Estimacija kretanja koristeći monocular vizuelnu odometriju

Ilma Okanović
Odsjek za automatiku i elektroniku
Elektrotehnički fakultet Sarajevo
Sarajevo, Bosna i Hercegovina
iokanovic1@etf.unsa.ba

Amila Šoše
Odsjek za automatiku i elektroniku
Elektrotehnički fakultet Sarajevo
Sarajevo, Bosna i Hercegovina
asose2@etf.unsa.ba

Amir Nakić
Odsjek za automatiku i elektroniku
Elektrotehnički fakultet Sarajevo
Sarajevo, Bosna i Hercegovina
anakic1@etf.unsa.ba

***Sažetak*—Vizuelna odometrija (VO) u prethodnom periodu dobiva veliku pažnju, s ciljem postizanja efikasne i tačne procjene kretanja sistema. Nedavna istraživanja pokazuju da algoritmi zasnovani na optimizaciji kretanja postižu visoku preciznost kada im dodijelimo dovoljan broj infromacija, ali i da povremeno pate od divergencije pri rješavanju izrazito nelinearnih problema. Akcenat je stavljen na spajanje vizuelnih informacija i unaprijed integriranih mjerenja u zajedničkom okviru za optimizaciju. U ovom radu ćemo obrazložiti algoritam za procjenu kretanja automobila u CARLA simulacijskoj platformi koristeći MONOCULAR kameru. CARLA simulator je razvijen s ciljem podržavanja razvoja, obuke i validacije sistema autonomne vožnje. Podržava fleksibilne specifikacije senzorskih aparata, uvjeta okoline te potpunu kontrolu svih statičkih i dinamičkih aktera.**

***Ključne riječi*—Vizuelna odometrija (VO), Monocular, Carla**

I. Uvod

U robotici, automatici i sličnim granama nauke, procjena kretanja je ključna. Posljednih godina, praćenje kretanja je temeljni problem u istraživanju proširene stvarnosti, samoupravljajućih vozila, bespilotnih letjelica, planiranja proizvodnje itd. Kao rezultat daje položaj, orijentaciju i brzinu objekta u nepoznatom okruženju. Sa ovim informacijama, autonomna oprema na osnovu procjene trenutne pozicije može donijeti odluku o narednom koraku vozila, letjelice ili drugog uređaja. Ova tehnologija postaje sveprisutna kako u industrijskoj primjeni, tako i u svakodnevnom životu. Tipični sistemi za praćenje kretanja uključuju više senzora koji se koriste za pribavljanje informacija o kretanju sistema u određenoj okolini i računara koji se koristi za obradu informacija pribavljenih sa senzora, te za davanje procjene o položaju i orijentaciji sistema.

Za procjene trenutnih pozicija robota i uređaja sa mogućnošću kretanja, koriste se različiti senzori poput GPS-a, brojača kilometara i kamera. Posljednjih godina algoritam vizuelne odometrije (VO) pobuđuje sve veće interesovanje jer prevazilazi nedostatke drugih senzora i omogućava robusan rad [1].

Vizuelna odometrija (VO) je preciznija od ostalih metoda za procjenu položaja uređaja, jer koristi dugotrajno praćenje vizuelnih karakteristika, međutim, pokazalo se da zakaže u slučajevima brzih pokreta, nedostatka tekstura scene te naglih promjena osvjetljenja. Nadalje, monokularni VO sistemi ne mogu procijeniti apsolutnu skalu kretanja zbog ograničenih mogućnosti projekcije prirodne kamere [1].

Najnoviji VO algoritmi se mogu klasificirati na pristup zasnovan na filtriranju, koji filtrima dodaje vizuelna i inercijalna mjerenja, te na algoritme čiji se pristup zasniva na optimizaciji, koristeći nelinearnu optimizaciju za procjenu stanja. Prvi pristupi koriste prošireni Kalmanov filter, koji predstavlja stanje kao normalnu raspodjelu sa srednjom vrijednošću i kovarijansom. Pristup zasnovan na filtriranju može procijeniti trenutne pozicije dovoljno brzo za aplikacije u stvarnom vremenu, međutim, oni su manje precizni od pristupa zasnovanog na optimizaciji zbog aproksimacije u koraku ažuriranja. Nedavno su razvijeni algoritmi zasnovani na optimizaciji koji imaju u cilju povećanje preciznosti, ali zahtijevaju veće troškove kalkulacija, te nisu pogodni za proračune kada je detekcija loša ili inicijalizacija nije ispravna. Svakako, kao i u svemu, postoji kompromis između validnosti performansi i brzine, te je teško optimizirati sve parametre u fazi inicijalizacije i ažuriranja.

Vizuelna odometrija nalazi svoju široku primjenu na dronovima, što je mnogo izazovnije nego na automobilima zbog težeg upravljanja usljed nametnutih ograničenja. Ove metode primijenjene na autonomno upravljanje drnova su korisne u slučajevima prirodnih nesreća, kada se GPS ne može primijeniti, s ciljem zamjene čovjeka u potrazi za preživjelim žrtvama zemljotresa, cunamija i slično. Kod takvih uređaja, izazov predstavlja ne samo percepcija, nego i upravljanje. Osnovno pitanje je pitanje lokalizacije u zatvorenim prostorima kada GPS nije primjenljiv. Dva su rješenja - ili izgradnja GPS-a koji će biti primjenljiv indoor-GPS, ili oslanjanje na snimak sa kamere koji se dalje procesira korištenjem VO. Razlika između pristupa je što je u prvom slučaju uređaj "slijep", oslanjamo se na snimak izvana, dok drugi omogućava uređaju viđenje okruženja.

