Šolski center velenje, Elektro in računalniška šola,
Trg mladosti 3, 3320 Velenje
Mladi raziskovalci za razvoj SAŠA regije

RAZISKOVALNA NALOGA

ANALIZA KOLESARSKIH POVRŠIN V ŠALEŠKI DOLINI

Področje:
Računalništvo, varnost v cestnem prometu

Avtor:
Anže Maj Blagus

Mentor:
Gregor Hrastnik, univ. dipl. inž. rač. in inf.

Velenje, 2024

Raziskovalna naloga je bila opravljena na Elektro in računalniški šoli Velenje.

Mentor: Gregor Hrastnik, univ. dipl. inž. rač. in inf.

Datum predstavitve:

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

ŠD Elektro in računalniška šola Velenje, šolsko leto 2023/2024

KG kolesarske površine / kolo / kolesarjenje

AV BLAGUS, Anže Maj

SA HRASTNIK, Gregor

KZ 3320 Velenje, SLO, Trg mladosti 3

ZA Elektro in računalniška šola Velenje

LI 2024

IN ANALIZA KOLESARSKIH POVRŠIN V ŠALEŠKI DOLINI

TD Raziskovalna naloga

OP XII, 63 str., 8 pregl., 18 sl., 27 graf., 1 pril, 24 vir.

IJ SL

JI sl/en

AI V raziskovalni nalogi sem si zadal izziv analize kvalitete in udobnosti vožnje po kolesarskih površinah kolesarskih površin po Velenju in okolici. Najprej sem izvedel anketo, v kateri sem zbral mnenja velenjskih kolesarjev o kvaliteti kolesarskih površin. Nato sem si pogledal nekaj možnosti zbiranja podatkov, s katerimi bi lahko ocenil kvaliteto kolesarskih površin. Izbral sem si uporabo senzorjev na pametnem telefonu. Za tem sem zbral senzorske podatke po večjem delu kolesarskih površin po Velenju in okolici in jih obdelal. Na koncu sem zbrane podatke prikazal na več različnih načinov, s katerimi si lahko pomagamo pri ocenjevanju udobnosti vožnje po poteh, in jih primerjal z rezultati ankete. Ugotovil sem, da je mogoče izdelati algoritem, ki iz zbranih podatkov vrne ocene odsekov, ki so podobne ocenam ljudi.

KEY WORDS DOCUMENTATION

ND	Elektro in računalniška šola Velenje, school year 2023/2024
CX	cycling surfaces / bike / cycling
AU	BLAGUS,Anže Maj
AA	HRASTNIK,Gregor
PP	3320 Velenje, SLO, Trg mladosti 3
PB	Elektro in računalniška šola Velenje
PY	2024
TI	ANALYSIS OF CYCLING SURFACES IN ŠALEK VALLEY
DT	Research work
NO	XII, 63 p., 8 tab., 18 fig., 27 graf., 1 ann., 24 ref.
LA	SL
AL	sl/en

AB The main challenge of this research work was to analyze the quality and comfort of cycling on different cycling paths around the town of Velenje. First, I created a survey to gather the opinions of cyclists in Velenje about the quality of cycling surfaces. Then, I looked at a couple of ways to gather data that could be used to assess the quality of the surfaces. I chose to use the sensors present in modern smartphones, which I used to gather accelerometer, gyroscope and GPS data for cycling paths around the town and it's surroundings. I then processed this data and displayed in different ways that can be used to rate the quality of the cycling paths. I also compared it to the results of the survey and found out that it's possible to create an algorithm that assesses the quality of cycling surfaces with similar results as cyclists' opinions.

KAZALO VSEBINE

1 Uvod **1**
1.1 Problem . . . 1
1.2 Hipoteze . . . 1

2 Pregled objav **2**
2.1 Obstoječe raziskave . . . 2
2.2 Načini zbiranja podatkov . . . 3
2.2.1 Video . . . 3
2.2.2 Mobilni telefon . . . 3
2.2.3 Namenska naprava . . . 5
2.2.4 Ultrasonični senzor in mikrofon . . . 5
2.3 Procesiranje podatkov . . . 5
2.3.1 Nizkoprepustni filter . . . 5
2.3.2 Standardni odklon . . . 6
2.3.3 Algoritmi za zaznavanje vrhov . . . 6
2.3.4 Valovna transformacija . . . 8
2.3.5 Fourierjeva transformacija . . . 9
2.3.6 Strojno učenje . . . 9
2.4 Prikaz podatkov . . . 9
2.4.1 Python . . . 10
2.4.2 OpenStreetMap . . . 10
2.5 Statistični testi . . . 12
2.5.1 Kolmogorov-Smirnov preizkus . . . 12
2.5.2 Wilcoxonov preizkus predznačenih rangov . . . 12

3 Materiali in metode dela **14**
3.1 Anketa . . . 14
3.1.1 Splošni del . . . 14
3.1.2 Odseki . . . 15
3.2 Zbiranje podatkov . . . 16
3.2.1 Namenska aplikacija . . . 16
3.2.2 Aplikacija SensorKraken . . . 16
3.2.3 Zajem podatkov . . . 21
3.3 Obdelava podatkov . . . 27
3.3.1 Nalaganje podatkov in konfiguracija . . . 27
3.3.2 Deljenje na odseke . . . 29
3.3.3 Ocenjevanje odsekov . . . 32
3.3.4 Izvoz podatkov . . . 37
3.4 Prikaz podatkov . . . 39

4 Rezultati **41**
4.1 Rezultati ankete . . . 41
4.1.1 Odsek Momax . . . 45

4.1.2 Odsek Jezero 46
4.1.3 Odsek Tomšičeva 47
4.1.4 Odsek Sončni park 48
4.1.5 Odsek Velenjka 49
4.1.6 Odsek Gorenje 50
4.2 Rezultati meritev 51
4.2.1 Cenilka mean 51
4.2.2 Cenilka std 51
4.2.3 Cenilka meanstd 51
4.2.4 Cenilka minmaxdiff 52
4.2.5 Cenilka bumpcount 52

5 Diskusija 53
5.1 Merjenje 53
5.2 Ocene odsekov 53
5.2.1 Povprečne ocene iz ankete 53
5.2.2 Ocene cenilk 53
5.2.3 Statistična primerjava 55
5.3 Izbira poti 56

6 Zaključek 57

7 Povzetek 58

8 Summary 59

9 Viri in literatura 60

SEZNAM TABEL

1 Vrednosti T izražene kot delež standardnega odklona 36

2 Ocene cenilke mean 51
3 Ocene cenilke std 51
4 Ocene cenilke meanstd 52
5 Ocene cenilke minmaxdiff 52
6 Ocene cenilke bumpcount 52

7 Odklon ocen cenilk od rezultatov ankete 55
8 Rezultati statističnih preizkusov 56

SEZNAM SLIK

1 Akcijska kamera GoPro . . . 3
2 Primer Jupyter notebook . . . 10
3 Izrisan zemljevid Slovenije . . . 11
4 Prvi zaslon aplikacije . . . 17
5 Nastavitve aplikacije . . . 17
6 Seznam senzorjev . . . 18
7 Trenutne vrednosti senzorjev . . . 18
8 Shranjene meritve senzorjev . . . 19
9 Izvožene meritve senzorjev . . . 19
10 Primer nosilca za telefon za kolo . . . 21
11 Primer torbe za telefon . . . 22
12 Namestitev telefona na kolo . . . 23
13 Cestno kolo . . . 24
14 Hibridno kolo . . . 24
15 Prva pot zajemanja . . . 25
16 Druga pot zajemanja . . . 25
17 Tretja pot zajemanja . . . 26
18 Interaktivni zemljevid . . . 40

SEZNAM FORMUL

1 Formula standardnega odklona 6

SEZNAM KODE

1 Primer branja podatkov senzorjev v Java aplikaciji 4
2 Enostaven primer kode za zaznavanje vrhov 6
3 Zaznavanje vrhov s knjižnico scipy 7
4 Primer uporabe PyWavelets 8
5 Primer Python kode 10
6 Primer uporabe OpenStreetMap zemljevida s Python 11
7 Primer izvedbe Wilcoxonovega preizkusa predznačenih rangov 13
8 Primer podatkov senzorja za pospešek 20
9 Konfiguracija programa 27
10 Nalaganje podatkov 28
11 Deljenje na odseke 30
12 Projekcija vektorjev pospeška na Z os 31
13 Ocenjevanje odseka 33
14 Iskanje vrednosti T za algoritem bumpcount 35
15 Ustvarjanje linij za zemljevid 37
16 JSON izvoz linij na zemljevidu 38
17 Prikaz na zemljevidu v Python 38
18 Prikaz na zemljevidu v JavaScript 39

SEZNAM GRAFOV

1 Primer rezultata zaznavanja vrhov 7
2 Primer rezultata valovne transformacije 9
3 Prvih 1000 podatkov senzorja pospeška 29
4 Primerjava projeciranih vrednosti in originalnih vektorjev pospeška . 32
5 Histogram vrednosti senzorja pospeška 34
6 Optimalne vrednosti T za cenilko bumpcount 36
7 Pogostost kolesarjenja po Velenju med anketiranci 41
8 Uporabljeni tipi koles pri anketirancih 42
9 Primerjava pogostosti kolesarjenja po cestah in kolesarskih površinah 42
10 Razlogi za pretežno kolesarjenje po cestah 43
11 Udobnost vožje po kolesarskih površinah 43
12 Udobnost vožje po cestah 44
13 Vpliv kvalitete površine na izbiro poti pri anketirancih 44
14 Mnenje o kvaliteti odseka Momax 45
15 Izognitev odseku Momax pri anketirancih 45
16 Mnenje o kvaliteti odseka Jezero 46
17 Izognitev odseku Jezero 46
18 Mnenje o kvaliteti odseka Tomšičeva 47
19 Izognitev odseku Tomšičeva 47
20 Mnenje o kvaliteti odseka Sončni park 48
21 Izognitev odseku Sončni park 48
22 Mnenje o kvaliteti odseka Velenjka 49
23 Izognitev odseku Velenjka 49
24 Mnenje o kvaliteti odseka Gorenje 50
25 Izognitev odseku Gorenje 50
26 Ocene segmentov 54
27 Primerjava povprečnih ocen anketirancev 55

SEZNAM KRATIC

GPS – globalni sistem za določanje položaja (angl. Global Positioning System)
CSV – vrednosti, ločene z vejico (angl. Comma Separated Values)
JSON – objektni zapis JavaScript (angl. JavaScript Object Notation)
HTML – jezik za označevanje besedila (angl. Hypertext Markup Language)

1 UVOD

Kolesarjenje je v zadnjem času vedno bolj razširjena aktivnost in prav tako način transporta. Kot posledica tega se v mestni občini Velenje vedno bolj spodbuja uporaba koles, prav tako pa se izvajajo projekti gradnje novih kolesarskih površin. [1] [2]

1.1 PROBLEM

Kljub razširjenosti kolesarjenja je v kolesarskih krogih pogosto slišati (in utemeljeno tudi z rezultati ankete, izvedeni pozneje v tej nalogi) mnenje kolesarjev, da so nekatere kolesarske površine v Velenju neprimerne za uporabo. Kot cilj te raziskovalne naloge sem si zadal objektivno izmeriti razne značilnosti površin in meritve primerjati z rezultati ankete.

Tukaj se je takoj pojavil izziv, kako te podatke izmeriti in kako najbolje zastaviti anketo, da pridobim čim bolj realne podatke pri obeh.

1.2 HIPOTEZE

Pred pričetkom raziskovanja sem si zadal naslednje hipoteze:

1. Mogoče je izmeriti kvaliteto odseka kolesarske površine brez specializiranih naprav.
2. Izmerjena kvaliteta posameznih odsekov kolesarske površine se ujema z mnenjem kolesarjev o kvaliteti tega odseka.
3. Kvaliteta kolesarskih površin vpliva na izbiro poti pri kolesarjih.

2 PREGLED OBJAV

Kolesarke površine in kolesarsko omrežje je v zadnjem času postalo vedno bolj pogosto uporabljeno za transport ali rekreacijo. V svoji uporabi je podobno cestnemu omrežju, zato je pri pregledu obstoječih objav relevanten pregled raziskav, ki so se ukvarjale z analizo kvalitete cestnega omrežja.

2.1 OBSTOJEČE RAZISKAVE

Prva raziskava, ki se ukvarja s tematiko ocenjevanja cestnih površin ima naslov "Road visualization for smart city: Solution review with road quality qualification". Ukvarja se z iskanjem in klasifikacijo znakov propadanja cest (luknje, razpoke ipd.). To so raziskovalci delali z uporabo nevronske mreže, s katero so klasificirali slike, zajete s kamero pri 15 sličicah na sekundo. Ta pristop je bil (odvisno od tipa znaka propada) uspešen med 50 % in 90 %. [3]

Druga raziskava, nosi naslov "Road quality analysis and mapping for faster and safer travel". Pri tej so se raziskovalci ukvarjali z naslednjim problemom: tradicionalni sistemi za navigacijo (kot na primer Google Maps) nas vodijo po najhitrejših poteh, kjer je upoštevana tudi kvaliteta ceste, saj delujejo na osnovi preteklih voženj, kjer ponavadi hitrejša vožnja pomeni boljšo cesto, vendar nam ne omogočajo pregleda nad kvaliteto posameznih cest. Raziskovalci so torej izdelali namensko napravo z uporabo mikrokontrolerja Arduino, GPS senzorja in pospeškovnega senzorja, s katero so merili tresljaje na cestni površini. Nato so izračunali standardni odklon za posamezni odsek ceste in s tem določili kvaliteto tega odseka. [4]

V članku z naslovom "Automated Sensing System for Monitoring of Road Surface Quality by Mobile Devices" so avtorji izdelali sistem, ki je z uporabo pospeškovnega senzorja v Android mobilnem telefonu in pa GPS senzorja določal "anomalije"na cesti. Te podatke je nato pošiljal na centralni strežnik, kjer so ustvarili zemljevid vseh anomalij, ki je pomagal pri varnosti na cestah. Rezultat tega bi bil tudi minimizacija ročnih pregledov cest, kar izboljša varnost za določene delavce. [5]

Naslednji članek z naslovom "On the analysis of road surface conditions using embedded smartphone sensors", sta avtorja opravila meritve cest z giroskopskim senzorjem, nato pa iskala način, kako primerjati in uskladiti meritve na enakem delu ceste, da sta dobila bolj realne rezultate. [6]

Obstaja še nekaj raziskav, ki se na podobne načine kot zgornje ukvarjajo bodisi z analizo kvalitete cest, bodisi s procesiranjem podatkov analize, bodisi z uporabo teh rezultatov za povečanje varnosti na cestah. Primer raziskave, ki je uporabljala za analizo cestne površine pospeškovni senzor v povezavi z nevronsko mrežo, je raziskava avtorja Alessia Martinelli ([7]). Podobna ideja, ki jo razvijam jaz s to raziskavo, je tudi v Kumarjevem članku ([8]), kjer razvija sistem za zaznavanje in upravljanje popravil na cesti z uporabo senzorjev v mobilnem telefonu.

2.2 NAČINI ZBIRANJA PODATKOV

V zgornjih raziskavah je predstavljenih več načinov zbiranja podatkov o stanju cest, ki jih lahko uporabimo tudi za zbiranje podatkov o stanju (in posledično kvaliteti) kolesarskih površin.

2.2.1 VIDEO

Prvi način je zajemanje videa med vožnjo in poznejša analiza prav tega. Za to je mogoče uporabiti akcijske kamere kot na primer GoPro (Slika 1) ali telefonsko kamero, ki je bolj praktična iz vidika cene in dostopnosti.

Vir: lasten

Slika 1: Akcijska kamera GoPro

2.2.2 MOBILNI TELEFON

Drugi način za zbiranje podatkov je uporaba senzorjev, ki jih imajo vsi moderni mobilni telefoni, torej pospeškovni in giroskopski senzor, prav tako pa vgrajeni GPS modul. Do podatkov teh senzorjev je mogoče dostopati s svojo aplikacijo, obstajajo pa že aplikacije, ustvarjene prav za ta namen.

Najbolj priljubljeni senzorji, ki jih ima večina pametnih telefonov, so merilnik pospeška, žiroskop, magnetometer, mikrofon in kamera. Dostop do teh je mogoč z uporabo sistemskih funkcij pri razvoju aplikacije. Primer kode, ki bere vrednosti senzorja svetlobe, je viden v kodi 1. [9]

```java
public class SensorActivity
extends Activity
implements SensorEventListener {
    private SensorManager sensorManager;
    private Sensor mLight;

    @Override
    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        sensorManager = (SensorManager) getSystemService(
            Context.SENSOR_SERVICE
        );
        mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
    }

    @Override
    public final void onSensorChanged(SensorEvent event) {
        // Senzor svetlobe vrne 1 vrednost - 1. element
        // Veliko senzorjev vrne 3 (x, y, z) vrednosti.
        float lux = event.values[0];
        // Tukaj naredimo nekaj s to vrednostjo ...
    }

    @Override
    protected void onResume() {
        super.onResume();
        sensorManager.registerListener(
            this,
            mLight,
            SensorManager.SENSOR_DELAY_NORMAL
        );
    }

    @Override
    protected void onPause() {
        super.onPause();
        sensorManager.unregisterListener(this);
    }
}
```

Koda 1: Primer branja podatkov senzorjev v Java aplikaciji

Zgornja koda je povzeta po [10].

2.2.3 NAMENSKA NAPRAVA

Naslednji način, ki sem ga zasledil pri obstoječih raziskavah, je izdelava namenske naprave, na primer z uporabo mikrokontrolerja Arduino ter senzorjev (senzor pospeška, žiroskop, ultrasonični senzor, mikrofon ...). Ta način je, v primerjavi z uporabo senzorjev v mobilnem telefonu (v predpostavki, da imamo mobilni telefon na voljo) malce zahtevnejši in dražji, omogoča pa prilagoditev naprave za naše namene. To pomeni namensko ohišje, možnost za senzorje z višjo natančnostjo ter zbiranje podatkov na način, ki je bolj praktičen za analizo.

Pri tem načinu so lahko težave z zbiranjem GPS podatkov in pa, v primeru, da je to zaželeno, pošiljanjem podatkov po internetu. Za to se je potrebno ukvarjati z internetno povezavo in pa pisanjem kode, ki vse to izvaja.

2.2.4 ULTRASONIČNI SENZOR IN MIKROFON

V raziskavi Jagatheesaperumala ([11]) je predstavljena uporaba ultrasoničnega senzorja za zaznavanje značilnosti v cestni površini. Za oceno stanja cest je oblikovan detektor cestne površine z mikrofonom, ki zbira zvočne, signale ter ultrazvočni modul, ki opazuje globinske informacije ceste. Celotna strojna oprema je nameščena v platišče koles vozil. Zbrane podatke s cest nato v raziskavi analizira z uporabo algoritmov strojnega učenja.

2.3 PROCESIRANJE PODATKOV

Pomemben del ocenjevanja kvalitete kolesarskih površin je dejanska analiza izmerjenih podatkov, zato so v tem delu predstavljeni koncepti in algoritmi, ki so pri tem opravilu relevantni.

