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

Računalništvo, varnost v cestnem prometu

Avtor:
Anže Maj Blagus

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

November 2023

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	2023
IN	ANALIZA KOLESARSKIH POVRŠIN V ŠALEŠKI DOLINI
TD	Raziskovalna naloga
OP	XII, 54 str., 7 pregl., 18 sl., 21 graf., 1 pril, 22 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 segmentov, 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	2023
TI	ANALYSIS OF CYCLING SURFACES IN ŠALEK VALLEY
DT	Research work
NO	XII, 54 p., 7 tab., 18 fig., 21 graf., 1 ann, 22 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

SEZNAM TABEL

SEZNAM SLIK

SEZNAM FORMUL

SEZNAM KODE

SEZNAM GRAFOV

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.

Prva relevantna raziskava z naslovom "Road visualization for smart city: Solution review with road quality qualification" se ukvarja 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.

2.1 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.

2.1.1 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

2.1.2 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]

```
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].

2.1.3 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.

2.1.4 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.

2.2 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.

2.2.1 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]

2.2.2 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

2.2.3 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
```

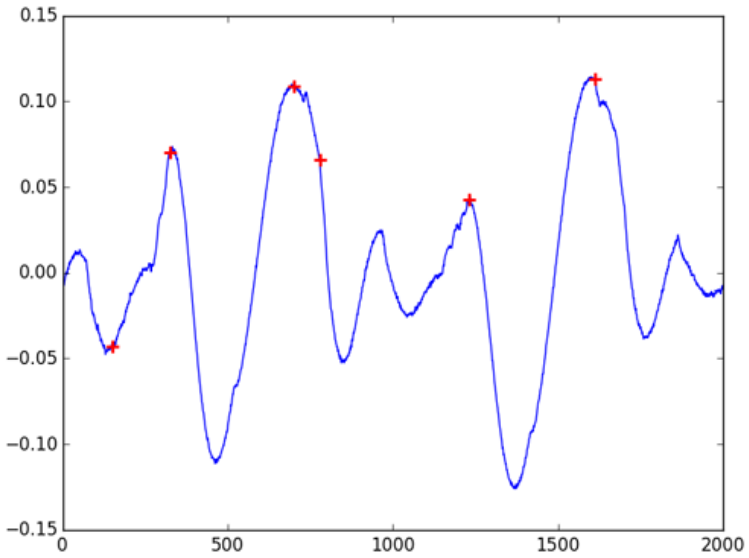
Koda 2: Enostaven primer kode za zaznavanje vrhov

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].

2.2.4 VALOVNA TRANSFORMACIJA

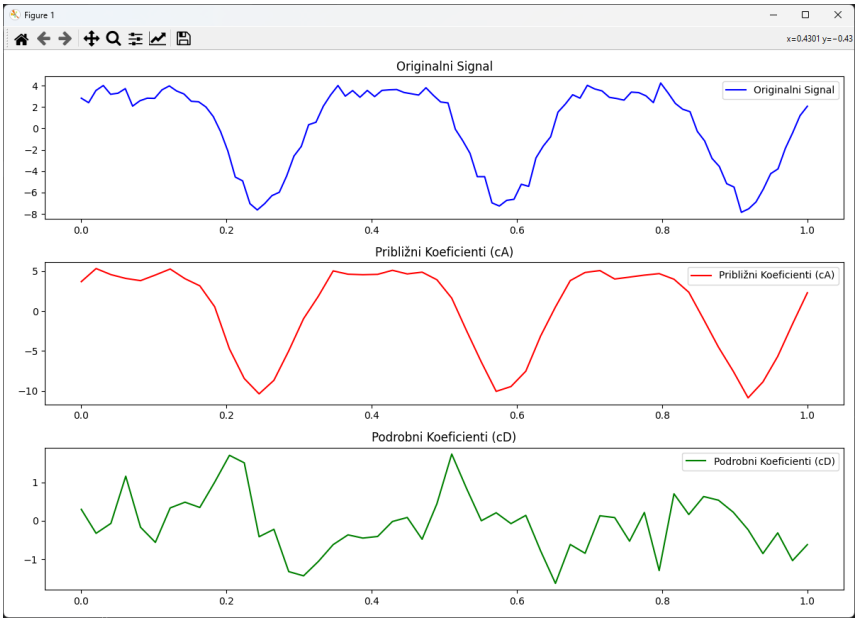
Valovna transformacija (angl. wavelet transfor) je tehnika, ki omogoča deljenje signalov ali podatkov v različne valovne komponente različnih frekvenc. S to metodo lahko identificiramo lokalne spremembe v signalih. [14]

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

```python
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

2.2.5 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]

2.2.6 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.

2.3 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.

2.3.1 PYTHON

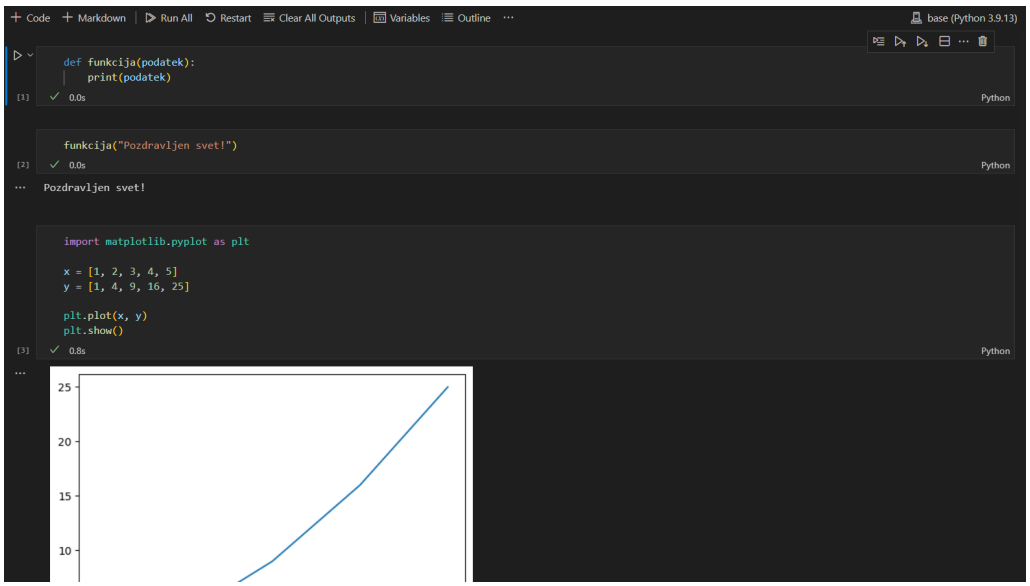
Python je visokonivojski programski jezik, namenjen uporabnikom z razponom znanja vse od začetnikov do izkušenih uporabnikov. Omogoča hiter razvoj raznih aplikacij. V realnem svetu se uporablja za vse od razvoja spletnih aplikacij do raznega procesiranja podatkov v raziskovalne namene. [16]

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

```
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.

2.3.2 OPENSTREETMAP

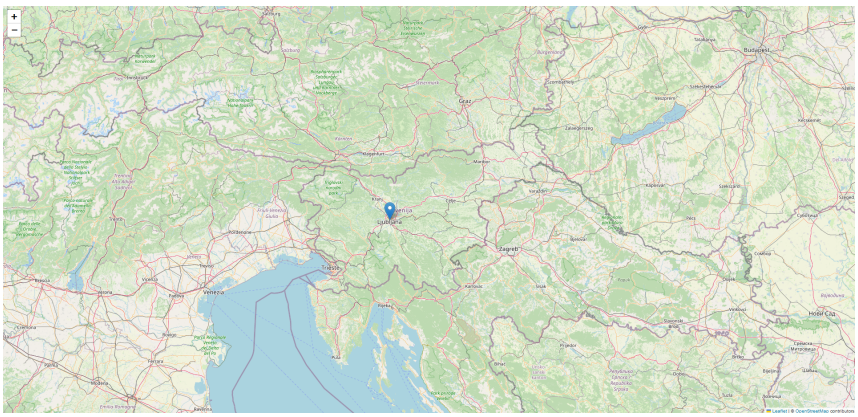
OpenStreetMap je baza geografskih podatkov, ki je odprta javnosti in zastonj. Je projekt, h kateremu prispevajo mnogi ljudje iz celega sveta in ponuja vse od lokacij podjetji do zastonj storitve, ki ponuja spletne zemljevide. [18]

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

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 segmentih, na katerih bi baziral nadaljnje raziskovanje. Drugi del je zbiranje objektivnih podatkov o kolesarskih površinah, torej nekakšno merjenje kvalitete segmentov. Tretji del je obdelava ter prikaz teh podatkov na uporaben način.

