UNIVERZA V LJUBLJANI

FAKULTETA ZA STROJNIŠTVO

Poročilo - Magistrski praktikum

Algoritem glajenja poti za avtonomno vodena vozila

Martin Knap, 23172069

Mentor: doc. dr. Rok Vrabič

Ljubljana, 22. 10. 2019

Kazalo

Slike

1 Uvod

Pri razvoju avtonomnega vozila, namenjenega razvozu materiala po proizvodnih prostorih, smo se soočili s problemom pri gibanju vozila. Problem se pojavi pri ubiranju poti, ki jo tvori navigacijski algoritem vozila. Ta pot vodi vozilo tako, da se le-to giblje energijsko neučinkovito ter sunkovito. Posledice takšne, grobo tvorjene poti se odražajo v hitrejšem praznenju baterij. To pomeni, da je potrebno baterije vozila večkrat polniti, ter da je vozilo manj časa razpoložljivo. Zaradi sunkovitega gibanja se lahko pojavijo tudi problemi v zevzi s prevozom tovora.

Naša naloga je bila izboljšati pot, ki jo tvori navigacijski algoritem in posledično omogočiti vozilu, da bo delovalo energijsko učinkovitejše, zanesljivejše, ter da bo posledično vozilo na razpolago daljše časovno obdobje.

2 Ozadje problema

Problem torej izvira iz poti, ki jo ustvari navigacijski algoritem vozila. Preden se lotimo podrobnejše obravnave izvora težav, bi bilo smotrno predstaviti zgradbo avtonomnega vozila in sistem, ki nam omogoča upravljanje z vozilom, ki se imenuje *Robot Operating System* ali krajše ROS.

2.1 Zgradba avtonomnega vozila

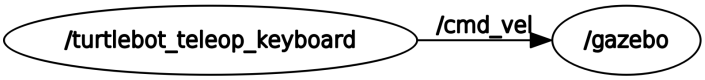
Obravnavano vozilo je tipičen predstavnik robotov z diferencialnim pogonom koles (*Angl.: differential wheeled robot*). Roboti te vrste imajo dve pogonski kolesi, pri čemer ima vsako od koles svoj motor. Poleg dveh pogonskih koles pa imajo ponavadi še eno ali več prosto vrtečih se koles za zagotavljanje stabilnosti. Če se pogonski kolesi vrtita z enako hitrostjo se robot giblje v ravni liniji. Zavijanje pa se doseže tako, da se eno od pogonskih koles vrti hitreje od drugega [1].

2.2 Robot Operating System

Krmiljenje obravnavanega vozila nam omogoča programsko okolje *Robot Operating System* - ROS. ROS ni operacijski sistem v pravem pomenu besede, temveč je odprtokodno ogrodje za programiranje robotov. Ideja za platformo je ta, da je delitev idej in znanja med razvijalci enostavnejša, še bolj pomembno pa je, da nam je olajšano razvijanje celotne programske infrastrukture pri razvoju robota, saj so to naredili že drugi razvijalci pred nami.

ROS je sestavljen iz krmilnikov, ki so namenjeni branju podatkov iz senzorjev in pošiljanju ukazov aktuatorjem v dobro definiranem formatu. ROS sestavlja tudi velik in rastoči nabor fundamentalnih robotskih algoritmov, ki omogočajo gradnjo zemljevidov, navigacijo po le-teh, interpretacijo podatkov iz senzorjev, planiranju gibanj, manipuliranju predmetov in še mnogo več. Poleg tega nam nudi še celotno računsko infrastrukturo, ki nam omogoča manipuliranje s podatki tako, da povežemo različne komponente kompleksnega robotskega sistema in vključevanje lastnih algoritmov. ROS je že v osnovi porazdeljen in nam omogoča delitev delovne obremenitve med več računalniki brez težav. Nudi nam tudi nabor orodij za vizualizacijo stanja robota (Gazebo, RViz), algoritmov, za odkrivanje napak in za zajemanje senzorskih podatkov. Navsezadnje pa nam širši ekosistem ROS-a nudi obsežen nabor virov, kot je spletna enciklopedija ter forum, kjer se postavljajo vprašanja ter deli znanje.

ROS torej tvori množica programov, ki delujejo istočasno in komunicirajo med sabo tako, da si med seboj pošiljajo sporočila (*message*), ki imajo natančno določeno obliko. Smiselna je grafična predstavitev sistema, kjer so programi vozlišča (*node*), robovi pa predstavljajo komunikacijske kanale (*topic*). Za primer si poglejmo spodnjo sliko (Slika 1), kjer imamo vozlišče `/turtlebot_teleop_keyboard` s katerim pretvarjamo ukaze tipkovnice v sporočila oblike `geometry_msgs/Twist`, ki nosijo informacijo o temu kako naj se robot premika (linearne, kotne hitrosti). Ta sporočila se nato po povezavi `/cmd_vel` posreduje do vozlišča simulacijskega okolja Gazebo z istim imenom v kateremu se nahaja testni robot. Obstajajo standardna sporočila ter sporočila, ki jih ustvari uporabnik sam.



Slika 1: Primer dveh vozlišč z medsebojno povezavo v okolju ROS.

