Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Peter Fajdiga

Samodejna tekmovalna vožnja po modelu dirkalne steze

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Bojan Orel

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Razvijte program, ki znotraj simulacije vodi vozilo po dirkalni stezi tako, da doseže čim boljši čas. Prilagojen naj bo posameznemu vozilu – iskanje optimalne poti naj upošteva njegove fizikalne lastnosti.

Kazalo

Seznam uporabljenih oznak

oznaka	pomen
$\mathbf{v}$	hitrost
$\widehat{\mathbf{T}}$	normalizirana tangenta
s	dolžina loka
t	čas
κ	ukrivljenost
w_v	širina vozila
$\mathbf{a}_{pogon}(\lvert\mathbf{v}\rvert)$	največji možni pogonski pospešek pri dani hitrosti
$\mathbf{a}_a$	največji možni pospešek vozila
$\mathbf{a}_b$	največji možni zavorni pospešek vozila
$\mathbf{v}_{mejna}$	največja hitrost, ki jo oprijem omogoča
P	točka na poti

Povzetek

Naslov: Samodejna tekmovalna vožnja po modelu dirkalne steze

Avtor: Peter Fajdiga

V diplomskem delu je predstavljeno iskanje optimalne poti, ki omogoča vožnjo po danem dirkališču v najmanjšem možnem času, in simulirana vožnja po njej. Razložene so relevantne fizikalne osnove ter vpliv lastnosti vozila na optimalno pot in na potreben čas za en krog po njej. Prikazano je ocenjevanje trajanja posamezne poti in dva načina minimiziranja tako izračunane vrednosti: enostaven linearen algoritem in genetski algoritem. Nazadnje je implementiran program, ki po najdeni poti vodi vozilo znotraj simulacije v igralnem pogonu Unreal Engine 4.

Ključne besede: samodejna vožnja, dirkalna steza, optimalna pot, optimizacija.

Abstract

Title: Autonomous Competitive Driving on a Race Track Model

Author: Peter Fajdiga

This thesis presents the problem of finding the optimal racing line, which enables completing a lap on a given race track in the shortest time possible, and a simulation of a car following the line. It explains the relevant physical basics and the effect vehicle properties have on the optimal racing line and lap time. A method of estimating the duration of each individual path is presented, along with two optimization methods for minimizing the calculated value: a simple linear algorithm and a genetic algorithm. Lastly, we implement a program that guides a simulated vehicle on the racing line using Unreal Engine 4.

Keywords: autonomous driving, race track, racing line, optimization.

Poglavje 1

Uvod

Računalniško vodena vozila v dirkalnih igrah in simulacijah morajo poznati pot po dirkališču in znati po njej voziti, da so tekmovalna in igralcu predstavljajo izziv.

Vsaka pot po progi pa ni enako dobra – biti mora čim manj vijugasta in čim krajša. Hkrati pa je treba upoštevati fizikalne lastnosti vozila. Boljša pot omogoča vozilu, da prevozi stezo v krajšem času. Eden od ciljev diplomskega dela je iskanje čim boljše poti po danem dirkališču. Optimalna pot ima širšo uporabo kot le to, da je uporabljena za računalniško vodeno vozilo. Tudi pri resničnih dirkah se preučuje kakšno linijo vzeti in vozniki se jo trudijo čim bolje upoštevati.

V naslednjem poglavju je predstavljenih nekaj obstoječih rešitev iskanja optimalne poti po stezi. Sledi poglavje o geometrijskih in fizikalnih osnovah, ki vplivajo na tekmujoče vozilo in posredno tudi na pot, ki mu omogoča najboljši čas, in za njim poglavje, kjer je natančneje predstavljen problem iskanja optimalne poti.

V poglavju Genetski algoritmi je razloženo, kaj to so in kako delujejo. Na kratko je tudi predstavljena knjižnica JGAP za delo z njimi.

V svojem poglavju je razložena implementacija optimizacije poti, kjer sta opisana dva načina njene izvedbe – preprost linearen algoritem in genetski algoritem. Oba se lahko uporabita za minimizacijo različnih vrednosti:

dolžine poti, njene ukrivljenosti ali ocene časa, ki ga določeno vozilo potrebuje za vožnjo po njej.

V poglavju Simulacija je predstavljena programska oprema, ki simulacijo poganja, in model vozila v njej.

V zadnjem poglavju diplome pa je implementiran program, ki vodi vozilo po prej izračunani poti v igralnem pogonu Unreal Engine 4.

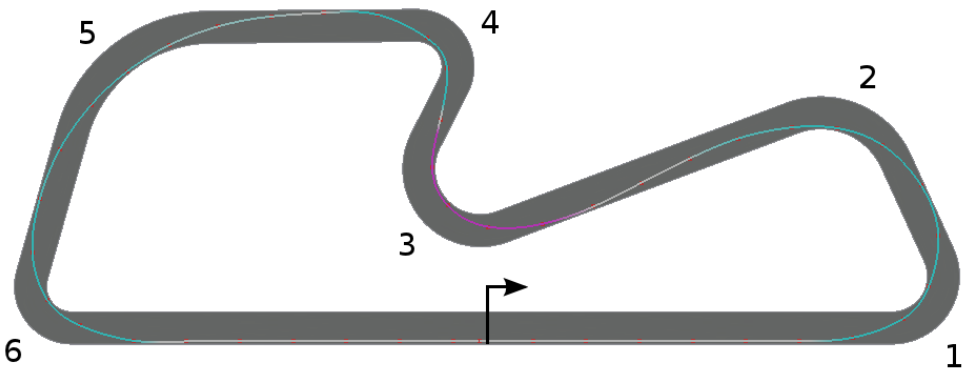
1.1 Pregled področja

1.1.1 Patent US 7892078 B2

Ameriški patent US 7892078 B2 [1] (ponovno objavljen kot US 8393944 B2 [2]) opisuje algoritem za minimizacijo ukrivljenosti poti po dirkališču, ne minimizira pa časa, kar pomeni, da fizika tu ne igra nobene vloge.

Omogoča tudi pripenjanje poti v delih proge, s čimer se omejijo dovoljene rešitve tako, da morajo potekati skozi pripete točke. Algoritem bo potem poiskal najmanj ukrivljeno pot, ki gre skozi vse pripete točke.

1.1.2 Vamos

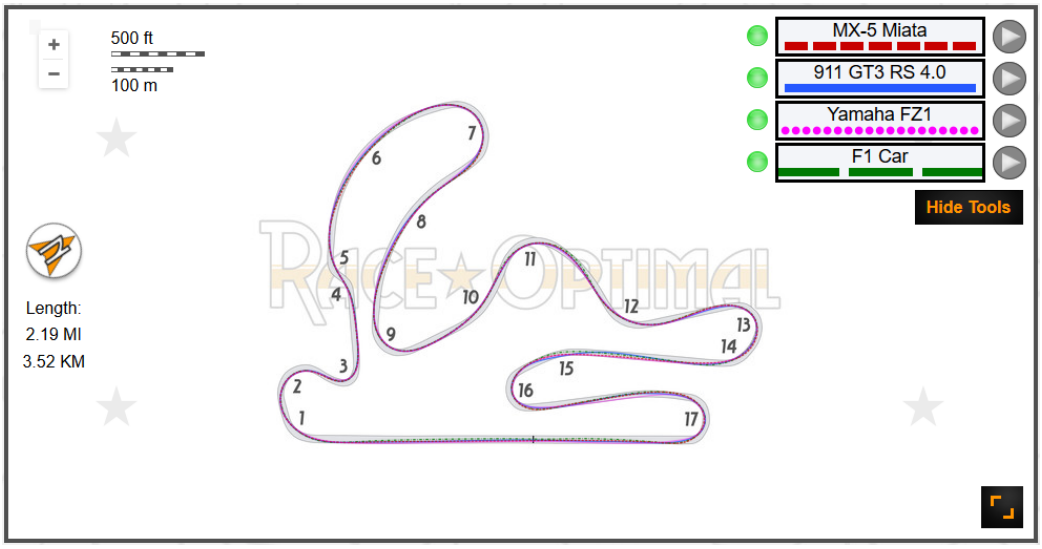


Slika 1.1: Vamosov rezultat po 800 iteracijah (139 vozlišč) [3]

Optimizacija v avtomobilskem simulacijskem ogrodju Vamos [4] prav tako minimizira le ukrivljenost poti.

Tu minimizacija ukrivljenosti deluje po unikatnem algoritmu, ki simulira verigo mas na vzmetnih tečajih [3]. Mase so na začetku razporejene po sredinski liniji proge in so omejene tako, da se ne morejo premakniti z roba proge. Simulacija potem poskrbi, da se minimizira energija, shranjena v vzmeteh.

1.1.3 Race Optimal



Slika 1.2: Spletna stran Race Optimal, ki prikazuje špansko progo Circuito de Cartagena s štirimi potmi po njej, vsaka optimizirana za svoje vozilo [5]

Race Optimal [6] je plačljiva spletna storitev namenjena voznikom, ki želijo izboljšati svoj čas na stezi.

Uporabnik najprej vnese podatke o svojem vozilu in izbere dirkališča, na katerih se želi izboljšati. Program mu nato izračuna optimalno pot po dirkališču. Ponujajo tudi ogled simulirane vožnje po izračunani poti in primerjavo optimalne vožnje z uporabnikovo lastno vožnjo, če jo je uporabnik posnel z GPS napravo in zaupal njihovemu sistemu.

Storitev za minimizacijo časa uporablja genetski algoritem, čas posamezne poti pa oceni glede na lastnosti vozila.

Poglavje 2

Geometrijske in fizikalne osnove

V tem poglavju so predstavljene geometrijske in fizikalne osnove, kot so fizikalne konstante, lastnosti vozil in enačbe, na katere se bom skliceval v tem diplomskem delu.

2.1 Frenetovo ogrodje

Gibanje po trirazsežni krivulji lahko opišemo z uporabo Frenetovega ogrodja. Za vsako točko na krivulji lahko izračunamo tri med seboj pravokotne vektorje:

- $\mathbf{T} = \mathbf{v}$ je tangenta oziroma hitrost in kaže v smer gibanja,
- $\mathbf{N}$ je normala,
- $\mathbf{B}$ je binormala.

