SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1935

Lokalizacija autonomnog vozila u simuliranom urbanom okruženju

Matija Vukić

Zagreb, svibanj 2019.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Zahvala

SADRŽAJ

1. Uvod

U zadnjih 30ak godina se može vidjeti ubrzan napredak u automatici i računarstvu. Tome su prethodile godine teorizacija iz kojih su kasnije nastali razni algoritmi za optimizacije i obradu podataka. Sada s razvojem računalnih aspekata poput memorija te procesorske snage možemo obrađivati sve više podataka u što manje vremena. To omogućava današnjim vozilima da u potpunosti budu električna što znači da su stalno u kontaktu s okolinom te mogu bez prestanka skupljati podatke iz okoline. Ta vozila su zapravo skupine senzora i uređaja konstantno povezanih na internet.

Skoro sva takva vozila danas imaju mogućnost autonomne vožnje. To omogućuju ljudima ugodnije iskustvo te veliko smanjenje broja prometnih nesreća koje uzrokuju vozači u stanjima koja nisu prigodna za vožnju ili čistome nemaru. S obzirom da je sigurnost u prometu jedna od najvažnijih stvari po pitanju svih sudionika ,ona mora biti prioritet. To znači da svi algoritmi za bilo kavo upravljanje vozilom moraju biti u potpunosti testirani te bez ikakvih pograšaka. Naravno to je nemoguć zahtjev zato što u takvoj domeni ti algoritmi za rad imaju previše varijabli koje se ne mogu uzeti u obzir.

Cilj ovoga rada je ukratko objesniti rad i prkazati rezultate nekoliko algoritama te njihove točnosti. Ulaz u algoritam su senzorska očitanja dok su izlazi lokacija vozila tj. relativna promjena lokacije između dva očitanja. Za usporedbu rezultata koristimo referentne podatke koji su prikupljni iz simulatora te je tako garantirana njihova točnost. Nekoliko primjera podataka je provedeno kroz algoritme te uspoređeno s referentnim podacima. Rezultati tih evaluacija su ilustrirani pomoću grafova.

2. Problem lokalizacije autonomnog vozila

2.1. Lokalizacija

Roboti i vozila u većini slučajeva se primjenjuju za izvođenje repetitivnih ili opasnih po život poslova. Čovjeka zamjenjuje robot ali to znači da je za upravljanje robota zadužen taj isti robot ili neki udaljeni sustav tj. čovjek više nema tu ulogu. U tu svrhu su roboti i vozila opremljeni raznim senzorima da bi se to omogučilo. Svi podaci prikupljeni iz tih senzora se koriste prilikom lokalizacije robota ili vozila.

Lokalizacija je postupak određivanja lokacije objekta u prostoru iz ulaznih podataka. Lokalizacija može biti vrlo zahtjevan zadatak te se u tu svrhu mogu koristiti algoritmi različitih složenosti. Što je algoritam složeniji to se sporije izvodi ali je točiji dok se neki više optimizirani tj. brži algoritmi brže izvode ali postoji veća vjerojatnost da je došlo do pogreške prilikom izvođenja.

Koriste se algoritmi za istovremenu lokalizaciju i mapiranje [dodaj SLAM link] tj. za stvaranje karte nepoznatog prostore kojime se robot kreće te koordinate u tome prostoreu. Lokalizacija odgovara 'Gdje je robot sada?' tj. gdje je sada naspram prethodne lokacije. Na to pitanje se može odgovoriti ovisno o tome radi li se o lokalizaciji u otvorenome ili zatvorenome prostoru. Lokacija robota je oglavnom prikazana u kartezijskom koordinatnom sustavu, bilo to u 2d ili 3d prostoru.

Postoje dvije vrste lokalizacije:

- Lokalna - informacije se prikupljaju pomoću senzora robota iz njegove okoline
- Globalna - informacije se dobiju iz GPS-a ili slično

Neke metode lokalizacije

Jedna od najjednostavnijih metoda je "Metoda najmanjih kvadrata" (eng. Least Squares Error) gdje se koristi metoda najmanjih kvadrata za regresijsku analizu podataka. Cilj te metode jest minimizacija pogreške gdje robot jest i gdje bi robot trebao biti tj. ona okvirno procjenjuje gradijent funkcije pomaka robota.

Praćenje pozicije (eng. Pose Tracking) metoda se koristi kada je poznata početna pozicija robota pa je potrebno samo pratiti njegovu poziciju kroz vrijeme. Metoda koristi ekstrakciju tj. izdvajanje značajki okoline koje se mogu uspoređivati te se tako kroz vrijeme može pratiti promjena položaja nekih uočljivih objekata.

Metoda višestrukih hipoteza (eng. Multiple Hypotesis Localization) pretpostavlja da početna pozicija nije poznata ali je poznata topografija mape. U ovome slučaju početnu poziciju može robotu pridodati korisnik ili robot uvijek može započeti iz iste pozicije. Ideja iza ove metode je da se detektira svojstvo te se preko njega stvaraju hipoteze o položaju robota naspram toga objekta kojemu pripada to svojstvo. Može se stvoriti nova hipoteza ili se može poboljšati neka od prethodnih hipoteza ili ipak eliminirati.