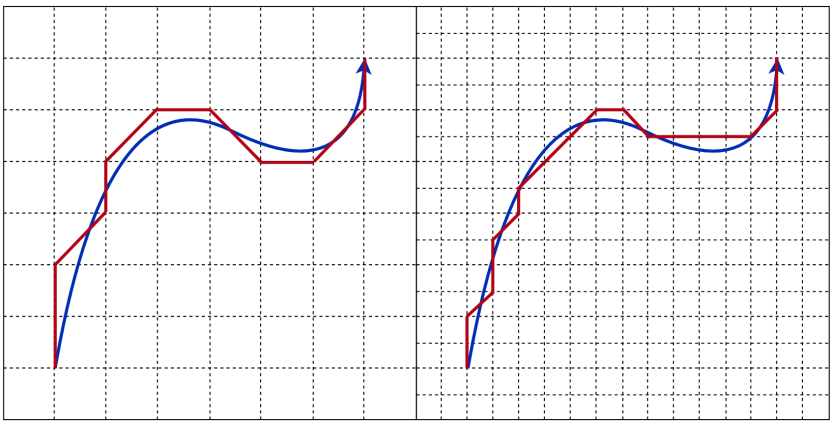
2.3 Opis problema

Seznanjeni z okvirnim delovanjem krmilnega okolja avtomatskega vozila se lahko osredotočimo na problem. Navigacija v okolju ROS se izvaja na podlagi zemljevidov uteži (Angl.: *Costmap*). To so dvodimenzionalne mreže pravokotne oblike, kjer so ovire predstavljene z utežmi. Razdelki v zemljevidu uteži imajo tako vrednost 0 v primeru odsotnosti ovire ter vrednosti 100 v primeru ovire.

Do teh zemljevidov pridemo z uporabo kartografskih algoritmov kot je na primer *Simultaneous Localization And Mapping* - SLAM. Mrežo okolja robota dobimo na podlagi izhodnih informacij senzorjev in sicer tako, da zaženemo kartografski algoritem in nato ročno ali avtomatsko vodimo robota po okolju tako, da zajamemo celotno okolico. Ko imamo zajeto celotno območje lahko izvozimo mrežo v obliki slike. Kartografske algoritme nam nudi okolje ROS.

Mreže, ki jih uporabljamo v ROS-u so v obliki rastrske sivinske slike. Črna polja mreže predstavljajo mesta v prostoru kjer so ovire, kjer se robot ne more gibati. Bela polja predstavljajo mesta v prostoru kjer ni ovir in tam se robot v splošnem lahko giblje. Siva polja pa predstavljajo območje, ki ga ne poznamo.

Izvor našega problema leži v ločljivosti zemljevida uporabljenega za navigacijo. Zaradi velikih dimenzij prostorov v katerih bo robot deloval se pojavi potreba po grobih zemljevidih, saj se tako prihrani računski čas ter prostor v računalniškem spominu. Posledica uporabe zemljevida z nizko ločljivostjo, velikim korakom mreže je v majhnemu številu točk na podlagi katerih navigacijski algoritem tvori pot vozila. Pot je v temu primeru groba in izrazito robata. Na spodnji sliki vidimo z modro obarvano idelano pot, z rdečo pa je obarvna pot določena na podlagi zemljevida vrednosti pri dveh različnih ločljivostih. V idealnem primeru bi z neskončno majhno diskretizacijo prostora rdeča in modra pot sovpadali.



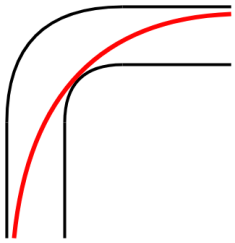
Slika 2: Primerjava poti, ki ju tvori navigacijski algoritem.

3 Namen in cilji

Kot smo spoznali v prejšnjem poglavju je uporaba zemljevidov z visko ločljivostjo neupravičena, saj je poraba računalniških sredstev tako prevelika. Naša naloga je najti način glajenja grobe poti, osnovane na zemljevidu z nizko ločljivostjo ter rešitev implementirati. V kontekstu ROS-a bo potrebno ustvariti vmesno vozlišče, ki bo prestreglo sporočila grobe poti, jih transformirala v gladko pot ter posredovla avtonomnemu vozilu. Pri implemetaciji je zaželena tudi enostavnost in intuitivnost implemetacije rešitve skozi oči uporabnika.

4 Pregled možnih rešitev

Navdih za rešitev problema glajenja poti smo dobili na področju motornih športov. Tam se vozniki dirkalnih vozil izključno poslužujejo načina vožnje skozi ovinke, ki se mu pravi dirkalna linija (Angl.: *racing line*). Dirkalna linija je pot preko ovinka, ki vozniku dirkalnika omogoča najkrajši čas proge [2]. Želja je, da je hitrost pri vožnji skozi ovinek karseda konstantna.



Slika 3: Dirkalna linija.

Ideja za dirkalno linijo je ta, da se vozilo oklepa zunanjega roba ovinka pri vstopu v ovinek in se skuša karseda približati notranjemu robu ovinka na njegovem vrhu. Na izhodu iz ovinka pa se vozilo ponovno približa zunanjemu robu. V teoriji je optimalna pot skozi ovinek takšna, ki ima največji radij.