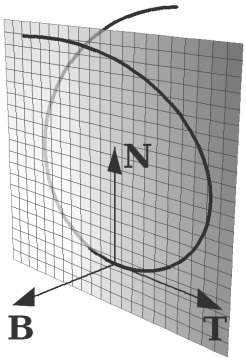
Naj bo $\mathbf{r}(t)$ trirazsežna krivulja, ki predstavlja krajevni vektor $\mathbf{r}$ položaja gibajočega se delca v času t. Tangento $\mathbf{T}$, normalo $\mathbf{N}$ in binormalo $\mathbf{B}$

izračunamo z enačbami: [7]:

$$\mathbf{T} = \frac{d\mathbf{r}}{dt} \tag{2.1}$$

$$\mathbf{B} = \mathbf{T} \times \frac{d^2\mathbf{r}}{dt^2} \tag{2.2}$$

$$\mathbf{N} = \mathbf{B} \times \mathbf{T} \tag{2.3}$$



Slika 2.1: Ilustracija vektorjev $\mathbf{T}$, $\mathbf{N}$ in $\mathbf{B}$ [8]

Če jih normaliziramo, dobimo enotske vektorje $\widehat{\mathbf{T}}$, $\widehat{\mathbf{N}}$ in $\widehat{\mathbf{B}}$ [7]:

$$\widehat{\mathbf{T}} = \frac{\mathbf{T}}{|\mathbf{T}|} = \frac{d\mathbf{r}}{ds} \tag{2.4}$$

$$\widehat{\mathbf{N}} = \frac{\mathbf{N}}{|\mathbf{N}|} = \frac{\frac{d\widehat{\mathbf{T}}}{ds}}{\left|\frac{d\widehat{\mathbf{T}}}{ds}\right|} \tag{2.5}$$

$$\widehat{\mathbf{B}} = \frac{\mathbf{B}}{|\mathbf{B}|} = \widehat{\mathbf{T}} \times \widehat{\mathbf{N}} \tag{2.6}$$

Pri tem s označuje dolžino loka.

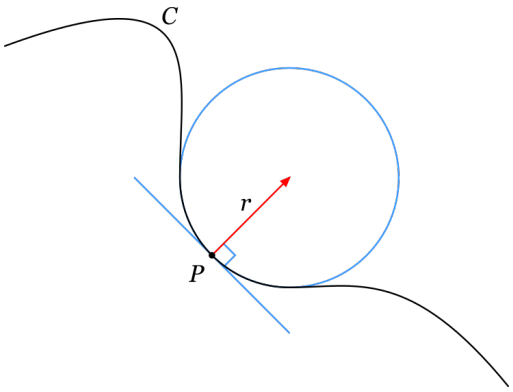
2.1.1 Ukrivljenost

Ukrivljenost κ krivulje v neki točki je definirana z enačbo [7]:

$$\kappa = \left|\frac{d^2\mathbf{r}}{ds^2}\right|. \tag{2.7}$$

Hkrati je ukrivljenost enaka obratni vrednosti polmera r v tej točki pritisnjene krožnice [7]:

$$\kappa = \frac{1}{r}. \tag{2.8}$$



Slika 2.2: Pritisnjena krožnica v točki P krivulje C [9]

2.2 Fizikalne konstante

g označuje gravitacijski pospešek objekta v vakumu blizu površine Zemlje. Definiran je kot $9.80665\frac{m}{s^2}$ [10].

Gostota zraka ρ se v tem delu smatra kot konstanta enaka $1.2041\frac{kg}{m^3}$, kar je gostota zraka pri temperaturi 20°C pri morski gladini [11].

2.3 Lastnosti vozila

Vsako vozilo je drugačno, zato bo na isti stezi doseglo drugačen čas. Prav tako se za različna vozila razlikuje tudi zanje optimalna pot po progi. Lastnosti vozila, ki vplivajo na optimalno pot, so:

- širina vozila w_v;
- masa m;

- koeficient trenja pnevmatik s stezo μ;
- največji možni pogonski pospešek pri dani hitrosti $\mathbf{a}_{pogon}(|\mathbf{v}|)$ – tak pospešek bi vozilo doseglo, če ne bi bilo omejeno z oprijemom in zračnim uporom;
- referenčna površina vozila S, ki vpliva na aerodinamiko;
- koeficient zračnega upora c_d;
- koeficient pritisne sile (angleško downforce coefficient) c_{df}, ki je nasproten koeficientu dinamičnega vzgona c_l.

2.4 Aerodinamika

Zračni upor (angleško aerodynamic drag) $\mathbf{F}_d$ je glavna ovira, s katero se sooča karkoli, kar želi hitro potovati skozi zrak. Pri visokih hitrostih je tok zraka turbulenten. Tedaj se enačba upora $\mathbf{F}_d$ glasi [12]:

$$\mathbf{F}_d = -\frac{\rho \mathbf{v} \left|\mathbf{v}\right| S c_d}{2}, \tag{2.9}$$

kjer je $\mathbf{v}$ hitrost gibanja vozila skozi zrak. Pri nizkih hitrostih pa je tok zraka laminaren in za zračni upor velja drugačna enačba [13]:

$$\mathbf{F}_d = -\zeta \mathbf{v}, \tag{2.10}$$

kjer je ζ konstanta, ki je odvisna od lastnosti tekočine in dimenzij predmeta.

Ta upor je vzrok pospeška $\mathbf{a}_d$, ki tako kot $\mathbf{F}_d$ kaže v smer nasproti potovanju vozila:

$$\mathbf{a}_d = \frac{\mathbf{F}_d}{m}. \tag{2.11}$$

Tok zraka pa je lahko tudi pozitivno izkoriščen. Večina dirkalnih avtomobilov je oblikovanih tako, da jih tok zraka pritiska v tla in s tem izboljša njihov oprijem. Nastala pritisna sila (angleško downforce) $\mathbf{F}_{df}$ je nasprotna sili dinamičnega vzgona (angleško lift) $\mathbf{F}_l$ in sledi naslednji enačbi [14]:

$$\mathbf{F}_{df} = \frac{\rho \widehat{\mathbf{k}} \left|\mathbf{v}\right|^2 S c_{df}}{2} = -\mathbf{F}_l, \tag{2.12}$$

kjer je $\widehat{\mathbf{k}}$ enotski vektor, ki je usmerjen od vozila navpično navzdol.

2.5 Oprijem

Oprijem omogoča vozilu, da ne zdrsi s ceste ter da se sploh lahko premakne in kasneje tudi ustavi. Ker je oprijem omejen, je s tem omejeno tudi zavijanje, zaviranje, pri močnejših vozilih pa tudi pospeševanje. Največji pospešek $|\mathbf{a}_{mejni}|$, ki ga oprijem omogoča, je definiran z enačbo [15]:

$$|\mathbf{a}_{mejni}| = \mu \left(g + \frac{|\mathbf{F}_{df}|}{m} \right). \tag{2.13}$$

Ko vozilo zavija in zato ukrivljenost κ ni enaka 0, to izkorišča del oprijema, ki je na voljo. Zato je pri zavijanju tangencialno pospeševanje oziroma zaviranje omejeno še bolj.

Da izračunamo tangencialni pospešek $\mathbf{a}_{mejni,t}$, ki nam je na voljo, moramo najprej izračunati normalno komponento $\mathbf{a}_{mejni,n}$:

$$\mathbf{a}_{mejni,n} = |\mathbf{v}|^2 |\kappa| . \tag{2.14}$$

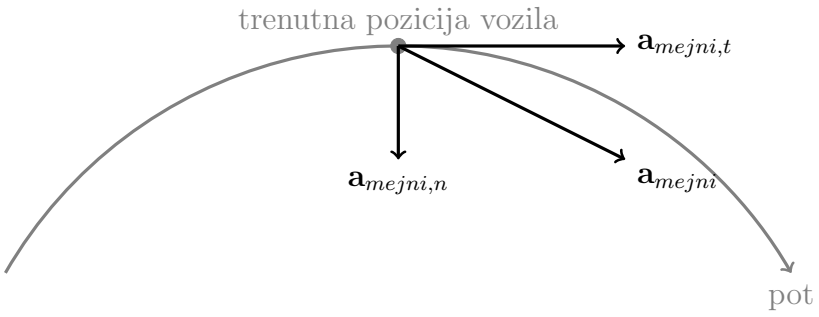
Zdaj lahko po Pitagorovem izreku izračunamo še tangencialno komponento $\mathbf{a}_{mejni,t}$, ki ima enako usmerjenost kot gibanje vozila:

$$|\mathbf{a}_{mejni,t}| = \sqrt{|\mathbf{a}_{mejni}|^2 - |\mathbf{a}_{mejni,n}|^2}. \tag{2.15}$$

Tangencialna komponenta $\mathbf{a}_{mejni,t}$ je največji možni tangencialni pospešek pri ukrivljenosti κ, ki ga omogoča oprijem.

2.6 Pospeševanje in zaviranje

Ko vzamemo v poštev še pospešek $\mathbf{a}_{pogon}(|\mathbf{v}|)$, ki ga vozilo s svojim pogonom lahko ustvari, in nasprotni pospešek zaradi zračnega upora $\mathbf{a}_d$, dobimo



Slika 2.3: Komponenti pospeška $\mathbf{a}_{mejni}$

končni največji mogoči pospešek vozila, ki se s hitrostjo $\mathbf{v}$ vozi po poti z ukrivljenostjo κ:

$$\mathbf{a}_a = \min \begin{pmatrix} \mathbf{a}_{pogon}(|\mathbf{v}|) + \mathbf{a}_d, \\ \mathbf{a}_{mejni,t} \end{pmatrix}. \tag{2.16}$$

Pri zmanjševanju hitrosti pa moramo upoštevati le oprijem in upor zraka. Z naslednjo enačbo dobimo pospešek $\mathbf{a}_b$, ki kaže v smer nasproti gibanju vozila in predstavlja maksimalno mogoče zaviranje:

$$\mathbf{a}_b = -\mathbf{a}_{mejni,t} + \mathbf{a}_d. \tag{2.17}$$

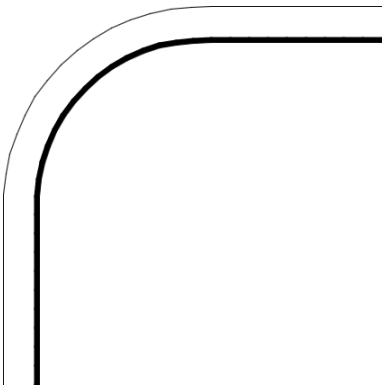
Poglavje 3

Iskanje optimalne poti

3.1 Opis problema

Za najboljši čas na dani dirkalni stezi je treba čim bolje izkoristiti prostor, ki nam je na voljo, in upoštevati fizikalne lastnosti vozila.