Postoje ograničenja koja su do sada ometala razvoj ovog polja nauke: algoritmi za percepciju nisu robusni, algoritmi i senzori imaju velike latentnosti (50-200ms), kontrola i percepcija su se do sada razmatrale odvojeno, potrebno je ostvariti kretanje uređaja kroz područja koja maksimiziraju teksturu prizora radi lokalizacije, pojave zamućenosti usljed kretanja. Glavna ideja key-frame vizuelne odometrije jeste praćenje kretanja korištenjem jedne kamere, te simultano praćenje okruženja. Ideja je prvenstveno vršenje triangulacije tačaka

*Kodove je moguće pronaći i preuzeti sa repozitorija: Mono-VO

korištenjem npr. 5-point algoritma, te lokalizacija svakog pronađenog frame-a prema postojećem cloud-u.

Dalje, postoji pitanje vršenja frame-to-frame estimacije kretanja. Postoje dva pristupa: feature-based metode i direktne (photometric) metode. Prva metoda se zasniva na odvajanju značajki prethodnog i trenutnog frame-a, te računanju kretanja minimizirajući reprodukcijsku grešku. Direktnom metodom se postiže veća preciznost i robusnost iz razloga što se koriste sve informacije sa slike. Ako se poveća frame-rate kamere, algoritam konvergira brže. Da bi ovaj algoritam radio efikasno, frame-to-frame kretanje mora biti vrlo ograničeno [2].

U ovom radu je predloženo rješenje koje koristi optimizaciju vizuelnog mjerenja kretanja automobila, zajedno sa inicijalizacijom početnih parametara mjerenja. Za rad u stvarnom vremenu, troškovi optimizacije za procjenu putanje ne bi trebali sadržavati veliki broj parametara. Smanjenjem broja parametara omogućavamo bolju optimizaciju što rezultira poboljšanom tačnošću samih proračuna. Osim tradicionalnih navigacijskih tehnologija, napredak u računarskom svijetu je motivirao nastanak niza aplikacija zasnovanih na praćenju lokacije, popunt simultane lokalizacije i mapiranja, te pružio mnoštvo novih mogućnosti za primjenu.

II. Metodologija

VO algoritmi su fokusirani na preciznu procjenu pozicije uređaja spajanjem vizuelnih informacija. Kamera pruža globalne i stacionarne informacije svijeta.

A. Monocular

Monocular kamera je uobičajena vrsta vision-senzora koji se koristi u aplikacijama automatizirane vožnje. Kada se postavi na vozilo, ova kamera može prepoznati predmete, otkriti granice trake i pratiti objekte kroz scenu.

Monocular vizuelna odometrija računa i relativno kretanje i 3D strukturu iz 2D podataka. Kako je apsolutna skala nepoznata, udaljenost između prva dva prikaza kamere se obično predstavlja kao jedan prikaz. Kako nova slika dolazi, relativna skala i prikaz u odnosu na prva dva frame-a se određuje koristeći znanja o 3D strukturi. U posljednjoj deceniji, uspješno su pribavljeni podaci putem jedne kamere korištene na velikim udaljenostima koristeći i prespektivu i višesmjernost kamera. Slični radovi se mogu podijeliti u tri kategorije: metode zasnovane na značajkama, metode zasnovane na izgledu i hibridne metode [6].

Metode zasnovane na značajkama su bazirane na istaknutim i ponovljivim značajkama koje se prate preko frame-ova. Metode zasnovane na izgledu koriste informacije o intenzitetu svih piksela na slici ili dijelova slike, a hibridne metode predstavljaju kombinaciju prethodne dvije.

Prva monocular vizuelna odometrija u realnom vremenu je korištena od strane Nister-a. Koristili su RANSAC algoritam za uklanjanje outlier-a i 3D u 2D estimaciju kamere za određivanje novog prikaza. Novost koju su oni uveli je upotreba minimalnog algoritma 5 tačaka za izračunavanje hipoteze u RANSAC-u. Nakon toga, RANSAC sa pet tačaka je postao vrlo popularan u metodama vizuelne odometrije.

Naredni pristupi su se bazirali na svesmjernim slikama katadioptrijske kamere i optičkog toka, prozorskog snopa zraka za rekonstrukciju i kretanja i 3D mape, ponovo koristeći RANSAC sa 5 tačaka za uklanjanje outliera.

RANSAC algoritam se koristi s ciljem uklanjanja outlier-a (odstupanja od nekog određenog modela, pojavljuju se zbog promjene perspektive, skale, osvjetljenja, šuma na slici itd). U slučaju velikih odstupanja, metod najmanjih kvadrata pokušava da uzme sve tačke jednako u obzir, što nije efikasno, jer se može dogoditi da mjerenje nije validno pod nekim uslovima. RANSAC ovaj problem rješava tako što ne postavlja jednake uvjete za outliere i inliere (tačke koje vjerodostojno preslikavaju model). RANSAC uzima dvije nasumične tačke, uspostavlja model između njih, izračunava ga i određuje grešku u odnosu na sve tačke data-seta od interesa. Samo dvije tačke koje su na pravcu, odnosno kroz koje prolazi pravac će imati grešku nula, sve ostale će imati grešku 1. Tu grešku izračunavamo kao kriterij, te ako je dovoljno velika, algoritam nastavlja dalje. Postupak se ponavlja dok se ne dobije dovoljan broj inliera. Cilj je dostizanje maksimalnog broja inliera, tokom k iteracija. U slučaju N tačaka, kompleksnost algoritma je $N(N-1)/2$. Najčešće se izvodi do 1000 iteracija.