Iskanje najboljše dirkalne linije za dano vozilo, na dani progi je eden od večjih problemov s katerim se srečujejo razvijalci dirkalnih računalniških iger [3]. Komercialne dirkalne igre se ponavadi zanašajo na dirkalne linije, ki jih zasnujejo ljudje - strokovnjaki s področja dirkanja. Obstajajo pa tudi izjeme, ki se izognejo omenjeni metodi in se problema lotijo na povsem drugačen način. Dober primer je Colin McRae Rally, kjer dirkalno linijo računajo s pomočjo nevronske mreže, ki so jo učili strokovnjaki iz področja motornih športov [3]. V Microsoft-ovi igri Forza Motorsport 2 s pomočjo tehnik nadzorovanega učenja (Angl.: *supervised learning*) učijo AI-je ter z aplikacijo evolucijskega računanja (Angl.: *evolutionary computation*) optimirajo njihove dirkalne linije [4].

Odprtokodne dirkalne igre se ponavadi zanašajo na hevristične pristope, ki tipično združujejo dobro prakso s hevristiko pri generaciji dirkalnih linij. Najuspešnejši primeri takšnih pristopov vključujejo K1999 algoritem [5], ki ga je razvil Remi Coulom in deluje na podlagi gradientnega spusta (Angl.: *gradient descent*) ter Simplix [6], ki ga je razvil Wolf-Dieter Beelitz za The Open Car Racing Simulator in bazira na preprosti hevristiki. Ostala uspešna primera sta tudi *the bot* [7], ki ga je razvil Jussi Pajala za Robot Auto Racing Simulator (RARS) in je zasnovan na A* algoritmu ter the *DougE1 bot* za RARS, ki ga je razvil Doug Elenveld, ta pa izkorišča genetski algoritem. Kljub temu, da omenjene rešitve temeljijo na različnih tehnikah je vsem skupno to, da skušajo zmanjšati ukrivlejnost poti vozila skozi ovinek, saj z manjšo ukrivlejnostjo poti vozilo lahko doseže večjo hitrost brez zdrsa.

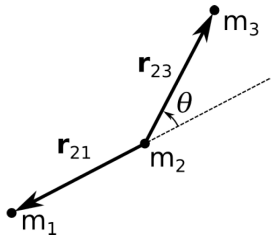
Zgoraj omenjene metode so sicer zelo impresivne, vendar obstajajo tudi metode, ki upoštevajo dinamski model vozila pri reševanju omenjenega problema in vodijo do še ustreznejših rešitev omenjenega problema. Na drugi strani pa imamo rešitve, ki so malce preprostejše kot je uporaba kubičnih parametričnih Bézier krivulj pri generaciji dirkalne linije [8] in pa edinstvena rešitev, ki jo nudi odprtokodni Vamos Avtomotive Simulator, kjer je za doesganje manjše ukrivlejnosti poti skozi ovinek uporabljena vergia masnih točk, ki jih povezuje torzijska vzmet. Do zglajene poti pridemo tako, da najdemo minimum potencialne energije shranjene v vzmeteh. Pri reševanju našega problema smo se odločili za seldnjo metodo, zato jo bomo v naslednjem poglavju predstavili podrobneje.

5 Izbrana rešitev

Vamos Avtomotive Simulator (VAS) je odprtokodno simulacijsko okolje s poudarkom na fizičnemu modeliranju [9]. Vamos modelira veliko večino ključnih podsistemov avtomobilov, kot so elementi pogonskega sklopa in sicer motor, sklopka, menjalnik in diferencial z omejenim zdrsom. Prav tako pa je modelirano tudi obnašanje pnevmatik pri temperaturnih vplivih, vzmetenje avtomobila in aerodinamski učinki na

avtomobilih. VAS igralcem nudi tudi igranje proti računalniškemu nasprotniku. V ta namen so razvijalci ustvarili algoritem izgradnje poti za računalniškega nasprotnika tako, da igralcem nudijo karseda realističen izziv. Ta algoritem bomo v naslednjem delu besedila vzeli pod drobnogled in ga nato tudi uporabili pri reševanju našega problema.

Pot vozila na začetku predstavlja nabor točk, ki tvorijo središčnico znotraj proge. Točke predstavimo kot masne točke, ki so med seboj povezane s torzijskimi vzmetmi.



Slika 4: Tri vozlišča poti. Moment pri kotu θ [9].

Tak sistem bo s časom silil v stanje minimalne energije v torzijskih vzmeteh in nam posledično ustvaril pot z manjšo ukrivljenostjo.