Pred začetkom je vredno razložiti mojo uporabo besede segment. Ko govorim o segmentu kolesarske površine, govorim o odseku kolesarske poti ali steze (primer: kolesarska ob neki glavni cesti), ki je dolg nekje med nekaj deset in nekaj sto metri. Primeri teh segmentov so vidni v anketi. Ko razlagam kodo za procesiranje signalov, se pojavi beseda segment v kontekstu GPS podatkov, ki so združeni s senzorskimi meritvami, toda ti segmenti 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 segmentih.

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 segmentih 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 segmenta na zemljevidu in nato o njem postavil naslednja vprašanja:

- *"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 segmentov, po katerih sem bolj podrobno spraševal. Tem segmentov 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 [20]). 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

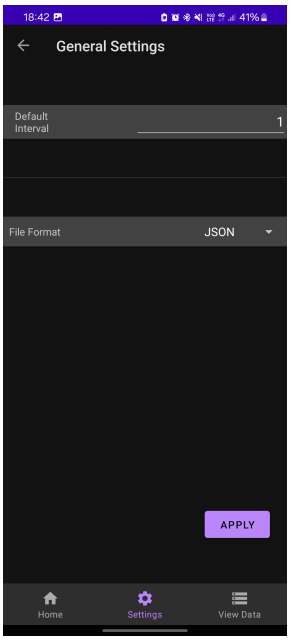
Aplikacija SensorKraken je odprtokodna aplikacija, izdelana z namenom zapisovanja senzorskih podatkov. [21]

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

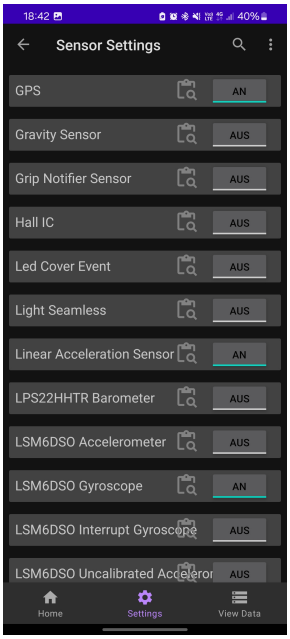


Slika 5: Nastavitve aplikacije

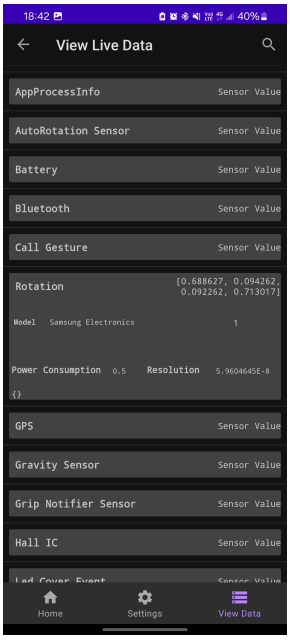
Aplikacija omogoča torej tudi izbiro, katere podatke naj zajema, kar vidimo na sliki 6. Izbral sem zajem GPS podatkov, podatkov senzorja linearnega pospeška (angl. linear acceleration sensor), rotacijskega vektorja (angl. rotation vector), ki nam pove trenutno orientacijo telefona v globalnem prostoru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

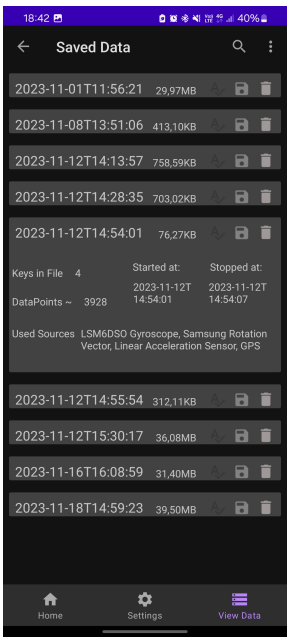


Slika 6: Seznam senzorjev

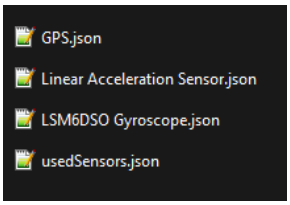


Slika 7: Trenutne vrednosti senzorjev

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 7. 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 7: 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

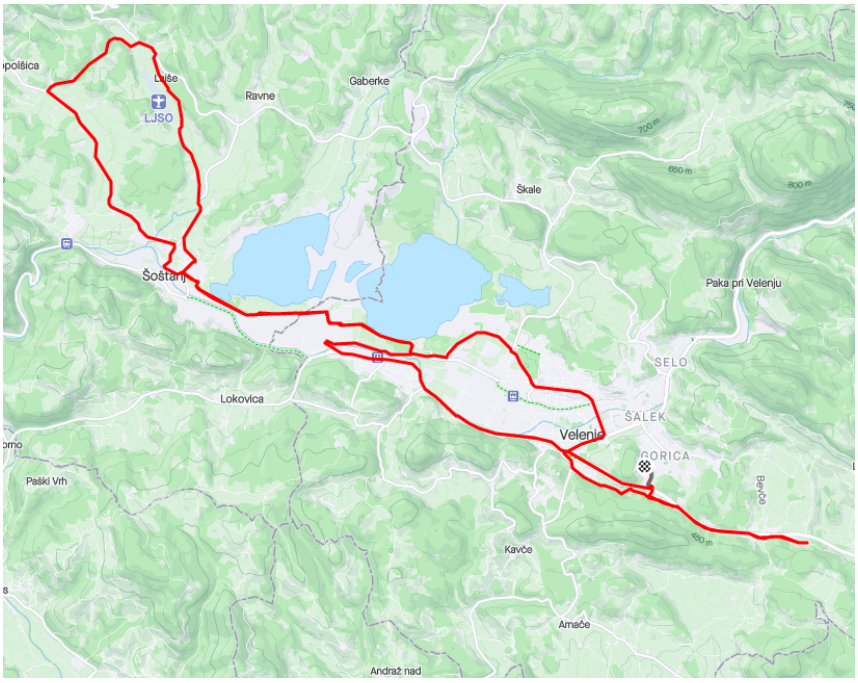
Vir: lasten

Slika 14: Hibridno kolo

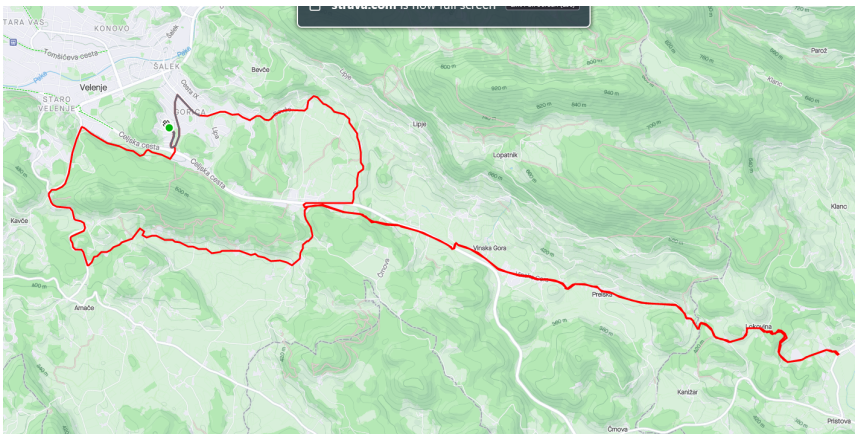
Dejansko kolesarjenje, torej ”zajemanje” sem izvedel v večih delih. Pot, po kateri sem kolesaril, sem si pred vsakim kolesarjenjem načrtoval tako, da sem se peljal po čim več segmentih, ki so bili vključeni v anketo.