Metoda iteracije najbližih točaka (eng. Iterative Closest Point) minimizira razliku između dvije skupine točaka tako da iterira između svake dvije točke te pronalazi onu kombinaciju koja daje najmanju grešku. Često se koristi pri rekonstrukciji 2D ili 3D površina nakon skeniranja. Tijekom izvođenja te metode jedna skupina točaka je fiksna tj. referentna dok se druga transformira tako da se najbolje slažu koordinatama u referentnom skupu. Postoje mnoge varijante ICP-a od kojih su point-to-point (usporedba točka-točka) , point-to-plane (usporedba točke-površina) i point-to-line (usporedba točka-linija) najpopularnije.

Metoda usporedbe očitanja (eng. Scan Matching) koristi dva uzastopna očitanja senzora robota poput lasera, sonara, ... da se pronađe relativan pomak robota u prostoru. Razlike između dva očitanja senzora se mogu uočiti vrlo lako zbog učestalosti skeniranja tj. frekvencije dohvaćanja senzorskih podataka te o gustoći lasera kojih uglavnom ima od nekoliko stotina do nekoliko tisuća. Načina na koji se zapravo traže razlike između dva očitanja ima mnogo. Koriste se laseri (eng. Laser Range Finders) da bi vidio prepreke i odometrija kotača (eng. Wheel Odometry) da dobije okvirno stanje robota. Odometrija iz kotača ima određenu grešku zbog proklizavanja kotača ili nekog drugog razloga te se ona tada ispravlja pomoću izračunatih vrijednosti odometrije iz lasera. U ovom radu će biti opisan te implementiran način pronalaženja odometrije pomoću korelacije histograma podataka iz lasera.

2.2. Problem lokalizacije

Sve prethodno navedene imaju nešto zajedničko, a to je da koriste algoritme čiji rezultati nikada nisu posve točni.
Ta netočnost može proizaći zbog sljedećih razloga:

- Šum u očitanjime - u podacima koji se dobiju iz senzora uvijek ima i podataka koji su nastali zbog privremenih objekata (npr. pas koji prolazi pokraj vozila)
- Sinkronizacija obrade i očitanja podataka - zbog prebrzog slanja podataka algoritmu, te se tako mogu neka očitanja preskočiti
- Samog načina izvedbe senzora - možda senzor zbog samog načina fizičke izvedbe ima uračunat šum
- ...

Razne metode lokalizacije već unutar svog tijeka izvođenja imaju metode koje prate veličinu relativne pogreške te ju pokašaju minimizirati nakon svake iteracije ali ta pograška i dalje postoji te će uvijek i postojati. Te pogreške se robota koji rade u skladištima ne moraju uzeti previše ozbiljno, dok se kod autonomnih vozila u svakodnevnome cestovnom prometu ili industrijskih robota te pogreške moraju uvijek uzeti u obzir.

3. Priprema podataka

3.1. Simulator

Za točne referentne podatke potrebno je imati simulirano okruženje. Takvo simulirano okruženje se zove simulator. Potreban je simulator koji već ima integrirane razne mape, razne senzore, vozila te način komunikacije s tim vozilima iz vanjskih skripti. Neki od simulatora su opisani u sljedećem tekstu.

Carla

Slika 3.1: Carla logo

Carla je simulacijsko okruženje koje služi za testiranje metoda i algoritama prilikom razvoja autonomnih vozila. U pozadini koristi Unreal Engine za izvršavanje simulacije. Simulator se ponaša kao poslužitelj koji prima naredbe iz vanjskih klijentskih programa. Ti klijentski programi su pisani u programskom jeziku pytohn. Carla ima integrirane razne senzore te su neki od njih:

- RGB kamera
- LIDAR senzor
- Senzor dubine
- GNSS

Sponzori projekta su Intel, Toyota, GM i Computer vision Center. Više o ovome simulatoru će biti u sljedećem poglavlju.

Apollo

Slika 3.2: Apollo logo

Apollo je također rješenje za testiranje autonomnoh vozila. Sadrži simulator ali također je i potpuno komercijalno rješenje. Podržava razne scenarije, ima sustav ocjenjivanja koji daje ocjenu na temelju desetak metrika. Simulacije zapravo provodi u oblaku tj. koristi Microsoft Azure. Sponzori projekta su mnoge azijske tvrtke kao i Ford, Microsoft, Daimler, Honda, Intel i ostali.

rFpro

Slika 3.3: rFpro logo

rFpro je kompletno rješenje za testiranje autonomnih vozila. U potpunosti je komercijalno rješenje ali je zato jedno od najboljih u svijetu. Uglavnom je usredotočeno na primjenu strojnog učenja u autonomnim vozilima. Ima jednu od najvećih baza digitaliziranih stvarnih likacija diljem svijeta. Dinamički sustav vremena omogućuje testiranje ponašanja vozila u raznim vremenskim uvjetima. Sponzori projekta su BMW, Shell, GM, Renault i ostali.