B. CARLA

Carla je open-source simulator, razvijen i namijenjen za podršku razvoja, obuke, istraživanja i validacije sistema autonomne vožnje. Pored pristupa kodu, CARLA nudi i otvorenu digitalnu imovinu (rasporedi urbane okoline, zgrada i vozila) koji su stvoreni za tu svrhu i mogu se slobodno koristiti. Simulacijska platforma podržava fleksibilne specifikacije senzora, potpunu kontrolu svih statičkih i dinamičnih aktera, te uslova okoline. Istaknute karakteriste CARLA simulatora su:

- skalabilnost preko arhitekture server-multi-klijent (više klijenata u istom ili u različitim čvorovima može kontrolirati različite aktere),
- fleksibilni API (CARLA nudi API koji omogućavaju korisnicima kontrolu svih aspekata povezanih sa simulacijom, uključujući stvaranje prometa, ponašanje pješaka, vremenske prilike, senzore i mnogo toga),
- paket senzora za autonomnu vožnju (korisnici mogu konfigurisati različite setove senzora, uključujući LIDAR, više kamera, senzore dubine i GPS-a),
- brza simulacija za planiranje i kontrolu (ovaj način onemogućava prikazivanje tako da nudi brzo izvršavaje simulacije prometa i ponašanja na ulicama, za koje grafici nisu neophodni),
- generisanje mapa (korisnici mogu jednostavno generisati vlastite mape slijedeći OpenDrive standard pomoću alata kao što je RoadRunner),
- simulacija saobraćajnog scenarija (ScenarioRunner omogućava korisnicima da definiraju i izvršavaju različite prometne situacije na osnovu modularnog ponašanja),
- integracija ROS-a (omogućena je integracija CARLA simulatora sa ROS-om putem ROS-bridge-a),

- polazne osnove za autonomnu vožnju

Slika 1. Izgled okruženja Carla simulatora

Klijent je modul kojeg korisnik pokreće s ciljem pribavljanja informacija ili promjena u simulaciji. Klijent radi na određenom portu i komunicira sa serverom putem terminala. Može biti istovremeno pokrenuto više klijenata.

Svijet je objekat koji predstavlja simulaciju. Djeluje kao apstraktni sloj koji sadrži glavne metode za upravljanje akterima, promjenu vremena, dobivanje trenutnog stanja u svijetu itd. Postoji samo jedan svijet po simulaciji, a biva uništen i zamijenjen novim pri promjeni mape.

Akteri predstavljaju sve objekte koji igraju ulogu u simulaciji kao što su automobili, pješaci, senzori, saobraćajni znakovi i semafori.

Blueprints (nacrti) su unaprijed generisani oblici aktera, neophodni za njihovo stvaranje. To su u osnovi modeli sa određenim animacijama i setom atributa. Neki od njih mogu biti određeni od strane korisnika, a neki se ipak ne mogu mijenjati. Postoji Blueprint biblioteka koja sadrži sve dostupne nacrte, kao i informacije o njima.

Mapa je objekat koji predstavlja simulacijski svijet, uglavnom grad. U CARLA simulatoru je dostupno 8 mapa, a sve koriste OpenDRIVE 1.4 standard za opisivanje puteva.

Putevima, trakama i raskrsnicama upravlja PythonAPI kojem se pristupa sa klijentske strane. Koriste se zajedno sa značajnim tačkama puta, s ciljem osiguravanja vozila navigacijskom stazom.

Senzori čekaju simulacijski događaj, a zatim prikupljaju podatke iz simulacije, te pozivaju funkciju koja definira način upravljanja podacima. Prema tome, senzori dobavljaju različite vrste podataka. Također, senzor je akter vezan za vozilo, pa na taj način prikuplja i podatke iz okoline. Dostupni senzori su definirani u Blueprint biblioteci.

Prije nego što pređemo na implementaciju algoritma estimacije kretanja osvrnut ćemo se na korake instalacije (eng. build) Carla simulatora u UnrealEngine editoru. Premda je osnovne korake moguće pronaći na [4] postoji par detalja koji nisu naglašeni. Napominjemo da sljedeće opaske će se odnositi na resurse kojim smo raspolagali: laptopi sa procesorima intel core i5 i intel core i7, integrisane grafičke karte, te Windows 7 i Windows 10 operativne sisteme. Ono što je moguće odmah primijetiti jeste da jedna od uputa dokumentacije [4] nije ispunjena, a odnosi se na grafičku kartu. Naime, integrisane grafičke kartice koje smo imali na raspolaganju su uspjevale postići tek 3 fps, što nije ni približno dovoljno kako bismo mogli govoriti o real-time simulaciji. No, kako ćemo pojasniti kasnije, ovaj problem smo odlučili riješiti na nešto drugačiji način. Premda u samoj dokumentaciji je navedeno da je potrebo imati 30 GB memorijskog prostora predviđenog za pohranu sveukupnog software-a, ipak se pokazalo da je potrebno puno više (čak dvostruko više) memorijskog prostora ako uključimo instalaciju svih programa potrebnih za instalaciju samog simulatora.

Kada je riječ o samom operativnom sistemu, prilikom preuzimanja Carla repozitorija primjećeno je da sve novije verzije Carla simulatora za Windows operativni sistem su obilježene sa terminom "experimental". Razlog jeste što je Carla simulator primarno bio razvijan za Linux operativni sistem, te je potom prilagođen Windows operativnom sistemu. Također, prilikom odabira verzije Carla simulatora potrebno je obratiti pažnju na moguće bug-ove i izmjene, pa je preporuka za odabranu verziju (u našem slučaju to je bila Carla 9.10.) pratiti tzv. [5]. Zbog korištenja Windows 7 operativnog sistema bilo je potrebno izvršiti upgrade Windows PowerShell na verziju koja podržava komandu Expand-Archive što je slučaj za verzije v5+. U protivnom, neće biti moguće izvršiti ekstrakciju file-ova potrebnih za instalaciju (build) Carle. Ova napomena odnosi se na korak "make launch" koji pozivamo po završetku pripreme za samu instalaciju, te posjedovanje starije verzije PowerShell će rezultirati greškom u ovom koraku. Još jedna napomena koja je posljedica onoga na što smo i sami naišli, jeste da prilikom pribavljanja source folder-a na osnovu kojih se vrši build (moguće ih je pronaći u Build folder-u) nedostajao je folder xerces-c-3.2.3-source stoga ga je bilo potrebno pronaći i manuelno dodati u Build folder, pa potom njega samog build-ati.