Silo, ki deluje na maso m_2 je možno izračunati na podlagi relativnih leg sosednjih mas. Začnimo s sklepom, da s povečanjem kota θ dobimo moment sorazmeren kotu.

$$\mathbf{N_2} = \mathbf{r}_{23} \times \mathbf{F_3} = -k\theta\hat{n} \tag{1}$$

Pri tem je $\mathbf{r}_{23}$ vektor od m_2 do m_3 in $\hat{n}$ je enotski vektor, ki kaže v smeri $\mathbf{r}_{23} \times \mathbf{r}_{21}$. Sila na m_3 je tako:

$$\mathbf{F_3} = -\frac{k}{|\mathbf{r}_{23}|}\theta(\hat{n} \times \mathbf{r}_{23}) \tag{2}$$

Produkt $\theta\hat{n}$ se izračuna kot vektorski produkt $\mathbf{r}_{23}$ in $\mathbf{r}_{21}$.

$$\mathbf{r}_{23} \times \mathbf{r}_{21} = |\mathbf{r}_{23}||\mathbf{r}_{21}| \sin(\phi - \theta)\hat{n} \tag{3}$$

$$\sin\theta\hat{n} = |\mathbf{r}_{23}||\mathbf{r}_{21}| \sin\theta\hat{n} \tag{4}$$

$$\sin\theta\hat{n} = \frac{\mathbf{r}_{23} \times \mathbf{r}_{21}}{|\mathbf{r}_{23}||\mathbf{r}_{21}|} \tag{5}$$

$$\sin\theta\hat{n} = \hat{r}_{23} \times \hat{r}_{21} \tag{6}$$

Ker pričakujemo majhne kote med vozlišči lahko $\sin\theta$ zapišemo kot θ.

$$\theta\hat{n} = \hat{r}_{23} \times \hat{r}_{21} \tag{7}$$

Z vstavljanjem v enačbo (2) dobimo:

$$\mathbf{F}_3 = \frac{k}{|\mathbf{r}_{23}|}(\hat{r}_{23} \times \hat{r}_{21}) \times \hat{r}_{23} \tag{8}$$

Upoštevajoč simetrijo ugotovimo, da je $\mathbf{F}_1 = \mathbf{F}_3$ in $\mathbf{F}_2 = -2\mathbf{F}_3$ pri majhnem kotu. Definiramo lahko vektor urkivljenosti $\mathbf{c}$ pri m_3:

$$\mathbf{c} = \hat{n}/R = \theta\hat{n}/|\mathbf{r}_{23}| \tag{9}$$

Kjer je R radij ukrivljenosti. Kot θ je kot med m_2 in m_3 od centra ukrivljenosti.

Enačbo lahko zapišemo kot:

$$\mathbf{F}_3 = -k\mathbf{c} \times \hat{r}_{23} \tag{10}$$

Velikost vekotrja ukrivljenosti bomo uporabili kot največjo hitrost vozila v dani točki na stezi.

Sile so izračunane za vsak povezan trojček n točk. Pri tem omejimo gibanje točk v vse smeri in namesto tega dovolimo le pomik prečno na pot. Ta omejitev nam omogoča večjo stabilnost, posledično pa lahko uporabljamo večje togostne koeficiente vzmeti in tako dobimo hitrejšo konvergenco.

Ko je izračunana skupna sila na vsakemu vozlišču lahko izračunamo novo lego točke z uporabo Newtonovovih zakonov gibanja ter z Euler-jevo metodo.

$$F = m\ddot{x} + c\dot{x} \tag{11}$$

$$\ddot{x} = \frac{F}{m} - \frac{c}{m}\dot{x} \tag{12}$$

Kvocient koeficienta dušenja in mase označimo z $d = c/m$.

$$\frac{\dot{x}_{i+1} - \dot{x}_i}{\Delta t} = \frac{F}{m} - d\dot{x}_i \tag{13}$$

$$\dot{x}_{i+1} = \dot{x}_i + (\frac{F}{m} - d\dot{x}_i)\Delta t \tag{14}$$

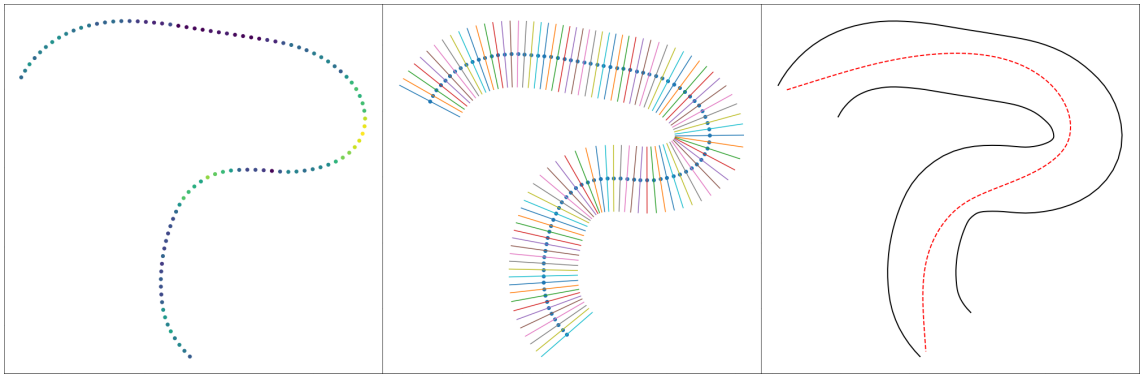
Iz definicije hitrosti sledi še druga enačba.

$$\dot{x} = \frac{x_{i+1} - x_i}{\Delta t} \tag{15}$$

$$x_{i+1} = x_i + \dot{x}_{i+1}\Delta t \tag{16}$$

Tako dobimo enačbo s katero lahko iterativno določimo novo lego vsake točke poti, ki jo gladimo. Algoritem vsebuje dva paramtera in sicer maso m in kvocient d. Pri tem se je potrebno zavedati, da je nova pot lahko groba v ostrih ovinkih (Enačba 6 in 7).