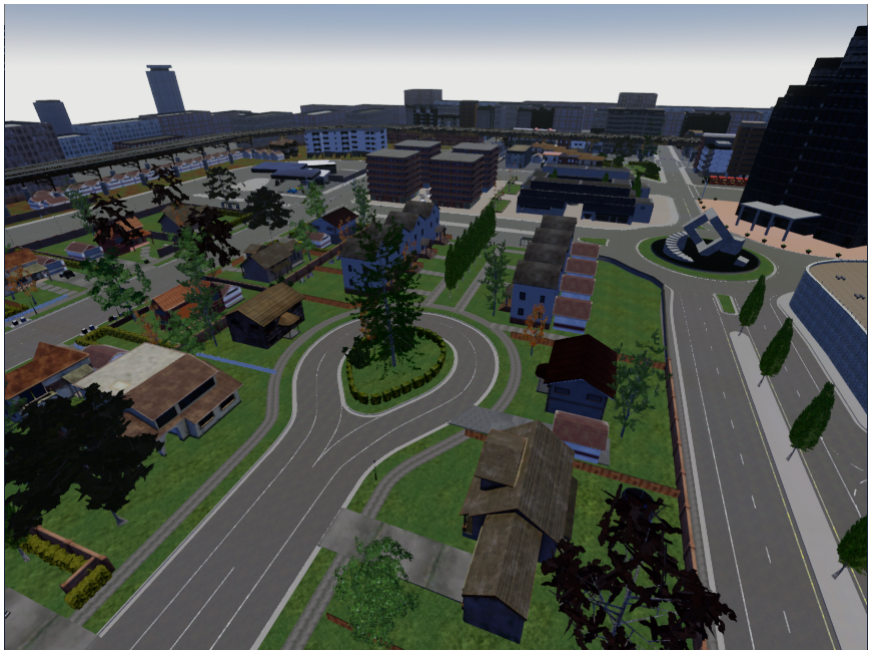
AVSimulation

Slika 3.4: AVSimulation logo

AVSimulation se zapravo sastoji od simulatora vožnje i samog simulatora SCANeR. SCANeR je skup aplikacija koji pružaju rute, senzore, vozila, dinamičko vrijeme, pisanje skripti. Vrlo je modularan. Smulator vožnje je zapravo kupola koja se sastoji od cijelog vozila te se zapravo kretanje tog vozila simulira unutar te kupole. Sponzori su Renault, PSA, Volvo, Microsoft, Mazda i ostali.

3.2. Opis okruženja

Simulacijsko okruženje će biti Carla zato što ima vrlo široko programsko sučelje za upravljanje aspektima simulacije te je besplatno za korištenje. Simulator se pokreće kao poslužitelj te se vozila dodaju pomoću skripte koja je napisana u programskom jeziku python.



Slika 3.5: Primjer mape pod nazivom Town03

Na slici 3.5 se vidi pogled na jedan od 7 mapa iz perspektive slobodne kamere.

Korištene mape su definirane OpenDrive standardom. Simulator podržava raznolike senzore. Svi ti senzori se mogu postaviti samostalno na mapu, ali su najkorisniji kada se postave na drugo vozilo.

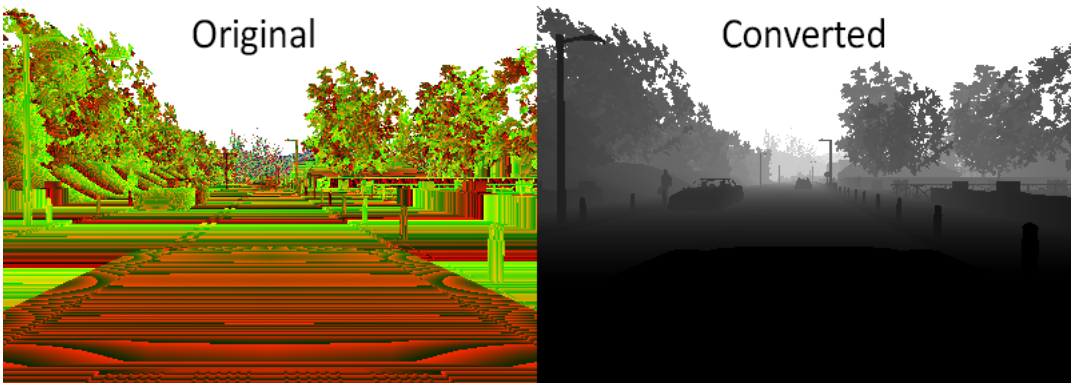
3.2.1. Senzori

RGB senzor

Slika 3.6: Primjer regularne kamere

RGB kamera je zapravo regularna kamera koja sliku onoga što vidi predstalja pomoću crvene, zelene i plave boje.

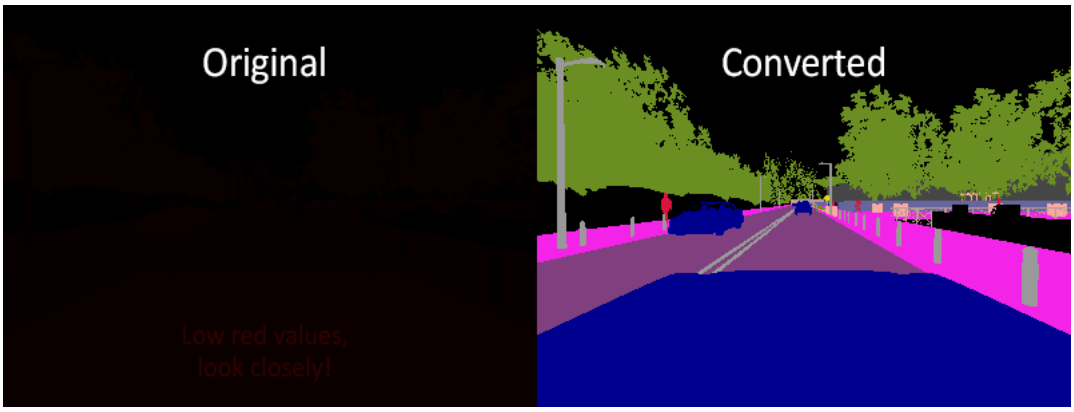
Senzor dubine



Slika 3.7: Primjer rezultata senzora dubine

Senzor dubine prikazuje svaki pixel na slici kamere u nijansama sive boje tj. ovisno koliko je objekt na određnome pixelu odaljen od kamere imat će svijetliju nijansu.