Da bi se izvršilo povezivanje klijenta i servera (Carla u okviru UE4 editora), nakon uspješno završenog koraka "make lauch", poziva se naredba "make PythonAPI". Ono što trebamo znati u vezi sa ovom komandom jeste da ona treba rezultirati kreiranje dist foldera unutar "PythonAPI/carla", a koji će sadržavati .egg file potreban za povezivanje klijenta i pokretanja python skripti. Ukoliko ne bude kreiran pomenuti .egg file moguće ga je kreirati pozivom komanda "py setup.py install" iz foldera "PythonAPI/carla". Još jednom naglašavamo da tokom postupka instalacije postoji mogućnost pojave drugih vrsta grešaka ili neočekivanih rezultata koji nisu pokriveni samom dokumentacijom, no ovdje su navedeni samo oni čije korekcije smo morali sami potražiti.

III. Implementacija

S obzirom na nemogućnost real-time izvršavanja realiziranog algoritma, to je bilo potrebno prethodno snimiti rutu kretanja vozila u okviru Carla simulatora. Za ovu svrhu kreirana je skripta *snimanje.py* koja pored snimanja samih frame-ova, također, vrši i pohranjivanje lokacije vozila odnosno kamere u trenutku pohranjivanja frame-a. Osnovna ideja za pokretanje

vozila preuzeta iz skripte *manual_control.py* koju je moguće pronaći unutar *PythonAPI/examples* foldera, a sastojala se od upravljanja vozilom tipkama na tastaturi ili automatskom kretanju vozila. Kao i za stvarnu kameru, tako i za kameru unutar Carla simulatora, su njeni intrinzični parametri. Tako ako pretpostavimo da je širina frame-a 1280 px, visina 780 px i fov (eng. field of view) 90 stepeni, intrinzični parametri fokalna dužina f (eng. focal length) i centari c_u i c_v se mogu odrediti prema relacijama 1 i 2:

$$f = \frac{sirina}{2tan(fov\frac{\pi}{360})} = \frac{1280}{2tan(90\frac{\pi}{360}\frac{180}{\pi})} = 640 \qquad (1)$$

$$c_x = \frac{sirina}{2} = 640 \quad i \quad c_y = \frac{visina}{2} = 360 \qquad (2)$$

Na osnovu navedenih podataka, matrica kamere K odnosno matrica instrinzičnih parametara kamere postaje:

$$K = \begin{bmatrix} f & 0 & c_u \\ 0 & f & c_v \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 640 & 0 & 640 \\ 0 & 640 & 360 \\ 0 & 0 & 1 \end{bmatrix} \qquad (3)$$

Algoritam estimacije kretanja moguće je podijeliti na par koraka koji se ponavljaju za svaki par frame-ova odnosno slika, pa ako pretpostavimo da je trenutno razmatrana k-ta iteracija imamo:

1) pribavljanje frame-a odnosno slike I_k,
2) izdvajanje značajki pribavljenjog frame-a I_k,
3) uparivanje značajki frame-a I_k i frame-a I_{k-1},
4) estimacija esencijalne matrice E, te određivanje matrica rotacije R i vektora translacije t,
5) triangulacija izdvojenih značajki na osnovu dobivene matrice R, vektora t i matrice intrinzičnih parametara kamere K,
6) procjena relativne skale r i korekcija vektora translacije t,
7) iscrtavanje nove tačke trajektorije,
8) ponavljanje koraka 1) do 7)

Premda se sami koraci algoritma čine intuitivnim, potrebno je znati da pojedini koraci mogu koristiti veoma sofisticirane i kompleksne alate za njihovo izvršavanje. Kroz kurs Robotske vizije imali smo priliku pristupiti internim funkcijama i organizaciji pojedinih alata, no za ovaj rad odlučeno je koristiti već prethodno implementirane funkcije s obzirom da je cilj bio dostići što veću brzinu izvršavanja algoritma. Lako je zaključiti da će to biti postignuto ukoliko odaberemo metode koje, sa stalnim porastom interesovanja za ovu oblast, bivaju usavršavane.

Ovdje možemo spomenuti da realizacija koraka 1) do 6) je izvršena tako da svi potrebni podaci o trenutnom frame-u, značajkama izdvojenim sa trenutnog i prethodnog frame-a, vrste detektora, intrinzičnim parametrima kamere i sl. budu pohranjeni kao atributi klase $VisualOdometry$. S obzirom na to da će biti potrebno nasljeđivati rezultate iz prethodne iteracije bilo je intuitivno odlučiti se za ovakvu implementaciju. Iako će u okviru ove dokumentacije biti spomenuti pojedini atributi i metode za potrebe pojašnjenja osnovnih koraka algoritma, pojašnjenja za sve atribute i metode moguće je pročitati u komentarima skripti proslijeđenih u prilogu dokumentacije.

Za prvi korak algoritma jedina pretpostavka koja se uvodi jeste da je dobavljeni frame predstavljen u sivoj skali (eng. grayscale) odnosno da je predstavljena kao matrica. Još jedna opcija koja je ponuđena u okviru ovog koraka jeste da umjesto pretraživanja značajki na osnovu cijelog frame-a, moguće je odabrati samo njegov segment tzv. regiju od interesa (eng. Region of interest ili RoI). Ovaj korak pretpostavlja određene apriorne informacije o poziciji kamere i onome što ona zapravo "vidi". Tako ako je pozicija kamera postavljena na način da donja polovina frame-a prikazuje cestu, dok gornja polovina prikazuje okruženje to je pogodno definisati RoI kao gornju polovinu frame-a. Ovim se može umanjiti dimezionalnost frame-a, a zatim i same matrice, što posljedično znači i manje vremena potrebno za računske operacije nad matricama.