2.3.1 NIZKOPREPUSTNI FILTER

Nizkoprepustni filter (angl. lowpass filter) je tehnika, ki pri procesiranju signala odstrani visoko-frekvenčne komponente in ohrani le nizko-frekvenčne. To omogoča odstranjevanje šuma pri signalu. Konkretno pri podatkih pospeškovnega senzorja je lahko uporaben za odstranjevanje manjših nepopolnosti v kolesarski površini, kar omogoča, da pridejo dejanske značilnosti (razpoke, luknje, robniki) bolj do izraza. [12]

2.3.2 STANDARDNI ODKLON

Standardni odklon nam pove, kako razpršeni so podatki v nekem skupku od njihove aritmetične sredine. V primeru procesiranja signala je to uporabno kot statistični pokazatelj, ki nam pove, kako "miren"oziroma stabilen je signal. Pri konkretni uporabi na podatkih senzorja pospeška je izračun standardnega odklona uporaben kot dober statistični indikator značilnosti tega odseka. Formulo standardnega odklona vidimo v formuli 1.

$$\sigma = \sqrt{\frac{\sum(x-\bar{x})^2}{N}}$$

$kjerje$

$\sigma = standardni\ odklon$
$x = posamezna\ vrednost$
$\bar{x} = aritmetična\ sredina\ vseh\ vrednosti$
$N = število\ vrednosti$

Formula 1: Formula standardnega odklona

2.3.3 ALGORITMI ZA ZAZNAVANJE VRHOV

Osnovna ideja postopka določanja vrhov je prepoznati vrednosti v podatkovnem nizu, ki so večje od svojih sosedov in s tem predstavljajo lokalne maksimume.

Zelo enostaven primer takšnega algoritma, ki pa vseeno predstavi osnovno idejo, je naveden v kodi 2. V tem algoritmu je *prag* vrednost, ki jo določimo in je mejna vrednost, ki jo mora preseči točka v nizu, da je označena kot vrh. Prilagajanje praga omogoča prilagajanje občutljivosti algoritma.

```
def detekcija_vrhov(seznam, prag=0.5):
    vrhovi = []
    for i in range(1, len(seznam)-1):
        if (
            seznam[i] > prag
            and seznam[i] > seznam[i-1]
            and seznam[i] > seznam[i+1]
        ):
            vrhovi.append(i)
    return vrhovi
```

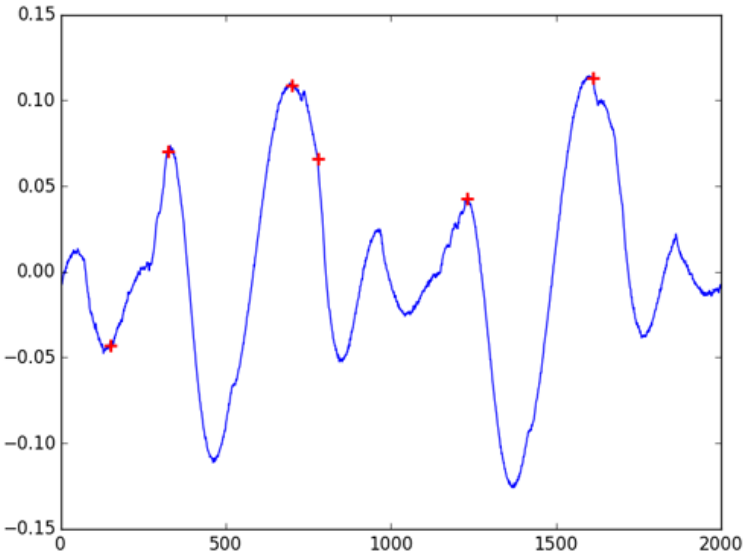
Koda 2: Enostaven primer kode za zaznavanje vrhov

Obstajajo knjižnice (kot na primer scipy.signal), ki implementirajo zahtevnejše algoritme za zaznavanje vrhov. Primer uporabe scipy je prikazan v kodi 3.

```
import numpy as np
from scipy.signal import find_peaks_cwt
cb = np.array([-0.010223, ... ])

# to nam vrne indekse vrhov
indexes = find_peaks_cwt(cb, np.arange(1, 550))
```

Koda 3: Zaznavanje vrhov s knjižnico scipy

Primer grafa z označenimi vrhovi, spet z uporabo scipy, vrne rezultat viden v grafu 1.



Vir: Yoan Tournade

Graf 1: Primer rezultata zaznavanja vrhov

Zgornji primer je povzet po [13].

2.3.4 VALOVNA TRANSFORMACIJA

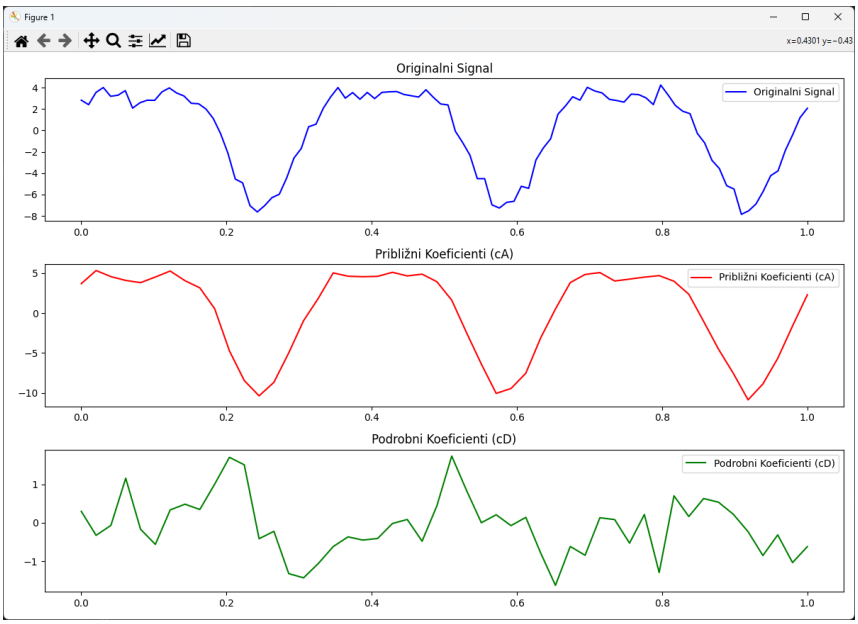
Valovna transformacija (angl. wavelet transfor) je tehnika, ki omogoča deljenje signalov ali podatkov v različne valovne komponente različnih frekvenc. S to metodo lahko identificiramo lokalne spremembe v signalih. [14]

Primer Python programa, ki z uporabo knjižnice PyWavelets nek signal razdeli na različne komponente, vidimo v kodi 4.

```
import numpy as np
import matplotlib.pyplot as plt
import pywt

# Generiranje primerjalnih podatkov z nihanji
cas = np.linspace(0, 1, 100)
podatki = (
    5 * np.sin(2 * np.pi * 3 * cas)
    + 2 * np.cos(2 * np.pi * 6 * cas)
    + np.random.normal(0, 0.5, 100)
)

# Izvedba diskretne valčne transformacije (DWT)
koeficienti = pywt.dwt(podatki, 'db1', mode='symmetric')

# Pridobitev časovnih vrednosti za koeficiente
cas_koeficienti = np.linspace(0, 1,
```

Koda 4: Primer uporabe PyWavelets

Izpis zgornje kode je viden na grafu 2. Vidimo 2 grafa, ki sta del (komponenti) osnovnega grafa.



Graf 2: Primer rezultata valovne transformacije

2.3.5 FOURIERJEVA TRANSFORMACIJA

Fourierjeva transformacija je matematična operacija, ki pretvori signal iz časovne domene v frekvenčno domeno. Ta transformacija se pogosto uporablja za analizo periodičnih signalov in omogoča razčlenitev kompleksnega signala na sestavne sinusne in kosinusne valove.

Za digitalne signale se pogosto uporablja diskretna oblika Fourierjeve transformacije, imenovana Diskretna Fourierjeva Transformacija (DFT), ki se izračuna s pomočjo hitre Fourierjeve transformacije (FFT) za učinkovito računanje. [15]

2.3.6 STROJNO UČENJE

Strojno učenje ima ključno vlogo pri analizi signalov, še posebej pri razpoznavanju vzorcev, kjer algoritmi omogočajo samodejno identifikacijo kompleksnih struktur in karakteristik v signalih. Ta pristop omogoča avtomatizirano klasifikacijo in filtriranje signalov. Za namene te raziskovalne naloge ta pristop ni preveč relevanten, saj ostajam na osnovnih pristopih k analizi signalov.

2.4 PRIKAZ PODATKOV

Ko so podatki (bodisi video, bodisi senzorski) zbrani z enim od zgoraj navedenih načinov, jih je potrebno obdelati. Za to obstaja več načinov, najbolj relevantni (torej ti, ki sem jih v raziskovalni nalogi tudi uporabil), so predstavljeni v nadaljevanju.

2.4.1 PYTHON

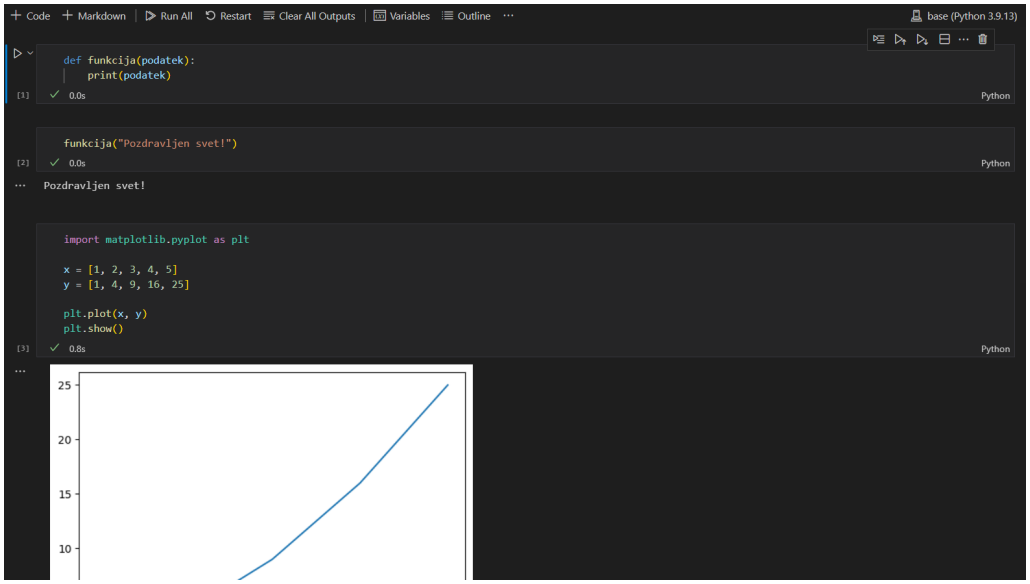
Python je visokonivojski programski jezik, namenjen uporabnikom z razponom znanja vse od začetnikov do izkušenih uporabnikov. Omogoča hiter razvoj raznih aplikacij. V realnem svetu se uporablja za vse od razvoja spletnih aplikacij do raznega procesiranja podatkov v raziskovalne namene. [16]

Primer enostavnega programa, ki izpiše "Pozdravljen, svet!", vidimo v kodi 5.

```python
def main():
    print("Pozdravljen, svet!")

if __name__ == "__main__":
    main()
```

Koda 5: Primer Python kode

Obstaja projekt Jupyter notebook, ki omogoča interaktiven razvoj Python kode. Primer Jupyter notebook-a je na sliki 2. [17].



Slika 2: Primer Jupyter notebook

Ta način razvoja oziroma ta format je uporaben predvsem, ko želimo imeti posamezne dele kode, ki jih razvijamo posamezno, kot celota pa izvajajo nek proces. Lahko se zgodi, da izvajanje nekega koraka traja dlje. V tem primeru lahko izvedemo ta korak enkrat, ostale (za njim) pa v lahko prihodnosti izvedemo večkrat. Pri teh izvajanjih bo uporabljeno takšno stanje programa, kot je bilo po zadnjem izvajanju.

2.4.2 OPENSTREETMAP

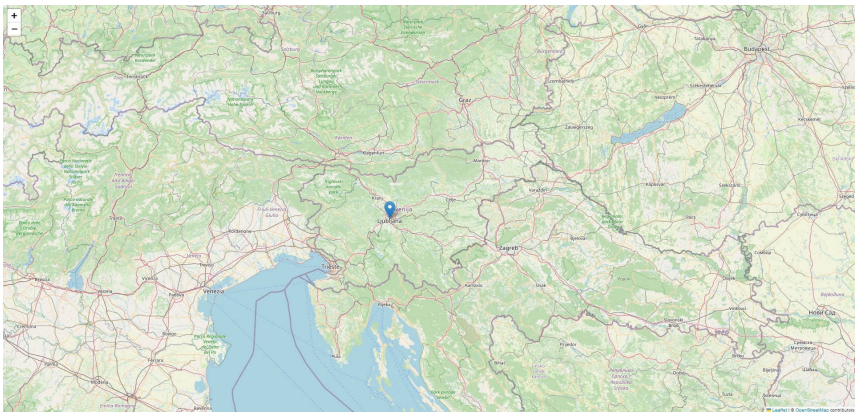
OpenStreetMap je baza geografskih podatkov, ki je odprta javnosti in zastonj. Je projekt, h kateremu prispevajo mnogi ljudje iz celega sveta in ponuja vse od lokacij podjetji do zastonj storitve, ki ponuja spletne zemljevide. [18]

Za uporabo OpenStreetMap zemljevidov obstajajo knjižnice v več programskih jezikih. Primer knjižnice za programski jezik Python je Folium. Primer kode, ki izriše zemljevid Slovenije vidimo v kodi 6.

```
# Ustvari zemljevid
map = folium.Map(location=[46.1512, 14.9955], zoom_start=8)

# Dodaj marker za Ljubljano
folium.Marker([46.0569, 14.5058], popup='Ljubljana').add_to(map)

# Shrani zemljevid v HTML datoteko
map.save('slovenia_map.html')
```

Koda 6: Primer uporabe OpenStreetMap zemljevida s Python

Izrisan zemljevid vidimo na sliki 3. Primeri povzeti po [19].



Slika 3: Izrisan zemljevid Slovenije

2.5 STATISTIČNI TESTI

V tem poglavju sta razložena dva statistična preizkusa, ki sem ju uporabil v končnem delu raziskovalne naloge.

2.5.1 KOLMOGOROV-SMIRNOV PREIZKUS

Namen

Ta preizkus (ali specifično različica tega preizkusa, ki deluje na dveh vzorcih) se uporablja za ocenjevanje smiselnosti domneve, da dva vzorca ne prihajata iz iste populacije. Lahko bi rekli, da preizkus preverja, ali sta dve skupini podatkov del iste porazdelitve ali ne. [20]

Hipoteze

Hipoteze v statistiki so trditve, ki jih želimo preveriti na podlagi razpoložljivih podatkov. V tem preizkusu imamo dve hipotezi:

- Ničelna hipoteza (H_0): Porazdelitvi vzorcev sta enaki.
- Alternativna hipoteza (H_1): Porazdelitvi vzorcev nista enaki.

2.5.2 WILCOXONOV PREIZKUS PREDZNAČENIH RANGOV

Namen

Ta preizkus se uporablja, kadar želimo primerjati dve povezani skupini podatkov. Uporablja se, kadar predpostavka normalne porazdelitve ni izpolnjena ali kadar so podatki rangirani (torej ne vrednosti v intervalski ali razmeriski lestvici). [21]

Hipoteze

Podobno kot pri prejšnjem preizkusu, imamo tudi tukaj dve hipotezi:

- Ničelna hipoteza (H_0): Ni razlike med parnimi opazovanji.
- Alternativna hipoteza (H_1): Obstaja razlika med parnimi opazovanji.

Primer

Primer, kjer je ta preizkus uporabljen na dveh skupinah podatkov je predstavljen v kodi 7.

```python
from scipy.stats import wilcoxon

# Generiranje vzorcev
before = [28, 31, 32, 30, 32, 29, 28, 30, 34, 30]
after = [31, 32, 30, 32, 33, 28, 31, 32, 33, 31]

# Izvajanje Wilcoxonov preizkusa
statistic, p_value = wilcoxon(before, after)

print("Statistika Wilcoxon:", statistic)
print("p-vrednost:", p_value)
```

Koda 7: Primer izvedbe Wilcoxonovega preizkusa predznačenih rangov

Ob zagonu zgornje kode, dobimo vrednost $p = 0.16015625$. S tem lahko trdimo, da ni dovolj statistično značilnih dokazov za zavrnitev ničelne hipoteze.

3 MATERIALI IN METODE DELA

Opravljanje raziskave sem si zamislil v 3 glavnih delih. Prvi del je anketa, iz katere bi pridobil podatke o splošnih mnenjih kolesarjev in konkretnih mnenjih o odsekih, na katerih bi baziral nadaljnje raziskovanje. Drugi del je zbiranje objektivnih podatkov o kolesarskih površinah, torej nekakšno merjenje kvalitete odsekov. Tretji del je obdelava ter prikaz teh podatkov na uporaben način.

Pred začetkom je vredno razložiti mojo uporabo besede odsek. Ko govorim o odseku kolesarske površine, govorim o delu kolesarske poti ali steze (primer: kolesarska ob neki glavni cesti), ki je dolg nekje med nekaj deset in nekaj sto metri. Primeri teh odsekov so vidni v anketi. Ko razlagam kodo za procesiranje signalov, se pojavi beseda odsek v kontekstu GPS podatkov, ki so združeni s senzorskimi meritvami, toda ti odseki niso enaki kot tisti, o katerih govorim drugje po nalogi.

Večkrat v tej nalogi se pojavi termin "v Velenju in okolici". Ta izraz ima v kontekstu naloge enak pomen kot "v Šaleški dolini", torej sta izmenljiva. Prav tako se na območje Šaleške doline večkrat sklicujem kot "to območje".

3.1 ANKETA

Raziskavo sem začel z anketo. Ciljna skupina za anketo so mi bili ljudje, ki so že kdaj kolesarili po Velenju, želel sem pa anketirati čim več ljudi, ki čim bolj pogosto kolesarijo po tem območju, saj zaradi tega bolj poznajo kolesarske površine in so njihova mnenja bolj izoblikovana.

Anketo sem logično razdelil na 2 dela. Prvi je namenjen pridobivanju splošnega mnenja kolesarjev o kolesarskih površinah v Velenju in okolici, drugi pa pridobivanju mnenja o konkretnih odsekih.

3.1.1 SPLOŠNI DEL

Za splošni del sem postavil nekaj splošnih vprašanj:

- *"Kako pogosto kolesarite po kolesarskih površinah in cestah v Velenju?"* – S tem vprašanjem pridobim približno oceno, kako dobro kolesar pozna kolesarske površine na tem območju.
- *"Katere tipe kolesa uporabljate za kolesarjenje?"* – To vprašanje ni vezano direktno na raziskavo, lahko pa se izkaže kot uporabno za povezovanje mnenj kolesarjev s tem, kakšna kolesa uporabljajo.
- *"Ali bolj pogosto kolesarite po cestah ali po kolesarskih površinah?"*
- *"Kaj je razlog za to izbiro?"* – To vprašanje je postavljeno le, če je odgovor na prejšnjega pozitiven. Odgovori na to vprašanje se znajo izkazati kot uporabni pri ugotavljanju, kako kolesarji izbirajo površine, po katerih kolesarijo.

- *"Od 1-5 ocenite kvaliteto in udobnost vožnje po kolesarskih površinah v Velenju in okolici."*
- *"Od 1-5 ocenite kvaliteto in udobnost vožnje po cestah v Velenju in okolici. (Če se po njih vozite)"* – S tem in prejšnjim vprašanjem dobimo neko splošno idejo o mnenju kolesarjev o različnih površinah, nista pa vezana direktno na raziskavo.
- *"Ali kdaj kvaliteta površine vpliva na vašo izbiro poti, po kateri se peljete s kolesom?"* – S tem vprašanjem direktno naslovimo tretjo hipotezo na splošno.

3.1.2 ODSEKI

Drugi logični del ankete je namenjen zbiranju podatkov o konkretnih odsekih kolesarskih površin v Velenju in okolici. Bolj specifično sem zbiral mnenje o udobnosti vožnje po posameznih odsekih (seveda le v primeru, da anketiranec ta odsek pozna).

V anketi sem prikazal sliko posameznega odseka na zemljevidu in o njem postavil eno do tri vprašanja.

Opomba: V anketi sem uporabljal namesto besede "odsek" besedo "segment".