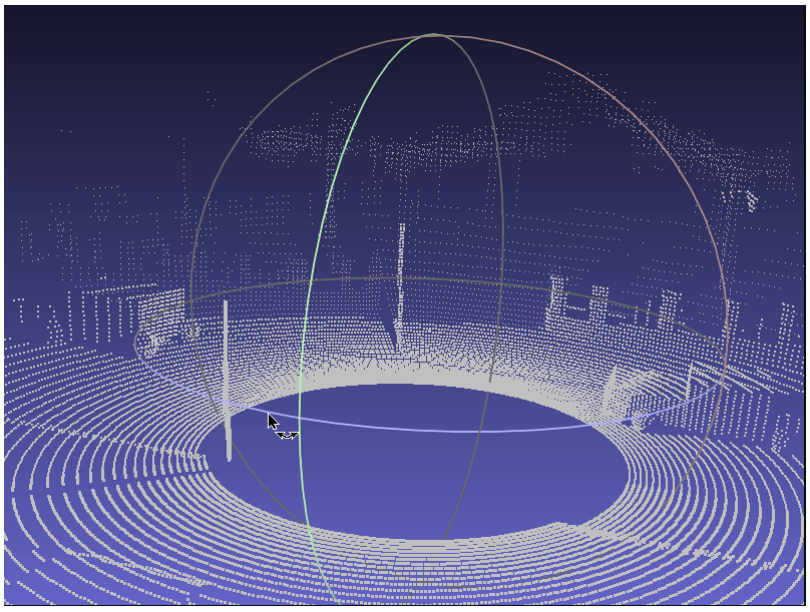
Senzor semantičke segmentacije



Slika 3.8: Primjer semantičke segmentacija

Ovaj senzor dijeli sliku kamere na semantičke dijelove tj. objekterazličitog tima predstavlja drigim bojama. Na slici 3.8 se vidi da je nebo crne boje, auti su plave boje, drveće je zeleno itd.

LIDAR senzor



Slika 3.9: LIDAr podaci

LIDAR senzor je zapravo vertikalan skup lasera koji simuliraju skeniranje od 360 stupnjeva tako da se rotiraju određeni broj puta u sekundi. Povratni podaci senzora su zapravo točke do kojih su laseri uspjeli doći. Na slici 3.9 se mogu vidjeti takvi podaci

vizualizirani u programu MeshLab. Više o ulaznim parametrima senzora kasnije u radu.

Senzor sudara

Ovaj senzor dojavljuje klijentskome programu ako se vozila sudarilo s drugim objektom u simulaciji.

Senzor prijelaza trake

Ovaj senzor dojavljuje klijentskome programu ako je vozilo prošlo preko trake na cesti.

GNSS senzor

Senzor koji dojavljuje klijentskome programu trenutnu GNSS lokaciju vozila. Ta lokacija se interno računa tako da se lokacija vozila dodaje na geografsku referentnu lokaciju definiranu za cijelu mapu.

Senzor prepreke

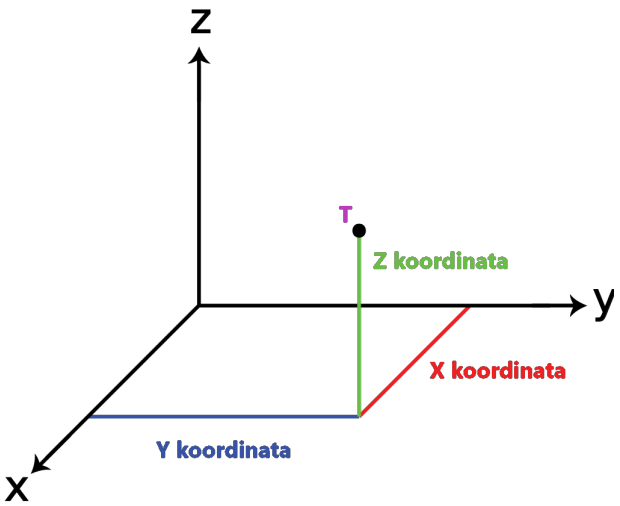
Ovaj senzor javlja klijentskome programu ako se ispred vozila nalazi prepreka.