U drugom koraku algoritma predviđeno je izdvajanje značajki, što je moguće izvršiti sa različitim detektorima kao što su ORB, FAST, SIFT... Zbog toga i u cilju provjere koji od njih daje najbolje rezultate po pitanju vremena izvršavanja i preciznosti estimacije odlučeno je omogućiti specificiranje detektora, pri čemu je moguće odabrati ORB, FAST, SIFT, SURF i SHI - TOMASI detektore dodjeljujući varijabli *detector* u *main.py* jedan od naziva detektora (npr. *detector* = 'FAST'). Treba obratiti pažnju da tokom prve iteracije će ovaj korak ujedno biti i jedini dok će u narednim iteracijama biti izvršavan barem jednaput. Kroz iteracije nasljeđuju se one značajke koje su bile izdvojene, a zatim i uparene sa značajkama prethodne iteracije. Zbog ovoga uz dovoljan broj iteracija i fizičko udaljavanje kamere broj značajki će se postepeno umanjivati, zbog čega je potrebno definisati minimalni broj značajki koje se nasljeđuju između iteracija. Ukoliko broj značajki postane manji od nekog definiranog praga (*kMinNumFeature*) tada je potrebno prije završetka iteracije izvršiti izdvajanje posve novih značajki sa trenutnog frame-a. Naravno, ponovno izdvajanje značajki bismo mogli vršiti i u svakoj iteracije, no moramo znati da bi ovaj korak bio računski jako skup i, za dovoljno bliske frame-ove, nepotreban*.

Korak koji započinje sa svojim izvršavanjem počevši od druge iteracije jeste uparivanje značajki koje je odlučeno vršiti pomoću Kanade-Lucas-Tomasi tracker-a†. Osnova ovog koraka, a dijelom i prethodnog, jeste u odabiru velikog broja parametara koje se proslјeđuju implementiranim funkcija. Premda je moguće pronaći opće preporuke za njihov odabir pokazuje se da tretiranje svake primjene zasebno će dovesti do najboljih rezultata. Također, ovaj korak podrazumijeva ne samo uparivanje značajki već ujedno izdvajanje onih koje je moguće okarakterizirati kao "dobre" i koje će posljedično dati bolju estimaciju esencijalne matrice. Upravo, njena estimacija predstavlja sljedeći korak algoritma zajedno sa rekonstrukcijom matrice rotacije R i vektora translacije t između dva susjedna frame-a. Ponovno su za realizaciju

*Također je primijećeno da u nekim testnim slučajevima može dovesti do nepredvidivih rezultata.

†Demonstrativno je u realizaciji priložen i kod kojim bi se uparivanje značajki vršilo brute force match-erom.

ovog koraka korištene već implementirane funkcije, no njihova interna realizacija je pojašnjena u okviru [3], [6] i [7]. Matrica R i vektor t nam dalje služe u koraku 5 kako bismo na osnovu njih mogli formirati oblak tačaka (eng. point cloud) što zapravo predstavljaju 3D koordinate izdvojenih značajki. Bitna napomena jeste da dobivene 3D koordinate ne predstavljaju koordinate tačke u svjetskom koordinatnom sistemu odnosno koordinatnom sistemu čijem početkom smatramo položaj kamere. U literaturi ćemo susresti termin "correct up to scale" kao osobinu određenih tačaka što zapravo znači da esencijalna matrica koju smo odredili na osnovu značajki nema jednoznačnost skale (eng. scale ambiguity). Zbog toga će biti potrebno odrediti skalirajući faktor kojim ćemo moći dobivene koordinate preslikati u svjetski koordinatni sistem. Skalirajući faktor bilo bi potrebno odrediti koristeći informacije dobivene na osnovu mjerenja drugih senzora kao što je IMU, no moguće je odrediti, do određene tačnosti, korištenjem triangulisanih tačaka trenutne i prethodne iteracije. Naime, u tom slučaju skalirajući faktor se naziva još i relativni faktor, a relacija prema kojoj se određuje je data kao [4]:

$$r = \frac{norm(X_{k-1,i} - X_{k-1,j})}{norm(X_{k,i} - X_{k,j})} \tag{4}$$

gdje razlika u brojniku predstavlja razliku između dvije proizvoljne traingulisane tačke prethodne iteracije (k-1) i, analogno, razlika u nazivniku predstavlja razliku između dvije proizvoljne traingulisane tačke trenutne iteracije (k). Premda se relativni faktor može definisati samo za dva para triangulisanih tačaka prethodne i trenutne iteracije, preporuka jeste da se odredi faktor r za sve triangulisane tačke, te da se njihov median usvoji kao relativni faktor. Nakon određivanja relativnog faktora, vektor translacije je potrebno korigovati množenjem sa skalirajućim faktorom r.

Konačno, posljednji korak algoritma jeste samo iscrtavanje trajektorije za što nam je potreban samo vektor translacije za trenutnu iteraciju. Razlog tomu jeste što svakom iteracijom odnosno izračunavanjem vektora translacije t mi, zapravo, dobijamo položaj koordinatnog početka trenutnog frame-a (pretpostavljamo da smo smjestili kameru u koordinatni početak) izraženog u koordinatnom sistemu prethodnog frame-a. Kako je cilj prikazati koordinatni početak svakog narednog frame-a u koordinatnom sistemu referentnog frame-a odnosno početnog to je potrebno vektor translacije izračunavati prema relacije:

$$t_{svjetsko} = r \cdot R_{svjetsko} \cdot t + t_{svjetsko} \tag{5}$$

pri čemu su $t_{svjetsko}$ i $R_{svjetsko}$, također, dobiveni rekurzivnim uvrštavanjem u gore navedenu relaciju, dok t predstavlja vektor translacije dobiven iz esencijalne matrice za trenutnu iteraciju. Naravno, kako se trajektorija iscrtava kao da posmatramo kretanje automobila odnosno kamere gledajući na njih odozgo to će nam za iscrtavanje biti potrebne samo dvije koordinate x i y. Potrebno je obratiti pažnju na to da u okviru Carla simulatora je korišten lijevo orijentisani koordinatni sistem, te da početni položaj kamere odnosno vozila nije ujedno koordinatni početak kao što je to pretpostavljeno za iscrtavanje trajektorije. Zbog toga je potrebno izvršiti konverziju, što činimo sa funkcijom *leftToRight()* uključenom u *main.py*. Ponovno napominjemo da u okviru ove sekcije su navedene smjernice za implementaciju, no o svakoj korištenoj naredbi ili objektu moguće je detaljnije pročitati u skriptama rješenja.