Postavljena vprašanja so bila naslednja:

- *"Ali prepoznate ta segment (in ste se po njem že vozili s kolesom)?"*
- *"Od 1-5 ocenite udobnost vožnje po tem segmentu kolesarske površine."* – Ta vprašanja so ključna, saj z njimi pridobim konkretna mnenja o posameznih odsekih.
- *"Ali bi se med vožnjo zaradi kvalitete površine izognili temu segmentu in peljali po alternativni poti?"* – S temi vprašanji bolj specifično ciljam na tretjo hipotezo.

Za anketo (in torej tudi za celotno raziskavo) sem izbral 6 odsekov, po katerih sem bolj podrobno spraševal. Tem odsekom sem podal imena, katera bodo uporabljena tudi v nadaljevanju naloge.

- **Odsek Momax** – Poteka po kolesarski poti od Žarove ceste 25 do nakupovalnega centra Velenjka.
- **Odsek Jezero** – Poteka po kolesarski poti ob velenjski plaži.
- **Odsek Tomšičeva** – Poteka po kolesarski stezi ob Tomšičevi cesti.
- **Odsek Sončni park** – Poteka po kolesarski stezi ob Cesti pod parkom.
- **Odsek Velenjka** - Poteka po kolesarski stezi od nakupovalnega centra Velenjka do gostilne na Hofu.
- **Odsek Gorenje** – Poteka po kolesarski stezi in poti ob Partizanski cesti do trgovine Hofer.

Anketo sem ustvaril s spletnim orodjem Google Forms. Deljenje sem začel s pošiljanjem vsem, za katere sem vedel, da se rekreativno ukvarjajo s kolesarjenjem v tej regiji, prav tako pa vsem, za katere sem vedel, da se včasih kam peljejo s kolesom. Nadaljeval sem z objavo na aplikaciji za rekreacijo Strava, prav tako pa objavo na socialnem omrežju Facebook. Družinske člane in znance sem prosil za deljenje teh objav.

3.2 ZBIRANJE PODATKOV

Po končani anketi sem se osredotočil na zbiranje podatkov o kvaliteti odsekov, ki jih bom nato primerjal z rezultati ankete.

Najprej sem si zbiranje zamislil z namensko napravo, toda ko sem začel pregledovati načine izdelave, sem hitro ugotovil, da bo ta način vzel precej časa, prav tako pa nimam dovolj znanja na tem področju, da bi lahko takšno napravo izdelal.

Za tem sem dobil idejo za uporabo senzorjev, ki so vgrajeni v mobilne telefone.

3.2.1 NAMENSKA APLIKACIJA

Sprva sem se lotil izdelave namenske aplikacije, ki bi ob kliku na gumb začela shranjevati senzorske podatke ter jih na koncu zapisala v JSON ali CSV datoteko za nadaljno obdelavo.

Za zgled sem si vzel kodo aplikacije, ki sem jo našel na spletu (projekt Sensors uporabnika bansalanurag na spletni platformi Github [22]). Ta aplikacija prikazuje trenutne podakte senzorjev, omogoča zajem teh podatkov in pa njihovo shranjevanje v CSV datoteko.

Aplikacijo sem želel spremeniti na način, da bi ta zajem deloval v ozadju. Želel sem tudi, da bi omogočala izbiro drugih senzorjev, v primeru, da bi te potreboval. Ob iskanju informacij ob razvoju aplikacije sem slučajno naletel na omembo aplikacije SensorKraken, zato sem nadaljeval z raziskovanjem v to smer – torej v smeri obstoječe končane aplikacije. Hitro sem ugotovil, da je aplikacija SensorKraken primerna za mojo uporabo, zato sem zavrgel idejo izdelave namenske aplikacije.

3.2.2 APLIKACIJA SENSORKRAKEN

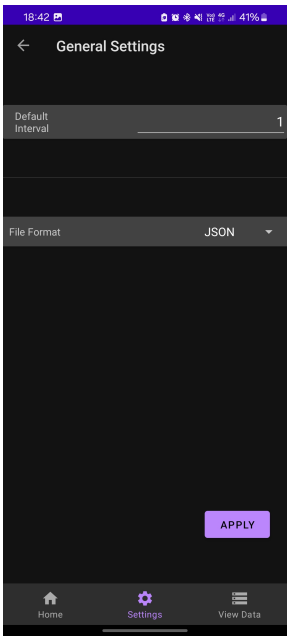
Aplikacija SensorKraken je odprtokodna aplikacija, izdelana z namenom zapisovanja senzorskih podatkov. [23]

Je zelo nastavljiva, saj omogoča spreminjanje intervala zajemanja podatkov, izbiro senzorjev, katerih meritve zajemamo, spremembo načina shranjevanja podatkov in mnogo drugih opcij.

Na sliki 4 vidimo prvi zaslon aplikacije SensorKraken, ki omogoča začetek zajema podatkov senzorjev ter testiranje senzorjev. Na sliki 5 vidimo nastavitve, kjer sem nastavil interval zajema (angl. default interval) na 1 (kar pomeni eno milisekundo), saj s tem na vseh senzorjih podatke aplikacija zajema tako pogosto, kot to senzor

omogoča. Prav tako vidimo, da sem izbral način izpisa JSON (JavaScript Object Notation), kar pozneje omogoča enostaven prenos in obdelavo zajetih podatkov.

Slika 4: Prvi zaslon aplikacije

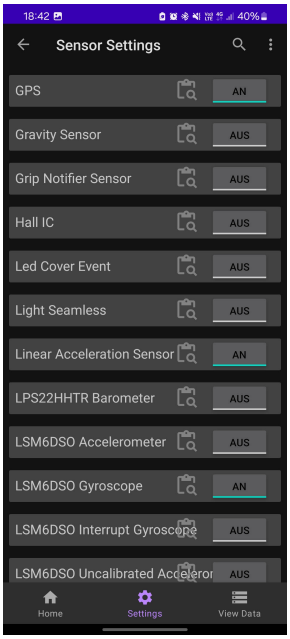


Slika 5: Nastavitve aplikacije

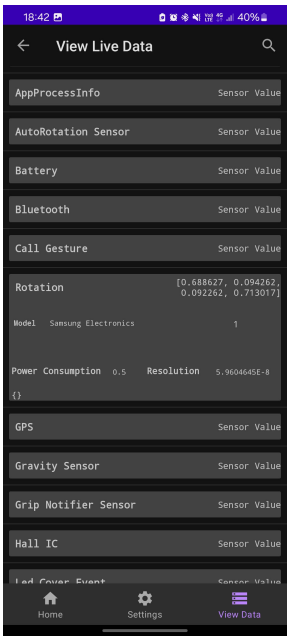
Aplikacija omogoča torej tudi izbiro, katere podatke naj zajema, kar vidimo na sliki 6. Izbral sem zajem GPS podatkov, podatkov senzorja linearnega pospeška (angl. linear acceleration sensor), rotacijskega vektorja (angl. rotation vector), ki nam pove trenutno orientacijo telefona v globalnem prostoru (glede na zemljino

magnetno polje) ter žiroskopa, ki nam pove spremembo rotacije telefona (podatkov tega pozneje nisem uporabil).

Trenutne vrednosti senzorjev (za namene testiranja oziroma razumevanja podatkov) si lahko prav tako ogledamo v aplikaciji, to vidimo na sliki 7.

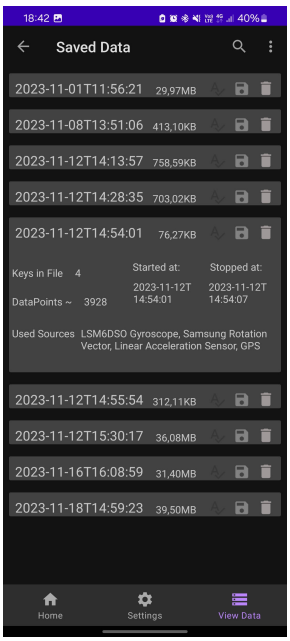


Slika 6: Seznam senzorjev

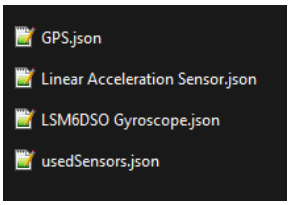


Slika 7: Trenutne vrednosti senzorjev

Po pripravljenih nastavitvah, lahko začnemo z dejanskim zajemov podatkov med vožnjo s kolesom. Kako je bilo to izvedeno v sklopu te raziskovalne naloge, je bolj podrobno opisano v poglavju Zajem podatkov 3.2.3.



Slika 8: Shranjene meritve senzorjev



Slika 9: Izvožene meritve senzorjev

Ko smo z zajemom podatkov končali, je potrebno te podatke izvoziti na način, da jih bo mogoče obdelati. To aplikacija omogoča na zaslonu s shranjenimi zajemi (slika 8). Pritisk na gumb za shranjevanje izvozi podatke v obliko JSON, vsak senzor v svojo datoteko (kot razvidno na sliki 9).

Struktura podatkov v datotekah je odvisna od tipa senzorja. Primer podatkov za senzor pospeška vidimo v kodi 8. Na vrhu JSON strukture vidimo nekaj splošnih podatkov o senzorju, nato pa v polju *readOuts* vidimo objekte, ki predstavljajo posamezne meritve. Vsaka meritev vsebuje časovni žig (*TimeStampSensor*) in pa 3 vrednosti (x, y, z) senzorja.

```
{
   "Linear Acceleration Sensor":{
      "SensorInfo":{
         "Name":"Linear Acceleration Sensor",
         "Type":10,
         "Vendor":"Samsung Electronics",
         "TimeStampBoot":1698581613277576085,
         "Resolution":0.0023942017,
         /* ostali podatki senzorja, izpuščeni za jasnost primera */
      },
      "readOuts":[
         {
            "TimeStampSensor":254569436453334,
            "TimeKrakenEvent":254569446207492,
            "Values":"[2.3642263, -1.4422553, 2.894227]",
            "Errors":[

            ]
         },
         {
            "TimeStampSensor":254569492184915,
            "TimeKrakenEvent":254569498198799,
            "Values":"[2.4079056, 0.27260303, 1.8801003]",
            "Errors":[

            ]
         },
         {
            "TimeStampSensor":254569529338915,
            "TimeKrakenEvent":254569537324453,
            "Values":"[2.380877, -0.32351637, 1.742856]",
            "Errors":[

            ]
         }, /* ... ostale meritve */
      ]
   }
}
```


Koda 8: Primer podatkov senzorja za pospešek

3.2.3 ZAJEM PODATKOV

Dejanski zajem podatkov sem izvajal z uporabo aplikacije SensorKraken, ki je zajemala podatke določenih senzorjev v mobilnem telefonu.

Ko tukaj govorim o zajemu podatkov, govorim o vožnji s kolesom po kolesarskih površinah, z istočasnim zajemom podatkov na telefonu. Za to je potrebna pritrditev telefona na kolo na način, ki bi omogočal čim bolj realne podatke (torej bi tresljaje čim bolj realno prenesel na ohišje telefona in posledično senzorje), saj me najbolj zanimajo prav ti podatki, torej podatki senzorja pospeška.

Za namestitev sem najprej pomislil na nosilec za telefon. V tem primeru bi telefon bil na nosilcu, ta bi pa bil pritrjen na kolo. Primer takega nosilca vidimo na sliki 10. Kot je razvidno iz slike, so takšni nosilci sicer iz trdnih, ne-upogljivih materialov, toda vseeno lahko sklepamo, da bi pri uporabi prišlo do nekolikšnega zvijanja materiala nosilca, kar bi lahko ublažilo tresljaje in senzorji ne bi zajemali realnih podatkov.

Vir: mimovrste=)

Slika 10: Primer nosilca za telefon za kolo

Naslednja možnost, na katero sem pomislil, je bila torba za telefon, primer katere je na sliki 11. Pri tem načinu pritrditve so problem upogljivi in mehki materiali, iz katerih je zgrajena torba, zato lahko sklepamo, da bi bil prenos tresljajev še manj natančen kot pri prejšnjem načinu.

Vir: Ebike Escape Shop

Slika 11: Primer torbe za telefon

Kot zadnja možnost pritrditve se je pojavila ideja uporabe lepilnega traku. To je torej pomenilo tesno pritrditev telefona na kolo z uporabo lepilnega traka. Ta možnost je sicer za enkratno uporabo, saj je ob vsakem zajemu potrebno telefon ponovno pritrditi, je pa telefon v tesnem stiku s kolesom, kar pomeni, da je prenos tresljajev še najbolj realen od vseh omenjenih možnosti. To možnost sem tudi izbral; izvedbo ob enem izmed zajemov, ki sem jih izvedel, vidimo na sliki 12.

Vir: lasten

Slika 12: Namestitev telefona na kolo

Kot je omenjeno zgoraj, je pri zajemu podatkov ključen prenos tresljajev na telefon. Zaradi tega je pomembno tudi kolo, katerega sem uporabljal pri snemanju in njegove karakteristike.

Na voljo sem imel 2 tipa kolesa, t. i. hibridno kolo, ki ga vidimo na sliki 14 in pa cestno kolo (slika 13). Ker ima hibridno kolo amortizerje, ki so namenjeni ublažitvi tresljajev, za zajem ni primerno. Izbral sem torej cestno kolo, prav tako sem pa pred zajemom vedno poskrbel, da je tlak v zračnicah enak (zato, da bi zmanjšal število zunanjih spremenljivk, ki bi lahko vplivale na zajete podatke). Izbral sem tlak 4.6 barov za zadnjo in 4.5 barov za sprednjo zračnico.

Vir: lasten

Slika 13: Cestno kolo

Vir: lasten

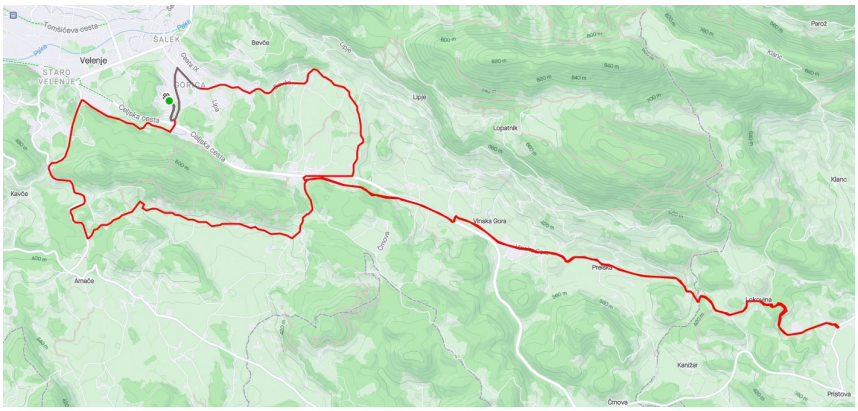
Slika 14: Hibridno kolo

Dejansko kolesarjenje, torej "zajemanje" sem izvedel v večih delih. Pot, po kateri sem kolesaril, sem si pred vsakim kolesarjenjem načrtoval tako, da sem se peljal po čim več odsekih, ki so bili vključeni v anketo.

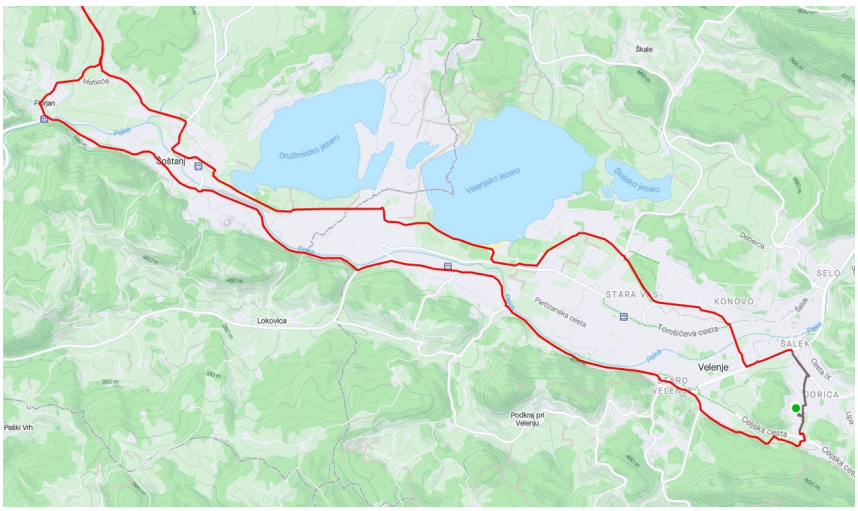
Spodaj so trije primeri poti (slika 15, slika 16, slika 17), po katerih sem zajemal podatke. S kombinacijo podatkov iz teh treh poti sem pridobil vsaj eno vrsto po-

datkov za vsak odsek, po katerem sem spraševal v anketi. Za nekatere odseke sem dobil tudi večkratne podatke, kar omogoči nadaljno analizo in primerjavo.

Slika 15: Prva pot zajemanja



Slika 16: Druga pot zajemanja



Slika 17: Tretja pot zajemanja

Na koncu sem vse skupaj izvedel 4 zajeme. S temi sem pridobil podatke na malo več kot 100 kilometrih kolesarskih površin in cest po Velenju in okolici. Iz tega sem pridobil za odseke Momax, Tomšičeva in Sončni park podatke za obe smeri vožnje, za odseke Jezero, Velenjka in Gorenje pa eno smer vožnje.

3.3 OBDELAVA PODATKOV

Ko so bili vsi podatki izvoženi iz aplikacije in prestavljeni na telefon, je bil čas za obdelavo. To sem se odločil narediti z uporabo programskega jezika Python, saj se mi je zdel za opravilo najprimernejši, prav tako pa ga dobro poznam. Bolj specifično sem ga napisal v obliki Jupyter Notebook, kar mi je omogočilo enostavnejšo testiranje in razvoj posameznih delov programa.

Cilj programa je obdelava surovih GPS in senzorskih podatkov. Rezultat programa naj bi bile ocene posameznih odsekov na lestvici od 1 do 5 in pa zemljevid, pobarvan z barvno lestvico od zelene (dobra kvaliteta) do rdeče barve (slaba kvaliteta).

3.3.1 NALAGANJE PODATKOV IN KONFIGURACIJA

Zajete podatke sem shranil po direktorijih, kjer je vsak zajem (torej posamezno "snemanje" oziroma kolesarjenje) svoj direktorij. Nato sem spisal kodo, ki je te podatke naložila v spomin za nadaljno obdelavo.

