Glajenje poti za avtomatsko vodena vozila

Marsel Rus[a], Martin Knap[a], Rok Vrabič[a]

[a] *Univerza v Ljubljani, Fakulteta za strojništvo*

Povzetek

Pri razvoju vozil na področju robotike se pogosto srečujemo s problemom tvorjenja gladke, energijsko učinkovitejše poti, namenjene navigaciji vozila. V tem delu preučimo možne rešitve ter eno od teh apliciramo na avtonomnem vozilu v razvoju. Rešitev problema najdemo na področju motornih športov, kjer se vozniki močno poslužujejo tehnike dirkalne linije za učinkovitejšo vožnjo preko zavojev. Same metode tvorbe dirkalne linije nudijo dirkalni simulatorji, kjer je uporabljena kot trajektorija AI vozil.

Ključne besede: ROS, Glajenje poti, Dirkalna linija, Avtonomno vozilo, Robotika, Navigacija robota

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. Problem predstavlja tudi potencialno nevarnost, da vozilo zaradi sunkovitih gibov izgubi svoj tovor.

Naša naloga je bila izboljšanje poti, 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. Definicija 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. V podpoglavju bomo predstavili tudi način krmiljenja vozila v programskem okolju *Robot Operating System* (ROS).

2.1. Zgradba avtonomnega vozila

Obravnavano vozilo je tipičen predstavnik robotov z diferencialnim pogonom koles (*angl.: differential wheeled robot*) [1]. 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.

Email addresses: rus.marsel@gmail.com (Marsel Rus), martin93.knap@yahoo.com (Martin Knap), rok.vrabic@fs.uni-lj.si (Rok Vrabič)

2.2. Robot Operating System

Upravljanje z vozilom nam v celoti nudi programsko okolje *Robot Operating System* (ROS). To je odprtokodno okolje, ki nudi širok nabor orodij za upravlajnje z robotom, kot so na primer krmilniki za branje senzorskih informacij, krmilniki za upravlajnje z aktuatorji, fundamentalni robotski algoritmi (kartografija, navigacija, planiranje gibanja, interpretacija senzorskih podatkov, ...) in računska infrastruktura za manipulacijo s podatki tako, da je omogočena komunikacija med različnimi komponentami kompleksnega robotskega sistema. Zelo pomembno je to, da nam ROS nudi vključevanje lastnih algoritmov.

Osnovni entiteti ROS-a sta vozlišče (*node*) in komunikacijska povezava (*topic*). Vozlišče je program z določeno funkcijo v sistemu, komunikacijska povezava pa služi povezovanju med vozlišči in tako omogoča tok informacij. Informacije, ki tečejo preko povezav imajo strogo določeno obliko.

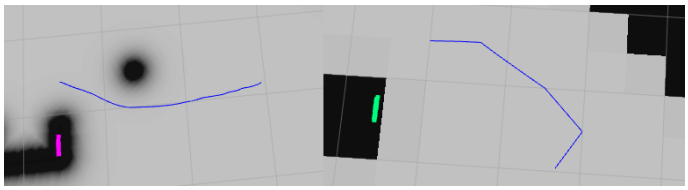
2.3. Opis problema

Seznanjeni z okvirnim delovanjem krmilnega okolja avtomatskega vozila se lahko osredotočimo na problem. Navigacija v okolju ROS se izvaja na podlagi dvodimenzionalnih zemljevidov, mrež. 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 levo vidimo pot pri visoki ločljvosti mreže, na desni pa pot pri nizki ločljivosti mreže.

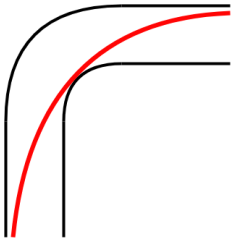


Slika 1: Primerjava poti, ki ju tvori navigacijski algoritem.

Naša naloga je torej najti metodo, ki nam bo omogočala transformacijo nabora točk, ki tvorijo grobo pot v tak nabor točk, ki bodo tvorile zglajeno, energijsko učinkovitejšo pot vozila. V kontekstu okolja ROS bo rešitev izvedena tako, da se bo ustvarilo dodatno vozlišče v katermeu se bo izvajala transformacija. To vozlišče bo bralo sporočila o poti, ki izhajajo iz ustreznega vozlišča navigacijskega algoritma, jih transformirala in pošiljala naslednjemu vozlišču, ki je izvaja sledenje poti.

