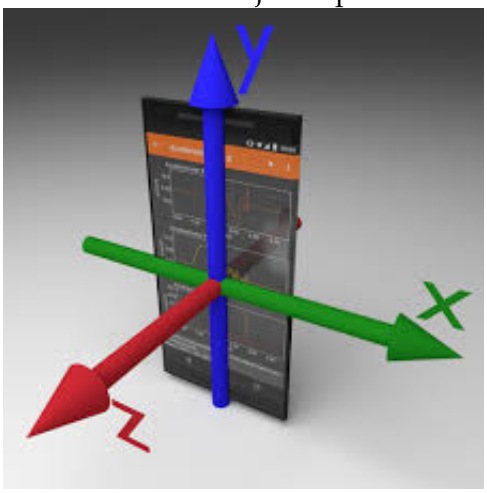
# KretanjeLifta

September 6, 2019

## 1 Kretanje lifta i prikaz ubrzanja

Ova analiza prikazuje primjer vizualizacije podataka prikupljenih koristei phyphox aplikaciju dostupnu za mobilne telefone. Vie onformacija o ovoj aplikaciji moe se pronai na: https://phyphox.org/

Na slici je prikazan koordinatni sustav kojeg koristi aplikacija




Podaci su zatim spremljeni u .csv datoteku i preneeni na raunalo. Datoteka je formatirana na nain da ima imena vrijednosti u zaglavlju te su koriteni zadani nazivi, a polja su odvojena znakom ";"

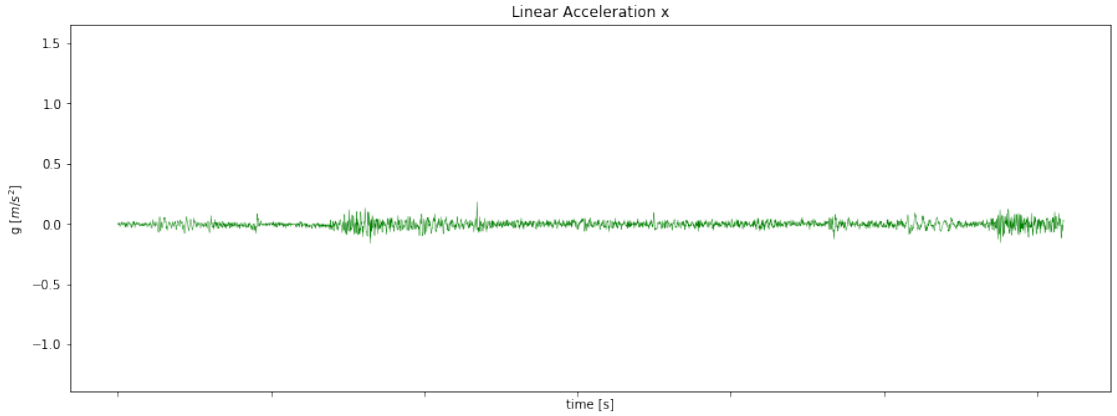
Kao i u predhodnoj analizi potrebno je prvo uitati potrebne module

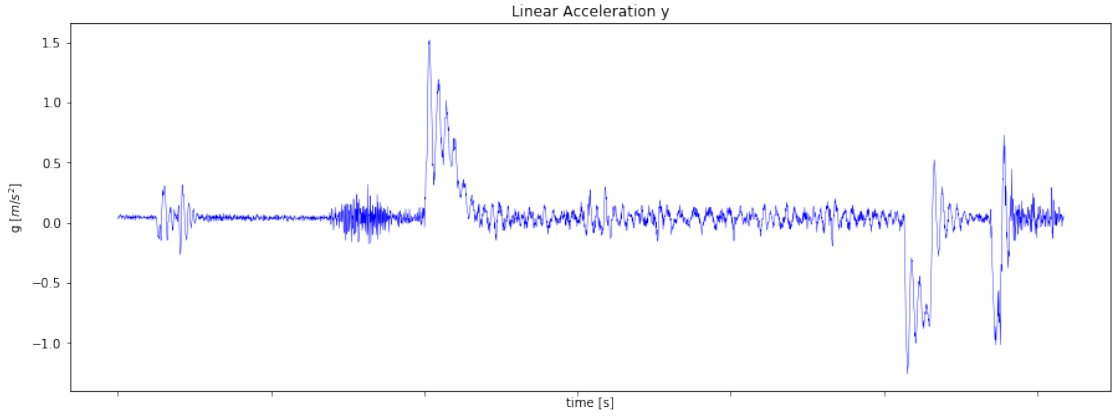
```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from matplotlib import style
```