Če bi predpostavili, da je oprijem vozila idealen – tak, ki bi omogočal vožnjo po poti s kakršnokoli ukrivljenostjo s kakršnokoli hitrostjo, bi bila optimalna pot tista, ki je najkrajša.

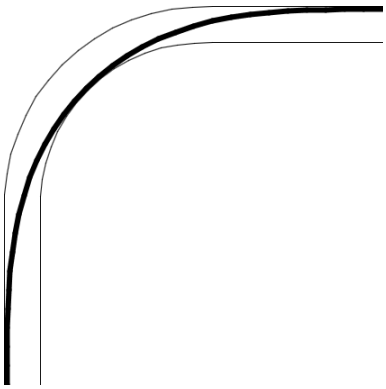


Slika 3.1: Minimiziranje dolžine poti skozi ovinek

Vendar pa je oprijem vsakega vozila omejen. Zato je najkrajša pot najboljša le, če oprijemu vozila ne predstavlja izziva in tako omogoča enako

hitrost, kot bi jo pot z manjšo ukrivljenostjo – na primer, če imamo že iz kakšnega drugega razloga nizko hitrost ali če ovinek ni zelo oster oziroma ima naše vozilo dovolj dober oprijem.

Če prestopimo v drugo skrajnost ter predpostavimo, da naše vozilo nima omejene najvišje hitrosti in da lahko pospeši (ali se upočasni) do kakršnekoli hitrosti v trenutku, vendar pa tokrat upoštevamo oprijem, pa je najboljša pot tista, ki je najmanj ukrivljena. Manj ostro zavijanje namreč omogoča vožnjo z višjo hitrostjo. Ponavadi se za najmanj ukrivljeno pot smatra tista, ki minimizira vrednost $\int_0^d \kappa(s)^2 ds$ [16], kjer $\kappa(s)$ predstavlja ukrivljenost v točki $P(s)$ (točka, ki leži na poti pri dolžini loka s), d pa je dolžina celotne poti.



Slika 3.2: Minimiziranje vrednosti $\int_0^d \kappa(s)^2 ds$ poti skozi ovinek

Če želimo najkrajši možni čas, potrebujemo pot, ki je glede na lastnosti vozila uravnotežena med obema skrajnostma, med najkrajšo in najmanj ukrivljeno potjo.

3.2 Ocena časa

Da lahko minimiziramo čas, potrebujemo nek način za določitev trajanja posamezne poti. Lahko simuliramo vožnjo po njej in čas izmerimo ali pa

trajanje matematično ocenimo:

$$t = \int_0^d \frac{ds}{|\mathbf{v}(s)|}, \tag{3.1}$$

kjer je d dolžina celotne poti, $\mathbf{v}(s)$ pa je hitrost, ki jo ima vozilo v točki $P(s)$.

3.2.1 Najvišja hitrost, ki jo ukrivljenost omogoča

Preden lahko določimo, kakšno hitrost ima vozilo v neki točki, moramo najprej izračunati, kakšna je najvišja hitrost, ki jo oprijem v tej točki in lastnosti vozila sploh dovoljujejo [15]:

$$\mathbf{v}_{mejna}(s) = \widehat{\mathbf{T}}\sqrt{\frac{|\mathbf{a}_{mejni}|}{|\kappa(s)|}}. \tag{3.2}$$

V tej enačbi je $\kappa(s)$ ukrivljenost v točki $P(s)$ in $\mathbf{a}_{mejni}$ največji možni pospešek, ki ga oprijem v tej točki omogoča. Da $\mathbf{v}_{mejna}$ kaže v smer gibanja, poskrbi normalizirana tangenta $\widehat{\mathbf{T}}$.

3.2.2 Pospeševanje in zaviranje

Le to, da ukrivljenost neko hitrost $\mathbf{v}_{mejna}(s)$ omogoča, še ne pomeni, da se bo vozilo v tej točki res lahko premikalo tako hitro. Vozilo zaradi omejenega pospeška mogoče te hitrosti v tej točki še ne more doseči. Prav tako morda ne sme voziti s hitrostjo $\mathbf{v}_{mejna}(s)$, če sledi del proge, kjer bo moralo voziti počasneje.

Zato mora za točko $P(s)$ ter bližnji točki $P(s-\Delta s)$ in $P(s+\Delta s)$ veljati naslednja zveza:

$$\mathbf{v}(s) = \widehat{\mathbf{T}} \min \begin{pmatrix} |\mathbf{v}_{mejna}(s)|, \\ \sqrt{|\mathbf{v}(s-\Delta s)^2 + 2\mathbf{a}_a \Delta s|}, \\ \sqrt{|\mathbf{v}(s+\Delta s)^2 + 2\mathbf{a}_b \Delta s|} \end{pmatrix}. \tag{3.3}$$

V tej enačbi $\mathbf{a}_a$ predstavlja največji možni povprečni pospešek avtomobila na poti od točke $P(s-\Delta s)$ do točke $P(s)$, $\mathbf{a}_b$ pa pomeni največji možni povprečni zaviralni pospešek, ko potuje od točke $P(s)$ do točke $P(s+\Delta s)$.

Poglavje 4

Genetski algoritmi

V naravi je evolucija zmožna prilagoditi vrsto na svoje okolje. Deluje zaradi naslednjih dejstev:

- Bolje prilagojeni osebki imajo večjo verjetnost preživetja.
- Njihove lastnosti so izražene v potomcih. Tako se lastnosti preživelih osebkov ohranjajo, slabše lastnosti pa počasi izgubijo.
- Naključne mutacije lahko osebku dajo nove lastnosti, ki jih ne bi mogel podedovati.

Genetski algoritmi posnemajo naravno evolucijo v namen optimizacije. Iz nabora rešitev boljše nadaljujejo v naslednjo iteracijo. Z njihovimi lastnostmi ter z naključnimi mutacijami se oblikujejo nove rešitve, ki se na enak način ovrednotijo v naslednji iteraciji. Cilj genetskega algoritma je na tak način producirati optimalno rešitev.

4.1 Kromosomi, geni, populacija

Tudi izrazoslovje pri tem področju se zgleduje po izrazih povezanih z evolucijo in genetiko.

Prvi tak izraz je kromosom. Ta predstavlja posamezno rešitev in je sestavljen iz genov, ki so tradicionalno biti [17], lahko pa so tudi cela ali realna števila, nizi znakov ali števil in podobno.

Skupek vseh kromosomov se imenuje populacija. Velikost populacije je nastavljiva in mora biti primerna aplikaciji. Če je populacija preštevilna, bodo po nepotrebnem zapravljeni računalniški viri; če pa je premajhna, lahko genetski algoritem konvergira prehitro in obtičimo z lokalno optimizacijo. [18]

Velikost populacije se med izvajanjem algoritma navadno ne spreminja, sama populacija pa se. Populaciji v določeni iteraciji pravimo generacija.

4.2 Delovanje

Prva generacija sestoji iz naključno generiranih kromosomov. Od njih je težko kateri dobra rešitev problema, kljub temu pa morajo nekateri biti boljše rešitve od drugih. Kako dober je posamezen kromosom oceni funkcija prilagojenosti (glej naslednji oddelek 4.3).

Boljši posamezniki so stohastično izbrani iz generacije. Isti kromosom je lahko izbran tudi večkrat. Vsak je spremenjen (glej oddelek 4.4) in dodan v naslednjo generacijo. Ta postopek se ponavlja, dokler nismo zadovoljni z rezultatom ali dokler ne obupamo.

4.3 Funkcija prilagojenosti

Algoritem mora biti sposoben primerjave kromosomov, da se lahko odloči, kateri bodo izbrani za v naslednjo iteracijo. Funkcija prilagojenosti (angleško fitness function) za posamezen kromosom izračuna kako dobro rešuje podan problem. Z njo se vsako iteracijo oceni celotno populacijo.

4.4 Genski operatorji

Kako se kromosomi spreminjajo, definirajo genski operatorji. Glavna tipa operatorjev sta mutacija in križanje.

Verjetnosti obeh sta nastavitvi, ki morata tako kot velikost populacije biti primerni problemu.

4.4.1 Mutacija

Mutacija skrbi za gensko raznolikost tekom algoritma in s tem preprečuje, da bi se algoritem ujel v lokalno optimizacijo, kjer so si kromosomi med seboj preveč podobni, da bi se lahko iz njih razvila boljša rešitev.

Pri mutaciji gre za naključno spremembo kromosoma, kaj točno se zgodi, pa je odvisno od implementacije. Lahko se na primer vsakemu genu dodeli nova naključna vrednost ali pa se, če so kromosomi nizi bitov, naključnemu izboru genov obrne vrednost.

Potrebna je pazljivost pri nastavitvi verjetnosti mutacije. Če je prenizka, lahko algoritem konvergira prehitro. Če je previsoka, bo algoritem podoben naključnemu iskanju in lahko se zgodi, da izgubimo dobre rešitve.

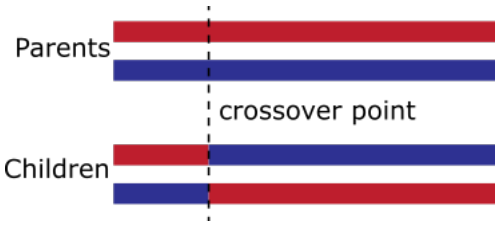
4.4.2 Križanje

Križanje omogoča, da se preživeli kromosomi kombinirajo in se tako ustvarijo novi, za katere upamo, da bodo podedovali dobre lastnosti staršev.

Pri križanju se iz dveh ali več kromosomov naredi eden ali več novih, ki imajo lahko gene obeh oziroma vseh staršev.