Spodnje slike prikazujejo delovanje dejanskega algoritma glajenja. Leva slika prikazuje graf niza točk nad katerimi bomo izvajali glajenje. Sredinska slika prikazuje pravokotne poti po katerih bodo točke spreminjale svoj položaj z iteriranjem algoritma. Te pravokotne poti posameznih točk tvorijo tudi progo. Proga je ključna za glajenje poti, saj določa območje v katerem ga izvajamo. Na desni sliki vidimo niz točk v novi legi.



Slika 5: Postopek glajenja.

6 Aplikacija algoritma

Z znanim algoritmom glajenja smo lahko pričeli na reševanju problema. Naloga je torej ustvariti vmesno vozlišče v ROS-u, ki jemlje podatke o obstoječi poti iz navigacijskega algoritma ter jih ustrezno transformira ter pošilja naslednjemu vozlišču, ki je odgovoren za izvajanje sledenja novo ustvarjeni poti. Pri določanju območja glajenja se bo upoštevala tudi prisotnost ovir, zato bo potrebno brati tudi informacije o zemljevidu vrednosti (Angl.: *costmap*). Preden zančenmo na delu s samim vozliščem pa je potrebno ustvariti okolje kjer naš program lahko preizkusimo.

6.1 Priprava razvijalnega okolja

Omenjeno vozilo v razvoju ni bilo lahko razpoložljivo, zato nam na srečo ROS paket nudi povezljivost s simulacijskim okoljem Gazebo. Gazebo je odprtokodno simulacijsko okolje za testiranje robotov, ki nudi dinamske simulacije, 3D prikaz, generacijo poljubnega modela robota in okolja in še marsikaj drugega. Omenjeni program smo uporabljali v kombinaciji z ROS paketom RViz, ki nudi 3D prikaz robota, okolja ter prikaz vseh vrst senzorksih podatkov, kot so na primer točke zajete z laserskim merilnikom razdalje. V bistvu nam prikazuje kaj robot vidi. Najpomembnejše je to, da nam RViz prikazuje podatke o poti, ki jo nudi navigacijski algoritem in tako nudi takojšnjo preverbo. Da okolje izpopolnimo, potrebujemo še testno vozilo. Uporabili bomo paket, ki ga nudi spletni blog moorerobots [10], ta vsebuje preprosto vozilo z diferencialnim pogonom ter prirejeno realizacijo navgiacijskega algoritma. Delali bomo z ROS izvedbo Kinetic Kame na operacijskem sistemu Linux Ubuntu. Z definiranim okoljem sedaj lahko začnemo na razvoju omenjenega vozlišča vokolju ROS.

6.2 ROS vozlišče

6.2.1 Pomembnejše povezave vozlišča

Za navigacijo robota v okolju ROS skrbi t.i. navigacijski sklad (Angl.: *navigation stack*), katerega vozlišče se imenuje `/move_base`. To vozlišče na podlagi podane ciljne točke, zemljevida okolice in bližnje okolice zaznane z laserskim merilnikom ustvari globalen načrt oziroma pot ter lokalno pot, katere namen je, da skuša vozilo karseda približati globalni. Glajena bo pot globalnega načrta, ki jo `/move_base` pošilja po komunikacijski povezavi (*topic*) `/move_base/TrajectoryPlannerROS/global_plan`. Sporočila te povezave so tipa `nav_msgs/Path` in nosijo informacijo o legah točk poti ter o orientaciji v ciljni točki. Ker se pri določanju območja glajenja upošteva tudi prisotnost ovir oziroma zemljevid vrednosti je potrebno brati tudi sporočila tipa `nav_msgs/OccupancyGrid`, ki jih navigacijski sklad pošilja po povezavi `/move_base/global_costmap/costmap`. Lega ovri se s časom lahko spreminja zato se tudi zemljevid vrednosti s časom spremija in posledično je potrebno njegovo informacijo dopolnjevati s sporočila tipa `map_msgs/OccupancyGridUpdate`, ki jih dobimo preko kanala `/move_base/global_costmap/costmap_updates`.

6.2.2 Delovanje vozlišča

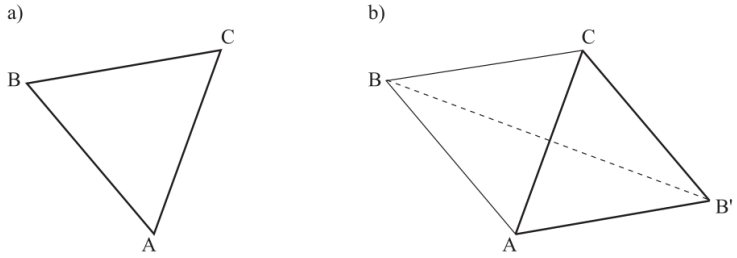
Vozlišče je zasnovano tako, da ima zavoljo enostavnosti uporabe avtomatizirano nastavljanje pramaterov ob njegovem zagonu. Ideja je realizirana tako, da vozlišče deluje kot avtomat z dvemi stanji. Naloga prvega stanja je nastavitev prametrov, naloga drugega pa je glajenje poti po tem ko so paramteri glajenja ugotovljeni.