V prvem delu programa se nahaja konfiguracija, preko katere lahko spreminjamo obnašanje programa. Del konfiguracijske kode je prikazan v kodi 9.

```
# Minimalna dolzina posameznega odseka
MIN_SEGMENT_LENGTH_METERS = 25

# Poti do direktorijev s podatki
DATA_PATHS = [
'data/real_data/' + str(ride_num) + '_bike' for ride_num in range(1, 4)
]

COLOR_MAP = "jet" # paleta za prikaz podatkov na zemljevidu

# Razred, ki naj bo uporabljen za branje podatkov,
# saj ima vsak telefon drugacna imena senzorjev
from includes import HuaweiSensors
UseSensors = HuaweiSensors()

# Pot do slike, kamor se shrani koncan zemljevid
OUT_FILE = 'out_images/_temp.png'

# ... ostala konfiguracija
```

Koda 9: Konfiguracija programa

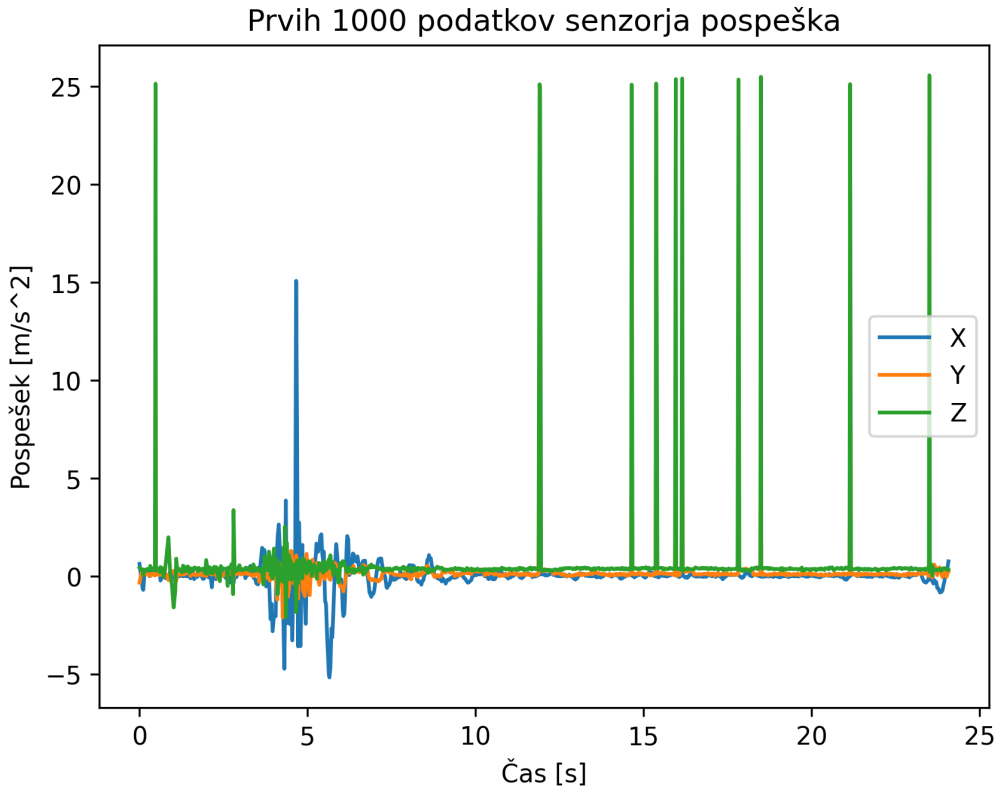
V drugem delu programa, so prebrani podatki iz zgoraj nastavljenih datotek. To branje je prikazano v kodi 10. Razred *DataLoader* je zadolžen za dejansko branje JSON datotek in njihovo pretvorbo v obliko, ki nam je v nadaljnji kodi bolj uporabna. Celotna koda razreda *DataLoader* je vidna v prilogi A.

```python
# Read data
all_data = []
for data_path in DATA_PATHS:
    # GPSDataLoader prebere podatke in
    # jih pretvori v bolj uporabno obliko
    gps_loader = GPSDataLoader(data_path)

    sensors_data = {
        sensor.get_type(): DataLoader(
            data_path,
            sensor.get_name()
        ).get_data() for sensor in UseSensors.get_sensors()
    }

    gps_data = gps_loader.get_data()

    all_data.append({
        'gps_data': gps_data,
        'sensors_data': sensors_data
    }) # Zdruzimo senzorske in GPS podatke
```

Koda 10: Nalaganje podatkov

3.3.2 DELJENJE NA ODSEKE

V tej točki imamo podatke pripravljene za dejansko analizo. Kot primer teh podatkov je na grafu 3 prikazanih prvih 1000 podatkov senzorja pospeška.



Graf 3: Prvih 1000 podatkov senzorja pospeška

Zaenkrat so senzorski podatki in GPS podatki še ločeni v smislu, da ne vemo, kje je bil zajet posamezni senzorski podatek. Za nadaljnjo obdelavo je potrebno združiti senzorske in lokacijske podatke, zato ustvarimo skupke podatkov, ki sem jih poimenoval "odseki poti". Ti odseki torej združujejo več GPS podatkov z ustreznimi senzorskimi meritvami.

Deljenje na odseke deluje tako, da dodajamo senzorske podatke v trenuten odsek, dokler ne pridemo do senzorskega podatka, ki je bil zajet časovno za trenutnim GPS podatkom. Za tem se prestavimo na naslednji GPS podatek in postopek ponovimo. Na ta način združimo GPS in senzorske podatke v en logičen "paket". Koda 11 prikazuje točno to delovanje.

```python
def split_into_segments(
    gps_data: np.ndarray,
    sensors_data: dict,
    min_len_meters:float = 10) -> list[PathSegment]:
    # sensor_names usebuje zadnje vrednosti in indekse za vsak senzor
    sensor_names = list(sensors_data.keys())
    last_indexes = {
        sensor_name: 0 for sensor_name in sensor_names
    }
    output = []; out_projected_accelleration_data = []
    current_segment = PathSegment()
    for i in trange(len(gps_data)):
        # Minimalna razdalja dosezena -> zacnimo nov odsek
        if current_segment.get_segment_distance() >= min_len_meters:
            output.append(current_segment)
            current_segment = PathSegment()
            # dodaj zadno GPS tocko novemu odseku, da sta povezana
            current_segment.add_gps_point(
                GPSDataPoint(gps_data[i - 1])
            )

        # Dodaj trenutne GPS podatke trenutnemu odseku
        current_gps_data = GPSDataPoint(gps_data[i])
        current_segment.add_gps_point(current_gps_data)

        # Dodaj senzorske podatke do naslednje GPS točke
        for sensor_name in sensor_names:
            for index in range(
                last_indexes[sensor_name],
                len(sensors_data[sensor_name])
            ):
                sensor_item = sensors_data[sensor_name][index]
                sensor_timestamp = sensor_item[0]
                if sensor_timestamp < current_gps_data.get_timestamp():
                    current_segment.add_sensor_data(
                        sensor_name,
                        sensor_item
                    )
                else:
                    break

                # Tukaj se zgodi še projekcija na Z os,
                # ki je bila za namene jasnosti izpuščena

            # Posodobimo indeks
            last_indexes[sensor_name] = index
    # Dodaj zadnji odsek
    output.append(current_segment)
    return output, out_projected_accelleration_data
```

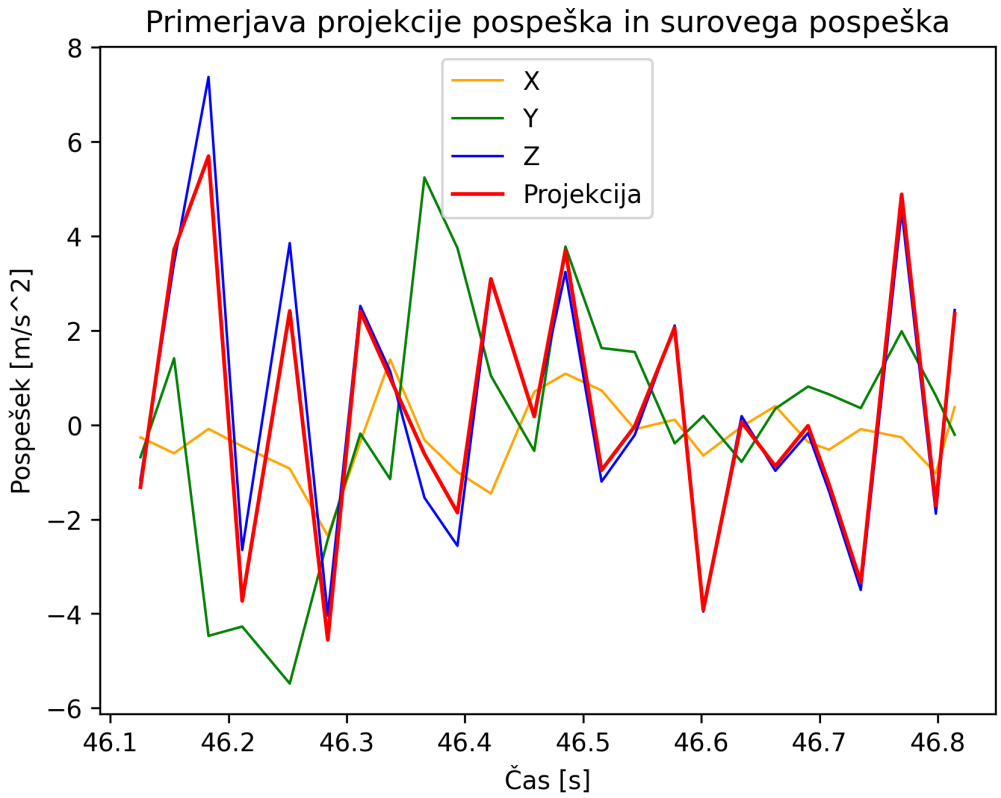
Koda 11: Deljenje na odseke

V zgornji kodi se zgodi še ena pomembna stvar: projekcija vektorjev pospeška na Z os. S to projekcijo dobimo navpično (pravokotno na tla) komponento vseh podatkov senzorja pospeška. S tem izluščimo le raziskavi relevantne podatke, saj nas pospeški levo, desno, naprej in nazaj (glede na smer vožnje) ne zanimajo. Izvleček kode, ki dela to, je prikazan v kodi 12.

```
# ...

# Funkcija, ki rotira vektor 'vector'
# za rotacijski vektor 'rotation_vector'.
def apply_rotation(
        rotation_vector: np.ndarray,
        vector: np.ndarray
    ) -> np.ndarray:
    rotation_matrix = st.Rotation.from_quat(rotation_vector).as_matrix()
    return np.dot(rotation_matrix, vector)

# Z os
Z_AXIS = np.array([0, 0, 1])
# Z os, rotirana v lokalni koordinatni sistem
rotated_z_axis = apply_rotation(rotation_vector[:4], Z_AXIS)

# Projekcija vektorja na vektor
def project_vec_on_vec(v1: np.ndarray, v2: np.ndarray) -> np.ndarray:
    return np.dot(v1, v2) / np.linalg.norm(v2)

# Projeciramo vektor pospeška na Z os
projected_acceleration = project_vec_on_vec(
                            acceleration_vector, rotated_z_axis
                        )

# ...
```

Koda 12: Projekcija vektorjev pospeška na Z os

Ker je lokalna os Z telefona na kolesu bila pod majhnim kotom glede na globalno os Z (tisto, ki kaže naravnost proti tlom), se projecirane vrednosti skoraj ujemajo z Z komponento vektorja, toda ne popolnoma. To vidimo v grafu 4.



Graf 4: Primerjava projeciranih vrednosti in originalnih vektorjev pospeška

3.3.3 OCENJEVANJE ODSEKOV

Korak ocenjevanja odsekov je namenjen pretvorbi mnogih senzorskih podatkov v eno število, ki pomeni oceno kvalitete tega odseka. Za to nalogo obstaja več načinov, zato sem ustvaril poenoten vmesnik (funkcijo *process* v razredu *PathSegment*), ki ima dostop do odseka in je njena naloga, da vrne eno decimalno vrednost – oceno odseka.

To funkcijo, oziroma njeno delovanje, v nadaljevanju raziskovalne naloge poimenujem cenilka (angl. estimator). Kasneje je predstavljenih več implementacij te funkcije, torej več cenilk.

Primer enostavne implementacije te funkcije, ki preprosto vrne povprečno hitrost na posameznem odseku, je viden v kodi 13. Ta v praksi ni uporaben, prikaže pa delovanje funkcije.

```
def process(self) -> float:
        # Hitrost
        dist = self.get_segment_distance()
        length = self.get_segment_time_length()
        speed = 0 if np.isnan(dist / length) else dist / length

        return speed
```

Koda 13: Ocenjevanje odseka

Za ocenjevanje kvalitete odseka sem izdelal 5 cenilk, katerim sem dodelil imena.

Preproste cenilke

Najprej sem kot enostaven primer izdelal cenilko, ki za oceno vrne povprečno vrednost projeciranih pospeškovnih podatkov. To cenilko sem v kodi in nadaljevanju raziskovalne naloge imenoval *mean*.

Drugo cenilko, ki sem jo uporabil, sem poimenoval *std*. Kot oceno odseka vrne standardni odklon vseh projeciranih vrednosti pospeškovnega senzorja za posamezni odsek.

Tretja cenilka je kombinacija prvih dveh, saj je polovica ocene sestavljena iz cenilke *mean*, polovica pa iz cenilke *std*. Poimenoval sem jo *meanstd*.

Naslednja cenilka, poimenovana *minmaxdiff*, za oceno odseka vrne razliko med največjo in najmanjšo vrednostjo pospeškovnega senzorja.

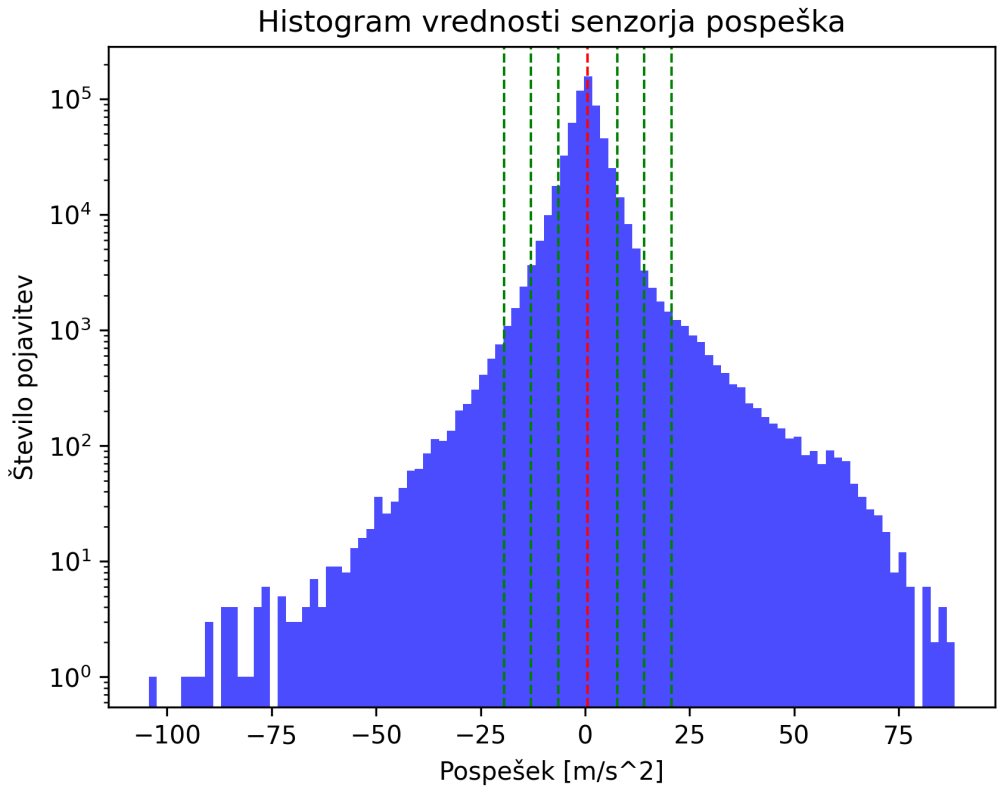
Zgoraj navedene 4 cenilke so v svojem delovanju preproste. S tem sem želel preveriti, če se je mogoče na zelo enostaven način, torej z enostavno statistično funkcijo, približati načinu, s katerim ljudje ocenjujejo odseke.

Cenilka bumpcount

Zadnja in tudi najbolj kompleksna cenilka, ki sem jo poimenoval *bumpcount*, deluje na principu štetja nepopolnosti (lukenj, robnikov) v posameznem odseku.

Ta cenilka ima dodaten parameter, ki določa, kaj je najmanjši pospešek, ki se naj šteje kot nepopolnost. Ta parameter sem poimenoval T.

Za izbiro tega parametra sem najprej izrisal histogram vseh vrednosti senzorja pospeška (graf 5). Graf ima na osi y logaritemsko skalo. Navpična rdeča črta predstavlja povprečno vrednost, zelene črte pa označujejo en, dva ali tri standardne odklone levo in desno od povprečja.



Graf 5: Histogram vrednosti senzorja pospeška

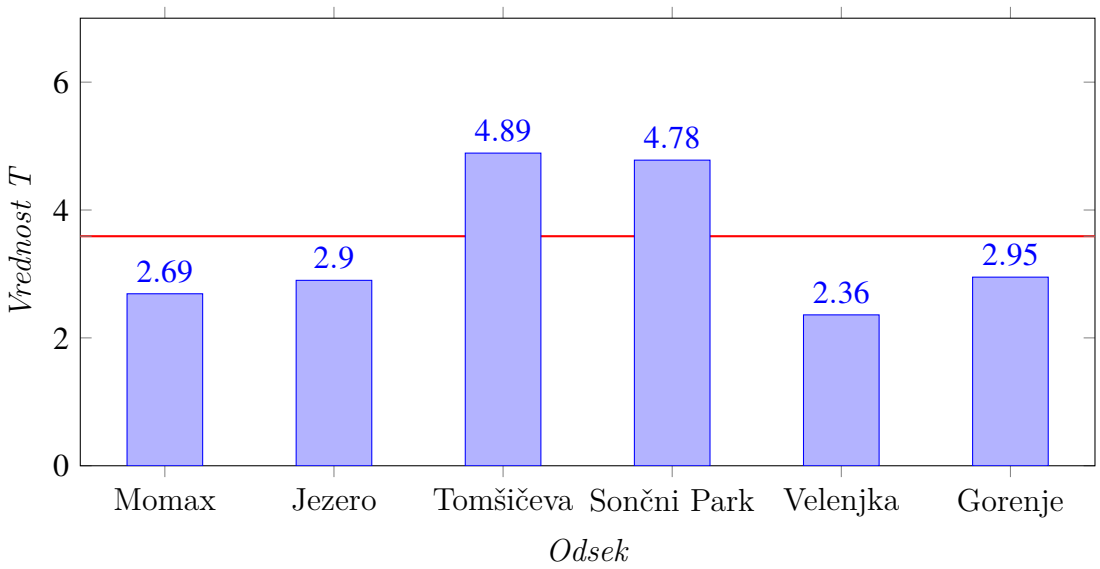
Kot T vzel vrednost, ki je bila za en standardni odklon desno. Ta vrednost je bila (zaokrožena na 1 decimalno mesto) 7.1, kar se je kasneje izkazalo za relativno slabo vrednost, saj so se ocene, dobljene z njo, zelo razlikovale od ocen iz ankete.

Ker sem želel izbrati čim boljši možen T, torej takega, ki se bi najbolj približal rezultatom ankete, sem uporabil funkcijo *minimize_scalar* Python knjižnice scipy, ki pomaga pri minimizaciji napake. V tem primeru sem spisal program, ki je našel za vsak odsek vrednost T, s katero se je ocena cenilke čim bolj približala oceni ankete za ta segment. To prikazuje koda 14.

```python
from scipy.optimize import minimize_scalar

best_Ts = {}
def get_abs_error(T: float, segment_name: str):
    segment_values = np.array(
        [segment.process("bumpcount", T) for segment in segments]
        )

    # Normalizacija vrednosti
    normalized_values = # ...
    normalized_segments = list(zip(segments, normalized_values))

    # Uporabi le vrednosti za ta segment
    this_segments = normalized_segments[
        SEGMENT_INDEXES[segment_name][0]:SEGMENT_INDEXES[segment_name][1]
    ]

    # Pretvorba v 1-5
    average_value = np.mean([segment[1] for segment in this_segments])
    rating = 1 + ((1-average_value) * 4)

    # Absolutna razlika
    error = np.abs(rating - SEGMENT_SURVEY_RESULTS_OBJ[segment_name])
    return error

# Poišči najboljši T za vsak segment
for segment_name in SEGMENT_INDEXES:
    print("Iščem T za segment:", segment_name)
        res = minimize_scalar(
        get_abs_error,
        args=(segment_name),
        bounds=(0, 10),
        method='bounded'
    )
    best_Ts[segment_name] = res.x

print("Najbolše vrednosti T:", best_Ts)
```

Koda 14: Iskanje vrednosti T za algoritem bumpcount

Zgornja koda je vrnila vrednosti, vidne na grafu 6. Na njemu je prikazana tudi povprečna vrednost T (3.59), označena z rdečo črto.