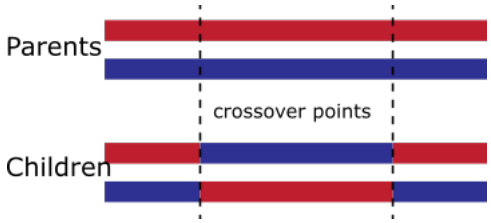
Najosnovnejši tehniki križanja sta križanje z eno točko in križanje z dvema točkama. Obe tehniki iz dveh staršev ustvarita dva otroka.

Pri prvi tehniki se naključno določi točka v nizu genov pri obeh starših. Eden od otrok podeduje gene prvega starša pred to točko in gene drugega starša za to točko, drugi pa obratno – gene drugega starša pred to točko in gene prvega starša za to točko.



Slika 4.1: Križanje z eno točko [19]

Tehnika z dvema točkama deluje zelo podobno. Pri tem načinu se izbere dve točki. Prvi otrok dobi gene prvega starša pred prvo točko in za drugo točko ter gene drugega starša med obema točkama. Drugi otrok pa podeduje preostale gene, torej tiste pred prvo in za drugo točko pri drugem staršu in tiste med obema točkama pri prvem staršu.



Slika 4.2: Križanje z dvema točkama [20]

4.5 Elitizem

Genski operatorji kromosome spremenijo v upanju, da se najde boljša rešitev. Večina tako spremenjenih kromosomov je navadno slabših. Lahko se celo zgodi, da so vsi slabši od najboljšega kromosoma prejšnje generacije. Da se zavarujemo pred tem, da bi izgubili trenutno najboljšo rešitev, uvedemo elitizem.

Pri elitizmu se najboljših nekaj kromosomov nespremenjenih prenese v naslednjo generacijo. Tako zagotovimo, da nobena iteracija ne more najboljše rešitve poslabšati.

4.6 JGAP

JGAP (Java Genetic Algorithms Package, slovensko Javanski paket za genetske algoritme) [21] je knjižnica za razvoj genetskih algoritmov. Je zelo nastavljiva, kljub temu pa dobro deluje tudi s privzeto konfiguracijo, ki ima naslednje lastnosti:

- Uporabljata se genska operatorja mutacija in križanje z eno točko, ki je določena naključno.
- Verjetnost mutacije posameznega gena je $\frac{1}{12}$.
- Verjetnost križanja je 0.35.
- Vsako križanje producira dva otroka.
- Iz prejšnje generacije je izbranih najboljših 90% kromosomov, preostalih 10% je zapolnjenih tako, da se najboljši izmed teh klonirajo, dokler ne dosežemo nastavljene velikosti populacije.

Knjižnica omogoča tudi elitizem, tako da se najboljši kromosom vsake generacije prenese v naslednjo v nespremenjeni obliki.

Poglavje 5

Implementacija optimizacije poti

5.1 Model steze

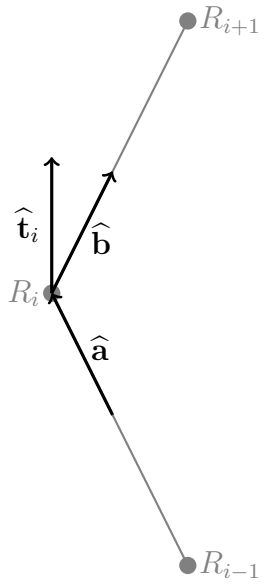
Steza je podana kot dvodimenzionalna sklenjena lomljenka z n oglišči, s to razliko, da lahko na isti premici leži tudi več kot dve zaporedni oglišči. Vsako oglišče R_i ima poleg koordinat $R_{i,x}$ in $R_{i,y}$ definirano tudi svojo širino w_i.

V vsakem oglišču steze R_i se izračuna približna tangenta $\widehat{\mathbf{t}}_i$ s pomočjo normaliziranih vektorjev $\widehat{\mathbf{a}}$ in $\widehat{\mathbf{b}}$. Prvi je usmerjen od oglišča R_{i-1} do R_i, drugi pa od R_i do R_{i+1}:

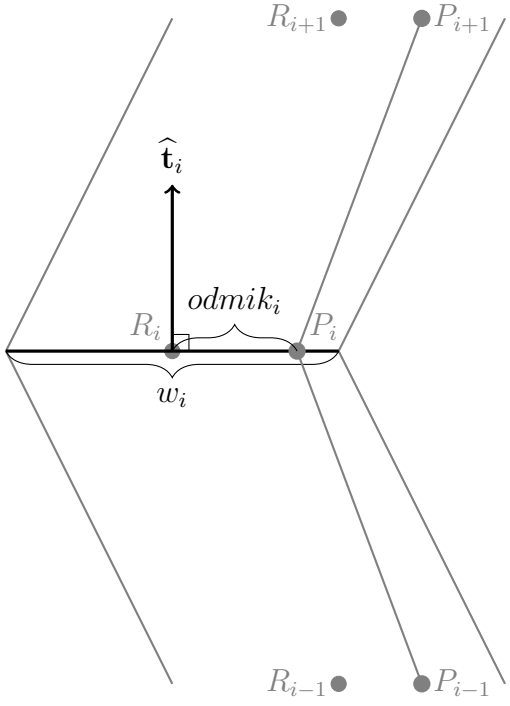
$$\widehat{\mathbf{t}}_i = \frac{\widehat{\mathbf{a}} + \widehat{\mathbf{b}}}{2}. \tag{5.1}$$

Program prav tako hrani trenutno najboljšo poznano pot p, ki je definirana kot nova poligonska veriga, katere oglišča $P_i \dots P_n$ ležijo na premicah, ki tečejo skozi oglišča steze $R_i \dots R_n$ in so pravokotne na tangente, izračunane v enačbi (5.1). Shranjena je kot niz odmikov $odmik_i \dots odmik_n$ od vsakega oglišča steze. Koordinati oglišča poti P_i sta:

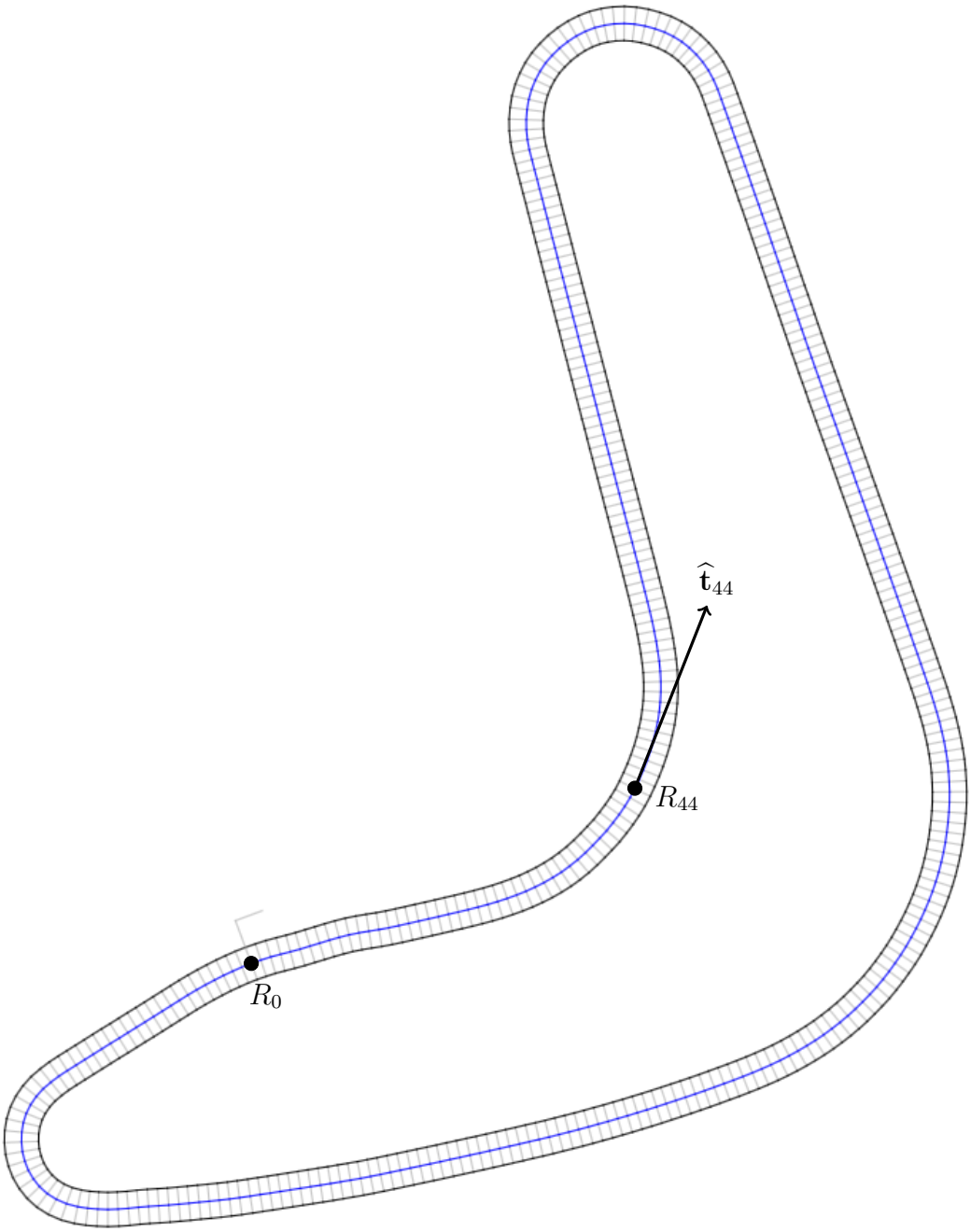
$$P_i \begin{pmatrix} R_{i,x} - odmik_i * \widehat{\mathbf{t}}_{i,y} \\ R_{i,y} + odmik_i * \widehat{\mathbf{t}}_{i,x} \end{pmatrix}. \tag{5.2}$$



Slika 5.1: Vektorji enačbe (5.1)



Slika 5.2: Lastnosti oglišča R_i



Slika 5.3: Primer dirkalne steze. Levi in desni rob sta črne barve. Skozi vsako vozlišče R_i je narisana prečna siva črta, ki je pravokotna na $\widehat{\mathbf{t}}_i$. Z modro pa je označena pot p, ki je pred optimizacijo enaka sredinski liniji proge.