3.2.2. Promet

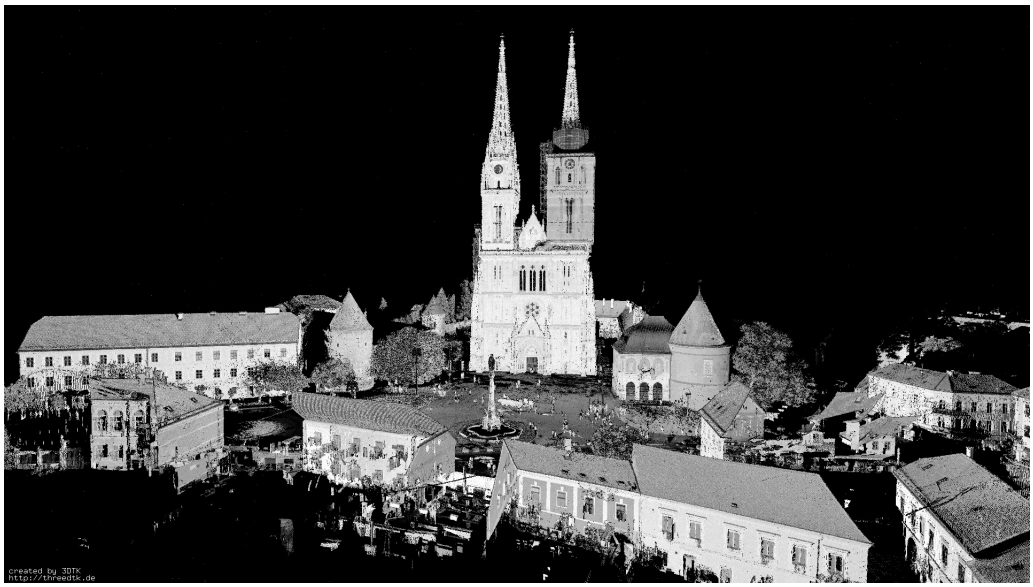
Postoji poveći broj već unaprijed definiranih vozila koja se mogu koristiti. Mogu se koristiti kao nositelji senzora ili kao ostali sudionici u prometu. Carla ima dobro definirana prometna pravila te semafore da bi simulacija izgledala što vjernije.

3.3. Point cloud

Oblak točaka je skupina podataka koji definiraju neki objekt u prostoru. Oblak točaka se obično generira pomoću trodimenzionalnog skenera. Oblaci točaka imaju veliku primjenu u rekonstrukcijama predmeta, vizualizaciji, animaciji, virtualnoj i proširenoj stvarnosti te industrijskoj proizvodnji i kontroli kvalitete. U trodimenzionalnome kartezijevom sustavu svaka točka je definirana s tri atributa, a to su njene x, y i z koordinate. Uz te osnovne podatke svaka točka također može sadržavati i podatke o njenoj boji.



Slika 3.10: Ilustracija točke u kartezijevom koordinatnome prostoru



Slika 3.11: Oblak točaka katedrale u Zagrebu

Na slici 3.11 se vidi skup točaka koji opisuje katedralu u Zagrebu te okolne objekte. Skup se sastoji od oko 22 milijuna točaka.

Skup točaka je mnogo jednostavnije koristiti za mapiranje objekata od slika zato što se lakše može obraditi na računalu. Ti podaci su zapravo spremljeni u tekstualne datoteke pa su lako prenosivi i čitljivi.

Konkretno će naše metode koristit oblak točaka prikupljen s rotirajućim laserima na vozilu. Jedan taj skup točaka predstavlja stanje okoline vozila u jednome trenutku.

3.4. Referentni podaci

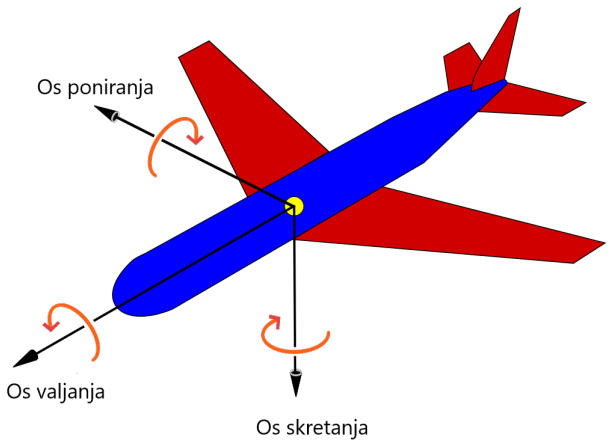
Referentni podaci su oni podaci s kojima se uspoređuju rezultati metoda. Ti referentni podaci su generirani u simulatoru te predstavljaju lokaciju i rotaciju vozila u jednome trenutku. Referentni podaci se zapravo sastoje od lokacije i rotacije vozila.

Lokacija

Lokacija vozila je također definirana kao točka u kartezijevom koordinatnome prostoru. Sastoji se od x, y i z koordinata. Slično kao prikazano na slici 3.10.

Rotacija

U trodimenzialnome prostoru objekt se zapravo može rotirati oko beskonaćnoga broja osi ali se u pravilu uzimaju 3 statičke osi. Te osi se nazivaju os skretanja (eng. yaw), os poniranja (eng. pitch) i os valjanja (eng. roll).



Slika 3.12: Ilustracija osi rotiranja

Os rotacije je os koja prolazi u smjeru kretanja vozila (x os), os poniranja je zapravo os okomita s os rotacija (z os), dok je os skretanja okomita na obje prethodne osi (y os). Te osi su ilustrirane na slici 3.12.

3.5. Izvor podataka

Kao izvor podataka za algoritme se koristi već prije spomenuti LIDAR senzor. Podaci koje nam vrati senzor su u obliku već spomenutog skupa točaka (eng. point cloud). Lidar senzor je zapravo vertikalan skup lasera.