6.2.3 Iskanje parametrov

Iskanje paramterov je izvedeno tako, da se ob zagonu vozlišča glajenja, ki smo ga poimenovali `/path_smoothing_node` vozlišču `/move_base` pošilja naključne cilje z ozirom na zemljevid vrednosti (če je cilj podan na mesto ovire nam navigacijski sklad ne ustvari poti). Namen tega je pridobitev ustrezno grobe poti na podlagi katere izvedemo optimizacijo s katero iščemo optimalne parametre. Ustreznost poti

na kateri temelji optimizacija ocenimo na podlagi srednje vrednosti in standardne deviacije absolutne vrednosti kotov med posameznimi daljicami, ki tvorijo grobo pot.

Ko je ustrezna pot najdena začenmo z iskanjem minimuma kriterijske funkcije, ki je definirana kot srednja vrednost absolutnih kotov med odseki poti. Metoda iskanja ekstrema je Nelder-Mead, ki deluje na sledeč način. Pri tej metodi iščemo ekstrem funkcije z uporabo t.i. simplexa. To je $n + 1$ dimenzionalen enakostraničen politop v n parametrskem prostoru (Slika 8, primer a). V našem primeru optimiramo dva paramtera (masa in dušenje) zato je simplex enakostraničen trikotnik.



Slika 6: Simplex [11].

Prvi korak iskanja minimuma funkcije s simplexom je tvorba simplexa na podlagi začetnih približkov [11]. Nato se najde točko na simplexu, ki vrne največjo vrednost kriterijske funkcije. To točko se nato preslika preko lika tako, da tvorimo nov simplex (Slika 8, primer b). Postopek ponavljamo toliko časa, dokler ne konvergiramo v okolico lokalnega minimuma.

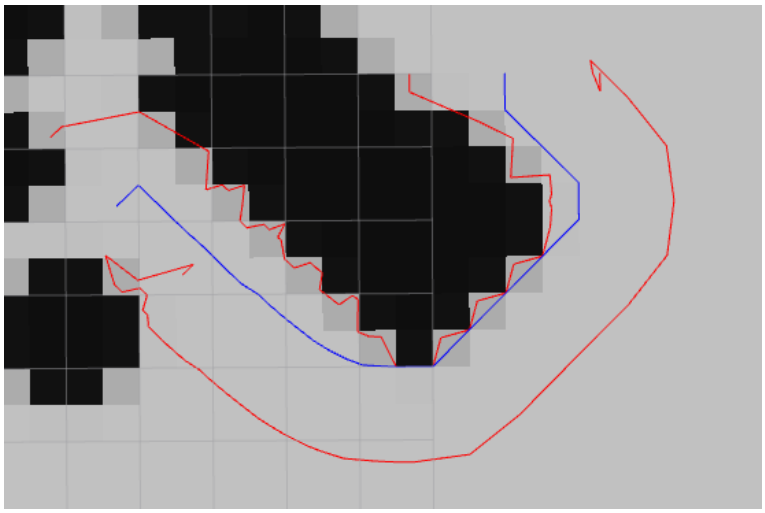
Prednost metode je v tem, da ne potrebujemo matematično definirane kriterijske funckije (lahko je funkcija v programskem smislu) in posledičnega odvoda, ki ga za razliko od te metode potrebujejo gradientne metode.

V našem primeru smo uporabili več začenih približkov z namenom bolj globalnega preiskovanja kriterijske funckije. Ko so z omenjenim pristopom najedni optimalni paramteri jih uporabimo v naslednjem stanju avtomata, ki predstavlja jedro vozlišča.

Prvo stanje v vozlišče vnaša nekaj ROS paramterov, ki jih lahko nadziramo s paketom `dynamic_reconfigure`. Te parametri so značilke na podlagi kerih iščemo grobo pot, to so torej srednja vrednost, standardna deviacija absolutnih kotov ter minimalna dolžina poti. Ko je nova pot pridobljena sledi še njena objava.