Graf 6: Optimalne vrednosti T za cenilko bumpcount

Vidimo, da je večina optimalnih vrednosti T med 2 in 3, z izjemo odsekov Tomšičeva in Sončni park. Možno je, da je korelacija bolj kompleksna, torej da je odvisna od standardnega odklona. Zato izrazimo vse vrednosti T še kot delež vrednosti standardnega odklona vrednosti senzorja pospeška za vsak odsek. Dobimo vrednosti v tabeli 1, kjer stolpec T prikazuje optimalno vrednost T, stolpec std prikazuje standardni odklon na tem odseku, stolpec T [std] pa prikazuje vrednost T, izraženo kot delež standardnega odklona. Vse vrednosti so za lažjo razumljivost zaokrožene na eno decimalno mesto.

Tabela 1: Vrednosti T izražene kot delež standardnega odklona

Segment	T	std	T [std]
Momax	2.7	4.6	12.3
Jezero	2.7	4.9	12.9
Tomšičeva	4.7	9.0	42.0
Sončni Park	2.0	9.3	18.3
Velenjka	2.3	4.3	10.0
Gorenje	3.0	5.4	16.0

Vidimo, da je zdaj še vrednost odseka Sončni Park bolj podobna drugim, kljub temu pa odsek Tomšičeva še vedno izstopa. Vseeno to pokaže, da bi bila optimalna vrednost nekje v okolici petnajstkratnika standardnega odklona posameznega odseka. Da dobimo dejansko najboljšo vrednost, lahko enako optimizacijo izvedemo na vseh odsekih na enkrat. To sem tudi naredil in dobil vrednost $T = 2.9$. To vrednost T sem za to cenilko uporabljal povsod v nadaljevanju.

Normalizacija dobljenih ocen

Ocenjevanje po posamezni cenilki se izvaja na vseh podatkih naenkrat. Ko imamo vrednosti ocen za vse odseke, jih normaliziramo na interval med 0 in 1. Za tem odstranimo vse vrednosti, ki so v zgornjem 1 % in vse vrednosti, ki so v spodnjem 1 %, saj se na ta način znebimo anomalij v meritvah, ki so nerealne. Vse te vrednosti nato prestavimo na skalo od 1 do 5 in tako dobimo končne vrednosti.

Iz zgoraj opisanega postopka je torej razvidno, da so ocene relativne na vse ostale podatke. To pomeni, da so odvisne od vseh podatkov. To bi lahko bil problem, če bi bili podatki večinoma enega tipa (npr. slaba površina), toda podatki, ki sem jih uporabljal tukaj, so mešani (imajo veliko vseh tipov površin).

3.3.4 IZVOZ PODATKOV

Zadnja naloga, ki jo ima ta Python program je izvoz končnih podatkov (torej ocen in GPS podatkov) na način, da so ti uporabni za nadaljni prikaz (na spletni strani). To sem se odločil narediti tako, da ustvarim iz "odsekov poti" linije na zemljevidu. Vsaka od teh linij dobi določeno barvno vrednost, glede na normalizirano oceno tega odseka. Ta potek, torej najprej normalizacijo in nato ustvarjanje teh linij, vidimo v kodi 15.

```
map_data = []

color_map = mpl.colormaps.get_cmap(COLOR_MAP)

for segment, normalized_value in zip(segments, normalized_values):

    # Kvadratni koren, saj so tako lepše vidne vse vrednosti
    real_value = np.sqrt(normalized_value)
    color = color_map(real_value)
    #color = color_map(normalized_value)

    map_data.append(
        {
            "points": [
                [point.get_lon(), point.get_lat()]
                for point in segment.gps_points
            ],
            "color": color,
            "value": real_value
        }
    )
```

Koda 15: Ustvarjanje linij za zemljevid

Po ustvarjanju seznama linij na zemljevidu, program naredi 2 stvari: linije izvozi v JSON obliko (koda 16) in pa nariše zemljevid ter ga prikaže (in sliko shrani v

datoteko).

```
import json
with open('temp/map_data.json', 'w') as f:
    json.dump(map_data, f)
```

Koda 16: JSON izvoz linij na zemljevidu

Naloga prikaza zemljevida je predana funkciji create_map, ki je prikazana v kodi 17. Uporablja knjižnico matplotlib ((v kodi *plt*)) v kombinaciji s contextily (v kodi *ctx*), kar omogoča uporabo zemljevida kot ozadnja v koordinatnemu sistemu; posledično je koda precej enostavna. Najprej ustvari LineString (iz knjižnice shapely), torej geometrijske koncepte, ki jih uporablja knjižnica geopandas (v kodi *gpd*), za tem nastavi nekaj vrednosti za pravilen izris zemljevida (postavitev), nazadnje pa vse linije nariše.

```
def create_map(data, dpi=300):

    plt.rcParams['figure.dpi'] = dpi
    plt.rcParams['savefig.dpi'] = dpi

    items = []
    colors = []

    for data_item in data:
        line = LineString(data_item['points'])
        items.append(line)
        colors.append(data_item['color'])

    # Ustvarjanje GDF in nastavljanje CRS
    gdf = gpd.GeoDataFrame(geometry=items)
    gdf.set_crs(epsg=4326, inplace=True)
    gdf = gdf.to_crs(epsg=3857)

    # Izris GDF
    fig, ax = plt.subplots(figsize=(10, 10))
    gpd.GeoSeries(gdf.geometry, crs=gdf.crs).plot(ax=ax, color=colors)

    # Ozadje (zemljevid)
    ctx.add_basemap(ax, source=ctx.providers.OpenStreetMap.Mapnik)

    # Odstranitev osi
    ax.set_axis_off()

    return fig, ax, gdf
```

Koda 17: Prikaz na zemljevidu v Python

3.4 PRIKAZ PODATKOV

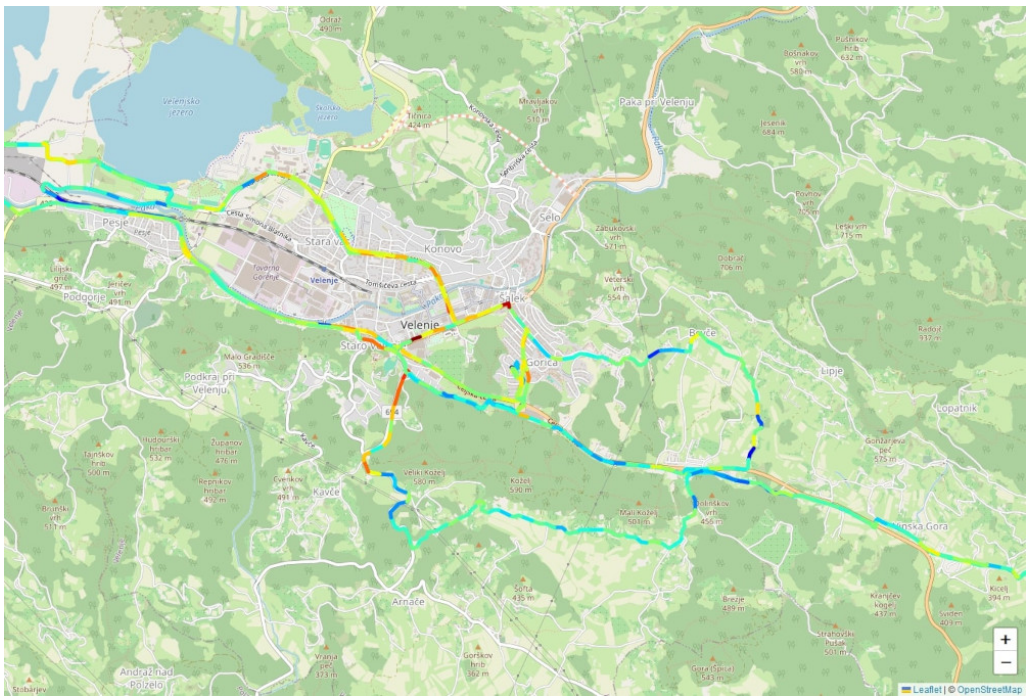
V prejšnjem poglavju je koda izrisala zemljevid s potjo, toda le kot statično sliko. Ker je bil cilj ustvariti interaktiven zemljevid, sem se odločil za izdelavo preproste spletne strani, ki bi uporabljala knjižnico Leaflet za prikaz zemljevida s podatki, izvoženimi v zadnjem koraku prejšnjega poglavja. [24].

Kodo, ki izvaja dejansko risanje, vidimo v kodi 18. Najprej s knjižnico Leaflet ustvari nov zemljevid, ga centrira na Velenje in pa doda ozadje (torej dejanski zemljevid). Za tem gre čez vse podatke in za vsako linijo ustvari linijo v knjižnici Leaflet in jo doda na zemljevid. Doda tudi možnost klika na linijo za prikaz njene vrednosti.

```
const data = [/*(SEM PRIDEJO IZVOŽENI PODATKI)*/];
# ... Tukaj ustvari zemljevid ...

for (let line of data) {
    const points = line.points;

    const lnclr = line.color;
    const color = `rgb(${lnclr[0]*255}, ${lnclr[1]*255}, ${lnclr[2]*255})`;

    const polyline = L.polyline(points, {
        color: color,
        weight: 5,
    }).addTo(map);

    const linePercent = line.value * 100;
    polyline.on('click', function (e) {
        L.popup()
            .setLatLng(e.latlng)
            .setContent('<p>' + linePercent.toFixed(2) + '%</p>')
            .openOn(map);
    });
}
```

Koda 18: Prikaz na zemljevidu v JavaScript

Rezultat te kode je interaktiven zemljevid, viden na sliki 18.



Slika 18: Interaktivni zemljevid

Ta način izrisa interaktivnega zemljevida sem uporabil za prikaz rezultatov na spletni strani, ki je dostopna na `http://kolesarske.anzeblag.us`. Izdelava te spletne strani je izven obsega te naloge.

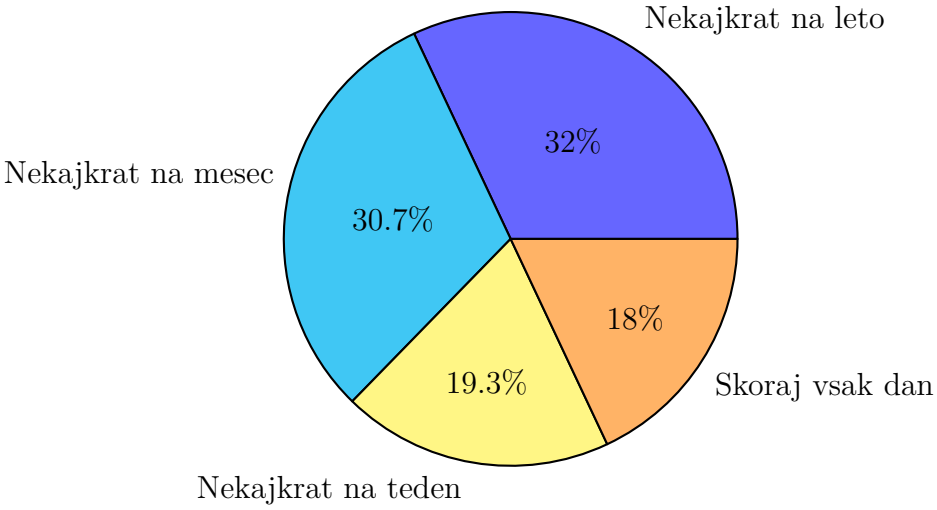
4 REZULTATI

Raziskovanje v tej nalogi je bilo sestavljeno iz 2 glavnih delov; ankete ter zbiranja podatkov.

4.1 REZULTATI ANKETE

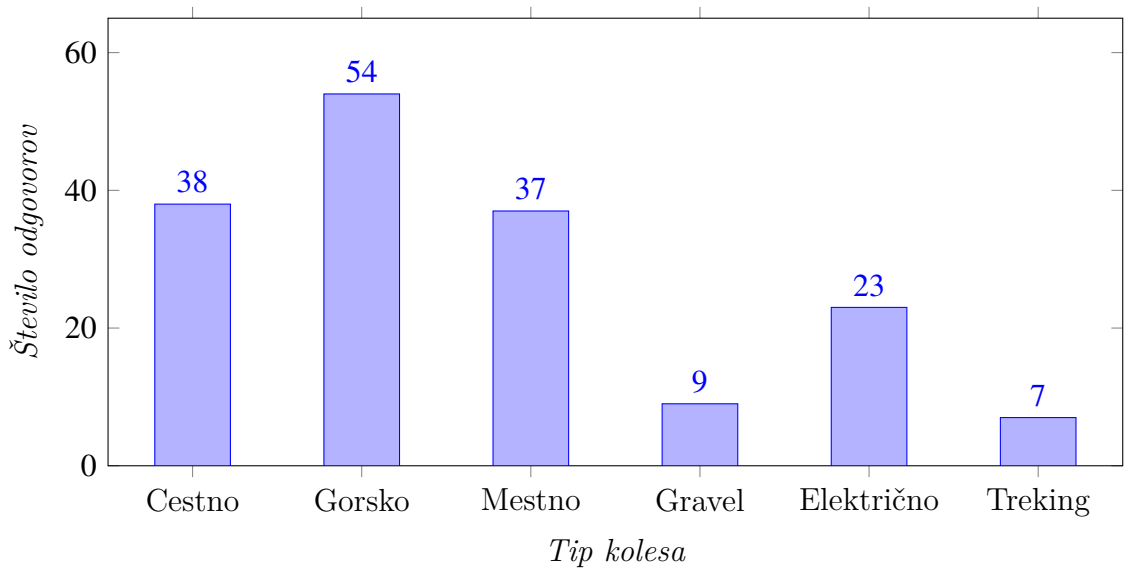
Vse skupaj je na anketo odgovorilo 150 anketirancev.

Prvo vprašanje se je nanašalo na pogostost kolesarjenja po kolesarskih površinah v Velenju. Iz grafa 7 je razvidno, da večina anketirancev kolesari nekajkrat na leto (32 %) ali nekajkrat na mesec (30.7 %). Približno petina jih kolesari nekajkrat na teden, 18 % pa skoraj vsak dan.



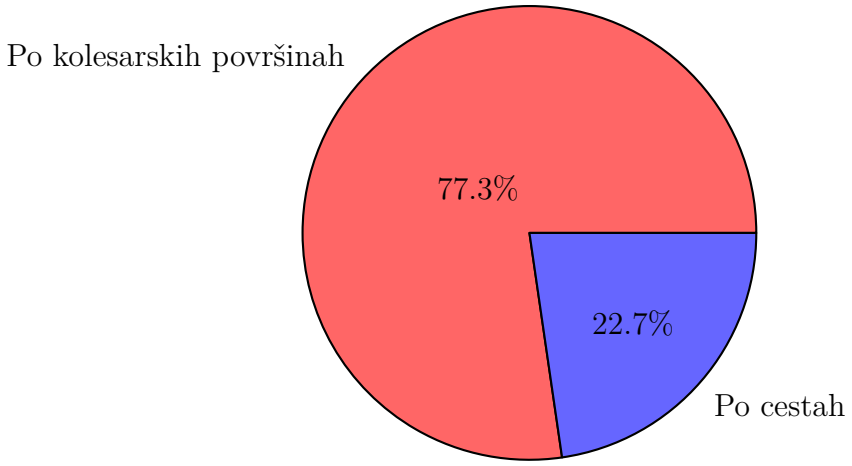
Graf 7: Pogostost kolesarjenja po Velenju med anketiranci

Iz grafa 8 je razvidno, da večina anketirancev za kolesarjenje uporablja gorsko kolo, takoj za tem sta po pogostosti cestno in mestno kolo. Ostali tipi koles so bolj redki. Nekaj anketirancev je izbralo odgovor, da se ne spoznajo na tipe kolesa. Njihovi odgovori v grafu niso prikazani.



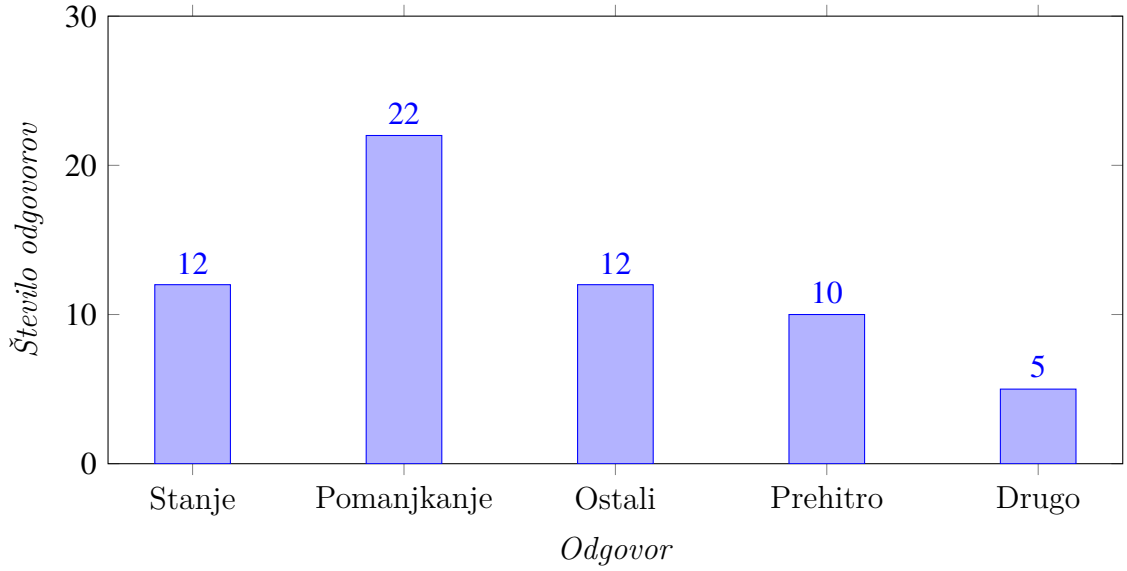
Graf 8: Uporabljeni tipi koles pri anketirancih

V grafu 9 vidimo razmerje med anketiranci, ki bolj pogosto kolesarijo po cestah (22.7 %) in tistimi, ki bolj pogosto kolesarijo po kolesarskih površinah (77.3 %).



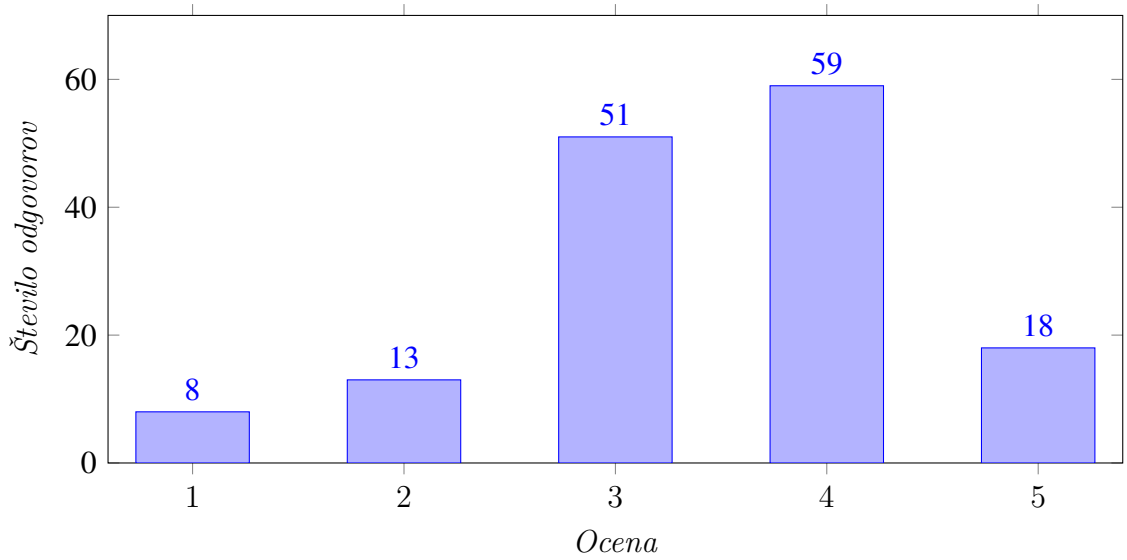
Graf 9: Primerjava pogostosti kolesarjenja po cestah in kolesarskih površinah

Največ od teh anketirancev, ki pretežno kolesarijo po cestah, se tako odloči zaradi pomanjkanja kolesarskih površin. To vidimo v grafu 10, ki prikazuje odgovore na vprašanje "Kaj je razlog, da pogosteje kolesarite po cestah?" (to je bilo postavljeno tistim, ki so izbrali, da pogosteje kolesarijo po cestah). V tem grafu, so razlogi zaradi jasnosti skrajšani v eno besedo. "Stanje" označuje odgovor "Slabo stanje kolesarskih površin", "Pomanjkanje" označuje odgovor "Pomanjkanje kolesarskih površin", "Ostali" označuje odgovor "Ostali udeleženci na kolesarskih površinah so moteči", "Prehitro" označuje odgovor "Vozim prehitro za kolesarske površine".