Vsak odmik $odmik_i$ je omejen tako s širino vozila w_v kot tudi s širino steze w_i v točki P_i:

$$-\frac{w_i - w_v}{2} < odmik_i < \frac{w_i - w_v}{2}. \tag{5.3}$$

Za vsako točko P_i se hranita tudi skalarja v_i in $v_{mejna,i}$. Prvi je hitrost vozila v tej točki, drugi pa hitrost, ki jo ukrivljenost v točki omogoča.

5.2 Model vozila

Model vozila ima vse lastnosti, ki so predstavljene v oddelku 2.3. Predpostavljeno je, da imajo vozila brezstopenjski menjalnik. Čeprav je to pri resničnih avtomobilih redkost, ta predpostavka poenostavi odvisnost pospeška od hitrosti brez večjih napak. Pogonski pospešek vozila $\mathbf{a}_{pogon}(|\mathbf{v}|)$ je tako definiran z naslednjo enačbo:

$$\mathbf{a}_{pogon}(|\mathbf{v}|) = \widehat{\mathbf{T}}\frac{a_{ref}v_{ref}}{|\mathbf{v}|}, \tag{5.4}$$

kjer je a_{ref} referenčni pospešek pri hitrosti v_{ref}. Konstanta a_{ref} je lastnost vozila, v_{ref} pa je konstanta, ki je ista ne glede na vozilo. Obe konstanti sta skalarja. Vidi se, da pospešek $\mathbf{a}_{pogon}(|\mathbf{v}|)$ pada z naraščanjem hitrosti v.

Pri računanju zračnega upora se predpostavlja turbulenten tok zraka in z njim enačba (2.9).

5.3 Ocena časa

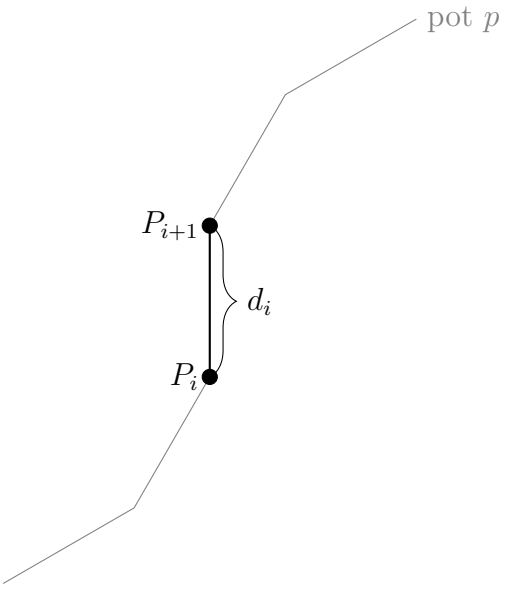
Program mora znati oceniti, koliko časa bi vozilo potrebovalo za neko pot (glej sekcijo 3.2).

Ker je pot razdeljena na odseke, je ocena časa t seštevek trajanja vseh odsekov:

$$t = \sum_{i=1}^{n} t_i. \tag{5.5}$$

Da dobimo trajanje odseka t_i, delimo dolžino odseka d_i s povprečjem hitrosti, ki ju vozilo doseže v krajnih točkah odseka (hitrosti v_i in v_{i+1}):

$$t_i = \frac{2d_i}{v_i + v_{i+1}}. \tag{5.6}$$

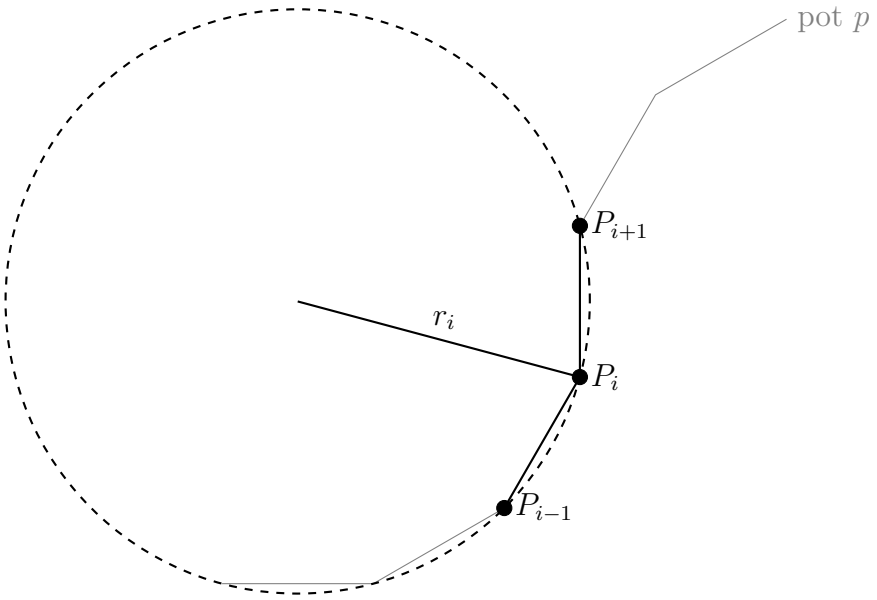


Slika 5.4: V točki P_i ima vozilo hitrost v_i, v točki P_{i+1} pa v_{i+1}.

Preden pa dobimo hitrost, ki jo ima vozilo v točki, moramo z enačbo (3.2) izračunati, kako hitro lahko vozimo skozi to točko, ne da bi prišlo do zdrsa koles – $v_{mejna,i}$, enačba (3.2). Za to potrebujemo ukrivljenost κ_i.

Ukrivljenost κ_i v točki P_i dobimo tako, da najprej izračunamo polmer r_i ocenjene pritisnjene krožnice v točki P_i. To je krožnica, ki gre skozi točko P_i in sosednji dve točki P_{i-1} in P_{i+1}. Ukrivljenost je potem:

$$\kappa_i = \frac{1}{r_i}. \tag{5.7}$$



Slika 5.5: Ocena pritisnjene krožnice v točki P_i

Hitrost v točki v_i ne sme biti višja od $v_{mejna,i}$. To pa je le ena od omejitev. Treba je tudi poskrbeti, da ni višja, kot dovoljujejo pospeševalne in zaviralne sposobnosti vozila – glej enačbo (3.3). Hitrosti v_i za vsako točko izračuna naslednji algoritem:

$v_1 \leftarrow 0$
for $i \leftarrow 2 \dots n$ **do**
  if $v_{mejna,i} \geq v_{i-1}$ **then** ▷ Moramo pospešiti
    $v_i \leftarrow \min \left(\begin{array}{l} v_{mejna,i}, \\ \sqrt{v_{i-1}^2 + 2\,|\mathbf{a}_a|\,d_{i-1}} \end{array} \right)$
  else ▷ Moramo upočasniti
    $v_i \leftarrow v_{mejna,i}$
    ZMANJŠAJ HITROST ČE JE TO POTREBNO$(i-1)$
  end if
end for

Zgoraj napisani algoritem predpostavlja, da se vozilo v 1. točki še ne giblje. To pa drži le v prvem krogu, zato v resnici algoritem ne začne s 1.

točko, ampak malo pred njo.

V primeru, da ne moremo upočasniti do hitrosti $v_{mejna,i}$, ker smo v točki P_{i-1} imeli previsoko hitrost, moramo predhodno hitrost v_{i-1} znižati. Za to skrbi naslednja funkcija:

function ZMANJŠAJ HITROST ČE JE TO POTREBNO(i)
  $v_b \leftarrow \sqrt{v_{i+1}^2 + 2\,|\mathbf{a}_b|\,d_i}$ ▷ Hitrost pred zaviranjem
  if $v_b < v_i$ **then**
    $v_i \leftarrow v_b$
    ZMANJŠAJ HITROST ČE JE TO POTREBNO($i-1$)
  end if
end function

5.4 Optimizacija

Cilj optimizacije je najti pot po dirkališču, za katero je ocena časa čim nižja. Uporabljena sta dva različna pristopa.

5.4.1 Linearni algoritem

Ta algoritem poskusi izboljšati trenutno najboljšo poznano pot p. Začnemo lahko s potjo po sredinski liniji proge.

$sprememba \leftarrow true$
while $sprememba$ **do** ▷ Dokler se pot p izboljšuje
  $sprememba \leftarrow false$
  for $i \leftarrow 1 \ldots n$ **do**
    $t_{start} \leftarrow$ OCENI ČAS($i, odmik_i$)
    $t_- \leftarrow$ OCENI ČAS($i, odmik_i - korak$)
    $t_+ \leftarrow$ OCENI ČAS($i, odmik_i + korak$)
    if $\min(t_-, t_+) < t_{start}$ **then** ▷ Našli smo pot, ki je boljša od p
      if $t_- < t_+$ **then**
        $odmik_i \leftarrow odmik_i - korak$ ▷ Popravi pot p

```
            else
                odmik_i ← odmik_i + korak                ▷ Popravi pot p
            end if
            sprememba ← true
        end if
    end for
end while
```

Funkcija OCENI ČAS$(i, odmik_{podan})$ vrne oceno časa, ki bi ga vozilo potrebovalo za en krog po trenutni najboljši poznani poti p, če bi njena točka P_i bila od točke na stezi R_i oddaljena za $odmik_{podan}$ namesto za $odmik_i$. Druge odmike upošteva take, kot so.