Građa lidara

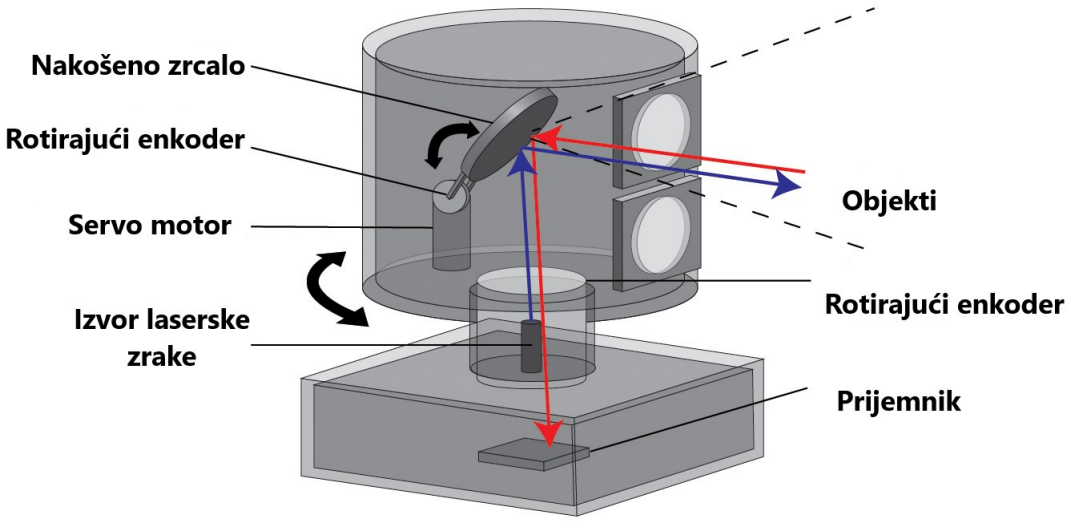
Lidar se sastoji od sljedećih glavnih dijelova:

1. Nakošeno zrcalo
2. Servo motor
3. Izvor laserske zrake
4. Detektor laserske zrake
5. Optički rotirajuči enkoder

Izgled je ilustriran na slici 3.13. Ovaj primjer ima samo jedan laser. Servo motor rotira kučište gdje se nalazi laser te tako omogućuje očitanje u svih 360 stupnjeva. Danas se zahtjeva da brzine rotacije budu veće od 1000 okretaja u minuti. Nakošeno zrcalo služi za usmjeravanje zrake. Izvor laserske zrake proizvodi lasere u pulsevima. Te zrake se nalaze u infracrvenome valnome području te su sigurne za vid i male energije. Laserski detektor detektira laserske pulseve te dekodira podatke.

Rad lidara

Lidar može sadržavati razne sustave za lokalizaciju i orijentaciju poput GPS-a, inercijskih sustava (IMU) i ostalih. Koristi dvodimenzionalno očitanje u horizontalnom smijeru i vertikalno polje vida za kreiranje panorame u 360 stupnjeva. Za dekodiranje podataka se koristi vrijeme povratka laserske zrake i valna duljina ako podržava detekciju boje. Postoji mogućnost nepotpunih očitanja zbog razlike između frekvencije rotacije i frekvencije očitanja senzora. Općenito se udaljenost predmeta s laserom



Slika 3.13: Ilustracija građe lidara

može izračunati formulom 3.1 gdje je c brzina svijetlosti, a t vrijeme povratka pulsa zrake.

$$D = \frac{ct}{2} \tag{3.1}$$

Ovakav način je pogodan za srednje do velike udaljenosti, dok za vrlo male udaljenosti se koriste druge metode zato što se teško računaju vremena povratka zrake.

Tipovi lidara

Jedna od podjela lidara je po tome gdje se koristi. Tako imamo zračne i zemaljske lidare. Zračni lidari se koriste za mapiranje topologije prostora. Zasada je to najbolji način za mapiranje površina zato što omogućuje filtriranje vegatacije te sa tako lako mogu dobiti topološki podaci. Takvi lidari su pričvršćeni za avion, dron ili helikopter. Zemaljski lidari su postavljeni statički na zemlji. Koriste se u forenzici i rekonstrukciji objekata.

Primjena

Lidar se koristi u geodeziji, geografiji, geologiji, seizmologiji, fizici a u zadnje vrijeme prilikom lokalizacije autonomnih vozila. Također se koriste u agrokulturi prilikom navodnjavanja ili sadnje. Koristi se prilikom klasifikacije biljaka uz pomoć strojnog

učenja. U arheologiji se primjenjuje od mapiranja arheoloških nalazišta koja su fizički nedostupna pa do mapiranja oštećenih predmeta.

Prednosti i nedostatci

Prednosti su što se može iskoristiti u mnoge svrhe i možemo dobiti razne informacije o okolini. Također su podaci koji se dobiju vrlo lako obradivi i čitljivi. Jedini nedostatak je brzina obrade tih podataka. Za velike skupove točaka se rijetko koristi obrada u stvarnome vremenu zato što je potrebna velika računalna snaga.

Parametri simuliranoga lidara

S obzirom da je lidar simuliran njegovi parametri nisu vezani uz njegovu izvedbu nego se mogu postaviti po želji. Jedina razlika je što simulirani lidar nema kašnjenje tj. njegova očitanja su gotovo trenutna što u stvarnome svijetu nikako ne može biti. Možemo mu postaviti sljedeće parametre:

- broj kanala - broj vertikalnih laserskih zraka
- donja granica polja vida - koliko nisko su orijentirani laseri
- gornja granica polja vida - koliko visoko su orijentirani laseri
- ukupan broj točaka - ukupan broj točaka po laseru u pojedinom očitanju
- frekvencija rotacije - koliko često se laseri rotiraju
- vremenski korak - koliko često se podaci prikupljaju

3.6. Prikupljanje podataka

Podaci su prikupljeni tako da se simulator pokrene u poslužiteljkom načinu rada te se tada pokreće klijentska skripta napisana u python jeziku. Ta skripta uspostavi kontakt s poslušiteljem te se tako šalju naredbe. Te naredbe će zapravo stvoriti naše vozilo, ostale sudionike i senzor. Nakon što smo prikupili dovoljno podataka skripta će obrisati stvorene objekte i spremiti podatke u datoteke. Tada simulator može prekinuti s radom te se ti podaci mogu obrađivati na bilo koji način.

