TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika (2476) õppekava

Eduard Rudi

Inimlik kiirus kurvides

Bakalaureusetöö (9 EAP)

Juhendaja: Tambet Matiisen

Tartu 2021

Inimlik kiirus kurvides

Lühikokkuvõte:

Bakalaureusetöö eesmärk oli leida võrrand, mille sisendiks on joone kõverus ja tagastab teelõigu kiiruse, mis on konstrueeritud inimese juhtimisandmete põhjal. Töös antakse lühiülevaade tänapäeva isejuhtimisüsteemi tarkvara ehitamise lähenemistest. Samuti tutvustatakse varasemaid uuringuid. Töös kirjeldatakse põhjalikult andmete töötlemisest ning mida nendega tehti. Kirjeldatakse detailselt lahti, kuidas võrrand implementeeriti Autoware'i ja kuidas algoritm arvutab kiirused mingi teekonnale. Lõpuks antakse ülevaade testimise tulemustest.

Võtmesõnad:

Autonoomne juhtimine, kiiruse arvutamine

CERC: 170 Arvutiteadus, arvanalüüs, süsteemid, kontroll

Human-like speed in curves

Abstract:

The purpose of this bachelor's thesis was to find equation that can be given a curvature and returns speed which is constructed on human driving data. A brief overview on approaches to build an autonomous system is given. Also, this thesis gives an overview of previous research. The preprocessing of data and further processes are thoroughly described. The implementation of the equation in Autoware software and how the algorithm calculates speed for a route is shown in detail. Finally, the testing results are discussed.

Keyword:

Autonomous driving, calculating speed

CERC: 170 Computer science, numerical analysis, systems, control

Sisukord

Sissejuhatus

Meie maailm areneb väga kiiresti, sealhulgas ka tehnoloogia. Tehnoloogia areng on mõjutanud peaaegu kõiki valdkondi, kaasa arvatud autotööstus, eriti viimase kümne aasta jooksul. Tänapäeval on autodes väga keerulised süsteemid, mis tagavad liikluses ohutuse, aga isejuhtimissüsteemid on osutunud populaarsemaks. Üks tuntumaid isejuhtimise süsteeme on Tesla Autopilot [17], mis on äratanud ka laiemat tähelepanu.

Süsteemid, mis on paljudel autovalmistajadel – nagu nt Mercedes-Benz, BMW, Volvo, Audi, Cadillac ja kindlasti ei tohi unustada Teslat –, ei ole täielikult isejuhtivad süsteemid, vaid pigem juhti abistavad süsteemid, sest nende süsteemidele ei saa anda asukohta, et sõiduk viiks inimest sinna, vaid nt hoiab kiirust, roolib teatud olukordades inimese eest ja hoiab automaatselt eespool sõitva autoga vahemaad. Suurem osa neist süsteemidest põhinevad suurtel algoritmidel ja väiksem osa tehisnärvivõrkudel. Waymo pakub seni ainukesena ilma juhita taksoteenust Phoenixi eeslinnas USAs [18]. Kuid, Tartu Ülikooli isejuhtivate sõidukite labor ehitab ka isejuhtimissüsteemi ja nende peamine eesmärk on uurida ja arendada tehnoloogiate tarkvara [16]. Teema on oluline, sest isejuhtimisautod on tulevik, kuhu liigutakse. Euroopa eSafety töögrupp on välja selgitanud, et 90%–95% liiklusõnnetustest on mingil määral seotud inimese veaga ning kolm neljandikku juhtumitest toimub ainult inimese vea tõttu [4]. Seega võib eeldada, et isejuhtivad autod teeksid liikumise autoga ohutumaks ja siinne lõputöö püüab sellele kaasa aidata.

Lõputöö keskendub võrrandi leidmisele, millele saab anda arvu, mida nimetatakse kõveruseks (ingl *curvature*) ja seda saadakse kui üks jagada ringi raadiusega, ning tagastab teoreetilise maksimumkiiruse, et läbida mingi kurv. Samuti implementeeritakse leitud valem Tartu Ülikooli Autoware'i repositooriumi klooni ja katsetatakse pärismaailmas, kuidas lahendus toimib. Autoware on avatud lähtekoodiga tarkvara isejuhtivate sõidukite jaoks ja mis on ehitatud ROS raamistiku peal [1]. ROS ehk Robot Operating System on operatsioonisüsteem, mille abil saab ehitada robotide tarkvara [13]. Võrrand peab kõige paremini iseloomustama graafikut, mille ühel teljel on auto kiirus ja teisel teljel kõverus. Kiirus on teoreetiline, sest pärismaailmas on palju muid segavaid faktoreid nagu valgusfoorid, teised auto, jalakäijad jt. Võrrand on konstrueeritud inimese juhtimisandmete põhjal. Oluline põhjus selleks on see, et inimestel on hea intuitsioon ja kogemus ning sõitja näeb, mis olukord on ja valib selle järgi sobiva kiiruse kurvi läbimiseks [5].

Selleks et lõputöö eesmärgid saavutada, on selle töö autor töötanud Tartu Ülikooli isejuhtivate sõidukite laboris, kust on saadud andmed ja juhiseid. Andmed on salvestatud labori meeskonna poolt. Nad kasutasid autot, millele on sobitatud andurid ja mille abil saab salvestada auto kiirust, koordinaate ja telgede muutusi.

Lõputöö on jagatud kuute peatükki. Peale sissejuhatust, esimeses peatükis kirjutatakse varasematest uuringutest. Teises peatükis selgitatakse isejuhtimissüsteemi ehitamise viisidest: mis on modulaarne lähenemine ja täielikult närvivõrkudel põhinev lähenemine. Kolmandas peatükis kirjeldatakse esmalt, milliseid andmeid kasutati, kuidas nad välja näevad ja millised eeltöötlused tehti. Seejärel kirjeldatakse, milliseid operatsioone tehti, et leida kolme punkti raadius, maksimum väärtus mingis vahemikus ja kuidas saadi võrrand. Kirjeldatakse ka, kuidas implementeeriti võrrand ja punktide valimise Autoware'i. Peatüki lõpus kirjeldatakse tulemustest ja mida tehti tulemustega. Peale seda tuleb neljas peatükk, mis keskendub tulemustele ja erinevatele võrdlustele, ning lõpuks on kokkuvõte.

1. Isejuhtimissüsteemi tarkvara ehitamise lähenemised

Kogu siinne peatükk põhineb Tambet Matiiseni interneti materjalil „Isejuhtivad autod“ [11].

Tänapäeval on kaks viisi, kuidas ehitada isejuhtimissüsteemi tarkvara:

1. modulaarne lähenemine (ingl *modular approach*), kus suur ülesanne on jaotatud väikstemadeks osadeks,
2. täielikult närvivõrkudel põhinev lähenemine (ingl *end-to-end approach*), kus auto käitmisloogikat juhib üks suur tehisnärvivõrk.

Suurem osa Tartu Ülikooli isejuhtivate sõidukite labori inimestest on seotud modulaarse lähenemisega, sest seda on arendatud kõige rohkem ja lühiperspektiivis tundub see lähenemine edukaim. Seetõttu on ka selle töö autor seotud modulaarse lähenemisega ning lõppvalem peab olema implementeeritud planeerimismoodulisse, mille ülesanne on anda juhtimismoodulile soovitatud kiiruse. Kuid on vaja mainida, et mõned inimesed laboris tegelevad ka täielikult närvivõrkudel põhineva lähenemisega.

1.1. Modulaarne lähenemine

Modulaarses lähenemises on tervik ülesanne jaotatud nelja moodulisse, millel on kindel sisend ja väljund:

1. Tajumoodul töötleb andurite poolt saadud andmeid ning saadab neid järgmistesse moodulitesse lihtsamal kujul.
2. Positsioneerimise moodul määrab auto asukoha sentimeetri täpsusega.
3. Planeerimise moodul on jaotatud kolmeks alammooduliks:
    a. teekonna planeerija, mis leiab praeguse asukohast antud asukohta lühima teekonna,
    b. manöövrite planeerija, mis otsustab, millist järgmist manöövrit teha,
    c. trajektoori planeerija, mis leiab trajektoori, et läbida mingit takistust või teha mingi manööver.
4. Juhtimismoodul, mille ülesanne on teisendada trajektoor juhtimis käskudeks.

Modulaarse lähenemise juures omab suurt tähtsust kõrglahutusega kaart (ingl *high-definition map, HD map*). Nagu nimigi ütleb, on see kaart väga detailne ning sisaldab elemente nagu

valgusfoorid, ülekäigurajad, liiklusmärgid, kiirusepiirangud, sõiduradade suunad ja kui palju neid on jne.

1.2. Täielikult närvivõrkudel põhinev lähenemine

Täielikult närvivõrkudel põhinevat lähenemist on kirjeldada palju lihtsam. Närvivõrku sisestatakse auto andurite andmed ja väljastus on rooli, gaasi või piduri positsioon. Teisisõnu, tehisintellekti proovitakse panna imiteerima inimese juhtimist.

Kuid kõik ei ole nii lihtne, sest närvivõrguga kaasnevad järgmised probleemid:

1. et õpetada edukalt ja korrektselt tehisintellekti, on vaja palju andmeid, sest on vaja näidata erinevaid situatsioone,
2. kui inimene sõidab halvasti, siis tehisintellekt hakkab ka imiteerima halba sõiduviisi,
3. kui tehisintellekt satub olukorda, mida ta ei ole enne näinud, ei saa ennustada, mida inimene võiks seal teha.

Sellest hoolimata, kasutatakse tänapäeval närvevõrke ikkagi ning suurt tähtsust omab stiimulõpe, kus närvivõrk saab punkte juurde õige trajektoori püsimisel ning kui inimene peab sekkuma, siis tehisintellekt saab aru, et tema trajektoor oli halb ja jätab selle meelde järgmiseks korraks.

