

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, 30. 4. 2018

Kazalo

1	Uvod	2
2	Ozadje problema	2
2.1	Zgradba avtonomnega vozila	2
2.2	Robot Operating System	2
2.3	Opis problema	3
3	Namen in cilji	3
4	Pregled možnih rešitev	4

Slike

1	Primer dveh vozlišč z medsebojno povezavo v okolju ROS.	3
2	Primerjava poti, ki ju tvori navigacijski algoritem.	3
3	Dirkalna linija.	4

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 praznjenju baterij. To pomeni, da je potrebno baterije vozila večkrat polniti, ter da je vozilo manj časa razpoložljivo. Problem predstavlja tudi potencialno nevarnost, da vozilo zaradi sunkovitih gibov izgubi svoj tovor.

Naša naloga je bila izboljšati pot, ki jo tvori navigacijski algoritem in posledično omogočiti vozilu, da bo delovalo energijsko učinkovitejše, zanesljivejše, ter da bo 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 smotno 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.

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 le 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, planiranje gibanj, manipuliranje 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 sabo 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 pogledajmo 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.



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

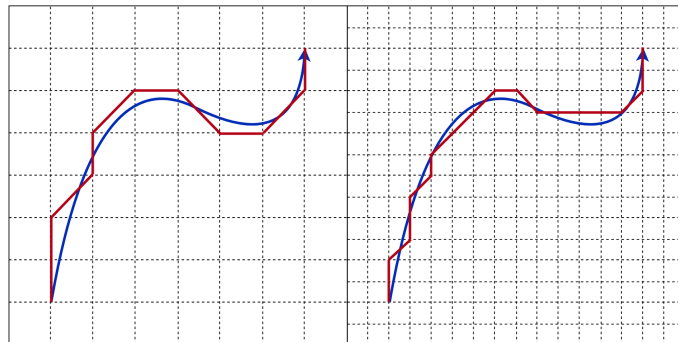
2.3 Opis problema

Seznajeni 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. Z neskončno majhno diskretizacijo prostora, bi rdeča in modra pot sovpadali.



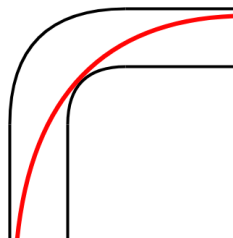
Slika 2: Primerjava poti, ki ju tvori navigacijski algoritem.

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 posredovala 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. 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. 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. 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.

Odprtokodne dirkalne igre se ponavadi zanašajo na hevristične pristope, ki tipično združujejo dobro prakso s hevristikami pri generaciji dirkalnih linij. Najuspešnejši primeri takšnih pristopov vključujejo K1999 algoritem, ki ga je razvil Remi Coulom in deluje na podlagi gradientnega spusta (*angl.: gradient descent*) ter Simplix, ki ga je razvil Wolf-Dieter Beelitz za The Open Car Racing Simulator in bazira na preprosti hevristici. Ostala uspešna primera sta tudi *the bot*, 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 ukrivljenost poti vozila skozi ovinek, saj z manjšo ukrivljenostjo 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 ustrežnejših rešitev omenjenega problema. Na drugi strani pa imamo rešitve, ki so malce preprostejše kot je uporaba kubičnih parametričnih Bézier krivulj pri generaciji dirkalne linije in pa edinstvena rešitev, ki jo nudi odprtokodni Vamos Avtomotive Simulator, kjer je za doseganje manjše ukrivljenosti 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.