IV. Rezultati

Premda nije bilo moguće testiranje algoritma izvršiti u realtime simulaciji zbog ograničenja korištenog sistema, testiranje je izvršeno snimanjem frame-ova putanje koju generiše autopilot počevši od proizvoljne lokacije. U prilogu rada moguće je pronaći različite testne rute, te će u okviru ove sekcije biti prikazana jedna od njih. Procjena uspješnosti algoritma je vršena prema dvije osobine i to su rekonstrukcija pomenutih ruta, te vrijeme izvršavanja, a svako pokretanje algoritma je vršeno sa drugim detektorom.

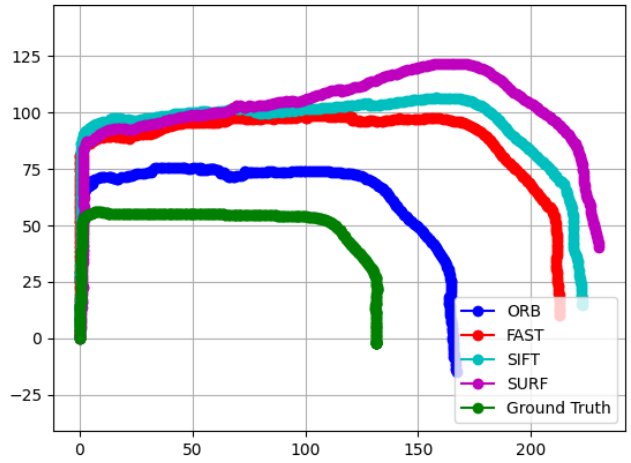
Za testiranje rekonstrukcije ruta korišteni su frame-ovi iz *test_1* izuzev za slučaj *SHI-TOMASI* detektora. Za uparivanje značajki korišten je Kanade-Lucas-Tomasi tracker[‡], a od primjene značajnih parametara bitno je spomenuti *fMATCHING_DIFF* parametar ključan za definisanje "dobrih" značajki koje će se koristiti dalje za estimaciju esencijalne matrice. Pokazuje se da postavljanje uslova za *fMATCHING_DIFF* na vrijednost 1 ili 2 dovodi do boljih rezultata, ali za slučaj korištenja *SHI-TOMASI* detektora može upotpunosti onemogućiti rekonstrukciju *test_1* rute (jer broj značajki postaje manji od 5, te nije moguće izvršiti estimaciju esencijalne matrice).

Na slici 2 možemo vidjeti rezultate dobivene za 4 korištena detektora: *ORB*, *FAST*, *SIFT* i *SURF* (*SHI-TOMASI* će biti naknadno detaljnije pojašnjen) testirane za iste postavke parametara. Premda vidimo da ruta rekonstruirana koristeći pomenute detektore zaista prati opći oblik ground truth rute čije su koordinate preuzete iz Carla simulatora, očigledan je problem skaliranja. Kao što je bilo pojašnjeno u sekciji III u okviru algoritma se koristi relativna skala računata prema relaciji 4. Možemo primijetiti da za detektore *FAST*, *SIFT* i *SURF* proračunata relativna skala je gotovo dvostruko veća, a to potvrđuju i testiranja kada je uz množenje sa relativnom skalom izvršeno množenje sa fiksnom vrijednošću unutar opsega 0.5 - 0.7 (slika 3).

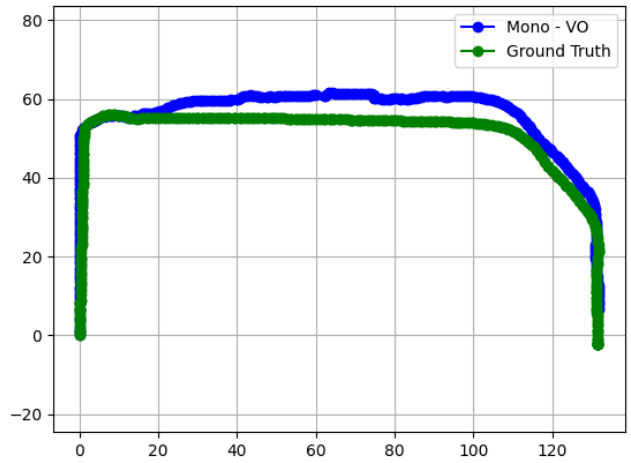
Testiranje prikazano na slici 2 podrazumijevalo je da se za sve detektore koriste iste postavke i vrijednosti parametara, zbog čega je posve očekivano da će neki od detektora davati bolje, a neki lošije rezultate. Na osnovu slike 2 možemo primijetiti da korišteni *ORB* detektor daje najpribližnije praćenje rute.