2. Varasemad uuringud

Kuigi isejuhtivate sõidukitega hakati katsetama juba 1920-ndate alguses, olid need sõidukid juhitud raadiosignaalide abil, seega olid need pigem juhita sõidukid või teisisõnu puldiautod [9]. Esimesed päriselt isesõitvad autod tulid 1980-ndatel, mis kasutasid kaameraid radari ja satelliitnavigatsioonisüsteemi asemel, et liikuda autonoomselt [9]. Kuigi on juba peaaegu 40 aastat esimestest isejuhtivatest sõidukitest möödas, hakkavad isejuhtimissüsteemid olema Euroopa turul alates 2020. aastast [14]. Sellest hoolimata on isejuhtivate sõidukite teema saanud järjest populaarsemaks.

Tehisnärvivõrgud on selliste ülesannete, nagu isejuhtiva süsteemi, jaoks väga populaarsed, kuna nad on robustsed ja oskavad üldistada, sest saavad edukalt müraseid andmeid käsitleda [8]. Mürased andmed sobivad hästi isejuhtiva auto mudeliks, sest ka pärismaailmas on tohutu hulk segavaid faktoreid ja seetõttu ongi palju teadustöid keskendunud närvivõrkude metoodikale. Järgnevalt on toodud välja, kuidas õpetati närvivõrku erinevate inimeste sõidu andmetest, ning teises alajaotuses, kuidas leiti kiirus kõveruse sõltumisest.

2.1. Kiiruse leidmine närvivõrkudega

Käesolev alajaotus põhineb peamiselt Geng, X. jt artiklil „Human-Driver Speed Profile Modeling for Autonomous Vehicle's“ [5], kus nad on nentinud, et kuigi on palju teadustöid tehtud kurvide kiiruse algoritmi leidmiseks, ei ole arvestanud suur hulk töödest tee ja ümbruskonna olukorda. Autorid tahavad teha mudelit, mis on n-ö tavaliste juhtide keskmine, seetõttu kasutasid nad SHRP 2 NDS andmebaasi, kus on salvestatud üle 3400 Ameerika Ühendriikide osaleja autosõidud. Sellega loodeti saada inimlikumat mudelit.

Andmete eeltöötluseks olid autorid algul silunud algandmeid ning seejärel arvutasid nad välja auto käitumistüübid. Vasak või parem pööre, konstantse kiirusega sõitmine ilma ristmiku ja sissesõidutee tee peal, tavalise tee peal sõitmine ilma eespool oleva autota, raja vahetamine – need kõik olid käitumistüübid, mida autorid defineerisid. Seejärel käidi läbi iga video algus ja kontrolliti, kas käitumine vastab tõele ning salvestasid ilmastikutingimused ja teekatte olud iga video kohta. Tulemuseks said nad iga käitumistüübi kohta 260 sündmust. Saadud andmed sisestati tehisnärvivõrkudesse, mis õpivad andmete sisestatud ja oodatud väljundi suhtest. Närvivõrk oli kolmekihiline koos 20 peidetud neuroniga, 200 näidet oli võetud treenimiseks ning

60 testimiseks. Parameetrid, mille peal mudelit õpetati, olid kurvi kõik punktide kõverused, kurvi tippkõverus, kurvi lõppkiirus ja suhteline pikivahe algusest lõppseisundini.

Tulemuseks saadi, et mida väiksem on ennustussamm, seda parem on mudeli täpsus. Teadlased tõid eraldi välja, et mudel ei saa täpselt ennustada inimese käitumist, aga mudeleid saab ikkagi kasutada teatud olukorras. Eriti sobiks mudel linnades, kuna seal on kiirused väikesed ja seetõttu saab väikese ennustussammuga hakkama.

2.2. Kiiruse leidmine valemi põhjal

Järgnev kirjeldus põhineb Serna ja Ruicheki artiklil „Dynamic Speed Adaptation for Path Tracking Based on Curvature Information and Speed Limits“ [15].

Serna ja Ruichek tegid süsteemi, kus kasutati kiiruse soovitamiseks võrrandit. Nende süsteemi nimetatakse DNS ehk *Dynamic Speed Adaptation*, mille ülesanne on kurvide jaoks aeglustada ja ohutult kurvi läbida.

Teadlased lähenesid soovitava kiiruse leidmiseks kasutades füüsikat. Nad nendivad, et kurvi kiirus sõltub tsentripetaal- ja tsentrifugaaljõust, mille tõttu kasutasid nad kõverust, hõõrdetegurit ja tee kaldenurka. Nad said järgmise valemi:

$$v = \sqrt{\frac{(e+\mu)g}{k}}, \tag{1}$$

kus $e = tan\,\theta$, kus θ on tee kaldenurk, μ on rehvide ja tee hõõrdetegur, g on Maa raskusjõud, k on kõverus ja v on kiirus. Kurvide leidmiseks arvutasid teadlased algul kolme järjestikuse andmepunkti kursid (ingl *bearing*), ning kõik järjestikused punktid, millel oli kurssi üle 1.25°. See tähendas, et see oli üks kurv.

Tulemusi polnud välja toodud, kuna nende töö põhines süsteemi ehitamisele ja mitte sellele, kui hästi nende valem töötas. Sellest hoolimata kirjutavad teadlased, et kasutades nende süsteemi, vähenes veamäär.

3. Metoodika

Andmete töötlemiseks on kasutatud Pythoni programmeerimiskeelt ning selle käivitamiseks Jupyter Notebooki. Andmete töötlemiseks oli vajalik ka kasutada järgmisi Pythoni teeke: Pandas, et lugeda andmed sisse, ning Numpy, et teha matemaatilisi arvutusi.

Töö autor kasutab sõna „skip“ kolme punkti valimisel, kui on massiiv või list koordinaatpunktidest ja ollakse indeksil i, siis „skip“ näitab, mitu punkti on esimesel teisega vahet ja teise kolmandaga, nt kui „skip“ on 3, siis jäetakse kaks punkti vahele ja võetakse kolmas punkt, ehk indeks – 3 ja indeks + 3.

3.1. Andmete eeltöötlus

Andmed on saadud Tartu Ülikooli isejuhtivate sõidukite laborist. Andmed koguti isejuhtivate sõidukite labori autoga Tartu linnas, mille käigus salvestati CSV formaadis auto koordinaadid ning kiirus. Seega, andmetes olid järgmised veerud (vt tabel 1): x, y ja z koordinaadid, „velocity” ehk kiirus, x_lest97 ja y_lest97, mis on x ja y koordinaadid teisendatud L-EST97 koordinaatsüsteemi [6] ning mõned muud veerud, mis selle töö eesmärki silmas pidades ei omanud tähtsust.

Tabel 1. Kümme esimest rida ühest andmefailidest

	A	B	C	D	E	F	G	H
1	x	y	z	yaw	velocity	change_fl	x_lest97	y_lest97
2	9958.208	11277.13	52.1319	1.1181	0	0	659958.2	6476277
3	9958.637	11278.04	52.1421	1.1365	4.7671	0	659958.6	6476278
4	9959.048	11278.96	52.134	1.1487	6.902	0	659959	6476279
5	9959.458	11279.9	52.1258	1.156	8.502	0	659959.5	6476280
6	9959.858	11280.84	52.1191	1.1603	9.7885	0	659959.9	6476281
7	9960.266	11281.79	52.1091	1.1612	10.921	0	659960.3	6476282
8	9960.663	11282.72	52.103	1.1608	11.8962	0	659960.7	6476283
9	9961.067	11283.67	52.0946	1.1592	12.7588	0	659961.1	6476284
10	9961.471	11284.61	52.0867	1.1571	13.5319	0	659961.5	6476285

Iga rida on salvestatud mingi kindla intervalli tagant ning seega oli kahte tüüpi faile:

1. iga rida oli salvestatud iga meetri tagant ja kiirus oli km/h,
2. iga rida oli salvestatud 0.01 meetri tagant ja kiirus oli m/s.

Kokku oli kuus faili, millest neli oli esimest tüüpi ning kaks teist tüüpi ning milles kokku oli umbes 40 000 rida. Need failid, kus oli iga rida salvestatud iga 0.01m tagant ja kiirusühik oli m/s, oli vaja teisendada ühe meetri kaugusteks ja m/s oli vaja teisendada km/h-ks.

3.1.1. Kiiruse teisendamine

Kiiruse teisendamiseks (vt joonis 1), täpsemalt m/s-st km/h-ni, loeti algul andmefail sisse, kasutades Pandast, mis teeb DataFrame'id. DataFrame on kahemõõtmeline andmestruktuur, kus on erinevad veerud [12] ja millest võeti kiiruse tulp. Seejärel itereeriti üle kiiruste veeru, kus iga kiiruse jaoks kasutati

$$v_{km/h} = v_{m/s} * 18 / 5 \quad (2)$$

valemit ja saadud kiirus lisati listi. Lõpuks lisati antud kiiruste list Pandase DataFrame'i ning salvestati faili.

```
data = pandas.read_csv(fileName, delimiter=",")
speedColumn = data['speed']
velocity = []
for speed in speedColumn:
      velocity.append(speed * 18 / 5)
data['velocity'] = velocity
data.to_csv(fileName)
```

Joonis 1. Kiiruse teisendamise koodijupp

Peab tähele panema, et tulbas nimega „speed" on kiirusühik m/s ning tulbas nimega „velocity" on kiirusühik km/h.