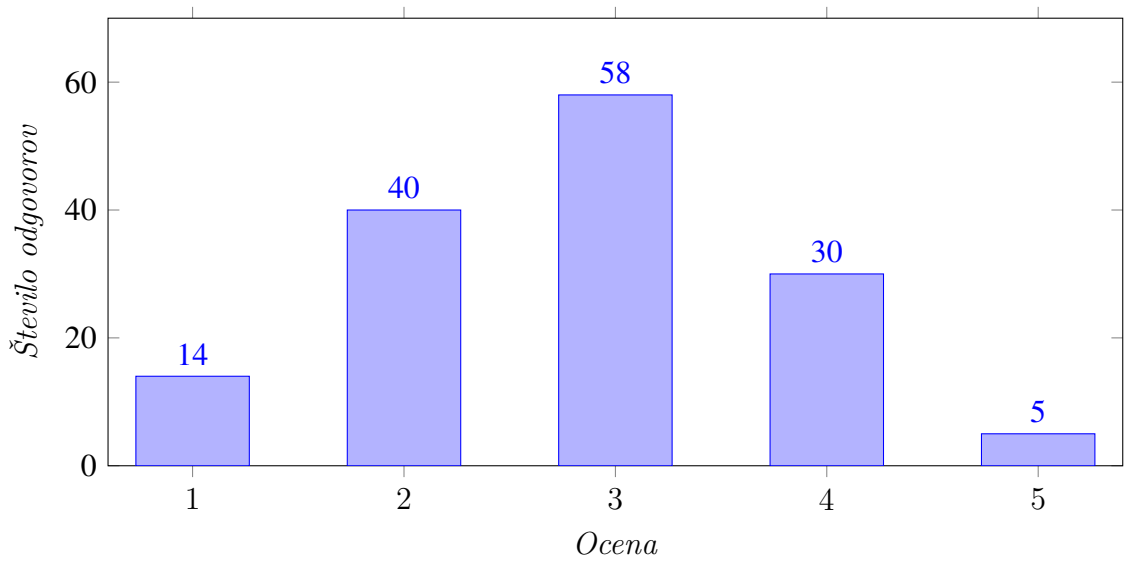
Graf 10: Razlogi za pretežno kolesarjenje po cestah

Večina anketirancev bi kolesarske površine v Velenju in okolici ocenila od 1 do 5 z oceno 4, povprečna ocena pa je približno 3.4. Vse ocene vidimo v grafu 11. Ta graf prikazuje odgovore na vprašanje "Ocenite kvaliteto in udobnost vožnje po kolesarskih površinah v Velenju in okolici. (Če se po njih vozite)".



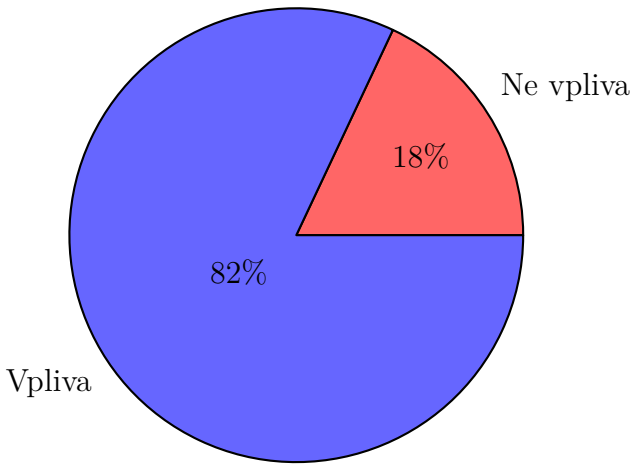
Graf 11: Udobnost vožje po kolesarskih površinah

Najpogostejša ocena kvalitete in udobnosti vožnje po cestah večine anketirancev je manjša kot ocena kolesarskih površin, in sicer 3. Povprečna ocena cest je bila približno 2.8. Vse ocene vidimo v grafu 12. Ta graf prikazuje odgovore na vprašanje "Ocenite kvaliteto in udobnost vožnje po cestah v Velenju in okolici. (Če se po njih vozite)".



Graf 12: Udobnost vožje po cestah

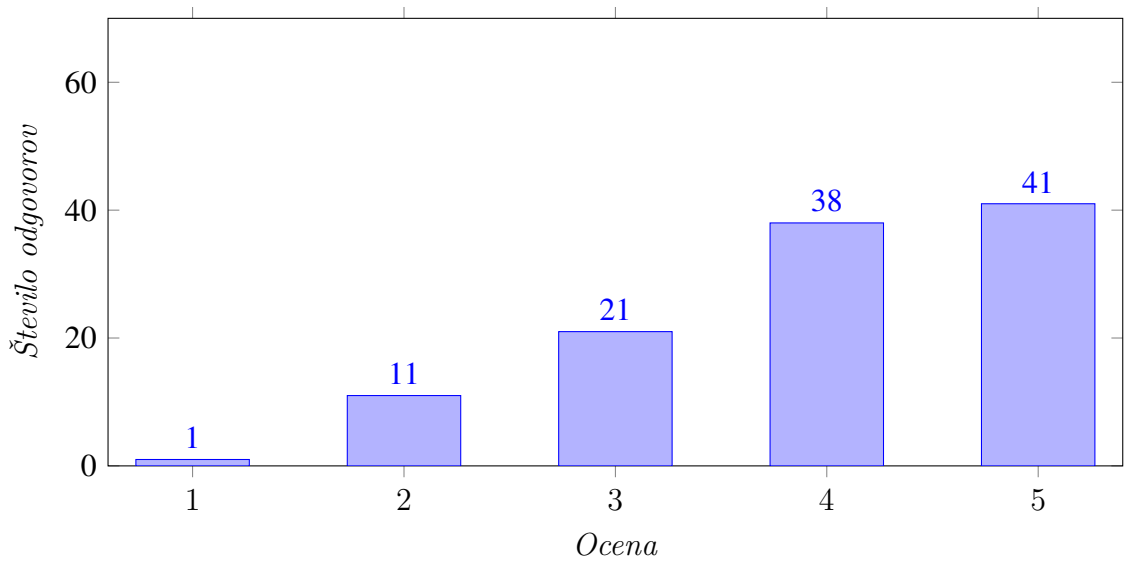
Zadnje vprašanje splošnega dela se nanaša na to, ali anketiranci mislijo, da kdaj kvaliteta površine vpliva na izbiro poti, po kateri se peljejo s kolesom. Iz grafa 13 je razvidno, da večina (82 %) anketirancev meni, da kvaliteta površine **vpliva** na njihovo izbiro poti.



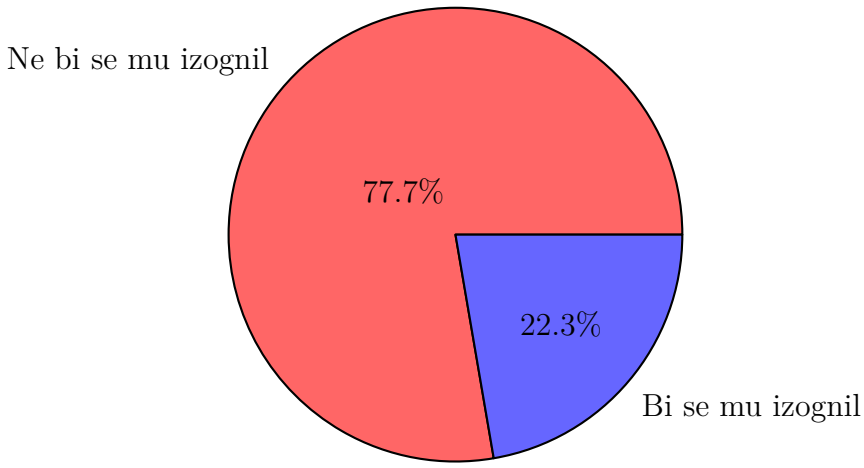
Graf 13: Vpliv kvalitete površine na izbiro poti pri anketirancih

4.1.1 ODSEK MOMAX

Prvi odsek, po katerem je spraševala anketa, je bil odsek Momax. Na grafu 14 je večina od tistih anketirancev, ki odsek poznajo, tega ocenila z oceno 5. Povprečna ocena je bila 3.9. Graf 15 kaže, da se 77.7 % anketirancev temu odseku ne bi izognilo.



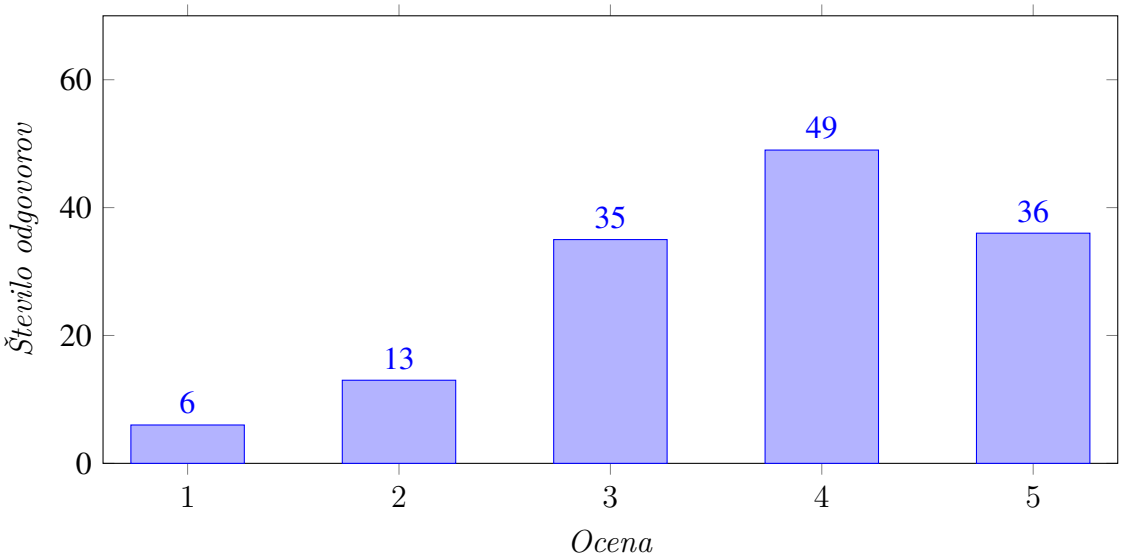
Graf 14: Mnenje o kvaliteti odseka Momax



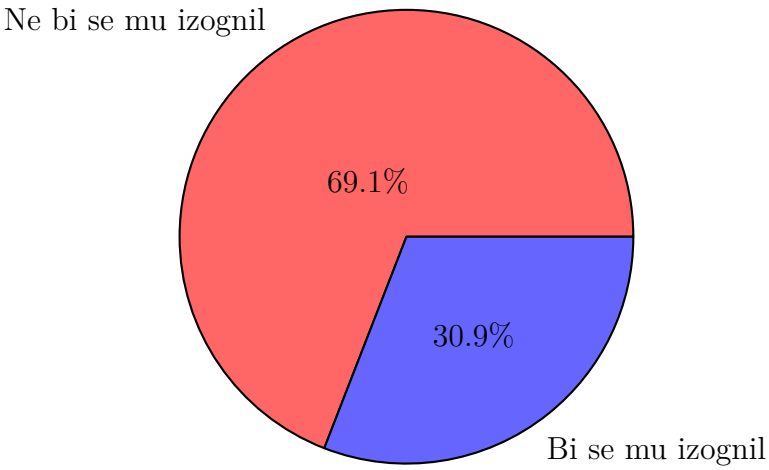
Graf 15: Izognitev odseku Momax pri anketirancih

4.1.2 ODSEK JEZERO

Pri odseku Jezero (graf 16 in graf 17) je najpogostejša ocena 4, povprečna ocena 3.7, 30.9 % anketirancev pa bi se odseku izognilo.



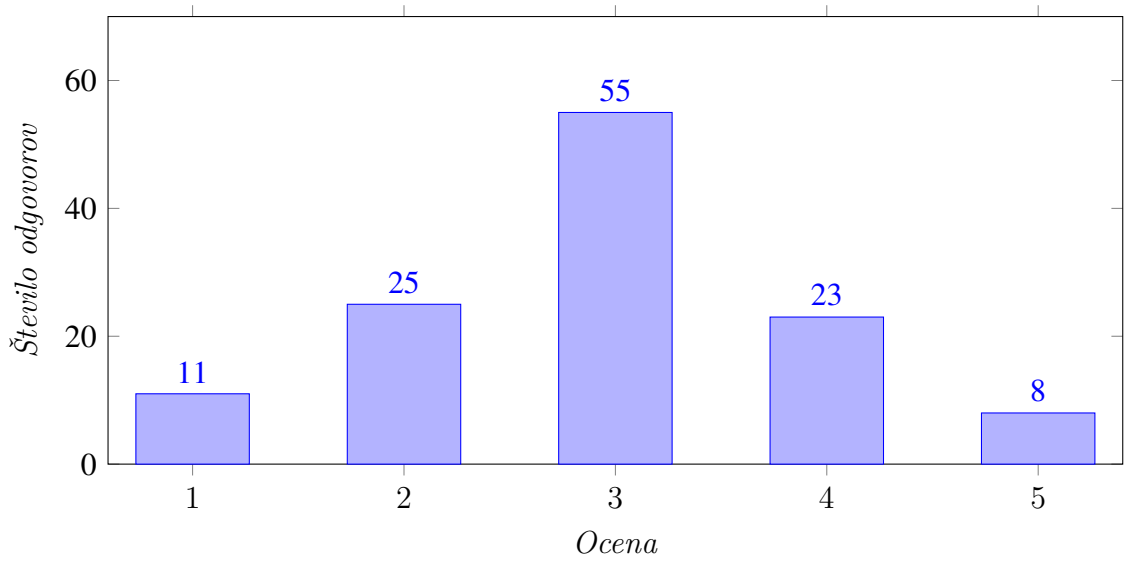
Graf 16: Mnenje o kvaliteti odseka Jezero



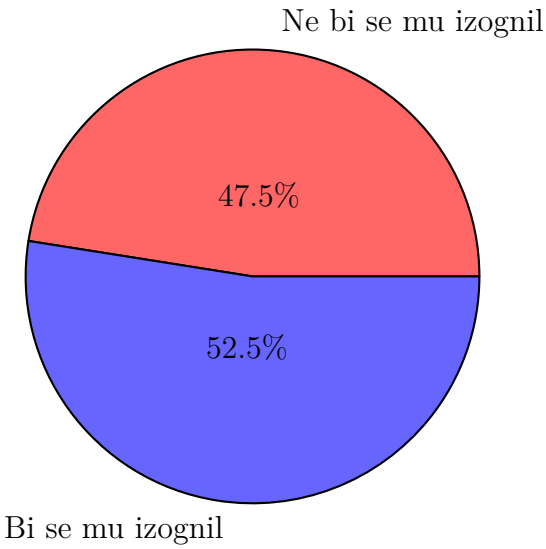
Graf 17: Izognitev odseku Jezero

4.1.3 ODSEK TOMŠIČEVA

Odsek Tomšičeva (graf 18 in graf 19) je največ anketirancev ocenilo z oceno 3, povprečna ocena je bila prav tako 3.0, 52.5 % anketirancev pa bi se odseku izognilo.



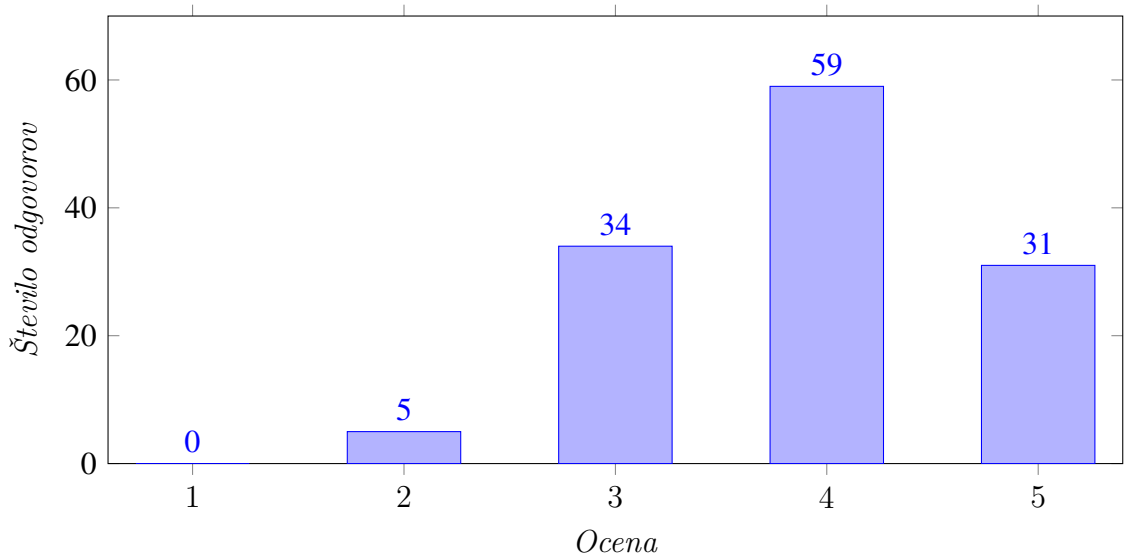
Graf 18: Mnenje o kvaliteti odseka Tomšičeva



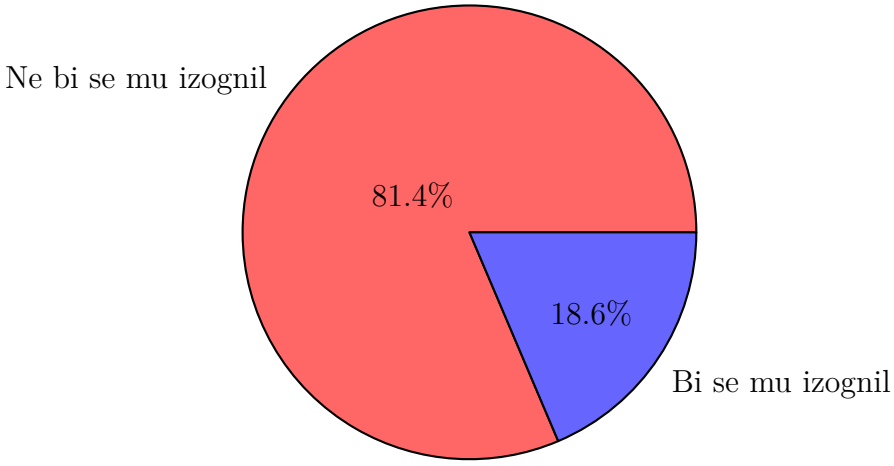
Graf 19: Izognitev odseku Tomšičeva

4.1.4 ODSEK SONČNI PARK

Odseku Sončni park bi se izognilo 18.6 % anketirancev (graf 21), največ jih je ocenilo ta odsek z oceno 4 (graf 20), povprečna ocena pa je bila 3.9.