Če je $odmik_{podan}$ prevelik ali premajhen, ker postavlja točko poti P_i izven steze, se namesto njega upošteva največji oz. najmanjši dovoljen odmik:

```
if odmik_podan < −(w_i − w_v)/2 then
    odmik_podan ← −(w_i − w_v)/2
else if odmik_podan > (w_i − w_v)/2 then
    odmik_podan ← (w_i − w_v)/2
end if
```

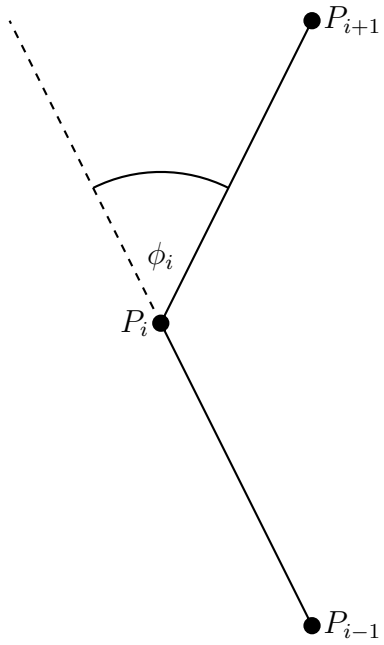
Minimizacija ukrivljenosti

Algoritem lahko uporabimo tudi za optimizacijo drugih vrednosti, na primer dolžine ali ukrivljenosti. Za oceno slednje lahko uporabljamo vrednost:

$$\sum_{i=1}^{n} \kappa_i^2. \tag{5.8}$$

Lahko pa namesto tega uporabimo zunanje kote med odseki (glej sliko 5.6):

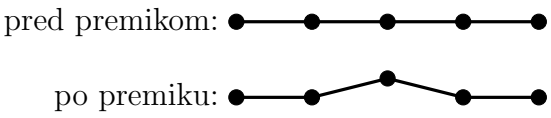
$$\sum_{i=1}^{n} \phi_i^2. \tag{5.9}$$



Slika 5.6: Zunanji kot med odsekoma

Resolucija

Ta algoritem deluje slabo na stezah z velikim številom oglišč, ker ima tedaj poskus premika posameznega odmika veliko verjetnost, da pot p poslabša (glej sliko 5.7). To lahko rešimo tako, da najprej optimiziramo poenostavljeno različico steze. Z rešitvijo poenostavljene steze si potem pomagamo pri optimizaciji prvotne steze.



Slika 5.7: Primer problema pri velikem številu oglišč: če je del poti p ravna črta, potem bo vsak posamezen premik v tem delu zvečal ukrivljenost in dolžino (ter s tem čas), tudi če je tak premik korak v pravo smer.

Stezo poenostavimo tako, da ji odstranimo vsako drugo oglišče. Ko poenostavljeno stezo optimiziramo, ji nazaj dodamo prej odstranjena oglišča in v njih odmik nastavimo na povprečje odmikov sosednjih točk:

$$odmik_i \leftarrow \frac{odmik_{i-1} + odmik_{i+1}}{2}. \tag{5.10}$$

Če ima tudi poenostavljena steza preveč oglišč, jo prav tako lahko poenostavimo po enakem postopku.

5.4.2 Genetski algoritem

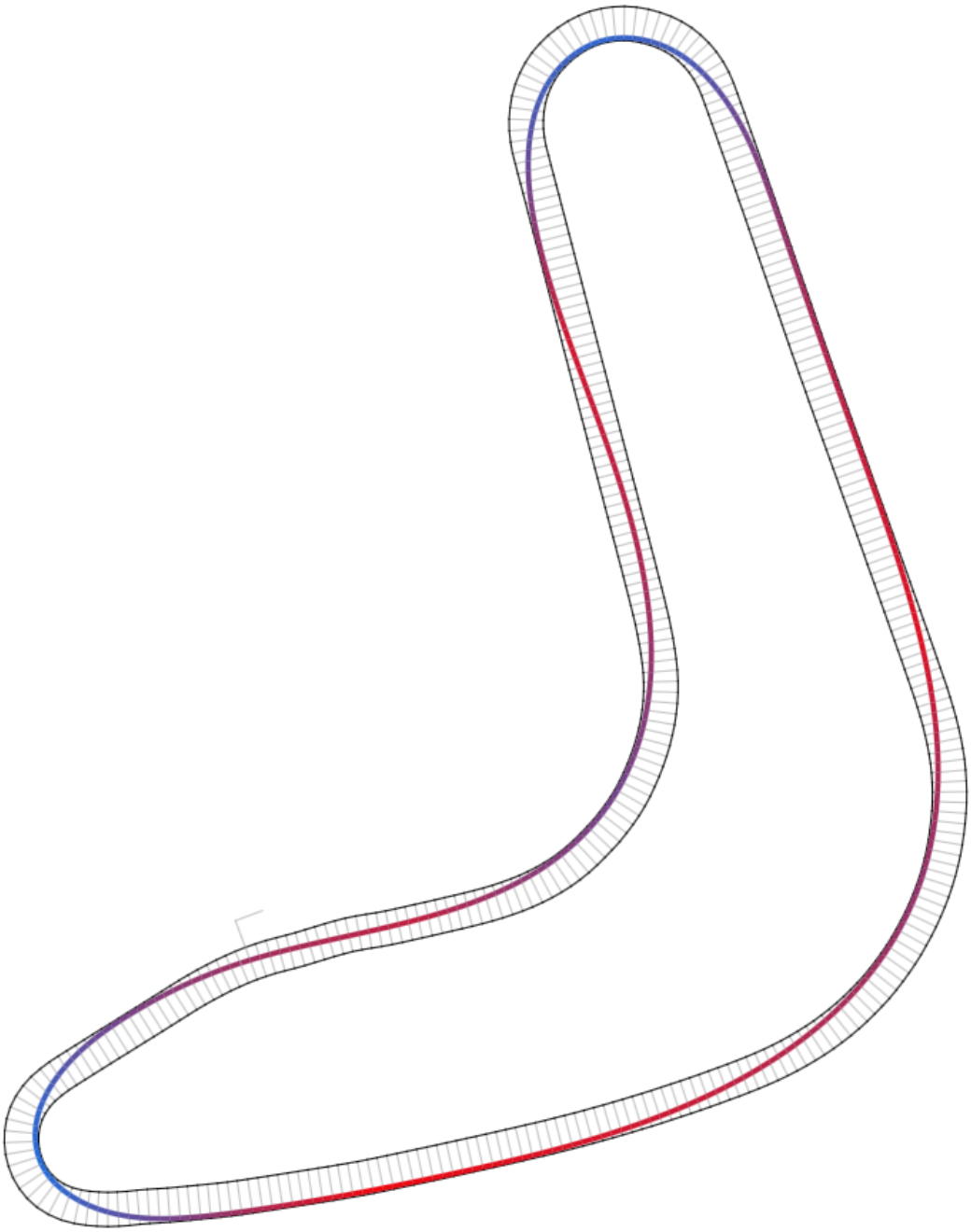
Genetski algoritem lahko začne z naključnimi kromosomi, kot je pri genetskih algoritmih običajno, lahko pa tako kot linearni algoritem poskusi izboljšati trenutno najboljšo poznano pot p. Implementacija uporablja knjižnico JGAP.

Uporabljene so privzete nastavitve in vključen je elitizem. Treba pa je problemu prilagoditi kromosom.

Da bo kromosom predstavljal rešitev problema, ga moramo definirati tako, da bo hranil neko pot skozi dirkališče. Kromosom naj ima toliko genov, kolikor ima naša steza oglišč, vsak gen pa je realno število, ki predstavlja odmik od središča steze v neki točki $odmik_i$ (glej enačbo (5.1)) in je omejen s širino steze v točki w_i in širino vozila w_v (glej enačbo (5.3)).

Če želimo začeti z neko že poznano rešitvijo p, na primer s potjo, ki jo je sproduciral linearni algoritem, to storimo tako, da začetno pot p dodamo prvi generaciji kot enega od kromosomov. Drugi kromosomi v prvi generaciji pa so še vedno naključno generirani.

Za funkcijo prilagojenosti uporabimo funkcijo, ki oceni čas t potreben za pot, ki jo kromosom predstavlja in je opisana v sekciji 5.3. Lahko pa namesto tega uporabimo druge lastnosti, na primer dolžino poti ali njeno ukrivljenost.



Slika 5.8: Minimizacija kotov (glej enačbo (5.9)) z linearnim algoritmom za vozilo široko 1,852 m

Poglavje 6

Simulacija

Za implementacijo vožnje po v prejšnjem poglavju izračunani poti potrebujemo simulacijo fizike. Izbral sem igralni pogon (angleško game engine) Unreal Engine 4, ki uporablja realnočasovni fizikalni pogon PhysX.

Unreal Engine [22] je razvilo podjetje Epic Games. Napisan je v programskem jeziku C++, za razvoj projektov pa poleg njega ponuja tudi skriptni jezik Blueprints.

Z njim razvit program teče v realnem času, lahko pa se določi, kako hitro.

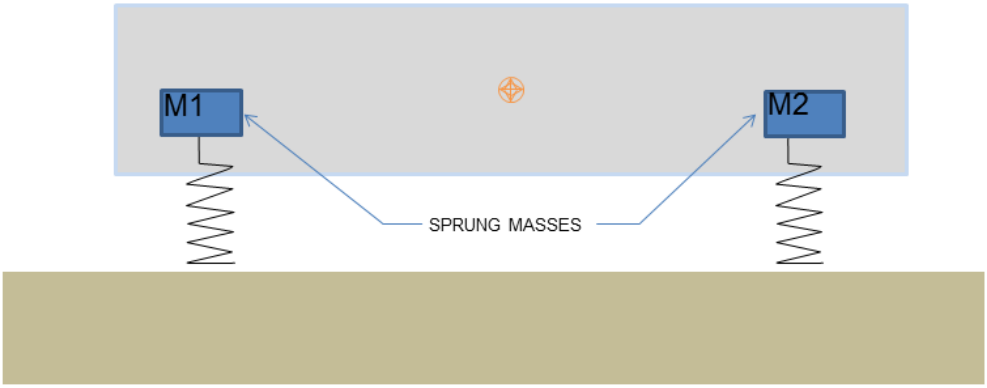
Igralni pogon za simulacijo fizike uporablja PhysX 3.3 [23] [24]. Gre za Nvidiin realnočasovni fizikalni pogon, ki podpira simulacijo trdnih in mehkih teles, delcev, lutk iz cunj, volumetričnih tekočin, tekstila in vozil.

6.1 Model vozila

Unreal Engine vsebuje razred za predstavitev vozil WheeledVehicle, ki razširja razred Pawn, kar pomeni, da predstavlja prostorski objekt, ki se ga da kontrolirati. Njegova komponenta WheeledVehicleMovementComponent (oziroma razred WheeledVehicleMovementComponent4W, ki jo razširja) skrbi za povezavo s PhysX modelom vozila, ki obsega motor, sklopko, menjalnik, diferencial, kolesa, pnevmatike, vzmetenje in šasijo. [25]

6.1.1 Vzmetenje

V PhysX ima vsako vozilo niz vzmetenih mas. Vsaka od njih je vzmetena s svojim amortizatorjem.