3.1.2. Ühe meetrite kaugusteks teisendamine

Ühe sentimeetri ühe meetri teisendamiseks (vt joonis 2) kasutati järgnevat loogikat. Algul leiti kõikide failide, kus olid kaugused üks meeter, kahe punkti x ja y koordinaadi kaugused, kasutades eukleidilist kaugust. Seejärel liideti x ja y kaugused kokku ning lisati ajutisse listi. Lõpus kasutati Numpy teeki, et arvutada keskmine ja lisati keskmine lõpp listi. Kui kõikide failide punktide keskmised on leitud, siis võetakse keskmiste keskmine.

```
means = []
for data in files:
    tempMeans = [] #Ajutine list
    for i in range(1, len(data)):
        x = (data.iloc[i]['x_lest97'] - data.iloc[i - 1]['x_lest97']) ** 2
        y = (data.iloc[i]['y_lest97'] - data.iloc[i - 1]['y_lest97']) ** 2
        tempMeans.append(x + y)
    means.append(numpy.mean(tempMeans))
mean = numpy.mean(means)
```

Joonis 2. Andmete punktide keskmine arvutamine

Nüüd, et teisendada 0.01m kaugustega failid ühe meetri kaugusteks, kasutati sellist loogikat (vt joonis 3), et esimese punkti ja järgmise punkti x ja y koordinaatide summa peab olema suurem või võrdne keskmisest. Kui see punkt on leitud, salvestatakse punkt ja tehakse samat loogikat ainult võttes nüüd teist punkti ja otsida järgmist punkti, mille nende kahe punkti x ja y koordinaatide summa oleks suurem või võrdne keskmisest, ning seda korratakse faili lõpuni.

```
data = pandas.read_csv(fileName, delimiter=",")
newDf = pandas.DataFrame()
current = None
for i, row in data.iterrows():
    if current is None:
        current = row
        newDf = newDf.append(row, ignore_index=True)
    else:
        x = (row['x_lest97'] - current['x_lest97']) ** 2
        y = (row['y_lest97'] - current['y_lest97']) ** 2
        if x + y >= mean:
            newDf = newDf.append(row, ignore_index=True)
            current = row
newDf.to_csv(fileName)
```

Joonis 3. Punktide leidmine, mis on ühe meetri kaugusel

Nüüd on failid teisendatud ja saab minna järgmisele etapile.

3.2. Kiiruse ja kõveruse suhte valem

Kui andmed on teisendatud, saab alustada valemi leidmisega, mis iseloomustab kõige paremini kõverust ja kiiruse suhet. Selleks arvutatakse lõigu kõverust ja selle kiiruse (vt joonis 4) ning

loogika on, et itereeriti üle andmete list, mille iga andmefail itereeriti üle ridade ning võeti eelmise rea, praeguse rea ja järgmise rea x ja y koordinaadid ning leiti nende punktidega kõverus. Kõveruse saamine toimus joonise 5 järgi, kus kasutati Heroni valemit kolmnurga pindala arvutamiseks, mida kasutati. Lisatud on ka tingimuslause, kus kontrolliti, kas antud kõveruse väärtus on väiksem või võrdne ühest ja kas kiirus on suurem 3km/h. Kõveruse piirang on selleks, sest peale väärtust üks on ainult üksikuid punkte ja väärtused peaaegu ei muutu, mis kõiguvad 0km/h ja 5km/h vahel, ning kiiruse piirang on selleks, et eemaldada auto seismised.

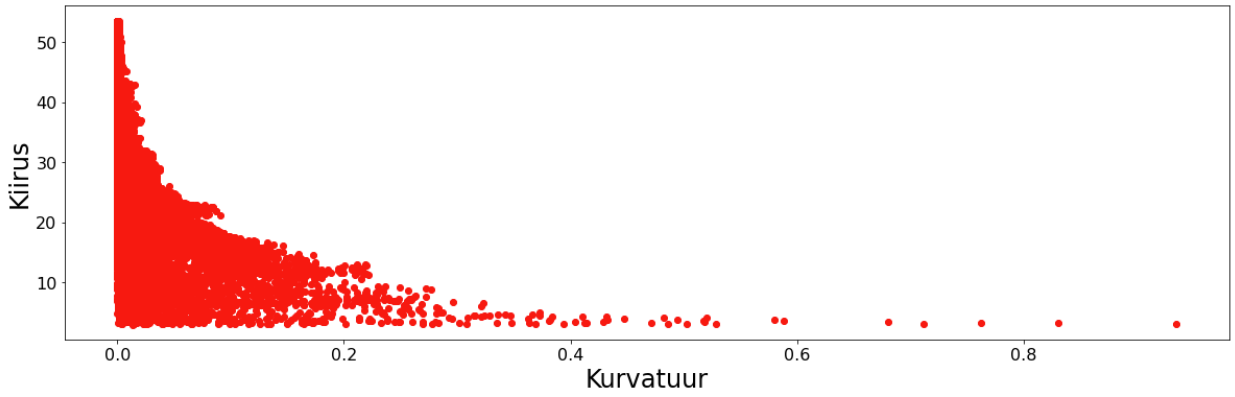
```
def getCurvatureAndSpeed(data):
    curvatureList = []
    speedList = []
    x_coord_tpye = "x_lest97"
    y_coord_type = "y_lest97"
    for d in data:
        dataLen = len(d)
        for index, row in d.iterrows():
            if index - 1 >= 0 and index + 1 < dataLen:
                row1 =
(d.iloc(0)[index - 1][x_coord_tpye] , d.iloc(0)[index - 1][y_coord_type])
                row2 =
(d.iloc(0)[index][x_coord_tpye], d.iloc(0)[index][y_coord_type])
                row3 =
(d.iloc(0)[index + 1][x_coord_tpye], d.iloc(0)[index + 1][y_coord_type])
                rows = [row1, row2, row3]
                curvature = findCurvature(rows)
                if row['velocity'] > 3 and curvature <= 1:
                    curvatureList.append(curvature)
                    speedList.append(row['velocity'])
    return curvatureList, speedList
```

Joonis 4. Kõveruse ja kiiruse leidmine

```
def findCurvature(points_utm): # points_utm is a 3-by-
2 array, containing the easting and northing coordinates of 3 points
    a = numpy.hypot(points_utm[0][0] - points_utm[1][0], points_utm[0][1] - point
s_utm[1][1])
    b = numpy.hypot(points_utm[1][0] - points_utm[2][0], points_utm[1][1] - point
s_utm[2][1])
    c = numpy.hypot(points_utm[2][0] - points_utm[0][0], points_utm[2][1] - point
s_utm[0][1])
    # Compute inverse radius of circle using surface of triangle (for which Heron
's formula is used)
    # Heron's formula for triangle's surface
    k = numpy.sqrt((a+(b+c))*(c-(a-b))*(c+(a-b))*(a+(b-c))) / 4
    den = a*b*c  # Denumerator; make sure there is no division by zero.
    if den == 0.0:  # Very unlikely, but just to be sure
        curvature = 0.0
    else:
        curvature = 4*k / den
    return curvature
```

Joonis 5. Kõveruse arvutamine kolme punkti järgi [3]

Kui kõverused ja kiirused on kätte saadud, võib neid graafikusse panna, et saada ettekujutus olemasolevatest andmetest (vt joonis 6).



Joonis 6. Kõveruse ja kiiruse suhe graafik

Võib näha probleemi, et kui võtta kõveruseks 0.1, siis kiirus saab olla nii 10km/h kui ka 20km/h, seega töö autor ja juhendaja otsustasid, et võtavad maksimumkiiruse antud kõveruse jaoks (vt joonis 7). Selleks tehti sõnastik, kus võtmeks on kõverus ja väärtuseks kiirus, pärast sorteeriti see sõnastik ära. Järgmisena itereeriti üle saadud sõnastiku ning leiti suurim kiirus antud vahemikus,

selleks oli valitud 0.05. Lõpus jälle sorteeriti sõnastik ära ja pakiti sõnastik lahti uuteks kõveruse ja kiiruse listideks.

```
def getHighestSpeedInGap(curvatures, speeds):
    speedCurvDict = {}
    for i in range(len(curvatures)):
        if curvatures[i] not in speedCurvDict:
            speedCurvDict[curvatures[i]] = speeds[i]
        else:
            if speeds[i] > speedCurvDict[curvatures[i]]:
                speedCurvDict[curvatures[i]] = speeds[i]
    speedCurvDict = sorted(speedCurvDict.items(), key=lambda kv: kv[0])

    highestSpeed = -math.inf
    highestPair = None
    increment = 0.05
    incrementTotal = increment
    newSpeedCurvDict = {}
    for i, val in enumerate(speedCurvDict): # val[0] - kurvatuur, val[1] - kiirus
        if incrementTotal <= val[0]:
            if highestPair is not None:
                newSpeedCurvDict[best[0]] = best[1]
            highestPair = None
            highestSpeed = -math.inf
            incrementTotal += increment
        if val[1] > highestSpeed:
            highestSpeed = val[1]
            highestPair = val
        if i == len(speedCurvDict) - 1:
            if highestPair is not None:
                newSpeedCurvDict[highestPair [0]] = highestPair [1]
    newSpeedCurvDict = sorted(newSpeedCurvDict.items())
    newCurvatures, newSpeeds = zip(*newSpeedCurvDict)
    return newCurvatures, newSpeeds
```

Joonis 7. Suurima kiiruse leidmine mingi vahemikkus

Edasi sai alustada valemi otsimisega, mis iseloomustab kõige paremini kõveruse ja kiiruse joont. Selleks kasutati funktsiooni „Function Equation Finder“ veebilehel „dCode“, kus kasutati „Exponential Using Curve Fitting“ [2] ehk eksponentsiaalfunktsiooni kõvera sobitamist. Kasutati veebilehte „dCode“, sest on tasuta, lihtne kasutada ja sisendiks on Exceli tabel, ning valiti

eksponentsiaalfunktsioon, sest joon kiiresti langeb või tõuseb, mis on vaja, sest kiirused kiiresti langevad, nagu seda näha joonisel 6. Muud valikud olid lineaar-, ruut-, astme-, logaritm- ja siinusfunktsioon.