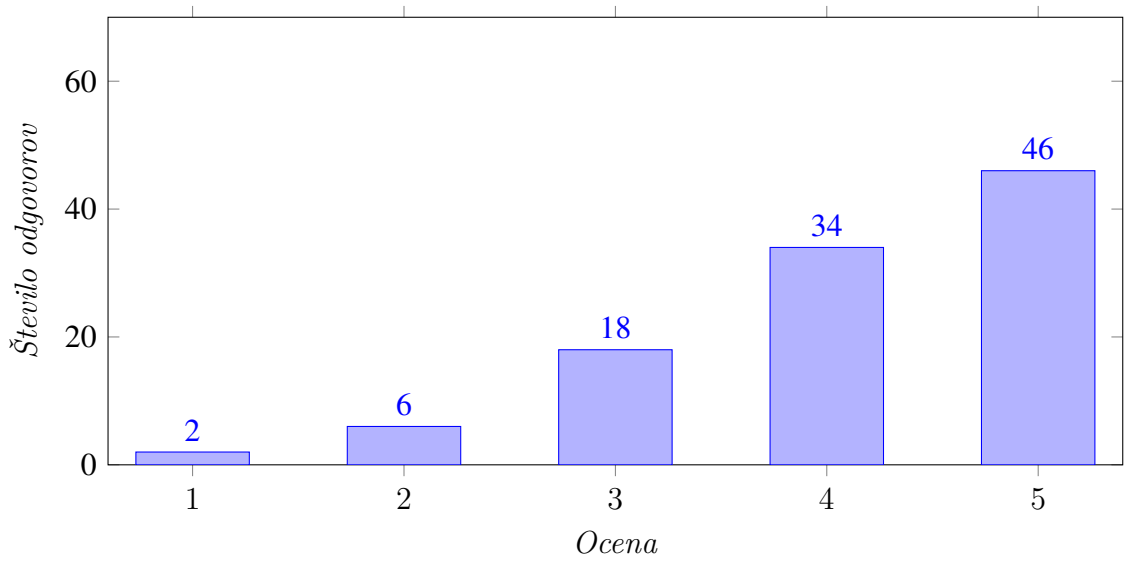
Graf 20: Mnenje o kvaliteti odseka Sončni park



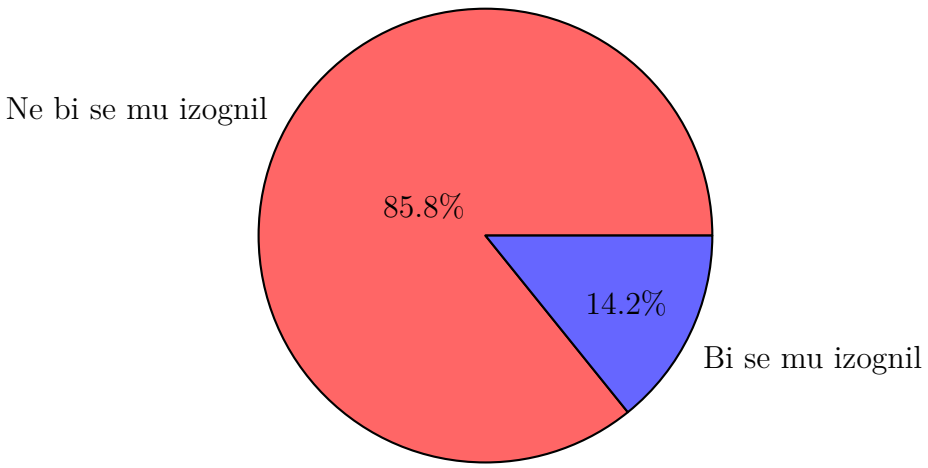
Graf 21: Izognitev odseku Sončni park

4.1.5 ODSEK VELENJKA

Odsek Velenjka je bil od vseh odsekov najbolje ocenjen. Imel je najpogostejšo oceno 5 (graf 22), povprečno oceno 4.1 in izognilo se bi mu le 14.2 % anketirancev (graf 23).



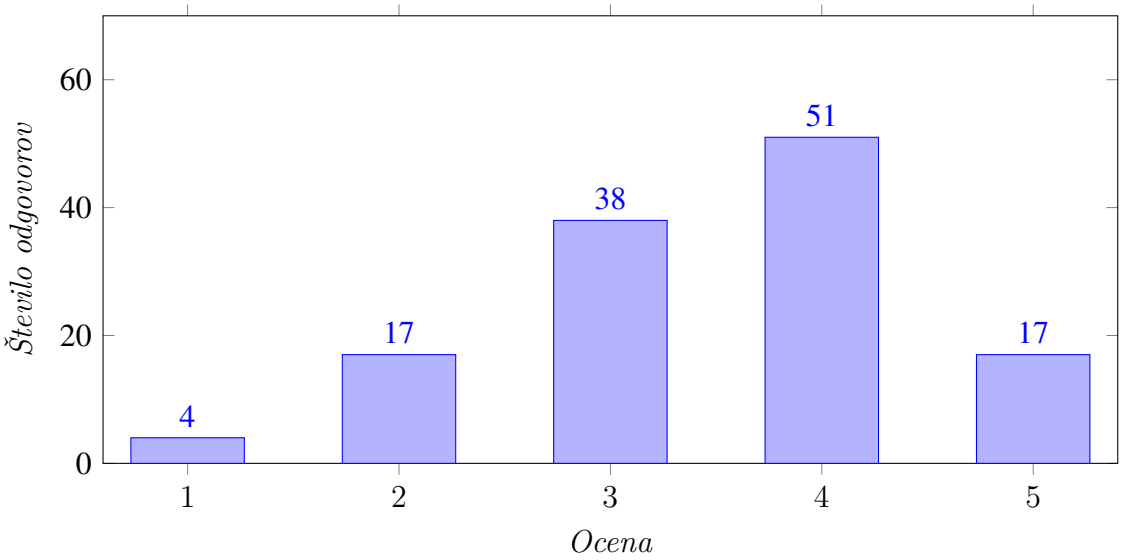
Graf 22: Mnenje o kvaliteti odseka Velenjka



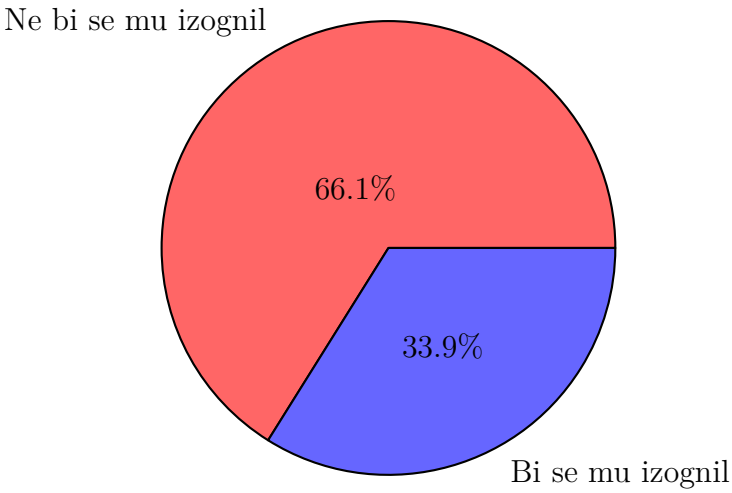
Graf 23: Izognitev odseku Velenjka

4.1.6 ODSEK GORENJE

Zadnji odsek, odsek Gorenje je dobil najpogostejšo oceno 4 (graf 24), povprečno oceno 3.4 in izognilo bi se mu 33.9 % anketirancev.



Graf 24: Mnenje o kvaliteti odseka Gorenje



Graf 25: Izognitev odseku Gorenje

4.2 REZULTATI MERITEV

Po obdelavi podatkov in analizi meritev z večimi cenilkami sem dobil več različnih rezultatov (ocen) za vsak odsek. V nadaljevanju so ti predstavljeni. Vse številke v tabelah v tem poglavju so zaokrožene na 2 decimalki.

4.2.1 CENILKA MEAN

Pri ocenah cenilke mean (tabela 2) opazimo, da so vse vrednosti med 3 in 3.4, torej ni bistvene raznolikosti.

Tabela 2: Ocene cenilke mean

Odsek	**Ocena kvalitete**
Momax	3.29
Jezero	3.22
Tomšičeva	3.08
Sončni Park	3.01
Velenjka	3.38
Gorenje	3.34

4.2.2 CENILKA STD

Po cenilki std, ocene katere so prikazane v tabeli 3, je bil najslabše ocenjen odsek Tomšičeva z oceno 3.14, najbolje pa odsek Velenjka z oceno 4.46.

Tabela 3: Ocene cenilke std

Odsek	**Ocena kvalitete**
Momax	4.36
Jezero	4.26
Tomšičeva	3.14
Sončni Park	3.36
Velenjka	4.46
Gorenje	4.20

4.2.3 CENILKA MEANSTD

Naslednja preprosta cenilka, meanstd, ki je kombinacija cenilk std in mean, se zaradi svoje narave ne razlikuje bistveno od cenilke mean. Prav tako je najslabše ocenila segment Tomšičeva, najbolje pa segment Velenjka. Ocene so vidne v tabeli 4.

Tabela 4: Ocene cenilke meanstd

Odsek	**Ocena kvalitete**
Momax	4.30
Jezero	4.18
Tomšičeva	3.06
Sončni Park	3.26
Velenjka	4.41
Gorenje	4.14

4.2.4 CENILKA MINMAXDIFF

Ta cenilka je prav tako kot ostale najslabše ocenila odsek Tomšičeva z oceno 3.11, najbolje pa odsek Velenjka z oceno 4.15. Vse ocene so vidne v tabeli 5.

Tabela 5: Ocene cenilke minmaxdiff

Odsek	**Ocena kvalitete**
Momax	4.10
Jezero	4.13
Tomšičeva	3.11
Sončni Park	3.47
Velenjka	4.15
Gorenje	4.07

4.2.5 CENILKA BUMPCOUNT

Zadnja cenilka je enako kot ostale najbolje ocenila odsek Velenjka, najslabše pa odsek Tomšičeva. Ocene so vidne v tabeli 6. Vredno je omeniti, da sem za te ocene uporabljal vrednost $T = 2.9$. Način pridobitve te vrednosti je razložen v poglavju 3.3.3.

Tabela 6: Ocene cenilke bumpcount

Odsek	**Ocena kvalitete**
Momax	4.12
Jezero	4.04
Tomšičeva	3.30
Sončni Park	3.53
Velenjka	4.40
Gorenje	4.02

5 DISKUSIJA

To poglavje je razdeljeno na tri logične dele, ki se nanšajo na tri hipoteze.

5.1 MERJENJE

Prvo hipotezo, torej *"Mogoče je izmeriti kvaliteto odseka kolesarske površine brez specializiranih naprav."* sem potrdil. Za merjenje fizičnih značilnosti odsekov kolesarske površine sem uporabil aplikacijo SensorKraken in iz nje uspešno pridobil podakte, ki so bili z nadaljno analizo pretvorjeni v ocene teh odsekov. To je bilo narejeno z mobilnim telefonom in lepilnim trakom, torej brez specializiranih naprav.

5.2 OCENE ODSEKOV

5.2.1 POVPREČNE OCENE IZ ANKETE

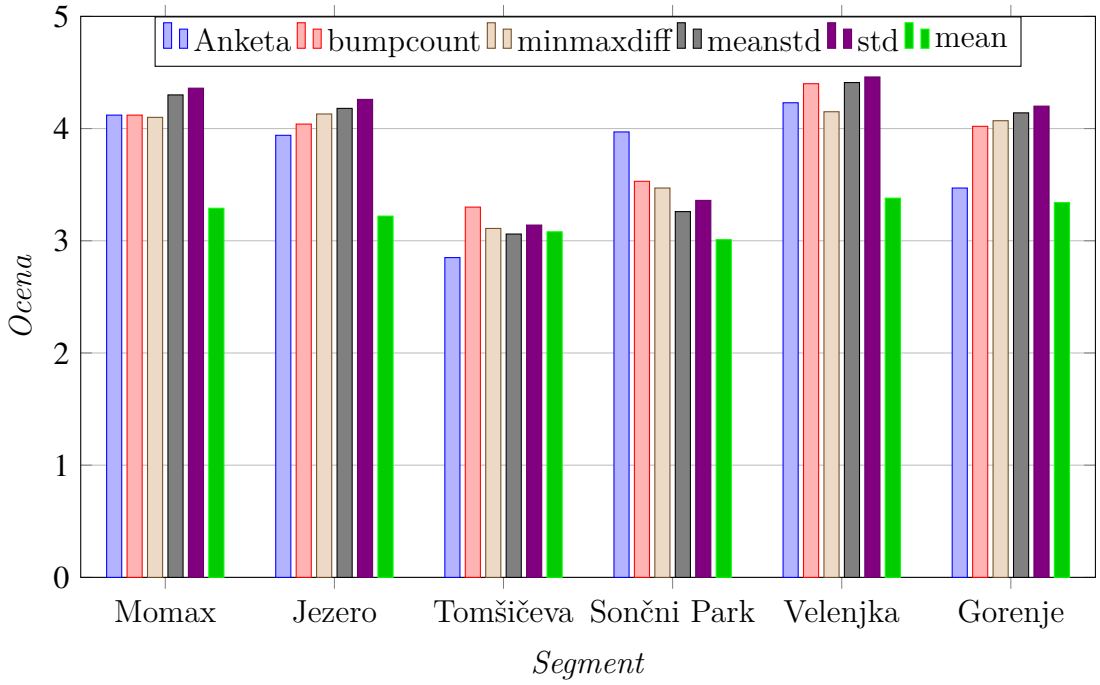
Iz rezultatov ankete lahko izračunamo povprečno oceno posameznih odsekov, ki so bili vključeni vanjo. S tem dobimo naslednje številke (zaokrožene na 2 decimalni mesti):

- Momax: 3.95
- Jezero: 3.68
- Tomšičeva: 2.94
- Sončni park: 3.90
- Velenjka: 4.09
- Gorenje: 3.48

5.2.2 OCENE CENILK

V nadaljnih tabelah so odseki po vrstnem redu: Momax, Jezero, Tomšičeva, Sončni park, Velenjka, Gorenje označeni s številkami od 1 do 6 za namen jedrnatosti tabel.

Vse ocene segmentov so predstavljene v grafu 26. Vidimo, da sta cenilki bumpcount in minmaxdiff pogosto blizu povprečne ocene ankete. Pri odseku Sončni Park vidimo največji odklon teh dveh cenilk od ocen ankete. Po mojem subjektivnem mnenju so ljudje zaradi lepote vožnje po parku ta segment kljub njegovi neudobnosti ocenili bolje, kot bi ga, če bi bil kje drugje, torej niso ocenjevali le kvalitete dejanske površine.



Graf 26: Ocene segmentov

Tabela 7 bolj podrobno predstavi odklone ocen cenilk od rezultatov ankete. Stolpec "Povprečje" predstavlja povprečni odklon od rezultatov ankete za to cenilko. Vidimo, da se rezultatom ankete s povprečnim odklonom 0.28 najbolj približata cenilki minmaxdiff in bumpcount, za njima je cenilka meanstd z odklonom 0.36, za to cenilka std z odklonom 0.40, sledi jim pa še relativno najslabša cenilka, in sicer mean z odklonom 0.62. V tej tabeli so vse vrednosti razen povprečja zaokrožene na 1 decimalno mesto, povprečje pa je zaokroženo na 2.

Razlog, da je mean najslabša cenilka, je jasen že iz njene narave, saj zajema le povprečje vrednosti, zato so vse ocene blizu vrednosti 3 (povprečje vrednosti vseh možnih ocen). Ta cenilka torej dejansko ne ocenjuje kvalitete odsekov.

Od cenilk, ki se bolj približajo rezultatom ankete, sta najboljši cenilki minmaxdiff in bumpcount. Očitno sta torej ta dva načina ocenjevanja najbolj podobna načinu, ki ga uporabljajo ljudje, ko ocenjujejo nek odsek kolesarske površine.

Cenilka minmaxdiff je uspela oceniti dane odseke z največjim odklonom 0.6, cenilka bumpcount pa še z manjšim največjim odklonom 0.5.

Tabela 7: Odklon ocen cenilk od rezultatov ankete

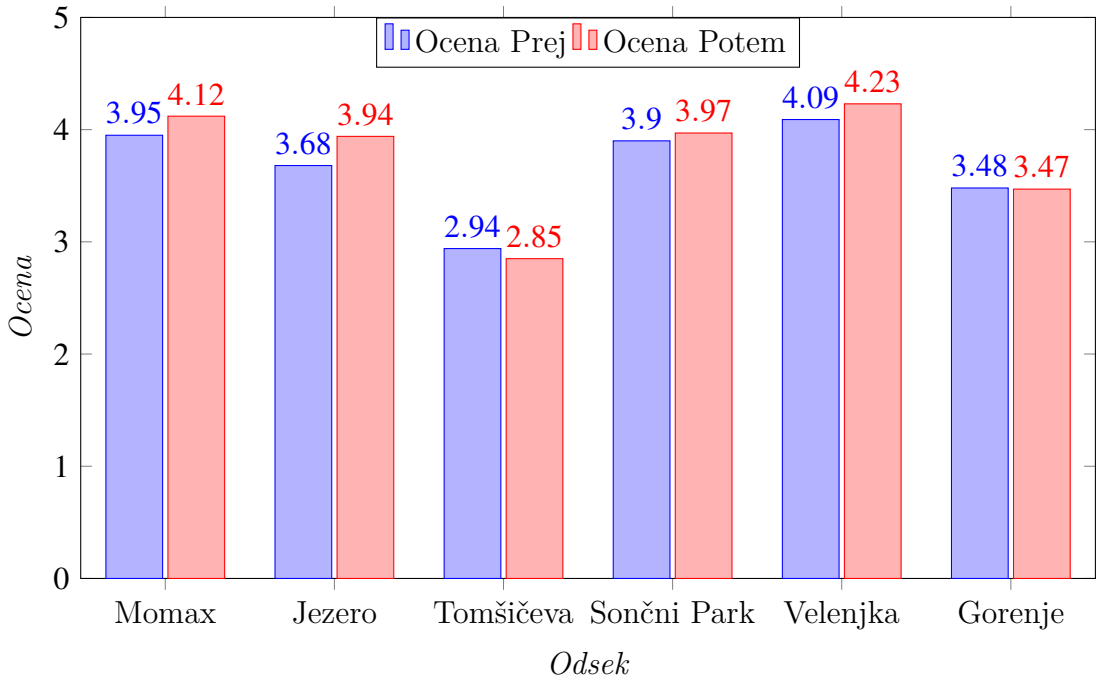
Cenilka/Odsek	**1**	**2**	**3**	**4**	**5**	**6**	**Povprečje**
Anketa	0.0	0.0	0.0	0.0	0.0	0.0	0.0
mean	0.8	0.7	0.2	1.0	0.9	0.1	0.62
std	0.2	0.3	0.3	0.6	0.2	0.7	0.40
meanstd	0.2	0.2	0.2	0.7	0.2	0.7	0.36
minmaxdiff	0.0	0.2	0.3	0.5	0.1	0.6	0.28
bumpcount	0.0	0.1	0.5	0.4	0.2	0.5	0.28

5.2.3 STATISTIČNA PRIMERJAVA

Samo z zgornjimi podatki hipoteze *"Izmerjena kvaliteta posameznih odsekov kolesarske površine se ujema z mnenjem kolesarjev o kvaliteti tega odseka."* še ni mogoče potrditi, saj nič dejansko ne nakazuje, da bi se ocene v anketi in izračunane ocene statistično ujemale.

Za največjo natančnost sem v nadaljnji statistični obdelavi uporabil le rezultate anketirancev, ki so poznali vse odseke. S tem odstranimo pristranskost, saj če na primer nek anketiranec ni poznal odseka, ki je relativno slab, je ostale odseke zaradi tega mogoče ocenil slabše.