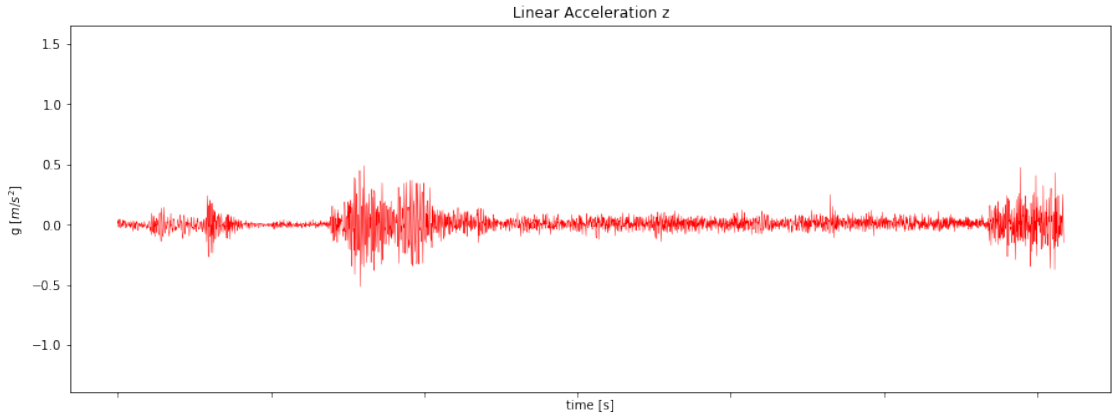
Unese se ime datoteke s podacima u varijablu *filename* koristei jednostuke navodne znakove

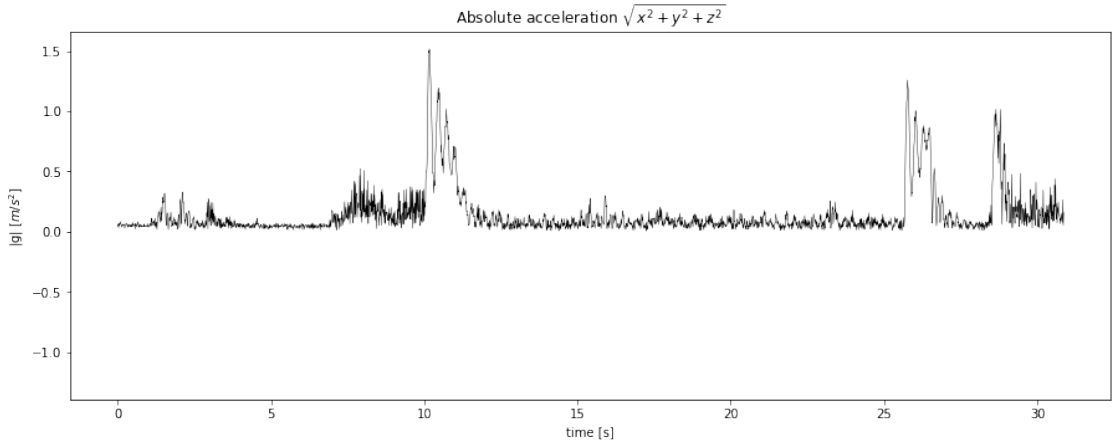
```
In [2]: filename='Lift.csv'
        #plt.style.use('ggplot')
        data=pd.read_csv(filename, delimiter=';')

In [3]: #otvori graf u novom prozoru
        #%matplotlib qt
```

```
plt.rcParams["figure.figsize"] = (15,25)
fig, axs = plt.subplots(4, sharex=True, sharey=True)


axs[0].plot(data['Time (s)'],data['Linear Acceleration x (m/s^2)'], 'g-',linewidth=0.4
axs[0].set_title('Linear Acceleration x')
axs[0].set(ylabel='g [$m/s^2$]')

axs[1].plot(data['Time (s)'],data['Linear Acceleration y (m/s^2)'], 'b-',linewidth=0.4
axs[1].set_title('Linear Acceleration y')
axs[1].set(ylabel='g [$m/s^2$]')

axs[2].plot(data['Time (s)'],data['Linear Acceleration z (m/s^2)'], 'r-',linewidth=0.4
axs[2].set_title('Linear Acceleration z')
axs[2].set(ylabel='g [$m/s^2$]')

axs[3].plot(data['Time (s)'],data['Absolute acceleration (m/s^2)'], 'k-',linewidth=0.4
axs[3].set_title('Absolute acceleration $\sqrt{x^2+y^2+z^2}$')
axs[3].set(ylabel='|g| [$m/s^2$]')

for ax in axs.flat:
    ax.set(xlabel='time [s]')
```


Linear Acceleration x
g [m/s²]
time [s]


Linear Acceleration y
g [m/s²]
time [s]


Linear Acceleration z
g [m/s²]
time [s]


Absolute acceleration $\sqrt{x^2 + y^2 + z^2}$
|g| [m/s²]
time [s]

Za probu zgodno je probati izraditi predhnodne grafove ali uz izmjenu da pojedini grafovi nemaju istu visinu ordinate vec se automatski prilagode rasponu vrijednosti koristeci zamjenjujuci sljedecu liniju koda

```
fig, axs = plt.subplots(4, sharex=True, sharey=False)
```

S obzirom da je defaultna vrijednost tog parametra *False* moe se i izostaviti te pisati samo

```
fig, axs = plt.subplots(4, sharex=True)
```

Prouciti kako ta promjena utjece na percepciju vrijednosti i vizualizaciju podataka