Slika 6.1: Predstavitev vozila kot niz dveh vzmetenih mas m_1 in m_2 [25]

Hkrati je vozilo predstavljeno kot trdno telo, katerega masa, masno središče in vztrajnostni moment so popolnoma skladni z masami in koordinatami vzmetenih mas. To trdno telo je končni produkt PhysX simulacije vozila in je v interakciji z drugimi telesi v sceni.

Z modelom vzmetenih mas se izračunajo sile vzmetenja in pnevmatik, ki se nato v obliki spremembe hitrosti in kotne hitrosti aplicirajo na trdno telo.

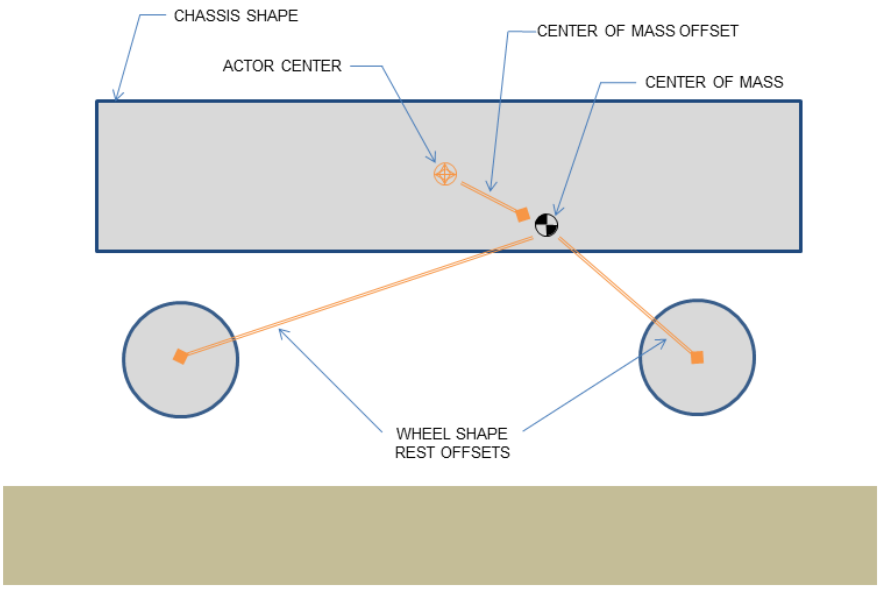
Masa trdnega telesa M v drugi predstavitvi je enaka seštevku vseh vzmetenih mas m_i prve predstavitve:

$$M = \sum_{i=1}^{n} m_i. \tag{6.1}$$

Masno središče trdnega telesa pa opisuje enačba:

$$\mathbf{R} = \frac{1}{M} \sum_{i=1}^{n} m_i \mathbf{r}_i, \tag{6.2}$$

kjer je $\mathbf{R}$ krajevni vektor masnega središča trdnega telesa, $\mathbf{r}_i$ pa masno središče posamezne vzmetene mase.

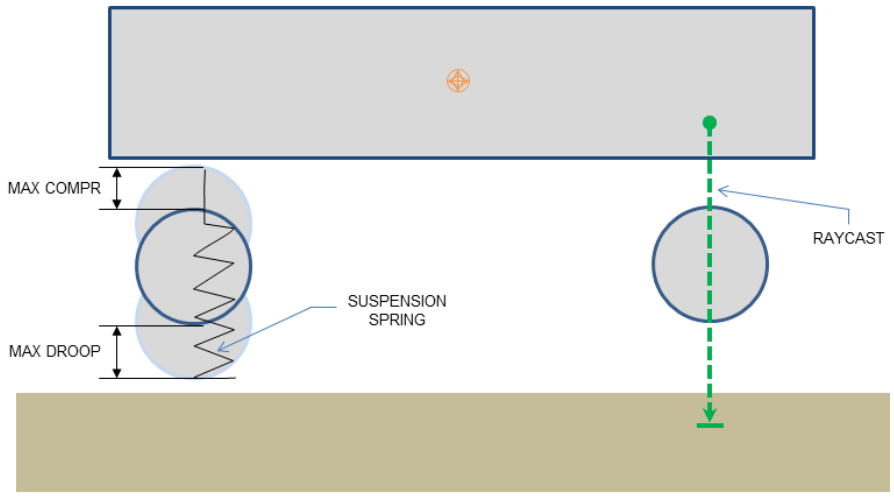


Slika 6.2: Predstavitev vozila kot trdno telo [25]

6.1.2 Metanje žarkov

Da se ugotovi, kako daleč so tla in s tem, kje ležijo kolesa, se proti tlom meče žarek. Za vsako vzmetenje je potreben svoj žarek, ki začne pot v točki malo nad vrhom pnevmatike pri maksimalni kompresiji vzmetenja. Potuje v smeri vzmetenja do točke malo pod dnom pnevmatike pri najizrazitejšem visenju (kolesa so najbolj oddaljena od šasije).

Sile vzmetenja vsakega raztegnjenega ali stisnjenega amortizerja so izračunane in dodane k skupni sili, ki bo vplivala na trdno telo. Poleg tega se sila vzmetenja uporabi pri izračunu obremenitve na pnevmatiko, ki skupaj z drugimi faktorji, kot so kot zavijanja, kot izbočenosti kolesa, trenje, hitrost vrtenja kolesa in gibalna količina trdnega telesa, vpliva na izračun sil pnevmatike, ki so prav tako aplicirane na trdno telo.



Slika 6.3: Meje vzmetenja in metanje žarka [25]

6.1.3 Pogon

PhysX podpira več modelov pogona. Unreal Engine uporablja PxVehicleDrive4W [26], ki omogoča do 4 pogonska kolesa. Porazdelitev navora med kolesi (angleško torque split ratio) je nastavljiva, kar je uporabno, če želimo simulirati avtomobil s štirikolesnim pogonom, saj imajo ti različne porazdelitve navora med prednjimi in zadnjimi kolesi.

V sredini modela je torzijska sklopka. Na eni strani sklopke je motor, ki ga neposredno upravlja vhod, ki predstavlja stopalko za plin, na drugi strani pa so menjalnik, diferencial in kolesa. Model sklopke omogoča ne le, da se navor motorja prenese do koles, temveč tudi, da kolesa vplivajo nazaj na motor.

6.1.4 Upravljanje

Razred WheeledVehicleMovementComponent definira naslednje funkcije za upravljanje vozila [27]:

- SETTHROTTLEINPUT(*Throttle*) igra vlogo stopalk za plin in zavore.

Argument *Throttle* je omejen z intervalom $[-1, 1]$. Če je nižji od 0, se sprosti stopalka za plin, stisnejo pa se zavore. Če je višji od 0, se sprostijo zavore, pritisne pa se na plin. Če je enak 0, se sprostita obe stopalki. Večja je absolutna vrednost argumenta, močneje se na stopalko pritisne. Če je avtomobil v vzvratni prestavi, se upošteva negativna vrednost ($-Throttle$). V tem primeru torej negativne vrednosti kontrolirajo plin, pozitivne pa zavore.

- SetSteeringInput(*Steering*) kontrolira zavijanje. Njen argument je prav tako omejen z intervalom $[-1, 1]$. Tukaj negativna vrednost pomeni zavijanje na levo, pozitivna pa na desno.

- setHandbrakeInput($bNewHandbrake$) aktivira ročno zavoro, če je podan Boolov argument $bNewHandbrake$ resničen. V nasprotnem primeru pa se ročna zavora sprosti.

Razred vsebuje tudi funkcije za upravljanje menjalnika, ki so potrebne, če izključimo avtomatično menjavanje prestav.

Poglavje 7

Implementacija vožnje po poti

Zdaj, ko poznamo optimalno pot, potrebujemo še program, ki bo znal vozilo voditi po njej.

V simulaciji je pot predstavljena z razredom SplineComponent, ki je del Unreal Engine. Vsebuje niz vozlišč $P_1 \dots P_n$ in predstavlja zlepek, ki teče skozi njih.

Naj bo $s(P_i)$ dolžina loka od začetka do točke P_i. $P(s)$ pa pomeni obratno – funkcijo, ki vrne točko na zlepku pri dolžini loka s. Kjer je točka napisana krepko, gre za krajevni vektor do te točke – $\mathbf{P}(s)$ je, na primer, krajevni vektor do točke $P(s)$. Za vsako vozlišče P_i je izračunana spremenljivka $lengthScale_i$, ki povezuje dolžino loka med točkama P_i in P_{i+1} z evklidsko razdaljo med njima:

$$lengthScale_i = \frac{s(P_{i+1}) - s(P_i)}{\sqrt{(P_{i+1,x} - P_{i,x})^2 + (P_{i+1,y} - P_{i,y})^2}}. \tag{7.1}$$

Za vozilo pa je definiran razred, ki razširja WheeledVehicle in uporablja samodejno menjavanje prestav.

7.1 Sledenje napredku vozila na poti

Najprej potrebujemo način za določevanje dela poti, na katerem se trenutno nahaja vozilo, ki mora za to hraniti naslednje podatke:

- kateri krog trenutno opravlja – $krog$,
- indeks zadnjega vozlišča, ki ga je prevozilo v tem krogu – u,
- napredek na poti izražen s prevoženo dolžino loka od začetka kroga – $s_{napredek}$.