3.3. Tulemuste salvestamine

Simulatsiooni tulemused salvestati ROSi vahenditega, mis salvestavad BAG faili, kust on võimalik eksportida koordinaadid ja kiirused. Programmi RViz abil on võimalik simulatsioone visualiseerida. Tartu Ülikooli isejuhtivate sõidukite labori inimesed on teinud mitmeid simulatsioone kaarte (HD map'e), mida töö autor sai kasutada, et testida lahendust.

Päriselu testimises, kasutati Tartu Ülikooli ja Bolti isejuhtivat autot. Testimisrajaks oli Eesti Rahva Muuseumi parkla, mida võib täpsemalt näha joonisel 8.

Joonis 8. Testimis rada, ERMi parkla. Joonis: Edgar Sepp[1].

Sõitude salvestamiseks kasutati ROSi, millel on käsklus „rosbag", mis salvestab sensorite andmed.

3.4. Tulemuste mõõtmine

Autor kasutas kiirust, kõrvalekaldumime trajektoorist ja põikikiirendust, et mõõta tulemusi. Põikikiirenduse mõõtmiseks kasutati inertsiaalandurit. Tulemused oli vaja keskmistada, selleks

[1] Edgar Sepp on Tartu Ülikooli isejuhtivate sõidukite labori liige, kes aitas autori lahendust testida, ning joonis ei ole avalikult kättesaadav. Joonise on teinud Edgar Sepp.

oli kasutatud kahe sekundilist akent, kuna põikikiirenduse mõõtmis andur on väga tundlik ning juhuslik rooli rapsimine võib põhjustada suurimat väärtust, mis tegelikult ei tulenenud auto kiirusest kurvis. Kiiruse hindamiseks kasutati satelliitpositsioneerimise ja inertsiaalanduri kobinatsiooni ning kõralekaldumine trajektoorist hindamiseks kasutati „MPC“ ehk „model predictive control“, mis hindas, mitu meetrit on auto kõrvale kaldunud trajektoori joonest. „MPC“ kasutab süsteemi mudelit, et ennustada selle tulevast käitumist [10]. Peab mainima, et peatüki 4.4. tulemuste arvutamiseks olid ainult need andmepunktid, kus oli autonoomne režiim sisse lülitatud.

4. Tulemused

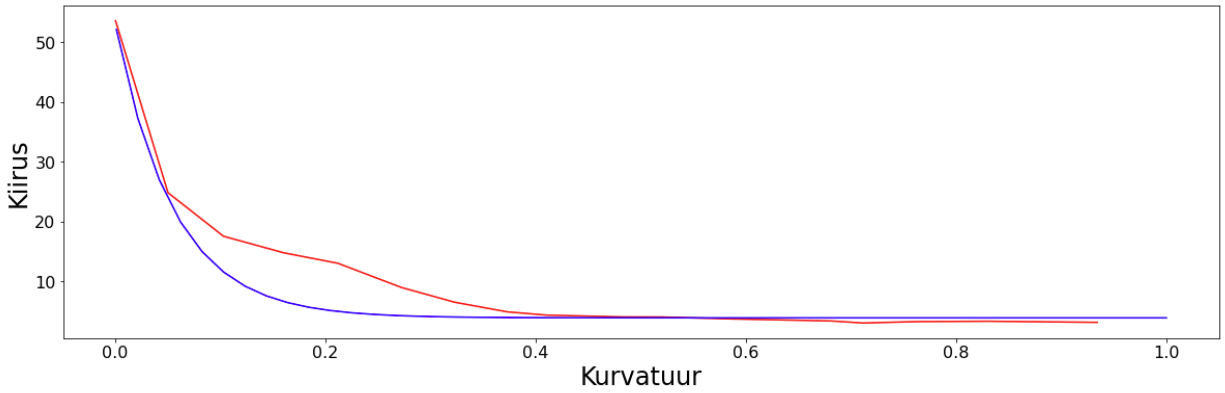
Selles peatükis tuuakse välja saadud võrrand ning tehaks võrdlusi, kõrvutatakse simulatsiooni tulemused ja päriselu tulemused.

4.1. Saadud võrrand

Veebilehel „dCode“ [2] oli saadud järgmine võrrand:

$$v = 3.91207 + 49.45\mathrm{e}^{-18.0329 \cdot k}, \tag{3}$$

kus v on kiirus ja k on kõverus. Tulemust on näha joonisel 9.

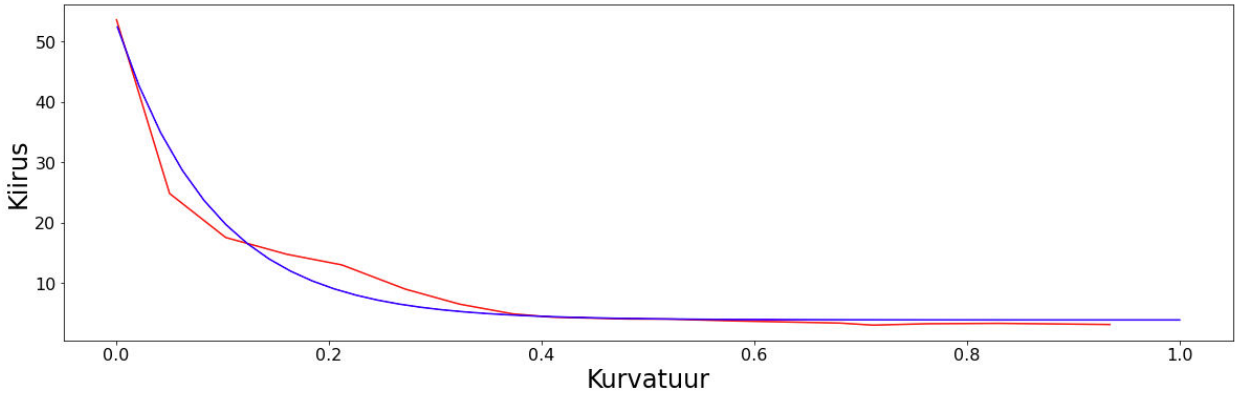


Joonis 9. Kõveruse ja kiiruse suhe, kus punane joon on saadud joonis 7 koodiga ja sinine joon (3) valemiga.

Kui (3) valemit muuta kujule

$$v = 3.91207 + 49.45\mathrm{e}^{-11 \cdot k}, \tag{4}$$

siis tulemust on näha joonisel 10. On näha, et sinine joon vastab andmetele palju rohkem.

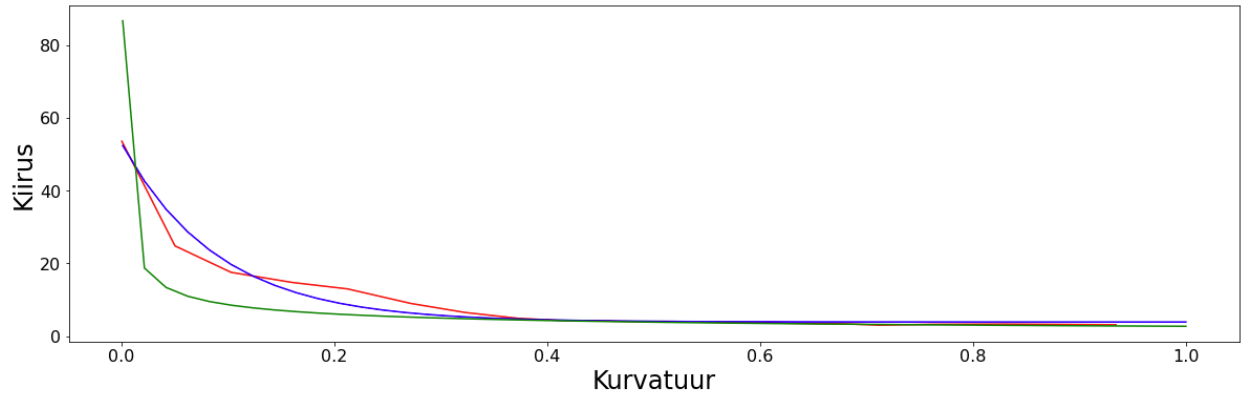


Joonis 10. Kõveruse ja kiiruse suhe, kus punane joon on saadud joonis 7 koodiga ja sinine joon (4) valemiga.

Saadud valem töötab sama hästi kui olemasolev lahendus, mida kasutab Tartu Ülikooli isejuhtivate sõidukite labor praegu (vt ptk 4.4.).

4.2. Võrdlemine füüsika põhilise valemiga

Kui võrrelda valemit (1), mis põhineb füüsikal, siinse töö valemiga, siis autori joon langeb sujuvamalt, aga valemis (1) langeb järsult ja stabiliseerub kiiresti ära. Joonisel 11 on roheline joon valemi (1) abil, sinine joon on saadud valemiga (4) ja punane joon on saadud joonise 7 oleva koodiga. Valemi (1) parameetrid olid järgmised: e oli 0, sest linnas ei ole tee kallet, μ oli 0.765 ja g oli 9.81.

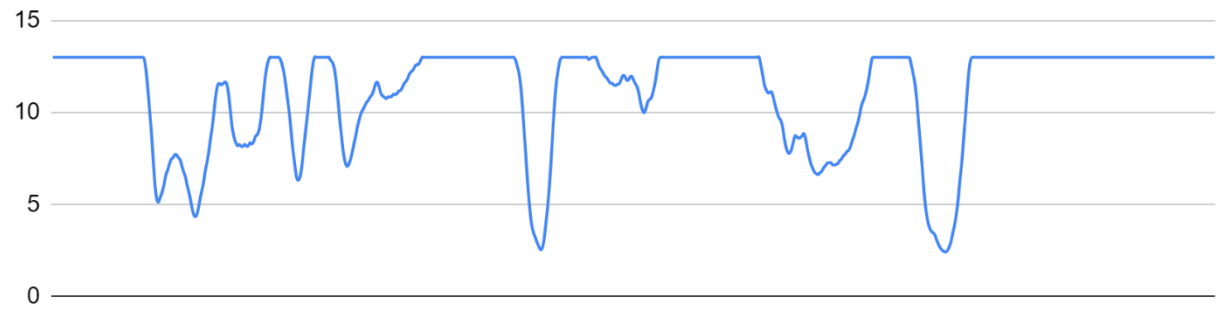


Joonis 11. Kõveruse ja kiiruse suhe, kus roheline joon on saadud (1) valemiga, sinine joon valemiga (4) valemiga ja punane joon joonise 7 koodiga.