3. 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 2: 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 Microsoftovi 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].

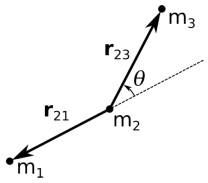
Odprtokodne dirkalne igre se ponavadi zanašajo na hevristične pristope, ki tipično združujejo dobro prakso s hevristiko pri generaciji dirkalnih linij. Najuspešnejši primeri takšnih pristopov vključujejo K1999 algoritem [5], ki ga je razvil Remi Coulom in deluje na podlagi gradientnega spusta (*angl.: 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.

4. Algoritem glajenja poti - Vamos

Vamos Avtomotive Simulator (VAS) je odprtokodno simulacijsko ogrodje s poudarkom na fizičnemu modeliranju [9]. Vamos modelira veliko večino ključnih podsistemov avtomobilov, kot so elementi pogosnkega 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 glajenja poti tako, da igralcem nudijo karseda realističen izziv. Ta algoritem bomo v naslednjem delu besedila vzeli pod drobnogled in ga nato tudi uporabili pri našem problemu.

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 3: Tri vozlišča poti. Kot θ povzorča moment [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.

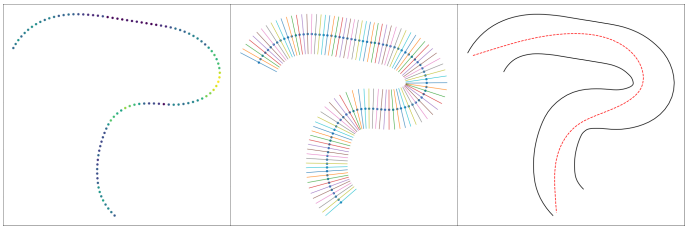
Ko je izračunana skupna sila na vsakemu vozlišču lahko izračunamo novo lego točke z uporabo Newtonov-ovih zakonov giabnja ter z Euler-jevo metodo.

$$v_{i+1} = v_i + (F/m - cv_i)\Delta t \tag{11}$$

$$r_{i+1} = r_i + v_{i+1}\Delta t \tag{12}$$

Z iteriranjem algoritma se lega zglajenega nabora točk stabilizira. Pri tem se je potrebno zavedati, da je pot lahko groba v ostrih ovinkih.

Spodnje slike prikazujejo delovanje dejanskega algoritma glajenja. Leva slika prikazuje graf niza točk nad katerimi bomo izvajali glajenje. Slika v sredini prikazuje tvorbo proge v okolici omenjenih točk. Proga je ključna za glajenje poti, saj določa območje v kateremu ga izvajamo. Zunanji in notranji del proge tvorijo točke, ki ležijo na premicah, ki so pravokotne na pot. Tukaj moramo biti pozorni na širino proge v ostrih zavojih saj se omenjene premice lahko medsebojno sekajo, kar vodi do težav. Na desni sliki pa vidimo niz točk v novi legi.



Slika 4: Postopek glajenja.

5. Aplikacija algoritma

Z znanim algoritmom glajenja sedaj lahko začnemo 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, odgovornemu 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. Preden zančenmo na delu s samim vozliščem pa je potrebno ustvariti okolje kjer naš program lahko preizkusimo.

5.1. 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. Še pomembnejše pa 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 pa vsebuje preprosto vozilo z

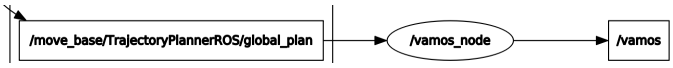
diferencialnim pogonom. Delali bomo z ROS izvedbo Kinetic Kame na operacijskem sistemu Linux Ubuntu. Z definiranim okoljem sedaj lahko začnemo na razvoju omenjenega vozlišča vokolju ROS.

5.2. Izdelava vozlišča

Za navigacijo robota v okolju ROS skrbi t.i. navigacijski sklad, katerega ključno 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`.