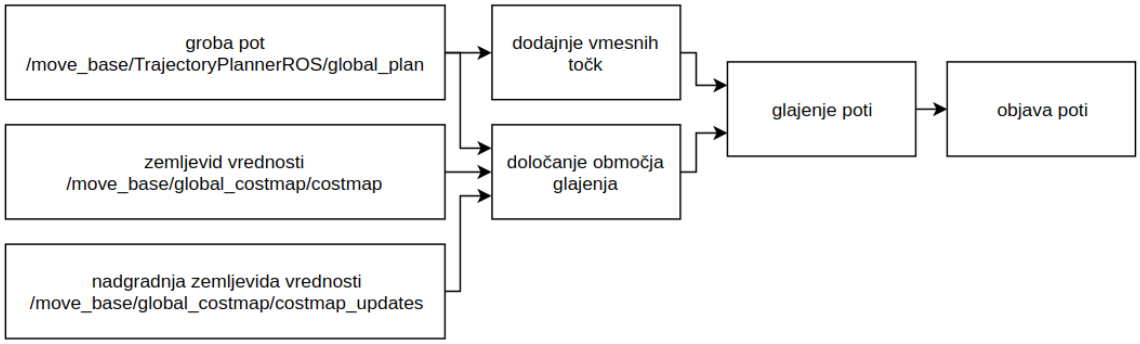
6.2.4 Glajenje poti

Jedro vozlišča, ki ga predstavlja glajenje poti je sestavljeno iz petih ključnih funkcij, ki so realizirane v programskem jeziku Python. Naloga prve funckije je branje obstoječe grobe poti, ki jo vrača navigacijski sklad. Prva funckija vrne grobo pot v urejeni obliki 2D polja točk. To polje točk nato prevzame druga funckija, ki doda vmesne točke kar poveča učinek algoritma glajenja. Na podlagi polja z dodanimi točkami ter zemljevida vrednosti se v tretji funkciji določa območje glajenja. Cilj te funkcije je določiti območje glajenja z ozirom na zemljevid vrednosti, to pa se izvede tako, da se skozi točke poti v pravokotni smeri inkrementalno premikamo vzdolž normale ter preverjamo pripadajoče mesto v zemljevidu vrednosti. Preverjanje zemljevida vrednosti je izvedeno tako, da se točko inkrementiranja na podlagi ločljivosti zemljevida pretvori v element matrike, ki predstavlja realizacijo zemljevida vrednosti in preveri vrednost omenjenega elementa. Ko dosežemo oviro ozrioma mejo inkremetiranja se postopek ustavi, dosežena razdalja pa se doda v polje razdalj od poti do ovire. Postopek se ponavlja za vsako točko poti na levi in desni strani glede na pot, tako dobimo dve polji (levo in desno) razdalj od poti do ovire. Primer območja v oklici poti lahko vidimo na sliki 7, kjer rdeči linji predstavljata meji območja, modra linija pa grobo pot. Na sliki 7 črna polja predstavljajo prisotnost ovire.



Slika 7: Primer določanja območja glajenja v programu RViz.

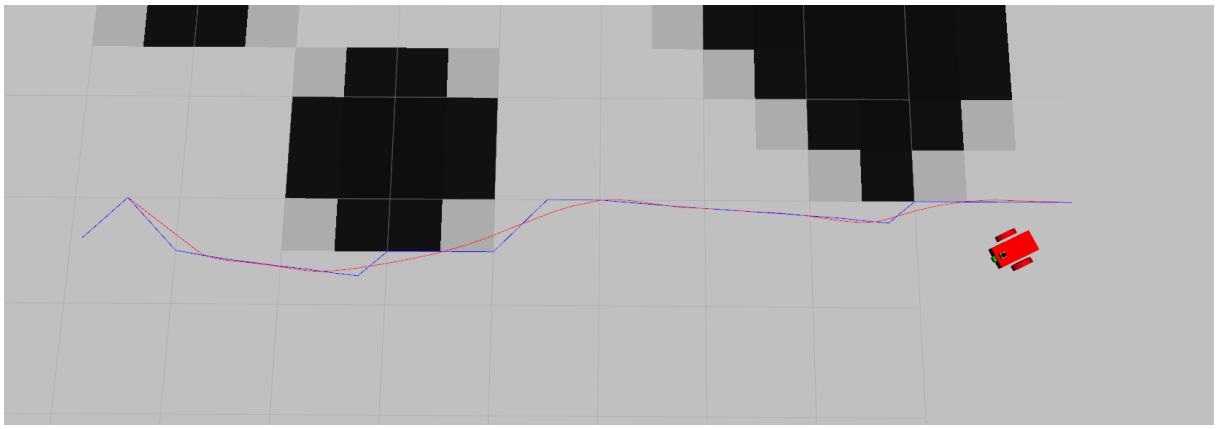
Polje točk grobe poti ter polji, ki tvorita območje galjenja se nato uporabi v funkciji s katero gladimo pot na način, ki je opisan v poglavju 5. Pri tem uporabimo paramtere ugotovljene v prvem stanju algoritma. Ostali paramteri glajenja so število iteracij s katerimi določamo lego po enačbi 16, korak inkrementalnega preverjanja območja glajenja, meja inkrementiranja ter prag vrednosti ovire v zemljevidu vrednosti. Ko je pot zglajena sledi njena objava v sporočilu oblike `/nav_msgs/Path`, ki vsebuje točke poti ter orientacijo v zadnji točki.



Slika 8: Shema delovanja algoritma glajenja poti.