Peab tähele panema, et valemis (1) on maksimum kiirus 90km/h juures, aga siinse töö jaoks 50km/h. Sellest hoolimata, on näha, et valemi (1) kõverusest 0.02st kuni 0.25ni on kiirus palju väiksem olevatest andmetest.

4.2. Simulatsiooni tulemused

Algul oli Autoware'is implementeeritud nii, et oli võetud teekonna listist kolm järjestikku punkti ning simulatsioonis saadi järgmine kiiruse graafik ühe testimisrajaga:



Joonis 12. Simulatsiooni kiiruse graafik (kiirused on m/s)

Jooniste 12. ja 13. järgi võib arvata, et valitud vahemaa on liiga väike ning saadud tulemus on liiga detailne. Seega muudeti funktsiooni ja prooviti erinevate vahemaadega.

Joonis 13. Kiirused mingi lõigu jaoks (kiirused m/s). Joonis: Edgar Sepp[2].

[2] Joonise on teinud Edgar Sepp ning ei ole avalikult saadaval.

Joonisel 14 on seitse erinevat joont, kus igal joonel on erinev „skip“ ning number on eraldatud sidekriipsuga. See tähendab, et „skip“ on võrdne selle arvuga. Näiteks kui võtta „skip_4“, siis „skip“ on võrdne neljaga ja indeksilt i liidetakse ja lahutatakse neli.

Joonis 14. Kiiruse graafik erinevate vahemaadega (kiirused m/s)

Jooniselt on näha, mida suurem on „skip“, seda suurem on miinimum kiirus. See võib osutuda ohuks, kuna kurvi kiirus suureneb. Seega, kui tee olukord on halb, võib auto kaotada juhitavuse ja põhjustada avarii. Veelgi, ka autos istuvatel inimestel on kurvides suurem ebamugavustunne.

4.3. Autoware'i integratsioon

Saadud valem oli vaja implementeerida Autoware'i, mis on kirjutatud C++ keeles [1], seega koodijupid, mis olid kirjutatud Pythonis, oli vaja ümber kirjutada.

Tartu Ülikooli isejuhtivate sõiduki laboril oli juba implementeeritud lihtne funktsioon, mis leiab soovitatud kiirused. Funktsioon asub planeerimismoodulis ning annab soovitatud kiirused juhtimismoodulile.

Valemi lisamine oli lihtne, kuna see on üks matemaatiline võrrand (vt joonis 15).

```
double PlanningHelpers::SpeedFormula(double curvature)
{
    return 3.91207 + 49.045*exp(-11 * curvature);
}
```

Joonis 15. Kiiruse arvutamis valem (4)

Kuid selle valemiga oli vaja arvutada kiirused antud teelõigule, seega oli vaja muuta olemasolevat funktsiooni. Funktsiooni muudeti kaks korda, kus teine implementatsioon oli tehtud peale esimest katset.

4.3.1. Esimene implementatsioon

Algul töötas muudetud funktsioon nii, et punktide valimine oli indeksi põhiselt. Selleks itereeriti üle teekonna massiivi ning iga indeksi i väärtusel liideti ja lahutati kindel „skip“ väärtus, mille tõttu saadi esimene ja komas punkt, ehk kolm punkti olid i – „skip“, i ja i + „skip“. Nende kolme punkti järgi arvutati ringi raadius. Juhul kui i on väiksem kui „skip“ või i on suurem kui teekonna massiivi suurus lahutatud „skip“, siis joonis 15 valemile anti teekonna massiivi i'nda kohal punkti atribuut „cost“, millest lahutatakse üks, kuna „cost“ on kõveruse vastand, ehk 1 – 1 / raadius, selle pärast lahutati atribuudist „cost“ üks, et saada tagasi kõverus. Kasutati „cost“, kuna oli juba välja arvutatud. Samuti teisendatakse kiirus km/h-st m/s-iks ning kontrollitakse, kas saadud kiirus on suurem maksimumkiirusest, kui on, siis pannakse maksimumkiirus, teisel juhul pannakse saadud kiirus.

4.3.2. Teine implementatsioon

Esimene implementatsioon oli sama hea kui olemasolev implementatsioon (rohkem saab lugeda 4.4. Päriselu tulemused). Seega prooviti piirata põikikiirendust, kuna see kiirendus teeb sõidu ebamugavaks. Seetõttu implementeeriti järgmine valem:

$$a = \frac{v^2}{r}, \tag{5}$$

kus v on kiirus, mille kiirusühik on m/s, ja r on raadius, mille ühik on meeter. Valemit (5) võib muuta kujule

$$v = \sqrt{a * r}, \tag{6}$$

et saada kiirus põikikiirendusest ja raadiusest. Samuti otsustati muuta kolme punkti valimist, et indeksite asemel on nüüd kaugusted, ehk et kui varem lahutati ja liideti kindel arv keskmisele indeksile, et saada äärmised punktid, siis nüüd peavad punktid olema kindla kaugusega.

Muudetud funktsiooni (vt joonis 16) idee oli nüüd, et itereeriti üle teekonna massiivi ning kui varem lahutati ja liideti indeksist i kindel „skip“ väärtus, nüüd on „skip“ tehtud kaheks, tagasi vaates kasutatakse „backward“ ning ette vaates „forward“, ehk kolm punkti on i – „backward, i ja i + „forward“. Nendega arvutatakse ringi raadius ning joonis 15 valemisse sisestatakse saadud raadius. „Backward“ punkt leitakse nii, et muutujale antakse algväärtuse üks ning kontrollitakse, kas i ja „backward“ indeksi punktide kaugus on suurem kui „distanceFromCenterPoint“, kui ei liidetakse üks „backward“ muutujale ning tehakse seda nii kaua kui i ja „backward“ indeksite punktide kaugus on suurem või võrdne „distanceFromCenterPoint“, sama loogika on „forward“ punkti leidmisel. Juhul kui i lahutatud „backward“ on väiksem kui null või i liita „forward“ on suurem või võrdne kui teekonna massiivi suurus, siis raadius saadi i'nda kohal punkti atribuudist „cost“, millest lahutatakse üks ja üks jagatakse saadud väärtusega. Saadud väärtus sisestatakse joonis 15 valemisse, et saada kiirus. Kui kiirus on nüüd kätte saadud, siis teisendatakse kiirus km/h-st m/s-iks, sisestatakse saadud kiirus ja raadius (5) valemisse ning kontrollitakse, kas saadud põikikiirendus ei ületa „lateralAccelerationLimit“ väärtusest, kui ületab, siis kiirus saadakse (6) valemiga, kus sisendiks on „lateralAccelerationLimit“ ja raadius. Lõpuks kontrollitakse, kas saadud kiirus on suurem maksimumkiirusest, kui on, siis pannakse maksimumkiirus, teisel juhul pannakse saadud kiirus.

```
double v = 0;
double r = 0;
GPSPoint center;
double lateralAccelerationLimit = 5.0;
int distanceFromCenterPoint = 1.0;

for (int i = 0; i < path.size(); i++) {

      int forward = 1;
      int backward = 1;
      bool gotForward = false;
      bool gotBackward = false;

      while (!gotForward || !gotBackward)
      {
            if (i - backward < 0 || i + forward >= path.size())
            {
                  r = 1 / (1 - path[i].cost);
                  v = SpeedFormula(1 / r);
                  break;
            }
            else
            {
                  if (distance2points(path[i - backward].pos, path[i].pos) <
             distanceFromCenterPoint && !gotBackward)
                        backward += 1;
                  else
                        gotBackward = true;

                  if (distance2points(path[i].pos, path[i + forward].pos) <
                  distanceFromCenterPoint && !gotForward)
                        forward += 1;
                  else
                        gotForward = true;

                  if (gotForward && gotBackward)
                  {
                        r =  CalcCircle(path[i-
                        backward].pos, path[i].pos, path[i+forward].pos, cen
                        ter);
                        if(r > 150.0 || std::isnan(r))
                              r = 150.0;
                        v = SpeedFormula(1 / r);
                  }
            }
```

```
        }
        v = KmhToMs(v);
        if (GetLateralAcceleration(v, r) > lateralAccelerationLimit)
              v = GetSpeedWithLateralAccelerationLimit(lateralAccelerationLimi
              t, r);

        if (v > max_speed)
              path[i].v = max speed;
        else
              path[i].v = v;
}
```

Joonis 16. Kiiruste leidmise muudetud funktsioon.

Saadud kiirused on soovituslikud ning kui ees ei ole takistusi, siis saab sõita saadud kiirusega.