Algoritem je napisan v programskem jeziku Python in je v grobem sestavljen iz petih funkcij. Naloga prve funkcije je branje obstoječe, grobe poti in tvorjenje izhodnega podatka v obliki 2D niza točk, kjer prvi stolpec predstavlja x koordinate, drugi stolpec pa y koordinate nezglajene poti. Naloga druge funkcije je dodajanje vmesnih točk med točke nezglajene poti, saj je groba pot sestavljena iz malega števila točk. Tako omogočimo, da se postopek glajenja izvaja nad večjim številom točk, kar še dodatno pripomore k tvorbi ustreznejše poti. Tretja funkcija bere zemljevid vrednosti in informacijo o zemljevidu pretvarja v matriko čitljive oblike, kjer lega posameznega elementa sovpada s točko kartografiranega prostora. V tretji funkciji se določa širina območja glajenja in sicer tako, da se inkrementalno povečuje leva in desna meja območja nakar se za vsak inkrement širine preveri, če katera od koordinat leve in desne meje leži v območju ovire. To se izvede tako, da se koordinate leve in desne meje območja glajenja prevede v pripadajoče elemente matrike vrednosti in preveri, če je pripadajoča vrednost pod mejo. Ko z inkrementiranjem dosežemo oviro je prag sprejemljive vrednosti presežen in inkrementiranje se zaključi, s tem pa določimo širino območja glajenja. Širina območja inkrementalnega povečevanja širine je omejeno, saj bi se v okolici brez ovir inkrementiranje izvajalo, do roba zemljevida. Postopek se izvajala ločeno za levo in desno mejo.

Četrta funkcija predstavlja jedro programa kjer se nahaja Vamos gladilni algoritem. V tem delu programa se torej izvaja transformacija točk prebranih iz druge funkcije. Ta funkcija vsebuje tudi vplivne parametre glajenja kot so masa točke, togostni koeficient, število iteracij algoritma ter širina proge, ki se jo določi v tretji funkciji. Izhodni podatek je ponovno 2D niz točk, le da so te transformirane. V tretji funkciji se izvaja prepis zglajenega niza točk v sporočila oblike `nav_msgs/Path` ter njihovo oglaševanje po komunikacijskem kanalu `/vamos`. Na ta kanal se nato poveže vozlišče odgovorno za vodenje robota po globalni poti.

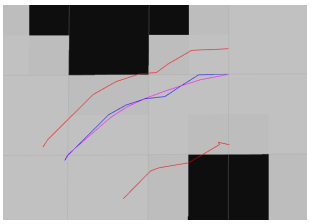


Slika 5: Grafičen prikaz povezav.

Vozlišče med drugim koristi tudi ROS-ov paket Dynamic Reconfigure, ki omogoča spreminjanje parametrov vozlišča tekom njegovega obratovanja. Z omenjenim paketom nudimo manipulacijo nad parametri v drugi, treji in četrti funkciji.

6. Diskusija in zaključek

Z izdelanim paketom v okolju ROS je sledil preizkus. Testnemu robotu smo podali ciljno točk in globalni načrtovalec navigacijskega algoritma je izdelal globalno pot, ki jo na spodnji sliki vidimo v modri barvi. Ker paket RViz omogoča prikaz dodatnih poti je sledil izris zglajene poti, ki jo vidimo v roza barvi ter izris leve in desne meje območja glajenja.



Slika 6: Prikaz poti.

Iz zgornje slike se jasno vidi upoštevanje okoliških ovir ter precej ugodnejši potek nove poti. Izdelan paket omogoca delovanje določeno v okviru zadane naloge, vendar še vedno obstaja mnogo možnosti za izboljšave, kot je na primer upoštevanje bremena in hitrosti.

[1] t. f. e. Wikipedia, Differential wheeled robot.
URL `https://en.wikipedia.org/wiki/Differential_wheeled_robot`

[2] t. f. e. Wikipedia, Racing line.
URL `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.
URL `http://www.simplix.wdbee-aorp.de/default.aspx`

[7] RARS, Robot auto racing simulator.
URL `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.
URL `http://vamos.sourceforge.net/computer-controlled-cars.pdf`

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