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-

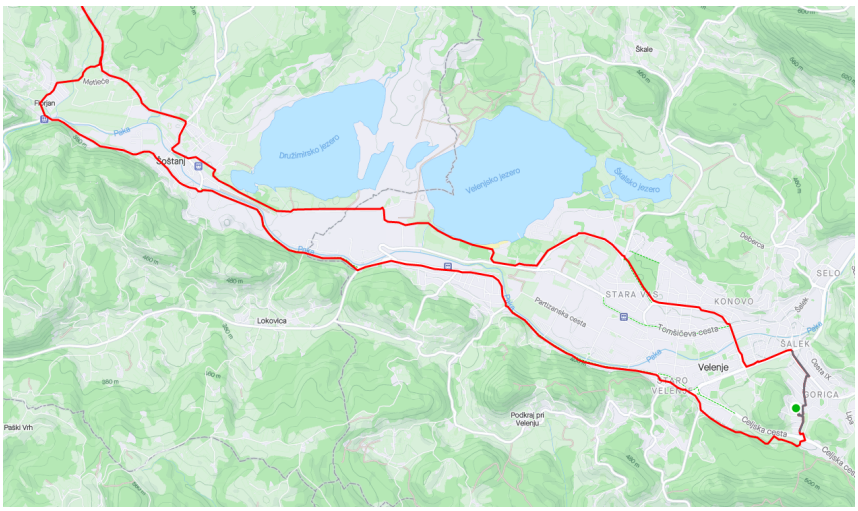
datkov za vsak segment, po katerem sem spraševal v anketi. Za nekatere segmente sem dobil tudi večkratne podatke, kar omogoči nadaljno analizo in primerjavo.



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 segmente 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 segmentov 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 8.

```
# Omejitev za korak predprocesiranja
LOWPASS_FILTER_ALPHA = 0.75
# Minimalna dolzina posameznega segmenta
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 8: Konfiguracija programa

V drugem delu programa, so prebrani podatki iz zgoraj nastavljenih datotek. To branje je prikazano v kodi 9. 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.

```
# 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 9: Nalaganje podatkov

3.3.2 PREDPROCESIRANJE

V koraku predprocesiranja imamo podatke že naložene v spremenljivki all_data, želimo jih pa obdelati na način, da nam bodo v nadaljnih korakih bolj uporabni. V ta namen, na vseh senzorskih podatkih uporabimo nizkoprepustni filter (razložen v poglavju poglavju 2.2.1). Ta korak je prikazan v kodi 10.

```
for data in all_data:
    for sensor_type, sensor_data in data['sensors_data'].items():
        preprocessed = preprocess(sensor_data, LOWPASS_FILTER_ALPHA)
        data['sensors_data'][sensor_type] = preprocessed

...

def preprocess(
    sensor_data: np.ndarray,
    lowpass_filter_alpha: float
) -> np.ndarray:
    out = sensor_data
    out = lowpass_filter(out, lowpass_filter_alpha)
    return out
```

Koda 10: Predprocesiranje

3.3.3 DELJENJE NA SEGMENTE

V tej točki imamo podatke pripravljene za dejansko analizo. Ker so senzorske podatki, ki se zajemajo več deset-krat na sekundo in GPS podatki, ki se zajemajo enkrat na sekundo, zaenkrat še ločeni, jih moramo nekako povezati. Za to ustvarimo idejo segmentov poti. Ti segmenti so način združevanja več GPS točk z ustreznimi senzorskimi meritvami; z namenom ocenjevanja.

Deljenje na segmente deluje tako, da dodajamo senzorske podatke v trenuten segment, dokler ne pridemo do senzorskega podatka, ki je bil zajet časovno za trenutnim GPS podatkom. Za tem se prestavimo na naslednji GPS podatek in postpopek 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 vsebuje 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 = []
    current_segment = PathSegment()
    for i in trange(len(gps_data)):
        # Minimalna razdalja dosezena -> zacnimo nov segment
        if current_segment.get_segment_distance() >= min_len_meters:
            output.append(current_segment)
            current_segment = PathSegment()
            # dodaj zadno GPS tocko novemu segmentu, da sta povezana
            current_segment.add_gps_point(
                GPSDataPoint(gps_data[i - 1])
            )

        # Dodaj trenutne GPS podatke trenutnemu segmentu
        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
            # Posodobimo indeks
            last_indexes[sensor_name] = index
    # Dodaj zadnji segment
    output.append(current_segment)
    return output
```

Koda 11: Deljenje na segmente

3.3.4 OCENJEVANJE SEGMENTOV

Korak ocenjevanja segmentov, ki smo jih izdelali s kodo v prejšnjem koraku, je namenjen pretvorbi mnogih senzorskih podatkov v eno število, ki "objektivno"oceni kvaliteto tega segmenta (lahko tudi neko drugo značilnost tega segmenta, toda za namen te naloge ocenjujemo kvaliteto). Za to nalogo obstaja več načinov, zato sem ustvaril poenoten vmesnik (funkcijo *process* v razredu *PathSegment*), ki ima dostop do segmenta in je njena naloga, da vrne eno decimalno vrednost – oceno segmenta.

Primer enostavne implementacije te funkcije, ki preprosto vrne povprečno hitrost na posameznem segmentu, je viden v kodi 12

```
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 12: Ocenjevanje segmenta

Za ocenjevanje kvalitete odseka sem si zadal 5 načinov:

- povprečna razlika med največjo in najmanjšo vrednostjo senzorja pospeška od vseh 3 osi v odseku (v kodi in rezultatih poimenovan *minmaxdiff*),
- število nepopolnosti (lukenj, robnikov) v odseku (v kodi in rezultatih poimenovan *bumpcount*),
- standardni odklon vrednosti senzorja pospeška na vseh 3 oseh (v kodi in rezultatih poimenovan *std*),
- aritmetična sredina vrednosti senzorja pospeška na vseh 3 oseh (v kodi in rezultatih poimenovan *mean*),
- kombinacija zgornjih 2 načinov (v kodi in rezultatih poimenovan *meanstd*).

Implementacije vseh zgoraj navedenih načinov so vidne v prilogi B, specifično v metodi *process*. Tukaj je vredno omeniti, kako deluje dejansko ocenjevanje, torej kako iz vrednosti, ki jih vrne metoda *process*, dobimo številske vrednosti za ocene od 1 do 5.

Obdelava se izvaja na vseh podatkih naenkrat. Ko imamo vrednosti ocen za vse segmente, vrednosti normaliziramo (torej so zdaj decimalne vrednosti 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).

3.3.5 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 "segmentov poti" linije na zemljevidu. Vsaka od teh linij dobi določeno barvno vrednost, glede na normalizirano oceno tega segmenta. Ta potek, torej najprej normalizacijo in nato ustvarjanje teh linij, vidimo v kodi 13.

```
map_data = []
segment_values = np.array(
    [segment.process() for segment in segments]
    )

# Normalizacija ocenjenih vrednosti
normalized_values = segment_values.copy()
normalized_values -= np.min(normalized_values)
normalized_values /= np.max(normalized_values)

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 13: Ustvarjanje linij za zemljevid

Po ustvarjanju seznama linij na zemljevidu, program naredi 2 stvari: linije izvozi v JSON obliko (koda 14) 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 14: JSON izvoz linij na zemljevidu

Naloga prikaza zemljevida je predana funkciji create_map, ki je prikazana v kodi 15. Uporablja knjižnico matplotlib v kombinaciji s contextily, kar omogoča uporabo zemljevida kot ozadnja v koordinatnemu sistemu; posledično je koda precej enostavna. Najprej ustvari LineString, torej geometrijske koncepte, ki jih uporablja knjižnica geopandas, za tem nastavi nekaj vrednosti za pravilen izris zemljevida (postavitev), nazadnje pa vse linije nariše.