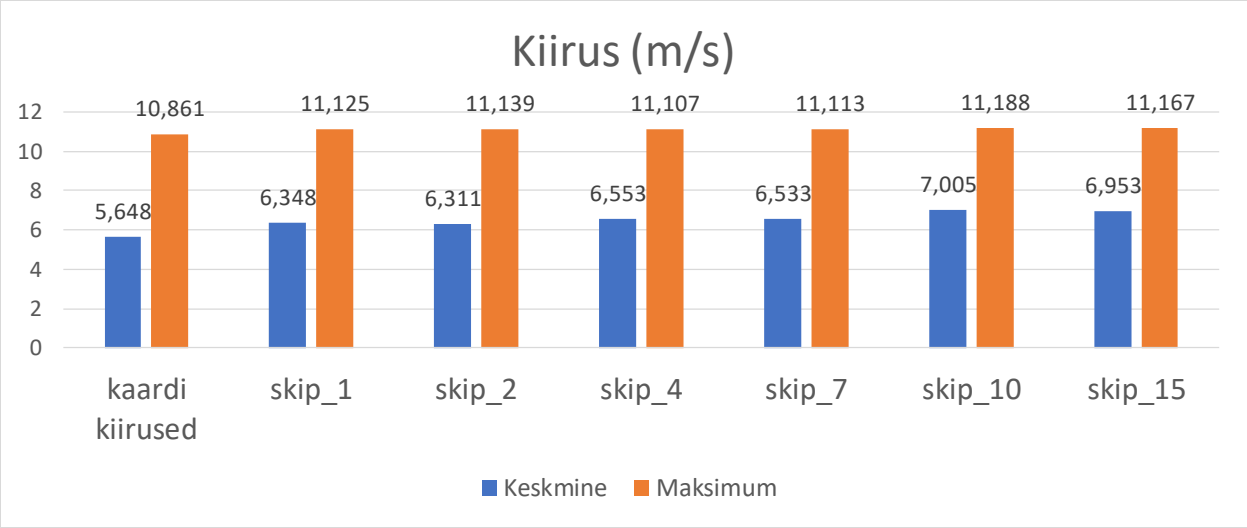
4.4. Päriselu tulemused

Terminit „kaardi kiirused“ kasutatakse siinses töös näitamaks, et testimis rada (vt joonis 8) kiirused olid käsitsi pandud.

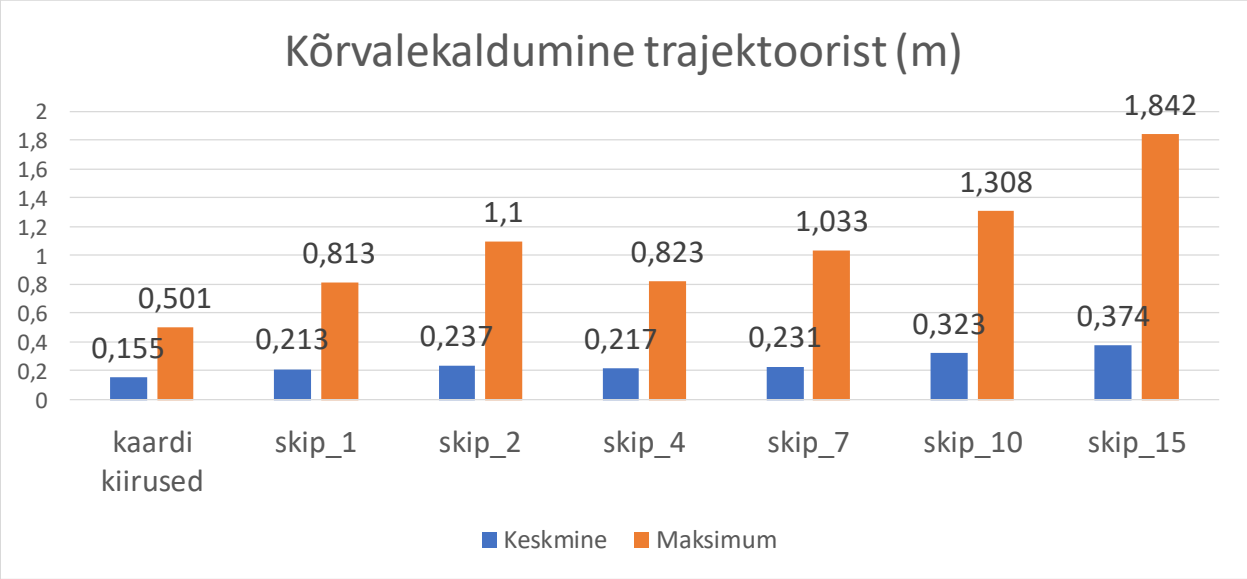
Oli tehtud kaks katset, kus esimeses katsetati indeksi põhilist punkti valikut ning teises katsetati kauguse põhilist punkti valikut ja oli ka põikikiirenduse piiraja. Peab märkima, et esimest ja teist katset otseselt ei tohi võrrelda, kuna muutus mitte ainult autori lahendus kui ka teised komponendid.

4.4.1. Esiemene katse

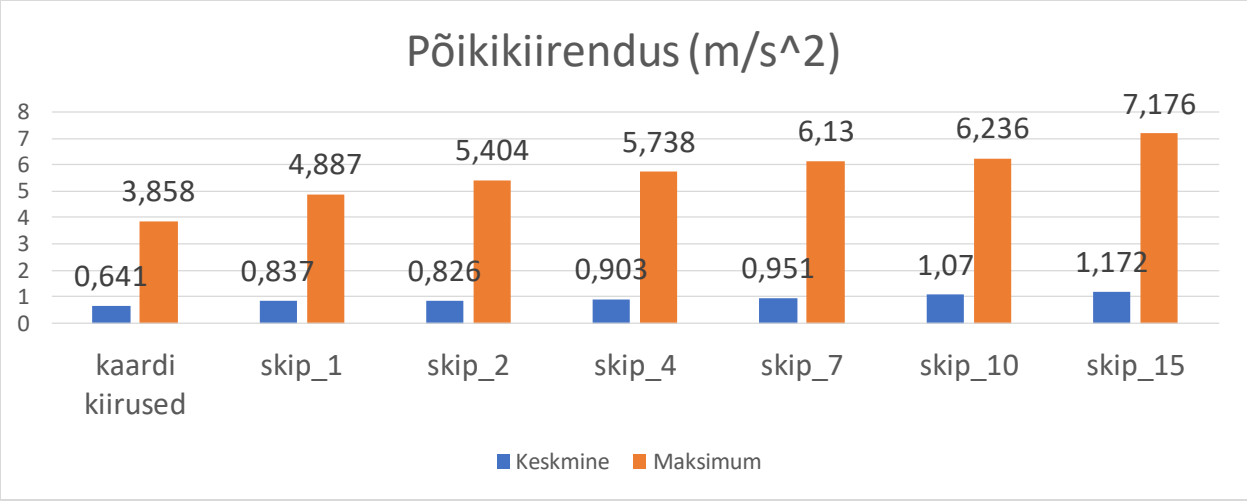
Joonisel 17 on näha, et tulemused vastavad simulatsioonile joonis 14 olevate tulemustega. Mida suurem on „skip“, seda suurem on kiirus kurvides, mille tagajärjel kõrvalekaldumine trajektoorist ja põikikiirendus suurenevad – seda kinnitavad ka joonised 18 ja 19. Samuti võib seda näha ka korrelatsioonides, kui arvutada keskmine kiirus, põikikiirendus või kõrvalekaldumine trajektoorist „skip“ suurusega korrelatsioonis. Saadakse vastavad väärtused 0.910423196, 0.99124758 ja 0.928086432, mis tähendab, et korrelatsioon on tugev.



Joonis 17. Esimese katse keskmine ja maksimum kiirus

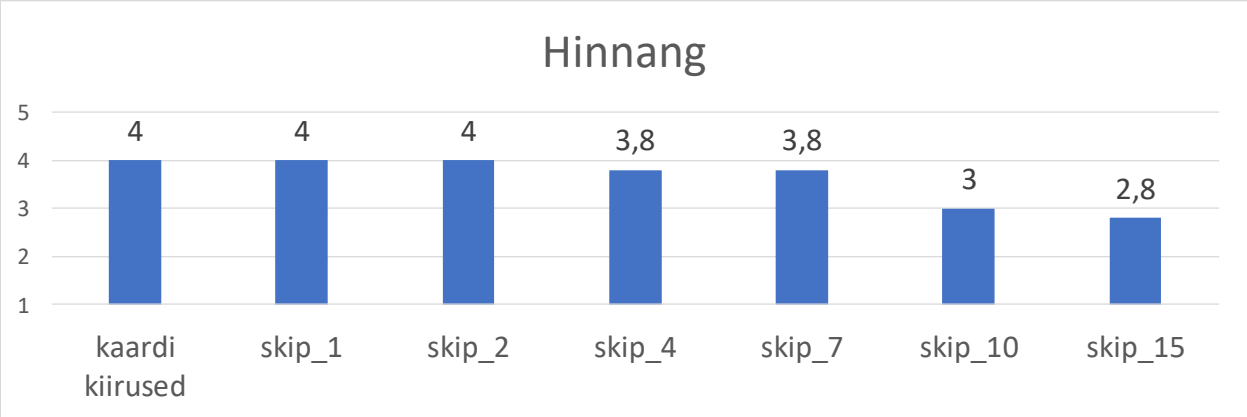


Joonis 18. Esimese katse keskmine ja maksimum kõrvalekaldumine trajektoorist



Joonis 19. Esimese katse keskmine ja maksimum põikikiirendus

Kuigi katses „kaardi kiirused“ kõrvalekaldumine trajektoorist ja põikikiirendus on väiksem kui autori implementeeritud valem, on ka keskmine kiirus väiksem. Teisisõnu, autot on võimalik panna sõitma 10km/h selleks, et saavutada väikseid põikikiirendusi ja vähem kõrvalekaldeid trajektoorist. Siiski selline lahendus ei sobi, sest sellise kiirusega ei ole mõistlik sõita. Seega proovitakse leida kiiruse ja mugavuse tasakaal. Joonisel 20 on näha juhendaja antud hinnang skaalal 1 kuni 5 testsõidu jaoks. Juhendaja oli hindaja, kuna tal on kogemus ja on Tartu Ülikooli isejuhtivate sõidukite labori tehnoloogiajuht.