Inspirisani javno dostupnim KITTI dataset-om koji pored frame-ova različitih testnih ruta, također, posjeduje lokacije za svaku od njih što omogućava računanje apsolutne skale, odlučili smo i mi testirati računanje apsolutne skale

[‡]U samom kodu moguće je pronaći funkciju *BFFeatureTracking()* ukoliko bi se koristio BFMatcher koji je prethodno bio testiran kroz laboratorijske vježbe kursa Robotska vizija

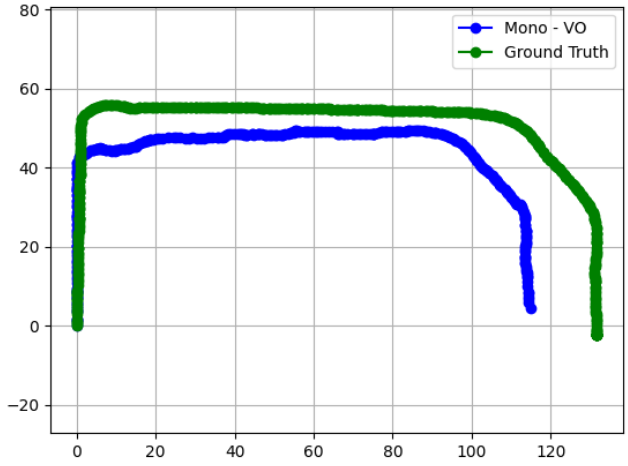


Slika 2. Rekonstrukcija rute korištenjem *ORB*, *FAST*, *SIFT* i *SURF* detektora



Slika 3. Umanjivanje relativne skale fiksnom vrijednošću 0.62 za korišteni *FAST* detektor

prema lokacijama zabilježenih u ground truth. Premda implementacija klase *VisualOdometry()* podrazumijeva mogućnost prosljeđivanja ground truth lokacija, opet smo bili dijelom spriječeni ograničenjima korištenih sistema. Naime, prilikom pohranjivanja frame-ova pojedini frame-ovi bi bili uništeni odnosno nepravilno pohranjeni, te ih je bilo potrebno ukloniti iz folder-a. Ovo je značilo da broj pohranjenih lokacija u .csv file-u nije bio jednak broju frame-ova, a slučajnost pojave nepravilnog pohranjivanja frame-ova je zahtijevalo mukotrpno pretraživanje pripadajućih lokacija za ukloniti nakon svakog generisanog testa. Zbog toga na slici 4 rekonstruisana ruta čija apsolutna skala je dobivena na osnovu lokacija pohranjenih u *koordinate_1.csv* file-u idalje odstupa od ground truth rute, a moguće je primijetiti da se odstupanje povećava što se približavamo kraju rute. Također, primjećujemo da je greška suprotnog predznaka u odnosu na prethodne rezultate u kojima je korištena relativna skala, odnosno da apsolutna skala čini rekonstruisanu rutu sve manjom.



Slika 4. Rekonstrukcija rute na osnovu apsolutne skale za korišteni *FAST* detektor

Još jedno ograničenje koje je postavljeno zbog nepravilno pohranjenih frame-ova jeste nemogućnost tačne evaluacije greške generisane trajektorije. Naravno, realizacija algoritma pretpostavlja da je pored vizuelne inspekcije rezultata neophodna i numerička procjena, pa zato uključuje atribut *error* unutar koje se pohranjuje srednja vrijednost odstupanja lokacije određene korištenjem VO i lokacije pohranjene u .csv file-u. Naravno, kao i slučaj određivanja apsolutne skale, problem ponovno predstavlja nepravilna korespondencija frame-a i ground truth lokacije, pa zbog toga ne možemo govoriti o tačnoj procjeni greške. Ipak, demonstrativno je odlučeno izvršiti estimaciju greški, te uporediti odstupnja za slučaj prikaz na slici 2. Rezultate je moguće vidjeti u tabeli I.

	ORB	FAST	SIFT	SURF
Sr. greška	28.7471	59.8825	65.4370	68.3732

Tablica I
SREDNJA VRIJEDNOST GREŠKE ZA DETEKTORE *ORB*, *FAST*, *SIFT* I *SURF* ZA *test_1*

Posljednje na što ćemo se osvrnuti po pitanju rekonstrukcije ruta jeste upotreba detektora *SHI-TOMASI*. Iako će se *SHI-TOMASI* detektor pokazati najbržim, njegova primjenljivost je ograničena u slučaju neprepoznavanja dovoljnog broja "dobrih" značajki prema postavkama KLT tracker-a korištenim za prethodna četiri detektora. Naime, u jednoj iteraciji broj značajki pada ispod 5 koliko je minimalno potrebno za estimaciju esencijalne matrice i ovim se algoritam prekida. U slučaju ovakvog ishoda moguće je izvršiti izmjenu postavki za KLT tracker ili parametara kojim definišemo instancu *SHI-TOMASI* detektora u cilju umanjivanje strogoće kvalitete

izdvojenih značajki. Iako je isto učinjeno za testnu rutu *test_1* nisu dobiveni rezultati približni onim predstavljenim na slici 2 odnosno dobivena trajektorija je značajno divergirala od ground truth trajektorije.

Pod relevantnim vremenom izvršavanja algoritma usvojili smo vrijeme potrebno za izvršavanje jedne iteracije procesiranja trenutnog frame-a (funkcija *processFrame()*). Pretpostavljeno je da funkcije *processFirstFrame()* i *processSecondFrame()* imaju kraće vrijeme trajanja, te kako se izvršavaju samo jedanput po pokretanju algoritma, to će njihovo vrijeme izvršavanja biti zanemareno. Također, odlučeno je vrijeme izvršavanja računati kao srednju vrijednost, pa je za te potrebe kreiran dodatni atribut klase *VisualOdometry()* unutar kojeg će se pohranjivati rekurzivno izračunata srednja vrijednost. Na ovaj način, za veći broj frame-ova dobijamo tačniju srednju vrijednost iteracije izvršavanja. Prije nego što navedemo dobivene rezultate za svaki od korištenih detektora osvrnut ćemo se na mogućnost ubrzavanja algoritma uvođenjem mogućnosti preskakanja frame-ova u slučaju da srednja udaljenost između uparenih značajki je manja od neke definirane udaljenosti d. Ovim zapravo postižemo da uzastopni frame-ovi koji prikazuju vozilo odnosno kameru na istom mjestu ili sporo kreće budu preskočeni, čime se vrijeme izvršavanja pojedinačne iteracije skraćuje za vrijeme potrebno za izvršavanje koraka 4-6 algoritma. Također, još jedna prednost koja je inherentna samoj realizacije jeste da se izdvojene i uparene značajke nasljeđuju kroz iteracije dokle god je njihov broj veći od nekog prethodno definiranog odnosno da se funkcija *detectNewFeatures()* poziva samo onda kada broj uparenih značajki trenutnog frame-a postane manji od npr. 1000.