```
import geopandas as gpd
import matplotlib.pyplot as plt
from shapely.geometry import LineString
import contextily as ctx

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 15: Prikaz na zemljevidu v Python

3.4 PRIKAZ PODATKOV

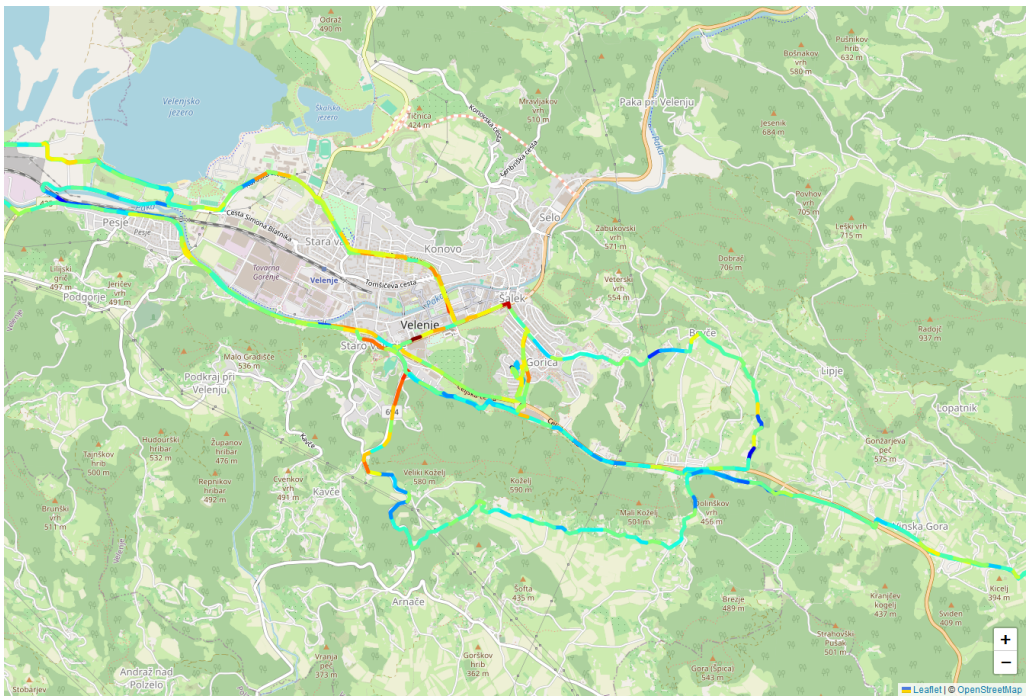
V prejšnjem poglavju je koda izrisala zemljevid s potjo, toda le kot statično sliko. Ker je bil cilj ustvariti interaktiven zemljevid, sem se odločil za izdelavo preproste spletne strani, ki bi uporabljala knjižnico Leaflet za prikaz zemljevida s podatki, izvoženimi v zadnjem koraku prejšnjega poglavja. [22].

Kodo, ki izvaja dejansko risanje, vidimo v kodi 16. 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)*/];
console.log(data);

var map = L.map('map', {
    zoomControl: false
}).setView([46.35563153539, 15.12551209295], 13);
L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19
}).addTo(map);

L.control.zoom({
    position: 'bottomright'
}).addTo(map);


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})`;

    // obrnimo geografsko visino in dolzino,
    // ker ju ta knjiznica zeli v drugi smeri
    for (let i = 0; i < points.length; i++) {
        points[i] = [points[i][1], points[i][0]];
    }
    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 16: 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.

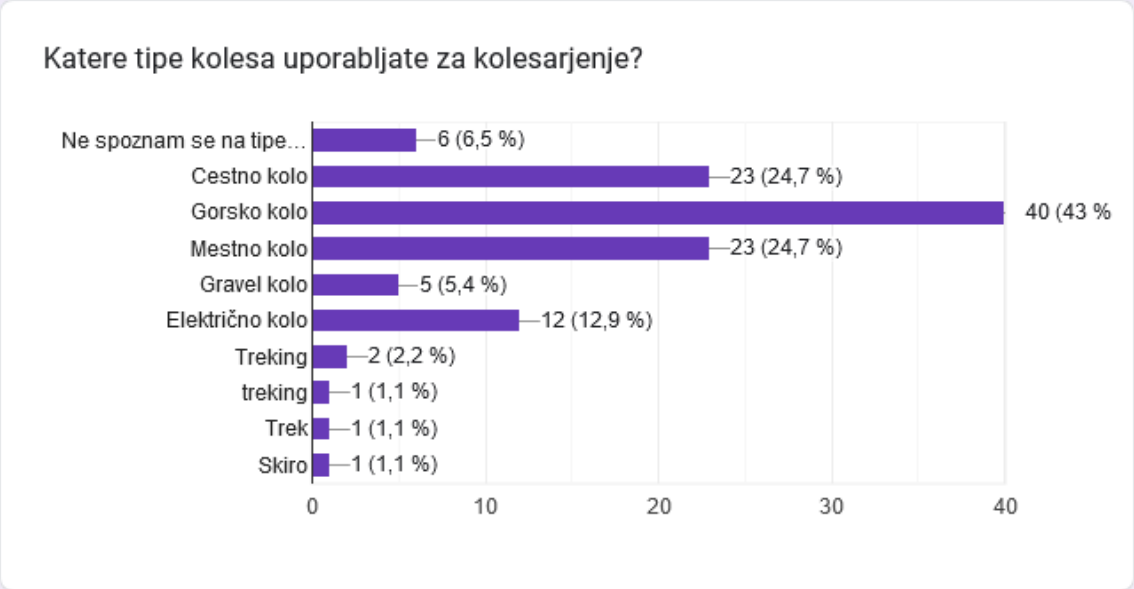
4.1 REZULTATI ANKETE

Vse skupaj je na anketo do časa pisanja odgovorilo 119 anketirancev.

Prvo vprašanje se je nanašalo na pogostost kolesarjenja po kolesarskih površinah v Velenju. Iz grafa 3 je razvidno, da večina anketirancev kolesari nekajkrat na mesec (32.3 %) ali nekajkrat na leto (29 %). Približno petina jih kolesari nekajkrat na teden, 17.2 % pa skoraj vsak dan.

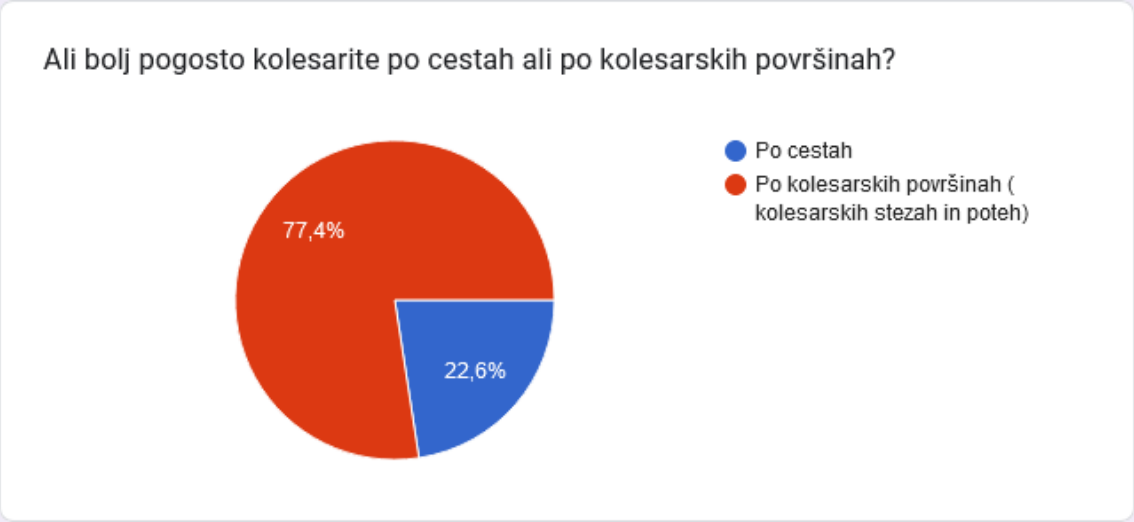
Graf 3: Pogostost kolesarjenja po Velenju

Iz grafa 4 je razvidno, da večina anketirancev za kolesarjenje uporablja gorsko kolo, takoj za tem sta po pogostosti mestno in cestno kolo. Ostali tipi koles so bolj redki.