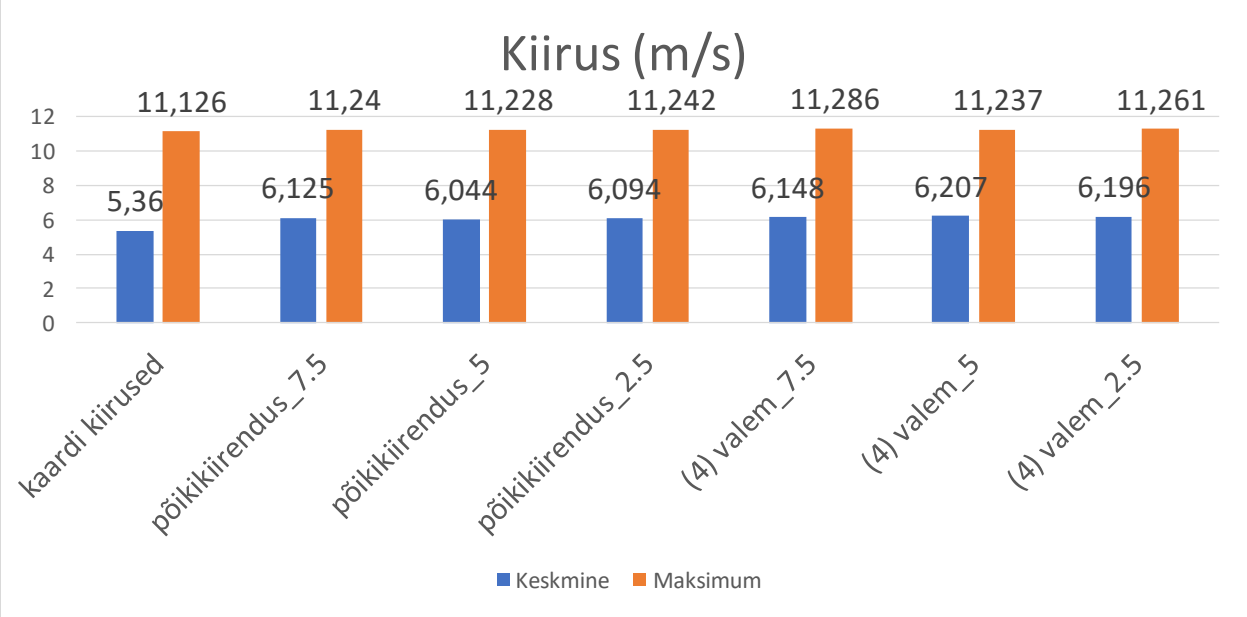


Joonis 20. Esimese katse juhendaja hinnang skaalal 1 kuni 5

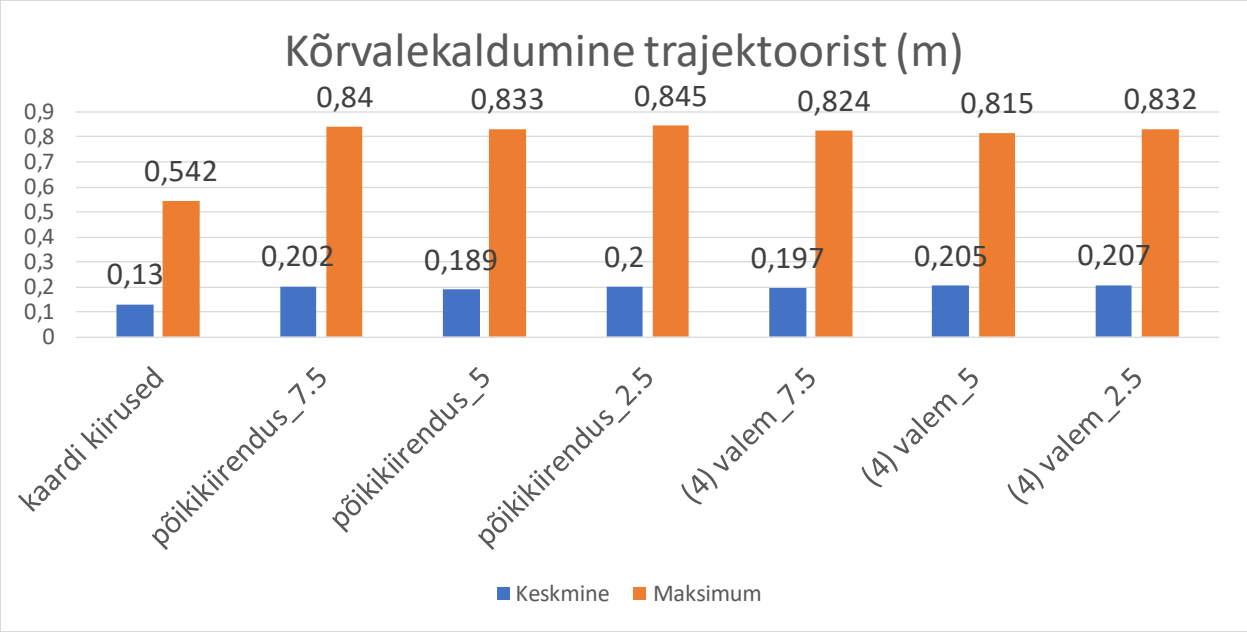
Juhendaja on andnud sama hinnangu katsele „kaardi kiirused“ ning autori implementatsioonile, kui „skip“ väärtus oli pandud ühele või kahele.

4.4.2. Teine katse

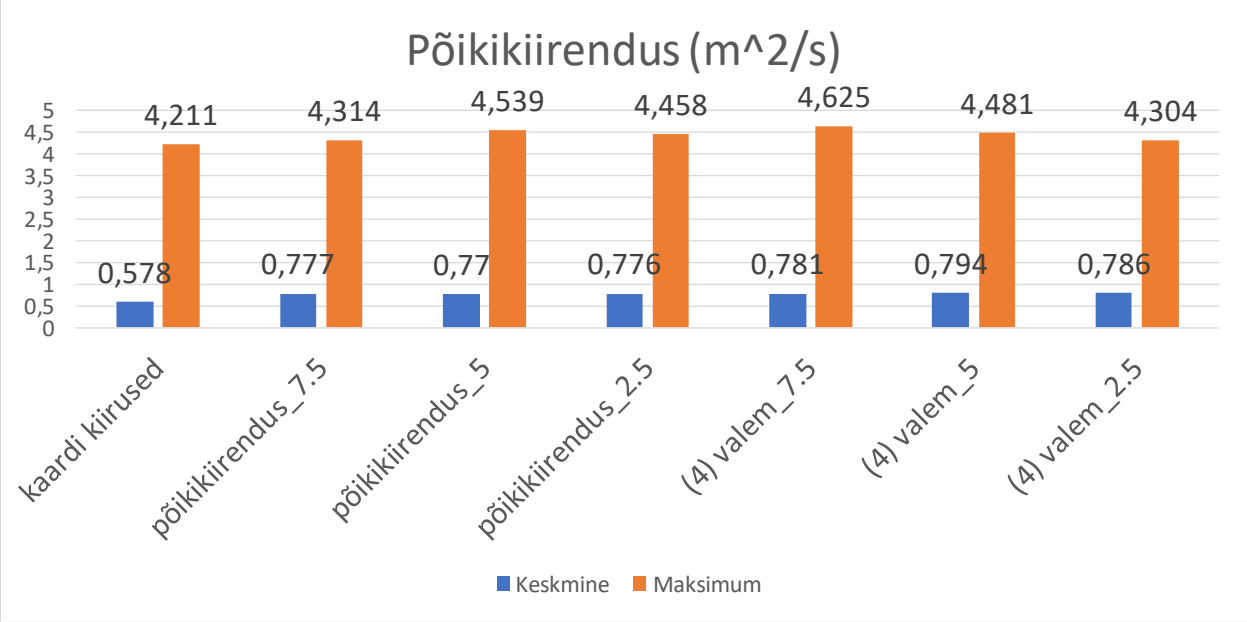
Teises katset otsustati, et punktide valimis kaugus on üks meeter, ehk esimese ja teise ning teise ja kolmanda punkti vahel on rohkem kui üks meeter. Seega, ainuke muutuv parameeter on põikikiirenduse piiramis arv, kuid prooviti kiirust saada ka ainult (6) valemiga, mis on väga erinev autori valemist. Jooniste 21, 22 ja 23 tulbad, kus on põikikiirendus ja number, tähendab, et kiiruse leidmiseks oli kasutatud (4) valemit ning põikikiirendus ei tohtinud ületada tollest arvust, ning tulbad, kus on (6) valem ja number, tähendab, et on kasutatud toda valemit kiiruse leidmiseks ja number oli põikikiirenduse võrrandi sisendiks. Joonistel 21, 22 ja 23 on näha, et autori katsetatud lahendustel ei ole suuri erinevusi ja kõige suurem erinevus on just käsitsi pandud kiirustega ning on sama muster nagu esimeses katses, et keskmine kõrvalkaldumine trajektoorist ja põikikiirendus on autori lahendusel suurem, kuid ka keskmine kiirus on suurem.