Kao za rekonstrukciju rute, pokrenutanje algoritma je vršeno svaki put sa drugim detektorom, no da bismo pokazali utjecaj preskakanja frame-ova mjerili smo i vrijeme izvršavanja kada je ova opcija omogućena. Naravno, napomena jeste da srednja vrijednost vremena izvršavanja će zavisiti od broja frame-ova koji su preskočeni, pa je zbog toga odabrana ruta u kojoj se vozilo zaustavlja samo na jednom mjestu odnosno ruta *test_1*. U tabelama II i III moguće je vidjeti ostvarene rezultate.

	ORB	FAST	SIFT
Sr. vrijed. [s]	Bez preskakanja 0,1225	Bez presk. 0,1866	Bez presk. 0,2992
Sr. vrijed. [s]	Sa preskakanjem 0,1092	Sa presk. 0,1161	Sa presk. 0,2199

Tablica II
SREDNJE VRIJEME IZVRŠAVANJA METODE *processFrame()*

	SURF	SHI-TOMASI*
Sr. vrijed. [s]	Bez preskakanja 0,1487	Bez presk. 0,1172
Sr. vrijed. [s]	Sa preskakanjem 0,1403	Sa presk. 0,0795

Tablica III
SREDNJE VRIJEME IZVRŠAVANJA METODE *processFrame()*

Na osnovu priloženih rezultata možemo vidjeti da je preskakanje frame-ova zaista umanjilo vrijeme izvršavanja koraka algoritma koji su obuhvaćeni funkcijom *processFrame()*. Dok je za slučaj detektora *ORB*, *FAST* i *SHI-TOMASI*[§] ubrzanje očito, preskakanje frame-ova za slučaj detektora *SIFT* i *SURF* nije bilo tako primjetno. Također, možemo primijetiti da korištenje detektora *SHI-TOMASI* je značajno ubrzava izvršavanje algoritma jer je njegova brzina izvršavanja i bez preskakanja frame-ova približna brzini sljedećeg najkraćeg izvršavanja sa *FAST* detektorom i uz preskakanje frame-ova.

V. ZAKLJUČAK

Vizuelna odometrija nalazi široku primjenu u sistemima autonomne vožnje, kao i upotrebi dronova, najviše s ciljem istraživanja područja pogođenih elementarnim nepogodama, kako bi se smanjila opasnost po čovjeka. Iz prethodne tvrdnje se može zaključiti kolika je već, i kolika će biti u budućnosti, važnost vizuelne odometrije. Za procjene trenutnih pozicija robota i uređaja sa mogućnošću kretanja, koriste se različiti senzori poput GPS-a, brojača kilometara i kamera. Vizuelna odometrija se pokazala kao efikasna metoda za vršenje procjene položaja uređaja jer koristi dugotrajno praćenje vizuelnih karakteristika. S ciljem realizacije algoritma, korištena je monocular kamera koja spojena sa postupkom vizuelne odometrije daje rezultate i relativnog kretanja i 3D strukture iz 2D podataka. Za samu implementaciju, korišten je Carla open-source simulator, namijenjen za razvoj, istraživanje i validaciju sistema autonomne vožnje.

S obzirom na ograničene resurse kojim raspolažemo, nije bilo moguće izvršiti testiranje navedenog algoritma u real-time simulaciji. Testiranje vršeno snimanjem frame-ova putanje, demonstrirano u sekciji rezultata je obavljeno prema dvije osobine: rekonstrukcijom ruta i vremenom izvršavanja. Oba postupka su prethodno opisana uz grafički prikaz dobijenih rezultata.

Kao što je prikazano u rezultatima, uspješno je određena trajektorija "up to scale", no zbog nedostatka poznavanja apsolutne skale, javljaju se odstupanja. Daljnji korak implementacije algoritma bi bio korištenje stereo vizije ili fuzije sa drugim senzorima kojima raspolažemo u Carla simulatoru.

LITERATURA

[1] Hong, E., & Lim, J. (2018). Visual-Inertial Odometry with Robust Initialization and Online Scale Estimation. Sensors, 18(12), 4287.
[2] Microsoft Research. (2017) Youtube. [Online]. Dostupno na: Robust, Visual-Inertial State Estimation: from Frame-based to Event-based Cameras
[3] Hartley, R. & Zisserman, (2003) A. Multiple View Geometry in Computer Vision. Cambridge University Press. Cambridge books online
[4] Read the docs [Online] Dostupno na: Windows build
[5] Github [Online] Dostupno na: Releases
[6] Scaramuzza, D., & Fraundorfer, F. (2011). Visual Odometry [Tutorial]. IEEE Robotics & Automation Magazine, 18(4), 80–92.
[7] Fraundorfer, F., & Scaramuzza, D. (2012). Visual Odometry: Part II: Matching, Robustness, Optimization, and Applications. IEEE Robotics & Automation Magazine, 19(2), 78–90.

[§]* - Kako nije bilo moguće kreirati trajektoriju za odabranu rutu, to je za razliku od prethodna 4 detektora vrijeme izvršavanja iteracije izračunat na osnovu rute koju je bilo moguće odrediti.