Graf 4: Uporabljeni tipi koles pri anketirancih

V grafu 5 vidimo razmerje med anketiranci, ki bolj pogosto kolesarijo po cestah (22.6 %) in tistimi, ki bolj pogosto kolesarijo po kolesarskih površinah (77.4 %).

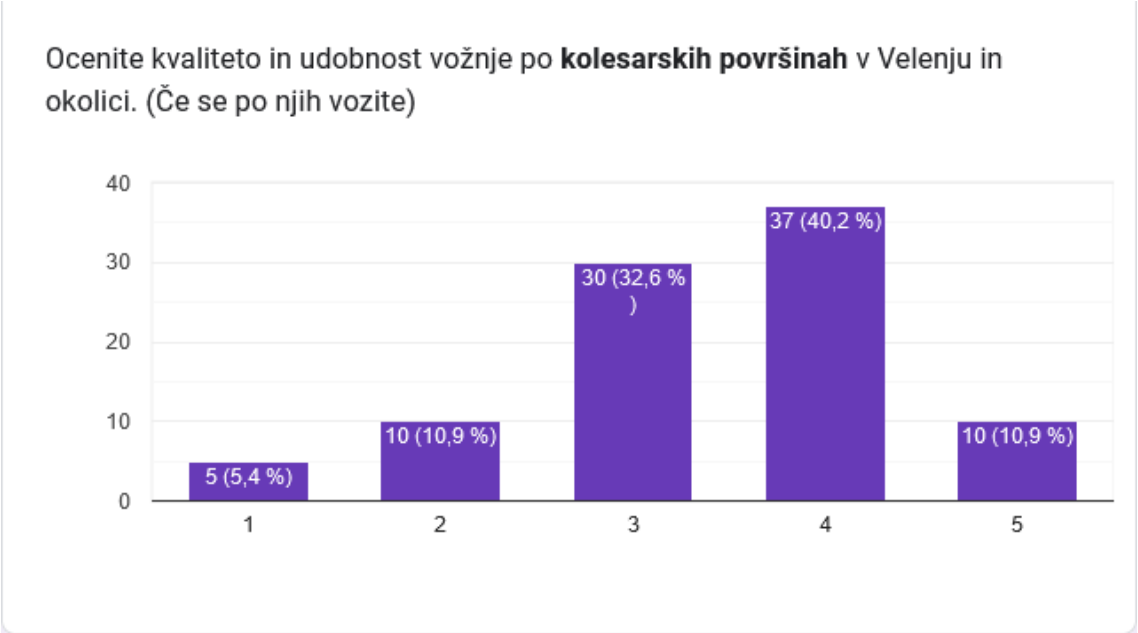


Graf 5: Primerjava pogostosti kolesarjenja po cestah in kolesarskih površinah

Največ od teh anketirancev, ki pretežno kolesarijo po cestah, se tako odloči zaradi pomanjkanja kolesarskih površin. To je razvidno iz grafa 6.

Graf 6: 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 7.



Graf 7: 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 2.9.

Graf 8: Udobnost vožje po cestah

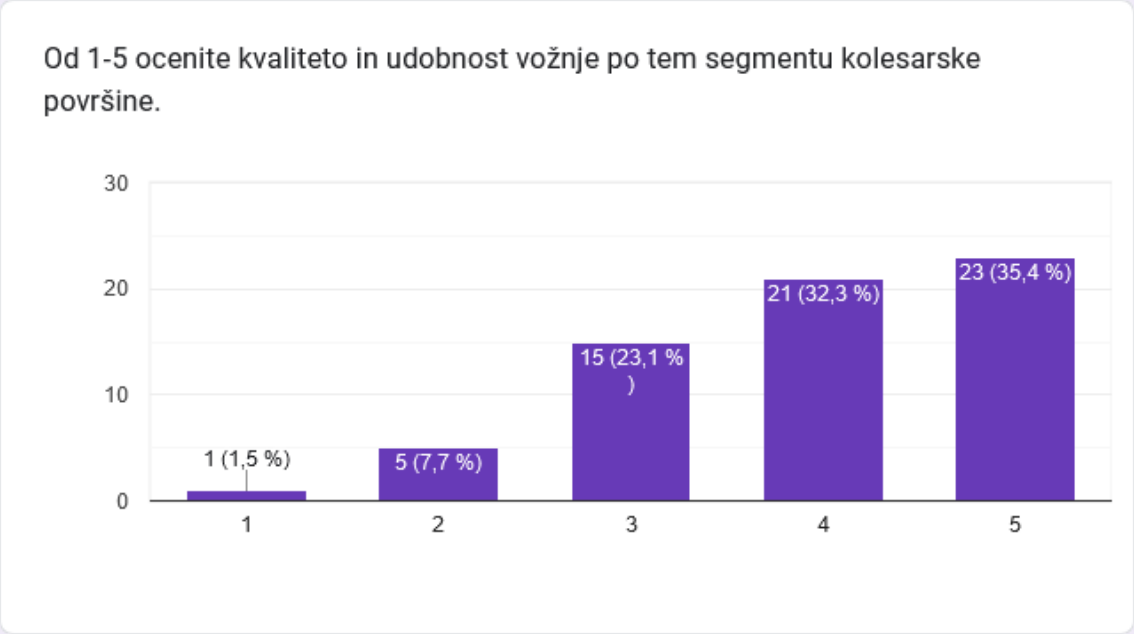
Zadnje vprašanje splošnega dela se nanaša na to, ali anketiranci mislijo, da kdaj kvaliteta površine vpliva na izbiro poti, po kateri se peljejo s kolesom. Iz grafa 9 je razvidno, da večina (79.6 %) anketirancev meni, da kvaliteta površine vpliva na njihovo izbiro poti.

Graf 9: Vpliv kvalitete površine na izbiro poti

4.1.1 ODSEK MOMAX

Prvi odsek, po katerem je spraševala anketa, je bil odsek Momax. Na grafu 10 je večina od tistih anketirancev, ki odsek poznajo, tega ocenila z oceno 5. Povprečna

ocena je bila 3.9. Graf 11 kaže, da se 80 % anketirancev temu segmentu ne bi izognilo.

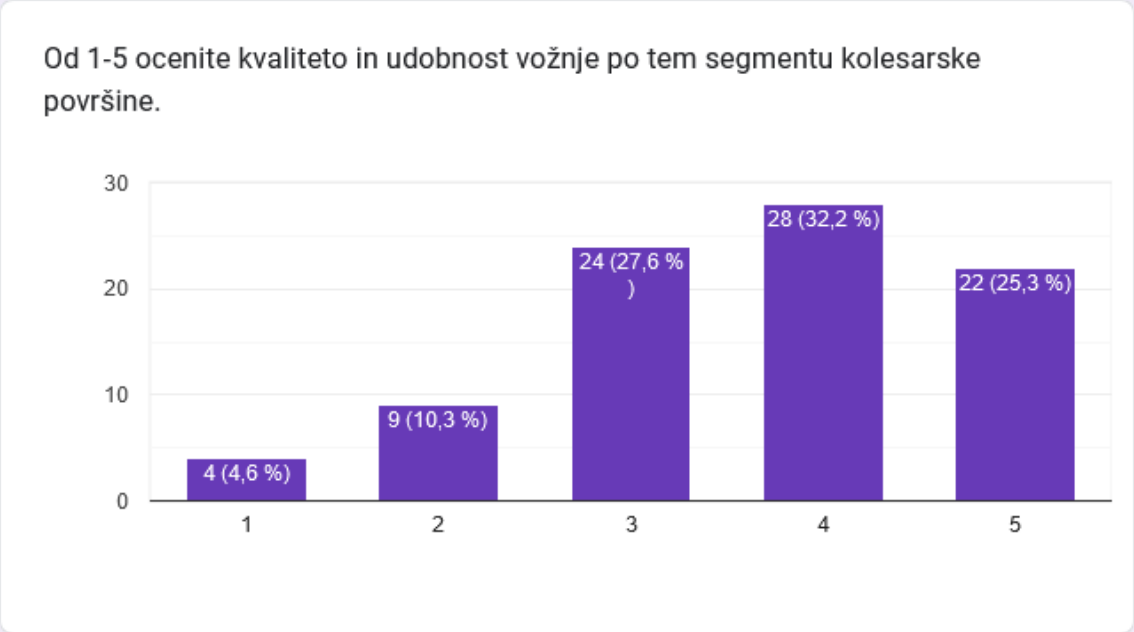


Graf 10: Mnenje o kvaliteti odseka Momax

Graf 11: Izognitev odseku Momax

4.1.2 ODSEK JEZERO

Pri odseku Jezero (graf 12 in graf 13) je najpogostejša ocena 4, povprečna ocena 3.7, 32.2 % anketirancev pa bi se odseku izognilo.