Pokretanje simulatora

Simulator Carla se pokreće pomoću python skripte zbog boljeg upravljanja parametrima ali se zapravo sastoji od naredbe pokazane u primjeru 1. Simulacija se po-

```
CarlaUE4.exe \
  /Game/Carla/Maps/Town01 \
  -quality-level=Low \
  -benchmark -fps=15 \
  -windowed -ResX=800 -ResY=600 \
  -carla-port=2000 \
```

Izvorni kod 1: Carla naredba

kreće u mapi pod nazivom Town01. Kvaliteta je postavljena na najnižu vrijednost kao i broj slika u sekundi zbog boljih performansi izvođenja. Prozor smo postavili na vrlo malu rezoluciju od 800 pixela širine i 600 pixela visine također zbog boljih performansi. Vrlo važan parametar je sučelje preko kojega klijentski program komunicira s poslužiteljem. Ovdje je definiran kao 2000.

Klijentska kripta

Referentni i testni podaci su prikupljeni iz simualtora ali iz različitih izvora. Referentni podaci su prikupljeni iz samoga simulatora dok su testni podaci prikupljeni pomoću senzora.

```
1  class CarlaProp:
2    spawn_delay = 1.0
3    host = "localhost"
4    port = 2000
```

Izvorni kod 2: Carla postavke

U primjeru izvornoga koda 3 klijent se spaja na Carla poslužitelj čiju smo lokaciju (IP adresu i sučelje) definirali u klasi `CarlaProp`. Također postavljamo sinkroni način rada simulatora, a razlog je taj što želimo upravljati frekvencijom slanja podataka iz poslužitelja prema klijentima. Varijabla `self.world` služi za izvođenje svih operacija koje su vezane uz svijet.

Svaka mapa ima već unaprijed definirane točke stvaranja tj. koordinate u svijetu na kojima možemo stvoriti objekte. Te koordinate se nalaze na cestama. Njih možemo dobiti naredbom prikazanom na primjeru izvornoga koda 4.

Sljedeće što slijedi je stvaranje ostalih sudionika prometa tj. ostalih vozila. Carla

```
  def connect_to_carla(self):
    self.client = carla.Client(CarlaProp.host, CarlaProp.port)
    self.client.set_timeout(2.0)
    self.world = self.client.get_world()
    settings = self.world.get_settings()
    settings.synchronous_mode = True
    self.world.apply_settings(settings)
```

Izvorni kod 3: Uspostava konekcije s poslužiteljem

```
    def get_spawn_points(world):
      return list(world.get_map().get_spawn_points())
```

Izvorni kod 4: Dohvaćanje liste koordinata stvaranja

ima već unaprijed definirane nacrte raznih objekata.

```
def get_vehicle_blueprints(world):
  blueprints = world.get_blueprint_library().filter('vehicle.*')
  blueprints = [x for x in blueprints if int(x.get_attribute('number_
  return [x for x in blueprints if not x.id.endswith('isetta')]
```

Izvorni kod 5: Dohvaćanje nacrta vozila

Na 5 se vidi kako koristimo knjižnicu nacrta da bi filtrirali nama potrebne nacrte. Koristiti će se samo vozila koja imaju 4 kotača.

Koristeću točke stvaranja i nacrte vozila sada se mogu ta vozila stvoriti u svijetu. Na 6 se koristeći petljom stvara unaprijed zadan broj ostalih sudionika definiranih u varijabli klase `self.npc_number`. Njihove reference se tada spremaju u listu zato što se na kraju izvođenja moraju uništiti. Stvaranje instance nacrta se izvodi naredbom na liniji 8. Također smo svakoj instanci definirali autonomni način rada na liniji 9.

Sada se definira vozilo koje zapravo promatramo tj. koje ima na sebi lidar senzor. To se radi na približno jednak način kao u primjeru 6. Izvorni kod je prikazan na primjeru 7.

Sada slijedi pronalazak nacrta za lidar senzor, postavljanje njegovih atributa, njegovo instanciranje i postavljanje na promatrano vozilo. Izvorni kod je prikazan na 9.

```
1  def spawn_npcs(self):
2    blueprints = utils.get_vehicle_blueprints(self.world)
3    points = self.spawn_points[1:self.npc_number+1]
4    for i in range(self.npc_number):
5      actor_blueprint = random.choice(blueprints)
6      print(f"\t{actor_blueprint}")
7      actor_spwn_point = points[i]
8      spawned_actor = self.world.try_spawn_actor(actor_blueprint, act
9      spawned_actor.set_autopilot()
10     self.npcs.append(spawned_actor)
11   self.tick()
```

Izvorni kod 6: Stvaranje ostalih vozila

```
1  def spawn_actor(self):
2    spawn_point = self.spawn_points[0]
3    actor_blueprint = utils.get_vehicle_blueprint(self.world.get_blue
4    print(f"\t{actor_blueprint}")
5    self.actor = self.world.spawn_actor(actor_blueprint, spawn_point)
6    self.actor.set_autopilot()
7    self.tick()
```

Izvorni kod 7: Stvaranje promatranoga vozila

Postavke LIDAR senzora se nalaze u klasi `LIDARProp`.