Preko tega filtra pride 78 anketirancev, povprečne ocene segmentov pa se minimalno spremenijo. To je razvidno v grafu 27 (Ocena Potem), kjer so te primerjane s povprečnimi ocenami vseh anketirancev (Ocena Prej).



Graf 27: Primerjava povprečnih ocen anketirancev

Zdaj imamo torej skupino podatkov (4.12, 3.94, 2.85, 2.97, 4.23 in 3.47), ki jo lahko z

različnimi statističnimi preizkusi primerjamo s skupinami podatkov (ocenami), ki so jih vrnili algoritmi cenilk. Opravil sem dva preizkusa, in sicer Kolmogorov-Smirnov preizkus ter Wilcoxonov preizkus predznačenih rangov. Vrednosti p obeh preizkusov za vse cenilke v primerjavi z rezultati ankete so predstavljene v tabeli 8. V tej tabeli predstavlja drugi stolpec p vrednost za Kolmogorov-Smirnov preizkus, tretji pa p vrednost za Wilcoxonov preizkus.

Tabela 8: Rezultati statističnih preizkusov

Cenilka	**P-vrednost (K-S)**	**P-vrednost (Wilcoxon)**
mean	0.03	0.09
std	0.47	0.31
meanstd	0.47	0.44
minmaxdiff	0.93	0.69
bumpcount	0.93	0.31

Iz tabele lahko razberemo, da lahko trdimo, da se ocene cenilke mean in rezultati ankete statistično značilno razlikujejo. Za nobeno drugo cenilko s temi vrednostmi ne moremo ne potrditi ne ovreči nobene od statističnih hipotez.

Kljub temu, da se je ena od cenilk (bumpcount) uspela približati rezultatom ankete z največjim odklonom, ki je bil manjši od 0.5 (kar je po mojem subjektivnem mnenju precej dobro), hipoteze *"Izmerjena kvaliteta posameznih odsekov kolesarske površine se ujema z mnenjem kolesarjev o kvaliteti tega odseka."* nisem niti potrdil niti ovrgel. Čeprav statistični preizkusi niso pokazali, da se ocene te cenilke ne bi ujemale z ocenami iz ankete (kar bi mi omogočilo, da to hipotezo ovržem), to ne pomeni, da jo lahko potrdim. Za potrditev te hipoteze bi bilo potrebno opraviti nadaljnje statistične analize in pa, v najboljšem primeru, izvesti še eno anketo na novih (kontrolnih) segmentih in za tem to cenilko preizkusiti še na njih ter dokazati statistično značilno ujemanje.

5.3 IZBIRA POTI

Zadnjo hipotezo, torej *"Kvaliteta kolesarskih površin vpliva na izbiro poti pri kolesarjih."* sem prav tako potrdil, saj je v rezultatih ankete očitna korelacija med mnenjem anketirancev, da je nek odsek relativno slabe kvalitete in procentom anketirancev, ki bi se temu odseku izognili. Primer tega vidimo pri odseku Velenjka, ki je bil obenem najbolje ocenjen in imel najmanjši procent anketirancev, ki se bi mu izognili. Nasprotni primer je najslabše ocenjeni odsek, Tomšičeva, ki je imel obenem najvišji procent anketirancev, ki se bi mu izognili.

6 ZAKLJUČEK

V sklopu te raziskovalne naloge sem bil postavljen pred veliko izzivov, vse od programerskih do čisto praktičnih. Zanimiv problem, ki se mi zdi vreden omembe, je težava z vremenom, saj sem začel podatke za raziskavo zbirati tik pred zimo. Ko se je začelo mrzlo zimsko vreme, mi je bilo onemogočeno kolesarjenje, zato sem na koncu uspel zajeti le 4 kolesarjenja. Vseeno se mi je uspelo z analizo pridobljenih podatkov približati rezultatom ankete. S tem sem razvil algoritem, ki lahko oceni kvaliteto odsekov kolesarskih površin in končne zemljevide objavil na spletno stran.

Manjša težava, ki se pojavi pri zajemu podatkov, je neponovljivost meritev. S tem ciljam na dejstvo, da bi se, tudi, če bi se po nekem odseku peljal večkrat, meritve vsaj malo razlikovale, saj je nemogoče s kolesom peljati po isti poti. Da bi minimiziral posledice te težave, bi rad v prihodnosti izvedel več meritev na vseh odsekih in spet izvedel obdelavo podatkov.

V prihodnosti bi rad prav tako bolj podrobno raziskal, kako različne značilnosti kolesa (tlak v zračnicah, tip kolesa ipd.) vplivajo na meritve. Rad bi tudi izboljšal program za ocenjevanje na tak način, da bi lahko združil več podatkov, ki so iz istega odseka, in jih analiziral naenkrat (v trenutnem programu to ni mogoče, odseki so analizirani posebej, tudi če se v fizičnem svetu prekrivajo).

Prav tako bi rad izboljšal, kako deluje samo ocenjevanje odsekov v programu. Zdaj so vse ocene relativne na ostale podatke, mogoče bi pa bilo nastaviti neke mejne vrednosti, ki ločujejo slabo cesto od dobre (seveda bi bila to skala, ne le da/ne) in s tem posplošiti ocenjevanje, da bi lahko program ocenil tudi le kratek odsek, samega po sebi.

Še ena možnost izboljšave te raziskave je iskanje boljšega načina za pritrditev telefona na kolo, ki je enako zanesljiv kot ta, ki sem ga uporabil, toda je za večkratno uporabo.

7 POVZETEK

V raziskovalni nalogi sem si zadal izziv analize kvalitete in udobnosti vožnje po kolesarskih površinah kolesarskih površin po Velenju in okolici. Najprej sem izvedel anketo, v kateri sem zbral mnenja velenjskih kolesarjev o kvaliteti kolesarskih površin. Nato sem si pogledal nekaj možnosti zbiranja podatkov, s katerimi bi lahko ocenil kvaliteto kolesarskih površin. Izbral sem si uporabo senzorjev na pametnem telefonu. Za tem sem zbral senzorske podatke po večjem delu kolesarskih površin po Velenju in okolici in jih obdelal. Na koncu sem zbrane podatke prikazal na več različnih načinov, s katerimi si lahko pomagamo pri ocenjevanju udobnosti vožnje po poteh, in jih primerjal z rezultati ankete. Ugotovil sem, da je mogoče izdelati algoritem, ki iz zbranih podatkov vrne ocene odsekov, ki so podobne ocenam ljudi.

8 SUMMARY

The main challenge of this research work was to analyze the quality and comfort of cycling on different cycling paths around the town of Velenje. First, I created a survey to gather the opinions of cyclists in Velenje about the quality of cycling surfaces. Then, I looked at a couple of ways to gather data that could be used to assess the quality of the surfaces. I chose to use the sensors present in modern smartphones, which I used to gather accelerometer, gyroscope and GPS data for cycling paths around the town and it's surroundings. I then processed this data and displayed in different ways that can be used to rate the quality of the cycling paths. I also compared it to the results of the survey and found out that it's possible to create an algorithm that assesses the quality of cycling surfaces with similar results as cyclists' opinions.

9 VIRI IN LITERATURA

[1] Amzs. Mobilnost: Razcvet kolesarjenja v sloveniji, 2021. URL https://www.amzs.si/motorevija/mobilnost/na-poti/2021-06-02-mobilnost-razcvet-kolesarjenja-v-sloveniji.

[2] 2024. URL https://www.velenje.si/o-velenju/mobilnost/gremo-s-kolesom/.

[3] Eric Leduc and Gabriel J. Assaf. Road visualization for smart city: Solution review with road quality qualification. *Internet of Things*, 12:100305, 2020. ISSN 2542-6605. doi: https://doi.org/10.1016/j.iot.2020.100305. URL https://www.sciencedirect.com/science/article/pii/S2542660520301372.

[4] S B Prapulla, Sriram N Rao, and Vivek A Herur. Road quality analysis and mapping for faster and safer travel. In *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, pages 2487–2490, 2017. doi: 10.1109/ICECDS.2017.8389899.

[5] Astarita Vittorio, Vaiana Rosolino, Iuele Teresa, Caruso Maria Vittoria, P. Giofrè Vincenzo, and De Masi Francesco. Automated sensing system for monitoring of road surface quality by mobile devices. *Procedia - Social and Behavioral Sciences*, 111:242–251, 2014. ISSN 1877-0428. doi: https://doi.org/10.1016/j.sbspro.2014.01.057. URL https://www.sciencedirect.com/science/article/pii/S1877042814000585. Transportation: Can we do more with less resources? – 16th Meeting of the Euro Working Group on Transportation – Porto 2013.

[6] Yazan Alqudah and Belal Sababha. On the analysis of road surface conditions using embedded smartphone sensors. In *2017 8th International Conference on Information and Communication Systems (ICICS)*, pages 177–181, 04 2017. doi: 10.1109/IACS.2017.7921967.

[7] Alessio Martinelli, Monica Meocci, Marco Dolfi, Valentina Branzi, Simone Morosi, Fabrizio Argenti, Lorenzo Berzi, and Tommaso Consumi. Road surface anomaly assessment using low-cost accelerometers: A machine learning approach. *Sensors*, 22(10), 2022. ISSN 1424-8220. doi: 10.3390/s22103788. URL https://www.mdpi.com/1424-8220/22/10/3788.

[8] G Arun Kumar, A Santhosh Kumar, A Ajith Kumar, and T Maharajothi. Road quality management system using mobile sensors. In *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pages 1–6, 2017. doi: 10.1109/ICIIECS.2017.8276014.

[9] Ming Liu. A study of mobile sensing using smartphones. *International Journal of Distributed Sensor Networks*, 9(3):272916, 2013. doi: 10.1155/2013/272916. URL https://doi.org/10.1155/2013/272916.

[10] 2024. URL https://developer.android.com/develop/sensors-and-location/sensors/sensors_overview#java.

[11] Senthil Kumar Jagatheesaperumal, Simon Elias Bibri, Shrivarshni Ganesan, and Poongkalai Jeyaraman. Artificial intelligence for road quality assessment in smart cities: a machine learning approach to acoustic data analysis. *Computational Urban Science*, 3(1):28, Sep 2023. ISSN 2730-6852. doi: 10.1007/s43762-023-00104-y. URL https://doi.org/10.1007/s43762-023-00104-y.

[12] MD Khan, MDM Alam, MDA Masud, and AA Amin. Importance of high order high pass and low pass filters. *World Applied Sciences Journal*, 34(9): 1261–1268, 2016.

[13] Yoan Tournade. Peak detection in the python world, Nov 2015. URL https://blog.ytotech.com/2015/11/01/findpeaks-in-python/.

[14] Ali N. Akansu and Richard A. Haddad. Chapter 6 - wavelet transform. In Ali N. Akansu and Richard A. Haddad, editors, *Multiresolution Signal Decomposition (Second Edition)*, pages 391–442. Academic Press, San Diego, second edition edition, 2001. ISBN 978-0-12-047141-6. doi: https://doi.org/10.1016/B978-012047141-6/50006-9. URL https://www.sciencedirect.com/science/article/pii/B9780120471416500069.

[15] Meinard Müller. *The Fourier Transform in a Nutshell*, pages 39–57. Springer, 08 2015. ISBN 978-3-319-21944-8.

[16] Allen Downey. *Think DSP: digital signal processing in Python*. "O'Reilly Media, Inc.", 2016.

[17] 2024. URL https://jupyter.org/about.

[18] Jonathan Bennett. *OpenStreetMap*. Packt Publishing Ltd, 2010.

[19] 2024. URL https://www.openstreetmap.org/about.

[20] F. J. Massey. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

[21] R. F. Woolson. *Wilcoxon Signed-Rank Test*, pages 1–3. John Wiley and Sons, Ltd, 2008. ISBN 9780471462422. doi: https://doi.org/10.1002/9780471462422.eoct979. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9780471462422.eoct979.

[22] Anurag Bansal. Sensors. https://github.com/bansalanurag/Sensors, 2018.

[23] Leo Zu. Sensorkraken. https://gitlab.com/sensorkraken/android-app, 2024.

[24] Volodymyr Agafonkin. Leaflet - an open-source javascript library for interactive maps, 2024. URL https://leafletjs.com/.

ZAHVALA

Za pomoč pri izdelavi raziskovalne naloge se zahvaljujem:

- mentorju g. Gregorju Hrastniku za osnovno idejo, nasvete in pomoč pri izdelavi raziskovalne naloge,
- staršema, ki sta zelo pripomogla pri deljenju ankete, prav tako pa sta pomagala z idejami, priporočili in splošno podporo pri izdelavi raziskovalne naloge,
- gospe dr. Nataši Meh Peer, prof. za lektoriranje raziskovalne naloge,
- Damjanu Vrenčurju za ideje in pomoč pri statistični analizi rezultatov.

PRILOGE

Priloga A

Razred Dataloader

```
from __future__ import annotations

import os
import sys
import json
import pathlib
import numpy as np

class DataLoader:
    def __init__(
        self,
        data_folder,
        sensor_name: str,
        extract_data_func = lambda x: json.loads(x["Values"]),
        extract_timestamp_func = lambda x: x["TimeStampSensor"],
    ):
        self.sensor_name = sensor_name
        self.raw_data: list[dict] = self._load_raw_data(
            data_folder + "/" + self.sensor_name + ".json"
        )

        self.extract_data_func = extract_data_func
        self.extract_timestamp_func = extract_timestamp_func

        # prvi stolpec je časovna oznaka, ostali stolpci so podatki
        self.data: np.ndarray = self._get_data(self.sensor_name)

    # Preberi surove podatke iz datoteke
    def _load_raw_data(self, data_path) -> dict:
        path = pathlib.Path(data_path)
        with open(path, "r") as f:
            data = json.load(f)
        return data

    # Pretvori surove podatke v uporabne podatke
    def _get_data(self, sensor_name) -> np.ndarray:
        # matrika matrik, kjer je prvi element časovna oznaka,
        # ostali elementi pa podatki
        first_timestamp = self.extract_timestamp_func(
```

```
            self.raw_data[sensor_name]["readOuts"][0]
        )
        readouts = []
        for readout in self.raw_data[sensor_name]["readOuts"]:
            try:
                data = self.extract_data_func(readout)
                timestamp = self.extract_timestamp_func(readout)
                timestamp -= first_timestamp

                timestamp_seconds = timestamp / 1_000_000_000
                readouts.append(np.array([timestamp_seconds, *data]))
            except KeyError as e:
                pass

        return np.array(readouts)

    # Pridobi vrednosti senzorja
    def get_data(self) -> np.ndarray:
        return self.data

    def get_data_count(self) -> int:
        return len(self.data)

    # Razdeli podatke (na primer: x, y, z)
    def get_arrays(self, n=3, skip=1) -> tuple[np.ndarray]:
        return tuple([self.data[:, i + skip] for i in range(n)])

class GPSDataLoader(DataLoader):
    def __init__(self, data_folder, sensor_name: str = "GPS"):
        super().__init__(data_folder, sensor_name,
                         extract_data_func=lambda x: [
            x["Location"]["Latitude"],
            x["Location"]["Longitude"],
        ],
        extract_timestamp_func=lambda x: x["Location"]["TimeStampLocation"]
        )
```

Priloga B

Razred PathSegment

```python
from __future__ import annotations
import numpy as np
from .GPSDataPoint import GPSDataPoint
import geopy.distance

import matplotlib.pyplot as plt
import scipy.spatial.transform as st

FEATURE_WEIGHTS = {'mean': 0.5, 'std': 0.5}

class PathSegment:

    def __init__(self):
        self.sensors_data = {}
        self.projected_accelleration_data: list[float] = []

        self.gps_points: list[GPSDataPoint] = []

    def add_gps_point(self, gps_data: GPSDataPoint):
        self.gps_points.append(gps_data)

    def add_sensor_data(self, sensor_name: str, sensor_data: np.ndarray):
        if sensor_name in self.sensors_data:
            self.sensors_data[sensor_name] = np.concatenate(
                (self.sensors_data[sensor_name], [sensor_data])
            )
        else:
            self.sensors_data[sensor_name] = np.array([sensor_data])

    """
    Vzame objekt
    {
        "acceleration": np.ndarray, # vektor
        "rotation": np.ndarray # kvaternion
        "timestamp": float
    }

    in projicira vektor pospeška na Z os (pravokotno na tla)"""
    def add_combined_data(self, data_point: dict) -> float:

        acceleration_vector = data_point['acceleration']
        rotation_vector = data_point['rotation']
```

```python
    # Ustvari rotirano Z os za vsako točko podatkov
    def apply_rotation(rotation_vector: np.ndarray, axis: np.ndarray):
        rotation_matrix = (
            st.Rotation.from_quat(rotation_vector).as_matrix()
        )
        return np.dot(rotation_matrix, axis)

    Z_AXIS = np.array([0, 0, 1])
    rotated_z_axis = apply_rotation(rotation_vector[:4], Z_AXIS)

    # Projiciraj podatke pospeška na rotirane Z osi
    def project_vec_on_vec(v1: np.ndarray, v2: np.ndarray):
        return np.dot(v1, v2) / np.linalg.norm(v2)

    projected_acceleration = (
        project_vec_on_vec(acceleration_vector, rotated_z_axis)
    )

    self.projected_accelleration_data.append(projected_acceleration)

    return projected_acceleration




def get_sensor_data(self, sensor_name: str) -> np.ndarray:
    return (
    self.sensors_data[sensor_name]
        if sensor_name in self.sensors_data else np.array([]
    )

def get_segment_time_length(self) -> float:
    first_gps_data = self.gps_points[0]
    last_gps_data = self.gps_points[-1]

    return last_gps_data.get_timestamp() - first_gps_data.get_timestamp()

# Vrne razdaljo v metrih
def get_segment_distance(self) -> float:

    if len(self.gps_points) < 2:
        return 0

    return np.sum([
            geopy.distance.distance(
                    (self.gps_points[  i  ].get_lat(),
```

```
                    self.gps_points[  i  ].get_lon()),
                    (self.gps_points[i + 1].get_lat(),
                    self.gps_points[i + 1].get_lon())
                ).meters
                for i in range(len(self.gps_points) - 1)
        ])

def get_segment_speed(self) -> float:
    if len(self.gps_points) < 2:
        return 0

    dist = self.get_segment_distance()

    if dist == 0:
        return 0

    length = self.get_segment_time_length()

    return dist / length


def process(self, algo = "speed", extra = None) -> float:


    if algo == "bumpcount":

        # Ta algoritem šteje število udarcev v
        # segmentu s pomočjo projiciranih podatkov pospeška

        # 2.9 je izračunana najboljša vrednost za ta parameter
        THRESHOLD = extra if extra is not None else 2.9

        # število vrednosti večjih od praga
        # (opomba: prej je bil uporabljen abs, zdaj ni več)
        bumps = np.sum(
            np.abs(self.projected_accelleration_data) > THRESHOLD
        )

        dist = self.get_segment_distance()

        if dist == 0:
            return 0

        return bumps / dist

    elif algo == "minmaxdiff":
```

```
        # Ta algoritem izračuna razliko med najmanjšo
        # in največjo vrednostjo projiciranega pospeška

        min_val = np.min(self.projected_accelleration_data)
        max_val = np.max(self.projected_accelleration_data)

        return max_val - min_val

    elif algo == "mean":

            # Ta algoritem izračuna
            # povprečje projiciranega pospeška
            return np.mean(self.projected_accelleration_data)

    elif algo == "std":

            # Ta algoritem izračuna
            # standardni odklon projiciranega pospeška
            return np.std(self.projected_accelleration_data)

    elif algo == "meanstd":
        # Ta algoritem izračuna povprečje
        # in standardni odklon projiciranega pospeška
        mean_val = np.mean(self.projected_accelleration_data)
        std_val = np.std(self.projected_accelleration_data)
        return mean_val + std_val

    raise ValueError(f"Neznan algoritem: {algo}")
```