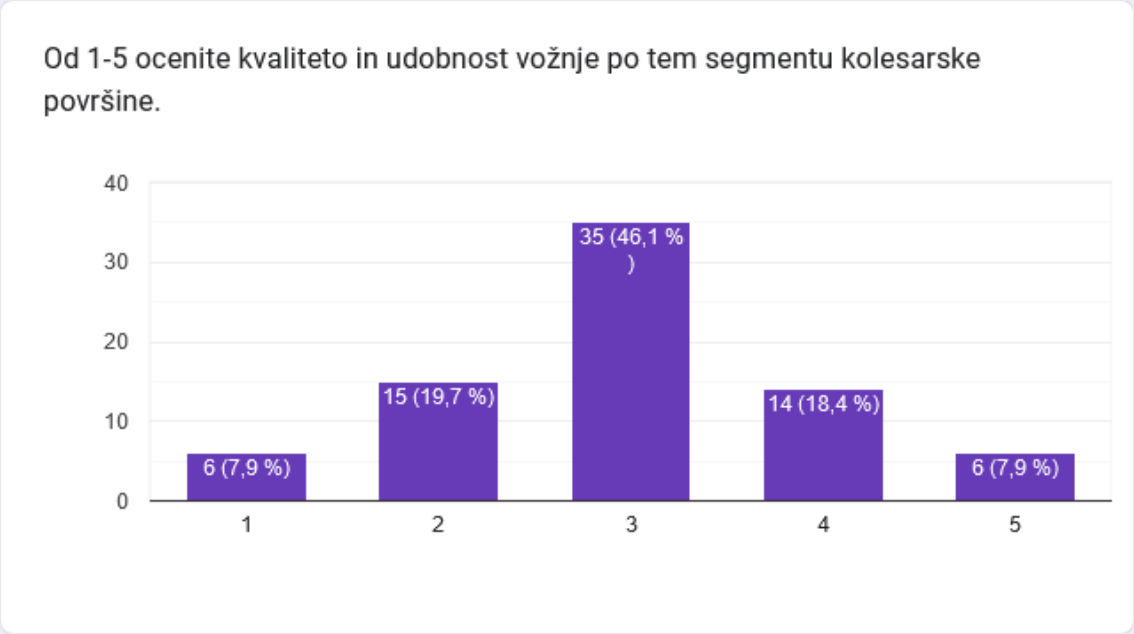


Graf 12: Mnenje o kvaliteti odseka Jezero

Graf 13: Izognitev odseku Jezero

4.1.3 ODSEK TOMŠIČEVA

Odsek Tomšičeva (graf 14 in graf 15) je največ anketirancev ocenilo z oceno 3, povprečna ocena je bila prav tako 3.0, 53.9 % anketirancev pa bi se odseku izognilo.

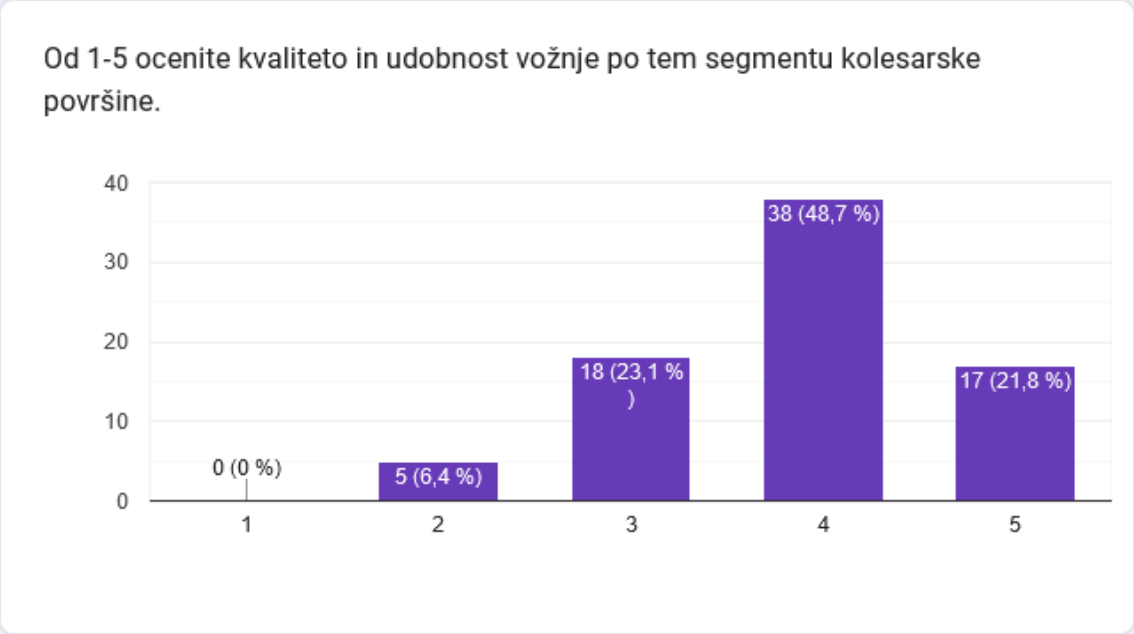


Graf 14: Mnenje o kvaliteti odseka Tomšičeva

Graf 15: Izognitev odseku Tomšičeva

4.1.4 ODSEK SONČNI PARK

Odseku Sončni park bi se izognilo 19.2 % anketirancev (graf 17), največ jih je ocenilo ta odsek z oceno 4 (graf 16), povprečna ocena pa je bila 3.9.

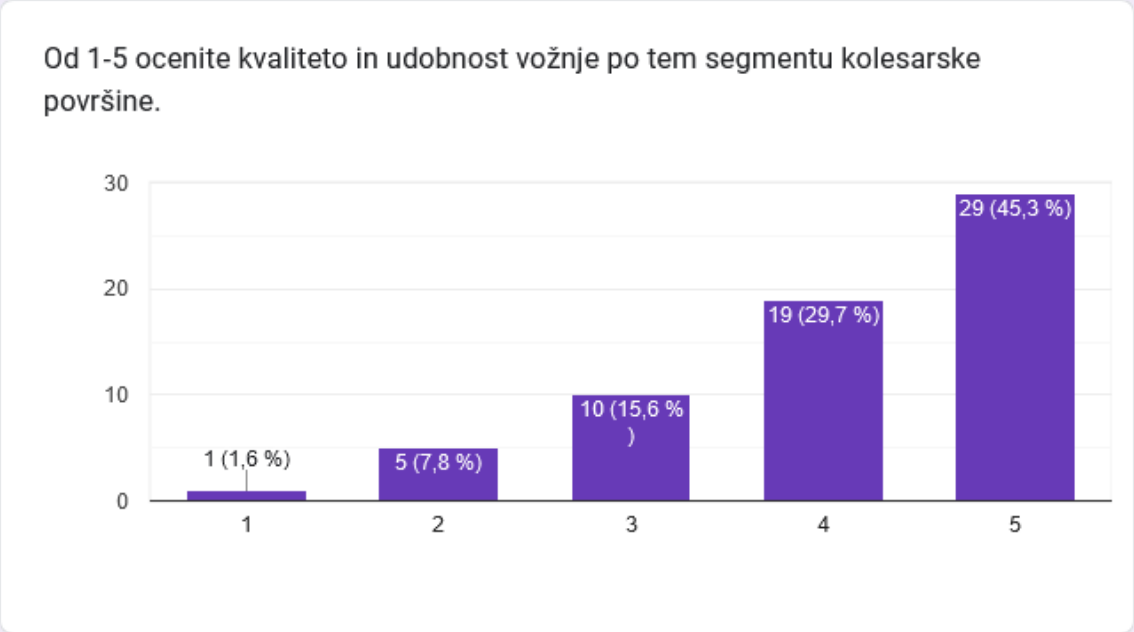


Graf 16: Mnenje o kvaliteti odseka Sončni park

Graf 17: Izognitev odseku Sončni park

4.1.5 ODSEK VELENJKA

Odsek Velenjka je bil od vseh odsekov najbolje ocenjen. Imel je najpogostejšo oceno 5 (graf 18), povprečno oceno 4.1 in izognilo se bi mu le 14.1 % anketirancev (graf 19).