Naj bo Q dejanska pozicija vozila. V funkciji TICK(), ki se izvede vsako iteracijo simulacije, se zgornji podatki posodobijo z naslednjim algoritmom:

$\widehat{\mathbf{a}} \leftarrow \frac{\mathbf{P}_{u+1}-\mathbf{P}_u}{|\mathbf{P}_{u+1}-\mathbf{P}_u|}$ ▷ Normaliziran vektor od P_u do P_{u+1}
$\mathbf{b} \leftarrow \mathbf{Q} - \mathbf{P}_u$ ▷ Vektor od P_u do Q
$d \leftarrow (\mathbf{b} \cdot \widehat{\mathbf{a}}) lengthScale_u$ ▷ Prevožena pot od zadnjega vozlišča
$s_{napredek} \leftarrow s(P_u) + d$
if $s_{napredek} > s(P_{u+1})$ **then** ▷ Prevoženo je bilo ogliščе.
    $u \leftarrow u + 1$
end if
if $u > n$ **then** ▷ Prevožen je bil cel krog.
    $krog \leftarrow krog + 1$
    $u \leftarrow 0$
    $v \leftarrow d$
end if

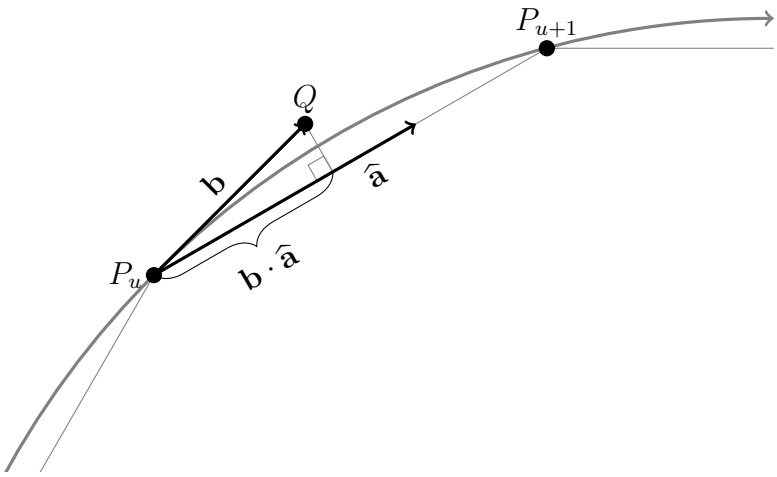
7.2 Upravljanje vozila

Upravljanje vozila je prav tako implementirano v funkciji TICK(). Vozilo mora slediti prej izračunani poti in upoštevati hitrost.

7.2.1 Pospeševanje in zaviranje

Naj bo $\mathbf{v}_{trenutna}$ trenutna hitrost vozila, $\mathbf{v}_{ciljna}$ pa hitrost, ki jo mora vozilo imeti v naslednjem vozlišču P_{u+1}. S konstanto k_{plin} nastavimo občutljivost na razliko med obema hitrostma. Stopalki za plin in zavore nastavimo takole:

$ThrottleInput \leftarrow (|\mathbf{v}_{ciljna}| - |\mathbf{v}_{trenutna}|)k_{plin}$



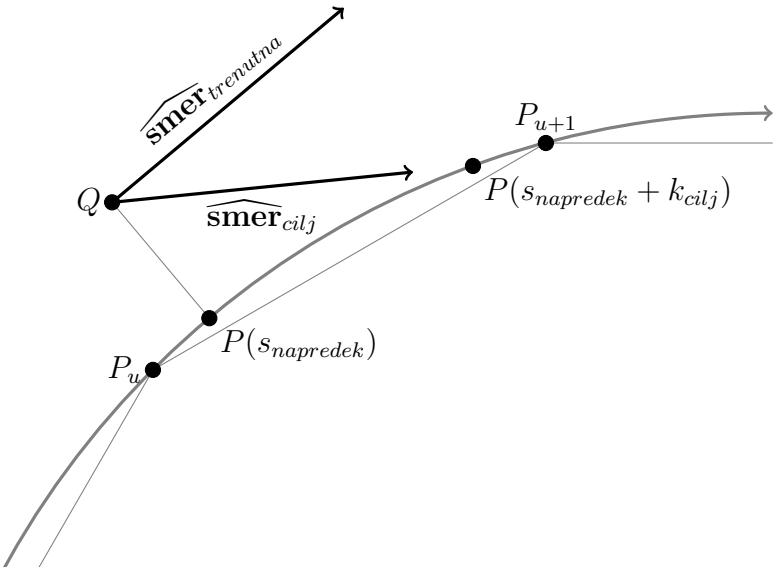
Slika 7.1: Ilustracija sledenja napredka

if $ThrottleInput < -1$ **then**
  $ThrottleInput \leftarrow -1$
else if $ThrottleInput \geq 1$ **then**
  $ThrottleInput \leftarrow 1$
end if
SetThrottleInput(sign $(ThrottleInput)\sqrt{|ThrottleInput|}$)

7.2.2 Zavijanje

Pri zavijanju skrbimo, da je vozilo obrnjeno proti neki točki na poti. Izberemo točko, ki je pred vozilom, saj lahko vozilo v nasprotnem primeru začne vijugati zaradi prevelikih popravkov. Koliko je ta točka pred doseženo dolžino loka $s_{napredek}$, nastavimo s konstanto k_{cilj}.

$\widehat{\mathbf{smer}}_{trenutna} \leftarrow \frac{\mathbf{v}_{trenutna}}{|\mathbf{v}_{trenutna}|}$
$\mathbf{smer}_{cilj} \leftarrow \mathbf{P}(s_{napredek} + k_{cilj}) - \mathbf{Q}$
$\widehat{\mathbf{smer}}_{cilj} \leftarrow \frac{\mathbf{smer}_{cilj}}{|\mathbf{smer}_{cilj}|}$
$SteeringInput \leftarrow (\widehat{\mathbf{smer}}_{trenutna} \times \widehat{\mathbf{smer}}_{cilj})_z$ ▷ Potrebujemo le komponento z skalarnega produkta.
SetSteeringInput(sign $(SteeringInput)\sqrt{|SteeringInput|}$)



Slika 7.2: Ilustracija zavijanja

Poglavje 8

Zaključek in možnosti izboljšave

Po dani stezi smo za določeno vozilo poiskali čim boljšo pot, tako da se upoštevajo njegove fizikalne lastnosti. To smo naredili z uporabo dveh algoritmov – preprostega linearnega in genetskega. Vsak od njiju lahko do rešitve pride sam, lahko pa eden nadaljuje delo, ki ga je začel drugi.

Model steze je dvorazsežen, kar omogoča predstavitev velike večine prog, ki nimajo nagnjenih delov, z dovolj veliko natančnostjo. Kljub temu pa bi trirazsežen model omogočal še večjo natančnost in predstavitev stez z nagnjenimi deli.

Predpostavljen je konstanten koeficient trenja. Namesto tega bi lahko implementacija dovolila različen oprijem na različnih delih proge, kar bi bilo še posebno koristno pri reliju, kjer ena tekma lahko obsega različne površine, ponavadi sta prisotna vsaj makadam in asfalt.

Nazadnje je implementiran program, ki po najdeni poti vodi vozilo znotraj simulacije v igralnem pogonu Unreal Engine 4.

Vodenje se lahko uporabi v dirkalni igri ali simulaciji, vendar bi za igro, kjer hkrati tekmuje več vozil, bila potrebna še detekcija in izogibanje drugim tekmovalcem. Implementacija namreč predpostavlja, da je vodeno vozilo na cesti samo oziroma, da ga nihče ne ovira.

Literatura

[1] M. E. Tipping, M. A. Hatton, and R. Herbrich. Racing line optimization, February 22 2011. US Patent 7892078.

[2] M. E. Tipping, M. A. Hatton, and R. Herbrich. Racing line optimization, March 12 2013. US Patent 8393944.

[3] Computer-Controlled Cars in Vamos. `http://vamos.sourceforge.net/computer-controlled-cars/computer-controlled-cars.html`. Dostopano: 2016-07-07.

[4] Vamos. `http://vamos.sourceforge.net/`. Dostopano: 2016-07-07.

[5] Circuito de Cartagena. `http://www.raceoptimal.com/Cartagena`. Dostopano: 2016-08-16.

[6] Race Optimal. `http://www.raceoptimal.com/`. Dostopano: 2016-07-06.

[7] B. P. Demidović. *Zadaci i riješeni primjeri iz više matematike*. Tehnička knjiga, 1968.

[8] Ilustracija Frenetovega ogrodja. `https://upload.wikimedia.org/wikipedia/commons/6/6f/Frenet.png`. Dostopano: 2016-07-28.

[9] Ilustracija pritisnjene krožnice. `https://upload.wikimedia.org/wikipedia/commons/8/84/Osculating_circle.svg`. Dostopano: 2016-07-28.

[10] Quantities and units – Part 3: Space and time. Standard ISO 80000-3:2006, International Organization for Standardization, Geneva, CH, 2006.

[11] Standard Atmosphere. Standard ISO 2533:1975, International Organization for Standardization, Geneva, CH, 1975.

[12] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge university press, 2000.

[13] R. Zwanzig. Hydrodynamic Fluctuations and Stokes' Law Friction. *J. Res. Natl. Bur. Std.(US) B*, 68:143–145, 1964.

[14] J. D. Anderson. *Introduction to Flight*. McGraw-Hill series in aeronautical and aerospace engineering. McGraw-Hill Higher Education, 2005.

[15] F. Braghin, F. Cheli, S. Melzi, and E. Sabbioni. Race driver model. *Computers & Structures*, 86(13):1503–1516, 2008.

[16] B. K. P. Horn. The Curve of Least Energy. *ACM Transactions on Mathematical Software (TOMS)*, 9(4):441–460, 1983.

[17] D. Whitley. A Genetic Algorithm Tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[18] GAVaPS – a Genetic Algorithm with Varying Population Size, author=Arabas, J. and Michalewicz, Z. and Mulawka, J., booktitle=Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on, pages=73–78, year=1994, organization=IEEE.

[19] Križanje z eno točko. `https://upload.wikimedia.org/wikipedia/commons/5/56/OnePointCrossover.svg`. Dostopano: 2016-07-27.

[20] Križanje z dvema točkama. `https://upload.wikimedia.org/wikipedia/commons/c/cd/TwoPointCrossover.svg`. Dostopano: 2016-07-27.

[21] JGAP. https://sourceforge.net/projects/jgap/. Dostopano: 2016-08-03.

[22] What is unreal Engine 4. https://www.unrealengine.com/what-is-unreal-engine-4. Dostopano: 2016-07-24.

[23] Unreal Engine – physics simulation. https://docs.unrealengine.com/latest/INT/Engine/Physics/. Dostopano: 2016-07-24.

[24] PhysX. http://www.geforce.com/hardware/technology/physx. Dostopano: 2016-07-24.

[25] PhysX Vehicles. http://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Vehicles.html. Dostopano: 2016-07-24.

[26] Unreal Engine 4 wheeledvehiclemovementcomponent4w.cpp. https://github.com/EpicGames/UnrealEngine/tree/release/Engine/Source/Runtime/Engine/Classes/Vehicles. Dostopano: 2016-08-24.

[27] Unreal Engine 4 wheeledvehiclemovementcomponent. https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/Vehicles/UWheeledVehicleMovementComponent/index.html. Dostopano: 2016-08-24.