Na liniji 2 dohvaćamo nacrt lidar senzora. Tada od linije 3 do 9 postavljamo zadane postavke nad nacrtom. Konačno na liniji 11 stvaramo instancu senzora ali metodi predajemo dodatan parametar `attach_to` koji je je jednak referenci na naše vozilo. Također umjesto stvarnih koordinata, za lokaciju senzora postavljamo lokaciju relativnu naspram lokacije vozila. NA liniji 12 postaljamo metodu `lidar_callback()` kao metodu koju će simulator pozvati svaki puta kada senzor očita okolinu i pošalje podatke. Spremanje podataka se izvršava tek nakon što smo sakupili konačan broj očitanja. Za spremanje podataka u datoteke se koristi posebna klasa `DataSaver` pokazana na primjeru 10.

Datoteke s informacijama o lokaciji vozila se sastoje od 2 reda. Prvi red sadrži vremensku oznaku a drugi sadrži lokaciju i transformaciju koji su opisani u prijašnjem

```
class LIDARProp:
    sensor_tick = str(0.0)
    channels = str(360)
    laser_range = str(1500.0)
    rotation_frequency = str(20.0)
    points_per_second = str(600_000)
    upper_fov = str(45.0)
    lower_fov = str(-80.0)
    location = carla.Transform(carla.Location(x=0, y=0, z=4))
```

Izvorni kod 8: LIDAR atributi

```
    def connect_LIDAR(self):
        blueprint = utils.get_lidar_sensor_blueprint(self.world.get_blu
        blueprint.set_attribute('sensor_tick', LIDARProp.sensor_tick)
        blueprint.set_attribute('channels', LIDARProp.channels)
        blueprint.set_attribute('range', LIDARProp.laser_range)
        blueprint.set_attribute('rotation_frequency', LIDARProp.rotatio
        blueprint.set_attribute('points_per_second', LIDARProp.points_p
        blueprint.set_attribute('upper_fov', LIDARProp.upper_fov)
        blueprint.set_attribute('lower_fov', LIDARProp.lower_fov)
        utils.print_sensor_blueprint_data(blueprint)
        self.lidar = self.world.try_spawn_actor(blueprint, LIDARProp.li
        self.lidar.listen(lambda data: self.lidar_callback(data))
        self.tick()
```

Izvorni kod 9: Stvaranje LIDAR senzora

poglavlju. Datoteke koje sadrže podatke o jednome očitanju se nalaze u tekstualnim datotekama s ekstenzijom .ply te se sastoji od zaglavlja i po jedan redak za svaku točku u očitanju. Također postoji još jedna datoteka koja samo sadrži relativnu transformaciju između senzora i vozila. Ona se nalazi u direktoriju output.

```python
class DataSaver:

  def __init__(self):
    self.pc_path = 'output/point_clouds'
    self.trans_path = 'output/actor_transforms'
    self.rel_path = 'output/relative_transform.txt'

  def initialize_folders(self):
    utils.create_directory(self.trans_path)
    utils.create_directory(self.pc_path)

  def save(self, scans):
    self.initialize_folders()
    self.save_rel_trans()
    start_time = time.time()

    with ThreadPoolExecutor(max_workers=10) as executor:
      jobs = list()
      for i, (scan, transform) in enumerate(scans[1:]):
        job = executor.submit(self.process_pair, scan, transform, i)
        jobs.append(job)
      for future in as_completed(jobs):
        future.result()
```

Izvorni kod 10: Klasa za spremanje podataka

```
  def process_pair(self, scan, trans, i):
    scan_path = f'{self.pc_path}/{scan.frame_number:06d}.ply'
    self.save_scan(scan, scan_path, i + 1)
    trans_path =  f'{self.trans_path}/{scan.frame_number:06d}.txt'
    self.save_transform(trans, scan.timestamp, trans_path, i + 1)

  def save_rel_trans(self):
    with open(self.rel_path, 'w+') as file:
      file.write(utils.transform_to_string(LIDARProp.location))

  def save_scan(self, scan, path, i = 0):
      scan.save_to_disk(path)

  def save_transform(self, trans, timestamp, path, i = 0):
    with open(path, 'w+') as file:
      content = f"{timestamp}\n{utils.transform_to_string(trans)}"
      file.write(content)
```

Izvorni kod 11: Klasa za spremanje podataka - nastavak

4. Algoritmi lokalizacije

4.1. Obitelj algoritama

Koruišteni algoritmi su ...

4.2. Opis algoritama

Algoritmi su implementirani u ...

4.3. Algoritmi

4.3.1. Algoritam 1

Opis algoritma

Ovaj algoritam radi tako da ...

Rezultat algoritma

Rezultati algoritma su ...

Evaluacija rezultata

U usporedbi s ground truth ovaj algoritam ...

4.3.2. Algoritam 2

Opis algoritma

Ovaj algoritam radi tako da ...

Rezultat algoritma

Rezultati algoritma su ...

Evaluacija rezultata

U usporedbi s ground truth ovaj algoritam ...

5. Eksperimentalni rezultati

Eksperimentalni rezultati ...

6. Zaključak

Zaključak.

LITERATURA

Lokalizacija autonomnog vozila u simuliranom urbanom okruženju

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: lokalizacija, simulacija

Title

Abstract

Abstract.

Keywords: simulation, localization.