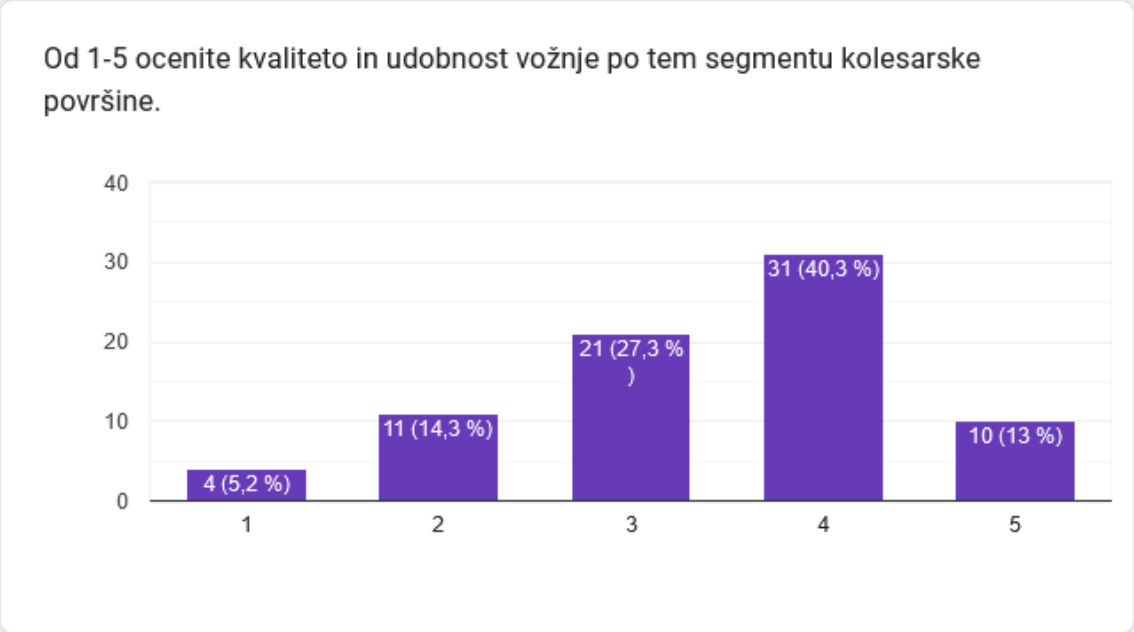


Graf 18: Mnenje o kvaliteti odseka Velenjka

Graf 19: Izognitev odseku Velenjka

4.1.6 ODSEK GORENJE

Zadnji odsek, odsek Gorenje je dobil najpogostejšo oceno 4 (graf 20), povprečno oceno 3.4 in izognilo bi se mu 36.4 % anketirancev.



Graf 20: Mnenje o kvaliteti odseka Gorenje

Graf 21: Izognitev odseku Gorenje

4.2 REZULTATI MERITEV

Po obdelavi in analizi meritev s programov na več različnih načinov sem dobil več različnih rezultatov oziroma ocen vsakega segmenta. V nadaljevanju so ti predstavljeni. Vredno je omeniti, da so za segmente Momax, Tomšičeva in Sončni park v tem poglavju rezultati ločeni na 2 smeri, saj sem pridobil podatke za obe smeri vožnje.

4.2.1 NAČIN MINMAXDIFF

Prvi način oziroma algoritem, ki sem ga uporabil za ocenjevanje segmentov, je bil minmaxdiff. Ta algoritem je podal ocene, razvidne v tabeli 1. Najslabše je ocenil obe smeri odseka Tomšičeva (obe 2.9), najbolje pa segment Velenjka (ocena 4.3).

Tabela 1: Ocene kvalitet načina ocenjevanja minmaxdiff

Odsek	**Ocena kvalitete**
Momax (1)	4.2
Momax (2)	4.2
Jezero	4.0
Tomsiceva (1)	2.7
Tomsiceva (2)	2.9
Sončni park (1)	2.9
Sončni park (2)	3.4
Velenjka	4.3
Gorenje	4.1

4.2.2 NAČIN BUMPCOUNT

Drugi uporabljen način je bumpcount. Pri tem načinu je vredno omeniti parameter, *DIFF_THRESHOLD*, ki določa, kaj bo algoritem štel kot nepopolnost v površini. Z njim nastavljamo minimalno vrednost razlike med dvema vrednostima, da se bo ta štela kot nepopolnost.

Da sem poiskal najbolj primerno vrednost tega parametra, sem poskusil več vrednosti: 0.1, 1, 2, 2.5, 3 in 5. Od teh sem pri vrednostih 0.1 in 5 takoj opazil, da so ocene vseh segmentov zelo podobne (bile so v razponu 1 pri oceni od 1 do 5), kar je pomenilo, da se preveč značilnosti šteje kot nepopolnosti. Od vseh teh vrednosti je vrednost 2 vrnila najbolj raznolike rezultate, zato sem uporabil v naslednjih rezultatih (tabeli 2) vrednost 2.

Ta algoritem je najslabše ocenil odsek Tomšičeva v obe smeri (oceni 2.4 in 2.6), kot najboljšega pa odsek Velenjka (4.6). Takoj za njim je bil odsek Momax z ocenama 4.4 in 4.3.

Tabela 2: Ocene kvalitet načina ocenjevanja bumpcount

Odsek	**Ocena kvalitete**
Momax (1)	4.4
Momax (2)	4.3
Jezero	4.1
Tomšičeva (1)	2.6
Tomšičeva (2)	2.4
Sončni park (1)	2.7
Sončni park (2)	3.2
Velenjka	4.6
Gorenje	4.1

4.2.3 NAČIN STD

Naslednji način je način std. Ta je prav tako kot prva dva najslabše ocenil segment Tomšičeva (2.3), najboljše pa odseka Velenjka in Momax (4.3). Ocene so vidne v tabeli 3.

Tabela 3: Ocene kvalitet načina ocenjevanja std

Odsek	**Ocena kvalitete**
Momax (1)	4.3
Momax (2)	4.1
Jezero	3.9
Tomšičeva (1)	2.3
Tomšičeva (2)	2.7
Sončni park (1)	2.5
Sončni park (2)	3.0
Velenjka	4.3
Gorenje	4.0

4.2.4 NAČIN MEAN

Pri načinu mean (tabela 4) opazimo, da so vse vrednosti med 2.5 in 2.8, torej ni bistvene raznolikosti.

Tabela 4: Ocene kvalitet načina ocenjevanja mean

Odsek	**Ocena kvalitete**
Momax (1)	2.7
Momax (2)	2.6
Jezero	2.7
Tomšičeva (1)	2.5
Tomšičeva (2)	2.8
Sončni park (1)	2.7
Sončni park (2)	2.4
Velenjka	2.7
Gorenje	2.8

4.2.5 NAČIN MEANSTD

Način meanstd je kombinacija načinov std in mean. Dejanski rezultati so zaradi tega podobni načinu std, saj, kot je prikazano pri načinu mean, ta ni imel velike variacije pri ocenah. Najbolje je ocenil odsek Velenjka, najslabše pa odsek Tomšičeva. Ocene tega načina so vidne v tabeli 5.