Joonis 21. Teise katse keskmine ja maksimum kiirus

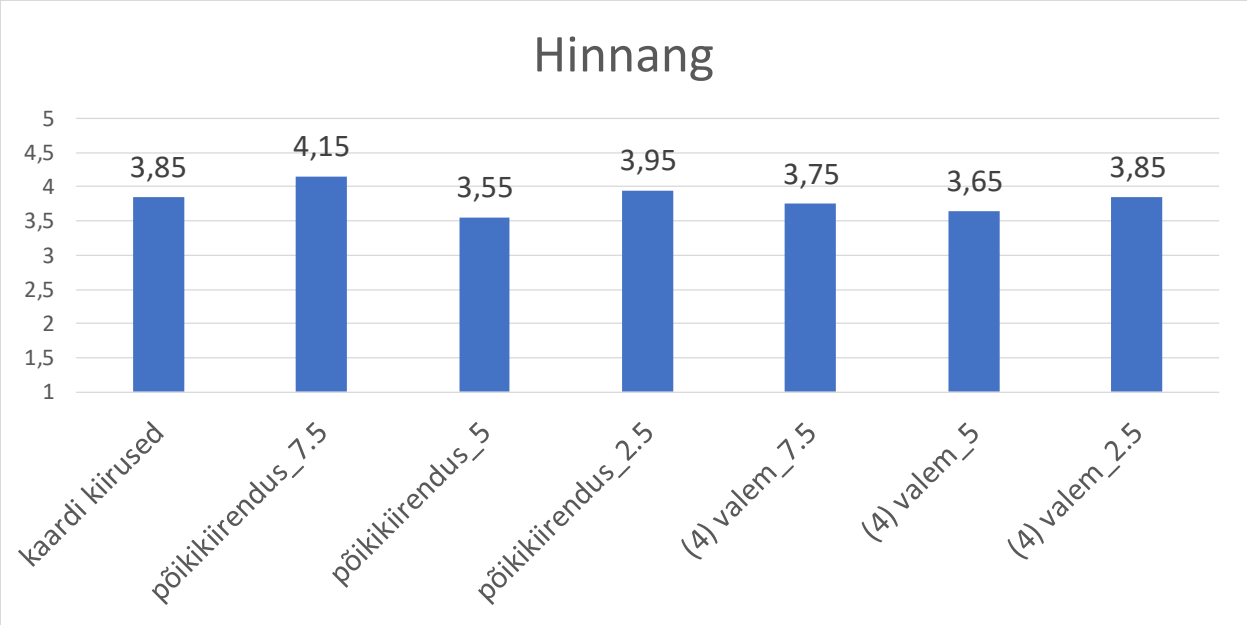


Joonis 22. Teise katse keskmine ja maksimum kõrvalekaldumine trajektoorist



Joonis 23. Teise katse keskmine ja maksimum põikikiirendus.

Teises katses olid hindajateks juhendaja ja Edgar Sepp. Kui vaadata hinnangu joonist (vt joonis 24), siis parim lahendus osutuks, kus kiirus oli leitud (4) valemiga ning oli põikikiirenduse piiraja, mille väärtus on 7.5.



Joonis 24. Teise katse juhendaja ja Edgar Seppa keskmine hinnang skaalal 1 kuni 5.

Kuigi autori lahendus osutus parimaks seekord, ei teeks autor suuri järeldusi, sest autor istus ka autos testide ajal ja väga suuri erinevusi ei tundnud, nagu seda võib näha joonistel, kuigi parameetrite muutused on mõjukad. Nt kahes testimis rajal kurvis oli kiirus langenud iga katsega alla 5km/h, isegi siis kui piirata miinimum kiirust. Seega, autor ütleks, et praeguse Tartu Ülikooli isejuhtivate sõidukite labori isejuhtimissüsteemi tarkvaras ei ole kiiruste valimis funktsioon väga mõjukas, vaid pigem mõjutavad teised komponendid palju tugevamalt autos viibivate inimeste mugavust. Üldiselt tõsteti keskmist kiirust ilma, et hinnang oleks langenud, ning autori lahendus on rohkem inimlik, kuna käsitsi pandud kiirustega sõitis auto liiga ettevaatlikult.

Kokkuvõte

Siinse töö eesmärk oli leida võrrand, mis iseloomustab kõige paremini graafikut, millel ühel teljel on auto kiirus ja teisel teljel kõverus. Võrrandi sisendiks on kolme järjestiku punkti kõverus ning tagastatakse kiirus vastava teelõigu jaoks. Andmed olid saadud Tartu Ülikooli isejuhtivate sõidukite labori käest, mille nad on salvestanud Tartu Ülikooli ja Bolti isejuhtiva autoga. Võrrandi leidmiseks kasutati veebilehte „dCode“ [2] ning muudeti parameetreid, et võrrandi joon vastaks paremini andmete joonele.

Samuti implementeeriti võrrand Tartu Ülikooli Autoware'i klooni, millega sai teha simulatsioone ja kasutada seda isejuhtivas sõidukis. Simulatsiooni tulemuste põhjal tundus, et implementeeritud lahendus arvutas liiga detailselt. Nimelt arvutati iga kolme järjestiku punkti kõverus ja kiirus. Seega muudeti funktsiooni ning prooviti erinevate kaugustega kõveruse arvutamist, ehk esimese ja teise ning teise ja kolmanda punkti kaugus suurendati. Selle tulemuseks saadi, mida suurem on vahemaa, seda suurem on kurvi miinimum kiirus, mis võib osutuda ohuks.

Päriselus tehti kaks katset, kus esimeses katses kinnitati, et mida suurem oli kiiruse arvutamise vahemaa, seda suurem oli kiirus. Seda kinnitasid nii testimises osalevad inimesed kui salvestatud andmed. Kinnituseks oli kiiruse, kõrvalekaldumine trajektoorist ja põikikiirenduse joonised. Kuigi käsitsi pandud kiirustega kõrvalekaldumine trajektoorist ja põikikiirendus oli väiksem kui autori implementatsioon, oli keskmine kiirus ka madalam ning juhendaja andis sama hinnangu. Peale esimest katset, otsustati lisada põikikiirenduse piiraja, et üle kindlat väärtust põikikiirendus ei tohtinud minna. Teises katses leiti, et autori katsetatud lahendustel ei olnud suuri erinevusi, kuigi prooviti ka (6) valemiga, mis on väga erinev autori valemist, ja põikikiirenduse piiraja väärtuse muutused oli suured, ning suurim erinevus oli käsitsi pandud kiirustega. Seega, autor pakub, et praeguse Tartu Ülikooli isejuhtivate sõidukite labori isejuhtimissüsteemi tarkvaras ei ole kiiruste valimis funktsioon väga mõjukas, vaid pigem mõjutavad teised komponendid palju tugevamalt autos viibivate inimeste mugavust. Üldiselt, keskmine kiirus oli tõstetud, mis tundus rohkem inimlik, ilma, et mugavuse hinnang oleks langenud.

Viidatud kirjandus

[1] Autoware. https://www.autoware.ai/ (01.04.2021)

[2] dCode. https://www.dcode.fr/function-equation-finder (01.04.2021)

[3] EdG. Numerical way to solve for the curvature of a curve. https://math.stackexchange.com/questions/2507540/numerical-way-to-solve-for-the-curvature-of-a-curve (31.03.2021)

[4] eSafety. Final Report of the eSafety Working Group on Road Safety. November 2002. https://op.europa.eu/en/publication-detail/-/publication/30836ab3-b0d1-4d76-bd6b-0ef9a4878c0d (07.12.2020)

[5] Geng, X., Liang, H., Xu, H., Yu, B.,Zhu, M. Human-Driver Speed Profile Modeling for Autonomous Vehicle's. *2016 IEEE Intelligent Vehicles Symposium (IV)*. Göteborg, Rootsi: IEEE, 2016, 755-760. https://www.depts.ttu.edu/transtech/documents/46.pdf (07.12.2020)

[6] Geoportaal. L-EST97 koordinaatsüsteem. https://geoportaal.maaamet.ee/est/Ruumiandmed/Geodeetilised-andmed/Eesti-geodeetiline-susteem/L-EST97-koordinaatsusteem-p173.html (21.04.2021)

[7] Gu, T., Dolan, J. M. Toward Human-like Motion Planning in Urban Environments. *2014 IEEE Intelligent Vehicles Symposium Proceedings*. Dearborn, USA: IEEE, 2014. https://ieeexplore.ieee.org/document/6856493 (07.12.2020)

[8] Jain, A. K., Mao, J., Mohiuddin, K. Artificial Neural Networks: A Tutorial. *Computer (Volume: 29, Issue: 3, Mar 1996)*. Washington, USA: IEEE, 1996, 31-44. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=485891 (07.12.2020)

[9] Kröger, F. Automated Driving in Its Social, Historical and Cultural Contexts. *Autonomous Driving*. Berliin, Saksamaa: Springer, 2016, 41-68. https://link.springer.com/chapter/10.1007/978-3-662-48847-8_3 (07.12.2020)

[10] MathWords. Understanding Model Predictive Control. https://www.mathworks.com/videos/series/understanding-model-predictive-control.html (07.05.2021)

[11] Matiisen, T. Isejuhtivad autod. *Tehisintellekti algkursus*. 2019. https://courses.cs.ut.ee/2020/Tehisintellekti_algkursus/spring/Main/PARTIVise (24.03.2021)

[12] Pandas. Intro to data structures. https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html#dataframe (21.04.2021)

[13] ROS. About Us. https://www.ros.org/about-ros/ (01.04.2021)

[14] Self-driving cars in the EU: from science fiction to reality. 2019. https://www.europarl.europa.eu/pdfs/news/expert/2019/1/story/20190110STO23102/20190110STO23102_en.pdf (07.12.2020)

[15] Serna, C. G., Ruichek, Y. Dynamic Speed Adaptation for Path Tracking Based on Curvature Information and Speed Limits. *Sensors 2017, 17(6), 1383*. 2017, 1-22. https://www.mdpi.com/1424-8220/17/6/1383 (24.03.2021)

[16] Tartu Ülikool. Isejuhtivate sõidukite labor. https://www.cs.ut.ee/et/isejuhtivate-soidukite-labor (30.04.2021)

[17] Tesla. Autopilot. https://www.tesla.com/autopilot?redirect=no (03.01.2021)

[18] Waymo. Waymo One. https://waymo.com/waymo-one/ (29.03.2021)

Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Eduard Rudi,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

Inimlik kiirus kurvides

mille juhendaja on Tambet Matiisen,

reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Eduard Rudi

07.05.2021