7 Rezultati

Na primeru navigacijskega paketa robota z bloga moorerobots [10] sedaj preizkusimo delovanje ROS-ovega vozlišča. Na spodnji sliki (Slika 9) vidimo z modro označeno grobo pot ter rdečo gladko pot. S slike je moč ugotoviti očitno razliko v trajektoriji. Za primer na sliki znaša srednja vrednost absolutnih kotov grobe poti 5.54°, gladke pa 2.72°. V kontekstu povprečja kotov vidimo precejšnje izboljšanje. Gladka pot v temu primeru temelji na grobi poti kjer je med vsako točko poti vrinjena ena dodatna točka. Z dodajanjem vmesnih točk se računski čas precej poveča, medtem ko znatne spremembe v učinku ni moč opaziti. Če pogledamo sliko bolj podrobno, na levi strani opazimo, da konca poti sovpadata. Namen tega je, da s potjo skušamo robota postaviti v ciljno orientacijo.



Slika 9: Prikaz delovanja algoritma glajenja poti.

8 Zaključek

Kot smo ugotovili je algoritem primeren za reševanje problema glajenja poti. Pri tem pa je potrebno omeniti, da je kvaliteta končne, gladke poti odvisna od najdenih optimalnih parametrov. Pogoj za uspešno glajenje je tudi stanje osnovne grobe poti, glajenje je namreč problematično ob prisotnosti zelo velikih kotov med odseki poti (okolica 90°). Končna pot je takem primeru neuporabna.

Problemi bazične izvedbe algoritma tičijo v težko interpretabilnih parametrih, kar povzroča težave pri njihovemu nastavljanju s strani končnega uporabnika. Poleg tega pa se ustreznost paramterov spreminja nelinearno z dimenzijo problema. Kot smo ugotovili je ta problem mogoče rešiti z aplikacijo optimizacijskih agloritmov, kjer iščemo paramtere tako, da nam le-te optimirajo ustrezno kriterijsko funkcijo, ki je odraz kvalitete poti. V našem primeru smo to storili tako, da poskušamo pridobiti ustrezno slabo pot iz navigacijskega algoritma ter na njeni osnovi poskušamo najti optimalne parametre. Takšna rešitev je ustrezna za primer kjer se inicializacija vozila izvaja v okolju s prisotnostjo ovir. Če ovir ni v okolici avtonomnega vozili, je pridobitev grobe poti iz navigacijskega algortma nemogoča in tako postane iskanje ustreznih paramterov neizvedljivo.

Omenjeni problem iskanja paramterov bi bilo moč izboljšati tako, da se optimizacija izvaja na osnovi grobe poti, polja točka, ki je programu vedno na voljo (polje v programu ali tekstovna datoteka) in se ga pred izvajanjem skalira v skladu z dimenzijo problema. Ustrezni faktor bi bilo moč najti z še eno optimizacijo, na primer z minimizacijo napake med povprečinimi razdaljami grobe poti ter neke poti navigacijskega okolja avtonomnega vozila. Skalirni faktor bi bilo mogoče določiti tudi na podlagi informacije o ločljivosti zemljevida vrednosti.

Ostali načrti za prihodnost bi lahko obsegali tudi preizkus na pravemu vozilu, prepis vozlišča v C++, ki ga ROS podpira poleg Python-a. S slednjim bi omogočili večjo učinkovitost v smislu zasedenosti računalniškega spomina in hitrosti računanja.

Na sliki 9 vidimo, da je glajenje poti dokaj lokalno ali z drugimi besedami, obstaja bolj neposredna pot od vozila do cilja. Za izboljšavo omnejenega opažanja bi lahko preizkusili uporabo lokalnih paramterov glajenja oziroma takšno glajenje, da ima vsaka točka glajenja poti svoj set paramterov. Pri tem problemu pa se je smiselno vprašati ali je to problem domene glajenja poti ali določanje globalne poti v okviru navigacijskega sklada.

9 Literatura

[1] Wikipedia, Differential wheeled robot, https://en.wikipedia.org/wiki/Differential_wheeled_robot

[2] Wikipedia, Racing line, https://en.wikipedia.org/wiki/Racing_line

[3] L. Cardamone, D. Loiacono, P. L. Lanzi, A. P. Bardelli. *Searching for the optimal racing line using genetic algorithms, in: Computational Intelligence and Games (CIG), 2010 IEEE Symposium on, IEEE, 2010, pp. 388–394.*

[4] M. Pfeiffer. *Machine learning applications in computer games, ÖGAI- Journal 23 (3) (2004) 4–7.*

[5] R. Coulom. *Reinforcement learning using neural networks, with applications to motor control, Ph.D. thesis, Institut National Polytechnique de Grenoble-INPG (2002).*

[6] W.-D. Beelitz, Simplix, http://www.simplix.wdbee-aorp.de/default.aspx

[7] RARS, Robot auto racing simulator, http://rars.sourceforge.net/

[8] M. Botta, V. Gautieri, D. Loiacono, P. L. Lanzi. *Evolving the optimal racing line in a high-end racing game, in: Computational Intelligence and Games (CIG), 2012 IEEE Conference on, IEEE, 2012, pp. 108–115.*

[9] S. Varner, Computer-controlled cars in vamos, http://vamos.sourceforge.net/computer-controlled-cars.pdf

[10] moorerobots, mybot, http://moorerobots.com/blog

[11] I. Škrjanc. *Inteligentni sistemi za podporo odločanju, Ljubljana 2016.*