Tabela 5: Ocene kvalitet načina ocenjevanja meanstd

Odsek	**Ocena kvalitete**
Momax (1)	4.1
Momax (2)	3.9
Jezero	3.8
Tomšičeva (1)	2.2
Tomšičeva (2)	2.6
Sončni park (1)	2.4
Sončni park (2)	2.8
Velenjka	4.2
Gorenje	3.9

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

Iz rezultatov ankete lahko izračunamo povprečno oceno posameznih odsekov, ki so bili vključeni v vanjo. S tem dobimo naslednje številke (zaokrožene na 1 decimalno mesto):

- Momax: 3.9
- Jezero: 3.7
- Tomšičeva: 3.0
- Sončni park: 3.9
- Velenjka: 4.1
- Gorenje: 3.4

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, združene v tabelo, so razvidne v tabeli 6 (kjer sta pri senzorjih ocenjeni 2 smeri, je uporabljena povprečna vrednost ocen). Vse vrednosti v tabeli so zaokrožene na 1 decimalno mesto.

Tabela 6: Vse ocene segmentov

Način ocenjevanja/Odsek	**1**	**2**	**3**	**4**	**5**	**6**
Anketa	4.2	4.0	2.8	3.2	4.3	4.1
minmaxdiff	4.2	4.0	2.8	3.2	4.3	4.1
bumpcount	4.4	4.1	2.5	2.5	4.6	4.1
std	4.2	3.9	2.5	2.8	4.3	4.0
mean	2.7	2.7	2.5	2.6	2.7	2.8
meanstd	4.0	3.8	2.4	2.6	4.2	3.9

V tabeli 7, kjer "Povprečje"pomeni povprečni odklon od rezultatov ankete za ta način ocenjevanja, vidimo, da je najbližje rezultatom ankete, s povprečnim odklonom 0.4, algoritem ocenjevanja minmaxdiff, za njim je način meanstd z odklonom 0.45, takoj za njim način std z odklonom 0.48, sledita jim pa še "najslabša"načina, in sicer bumpcount z odklonom 0.67 in mean z odklonom 1. V tej tabeli so vse vrednosti razen povprečja zaokrožene na 1 decimalno mesto, povprečje pa je zaokroženo na 2.

Razlog, da je mean najslabši način, je jasen že iz njegove narave in podatkov v zgornji tabeli, saj zajema le povprečje vrednosti, zato so vse ocene blizu vrednosti 2.5 (povprečje vrednosti vseh možnih ocen). Ta način torej dejansko ne ocenjuje kvalitete odsekov.

Od načinov, ki se bolj približajo rezultatom ankete, je najboljši torej način minmaxdiff, ki v bistvu gleda največji tresljaj v posameznem segmentu. Očitno je torej ta način ocenjevanja najbolj podoben načinu, ki ga uporabljajo ljudje, ko ocenjujejo nek segment kolesarske površine. Ta način je uspel oceniti posamezne segmente z največjim odklonom od rezultatov ankete 0.7.

Tabela 7: Odklon ocen algoritmov z rezultati ankete

Način ocenjevanja/Odsek	**1**	**2**	**3**	**4**	**5**	**6**	**Povprečje**
Anketa	0.0	0.0	0.0	0.0	0.0	0.0	0.0
minmaxdiff	0.3	0.3	0.2	0.7	0.2	0.7	0.4
bumpcount	0.5	0.4	0.5	1.4	0.5	0.7	0.67
std	0.3	0.2	0.5	1.1	0.2	0.6	0.48
mean	1.2	1.0	0.5	1.3	1.4	0.6	1.0
meanstd	0.1	0.1	0.6	1.3	0.1	0.5	0.45

Zaradi zgoraj navedenih dejstev sem hipotezo *"Izmerjena kvaliteta posameznih odsekov kolesarske površine se ujema z mnenjem kolesarjev o kvaliteti tega odseka."* potrdil, saj se je eden od načinov (minmaxdiff) uspel približati rezultatom ankete z odklonom, manjšim od 0.7; to pokaže očitno ujemanje med ocenami anketirancev in oceno algoritma.

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 segment relativno slabe kvalitete in procentom anketirancev, ki bi se temu segmentu 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. To seveda nima bistvenega vpliva na končne rezultate, je pa vredno omembe.

V prihodnosti bi rad bolj podrobno raziskal, kako različne značilnosti kolesa (tlak v zračnicah, tip kolesa ipd.) vplivajo na meritve. Prav tako bi rad izboljšal program 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 segmentov 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 segment, 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 segmentov, 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. URL https://www.amzs.si/motorevija/mobilnost/na-poti/2021-06-02-mobilnost-razcvet-kolesarjenja-v-sloveniji.

[2] 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. 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] 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. 08 2015. ISBN 978-3-319-21944-8.

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

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

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

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

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

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

[22] Volodymyr Agafonkin. Leaflet - an open-source javascript library for interactive maps. 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.

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

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

import matplotlib.pyplot as plt

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

class PathSegment:

    def __init__(self):
        self.sensors_data = {}
        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])

    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") -> float:

    if algo == "speed":
        # Hitrost
        dist = self.get_segment_distance()
        length = self.get_segment_time_length()
        speed = 0 if np.isnan(dist / length) else dist / length
        return speed
    elif algo == "minmaxdiff":
        # Razlika med minimumom in maksimumom na pospeškomeru
        accel_data = self.get_sensor_data(SensorType.ACCELEROMETER)
        if len(accel_data) == 0:
            return 0
        accel_min = np.min(accel_data, axis=0)
        accel_max = np.max(accel_data, axis=0)
        return np.linalg.norm(accel_max - accel_min)
```

```
elif algo == "bumpcount":
    # Število sprememb pospeška
    DIFF_THRESHOLD = 2.5

    accel_data = self.get_sensor_data(SensorType.ACCELEROMETER)
    if len(accel_data) == 0:
        return 0
    accel_diff = np.diff(accel_data, axis=0)
    accel_diff = np.linalg.norm(accel_diff, axis=1)

    bumps = np.sum(accel_diff > DIFF_THRESHOLD)

    print(f"Bumps: {bumps}")

    dist = self.get_segment_distance()

    if dist == 0:
        return 0

    return bumps / dist
elif algo == "meanstd":
    # Izračun večih metrik kakovosti
    accelerometer_data = self.get_sensor_data(
        SensorType.ACCELEROMETER
    )

    # Preveri, ali obstajajo podatki pospeškomera
    if (
        accelerometer_data is None
        or accelerometer_data.size == 0
    ):
        return 0

    # Izračunaj značilnosti za vsako os
    mean_vals = np.mean(accelerometer_data[:, 1:], axis=0)
    std_devs = np.std(accelerometer_data[:, 1:], axis=0)

    # Uporabi uteži za vsako značilnost
    weighted_mean = np.dot(
        mean_vals,
        FEATURE_WEIGHTS.get('mean', 1)
    )
    weighted_std = np.dot(
        std_devs,
        FEATURE_WEIGHTS.get('std', 1)
    )
```

```
        # Združi utežene značilnosti
        road_quality_metric = np.sum(
            [weighted_mean, weighted_std]
        )

        return road_quality_metric
    elif algo == "std":
        # Standardni odklon pospeškomera
        accelerometer_data = self.get_sensor_data(
            SensorType.ACCELEROMETER
        )
        # Preveri, ali obstajajo podatki pospeškomera
        if (
            accelerometer_data is None
            or accelerometer_data.size == 0
        ):
            return 0

        # Izračunaj standardni odklon za vsako os
        std_devs = np.std(accelerometer_data[:, 1:], axis=0)

        # Združi standardne odklone
        road_quality_metric = np.sum(std_devs)

        return road_quality_metric
    elif algo == "mean":
        # Povprečje pospeškomera
        accelerometer_data = self.get_sensor_data(
            SensorType.ACCELEROMETER
        )

        # Preveri, ali obstajajo podatki pospeškomera
        if (
            accelerometer_data is None
            or accelerometer_data.size == 0
        ):
            return 0

        # Izračunaj povprečje za vsako os
        mean_vals = np.mean(accelerometer_data[:, 1:], axis=0)

        # Združi povprečja
        road_quality_metric = np.sum(mean_vals)
        return road_quality_metric
```