# Plánování závodní trajektorie na dráze s překážkami

Michaela Cihlářová
Čtvrtek 9:15
*Kybernetika a robotika*
*cihlami1@fel.cvut.cz*

## I. Zadání

### A. Popis problému

Závod na čas je jedna z mnoha výzev autonomního řízení. Jelikož závodní auta, díky dostupnosti současných technologií, mají velmi podobné vlastnosti, jako maximální rychlost, maximální zrychlení apod., vítěze určuje především sledovaná trajektorie. Ta neobsahuje pouze posloupnost bodů na dráze, ale také údaje o okamžité rychlosti, křivosti a okamžitém zrychlení. Tato, už tak náročná úloha, se ještě více komplikuje, přidáme-li do dráhy překážky.

Cílem této práce je upravit již existující algoritmus [1], který byl vytvořen týmem vědců z Českého institutu informatiky, robotiky a kybernetiky a zabývá se právě hledáním optimální trajektorie, tak aby byl schopen najít optimální trajektorii i okolo statických překážek rozmístěných náhodně po dráze. Jedná se o genetický algoritmus (GA) [2], tedy o heuristický postup, který je inspirován evolučním vývojem. Tyto algoritmy typicky začínají vytvořením inicializační populace, kde je každý jedinec z této množiny ohodnocen tzv. fitness funkcí, která určuje jeho kvalitu. Následně je na základě výsledků z ohodnocení vybrána skupina, ze které jsou, pomocí např. mutací nebo křížení, vytvořeni potomci a ti vytvoří novou populaci, která je opět ohodnocena atd.. Algoritmus končí při nalezení kvalitního řešení, nebo dokud není vyčerpán optimalizační rozpočet.

Podrobné informace o použitém algoritmu lze nalézt v [1], zde bude uveden pouze zkrácený popis. Jako první se dráha rozdělí na $n$ nepřekrývajících se částí, neboli segmentů $S_i$, kde $i \in \{1, 2, ..., n\}$, $S_i \cap S_j = \emptyset$ pro $i \neq j$. Tuto hodnotu je nutné volit manuálně a z výsledků práce [1] je vyplývá, že kvalita výsledné trajektorie závisí na počtu segmentů a s jejich zvyšujícím se počtem se rychlost kola značně zhoršuje.

Aby bylo možné využít GA je nutné závodní dráhu správně zakódovat. Možné řešení GA tvoří série bodů $P_i$, kde $P_i \in S_i$, $i \in \{1, 2, ..., n\}$ a tzv. fitness hodnotu $f$ lze získat jako $f = F(I(P), A)$, kde $I(P)$ reprezentuje proložení bodů $P$ kubickou křivkou, $A$ jsou parametry vozidla a $F(I(P), A)$ je funkce jejíž výstupem je čas potřebný k objetí jednoho kola po dané trajektorii $I(P)$. Tím je vytvořen následující optimalizační problém

$$\begin{gathered}\min_{P_i \in \mathbb{R}} F(I(P_1, ...P_n), A), \\ \text{za podmínky:} P_i \in S_i, \ \forall i \in \{1, ..., n\}.\end{gathered} \tag{1}$$

Aby bylo možné i podmínku vyjádřit pro GA, je využito zobrazení $H_i : [0, 1]^2 \to S_i, i \in \{1, ..., n\}$. Díky tomu lze předchozí problém popsat jako

$$\min_{P'_i \in [0,1]^2} F(I(H_1(P'_1), ...H_n(P'_n)), A). \tag{2}$$

Problémy (1) a (2) jsou ekvivalentní v případě, že zobrazení $H$ je homeomorfismus [3], tedy zobrazení je spojité, bijektivní a inverze zobrazení je také spojitá. Toto zobrazení je velmi náročné získat a proto je v [1] představo tzv. Matryoshka zobrazení, které je vhodnou aproximací zobrazení $H$.

V následujících sekcích práce bude na výše popsaný algoritmus odkazováno jako na knihovnu *ng_trajectory*.

## II. Related works

Plánování trajektorie pomocí GA se v dnešní době stává populární jako alternativa k hledání nejkratší cesty, nebo cesty s nejmenším zakřivením, jelikož fitness funkce dovoluje dobře definovat vlastnosti hledané trajektorie. Právě možnost hledání optima mezi nejkratší trajektorií a nejmenším zakřivením je hlavním předmětem GA využitých ve světě závodního autonomního řízení. Jako první se tomuto přístupu věnoval Barghin et al. [4].V této práci je také dráha rozdělena na segmenty, ovšem body, které tvoří výslednou trajektorii, se nachází na hranicích těchto segmentů. Segmenty tedy nejsou myšleny jako 2D boxy jako v [1], ale jako přímky.

Podobnou fitness funkcí jako [4] se zabývá i práce [6]. Segmenty jsou brány jako 2D boxy stejně jako v [1], ovšem nejsou ekvidistanční, ale jejich umístění závisí na tvaru trajektorie, přesněji jejím zakřivení. V těchto jednotlivých segmentech se hledají lokální optimální trajektorie, které jsou následně propojeny.

Offline plánování kolem statických překážek není však hojně probíráno, jelikož typické závody v zásadě probíhají na dráze bez překážek a při hrozící kolizi s nečekaným objektem se často využívá rychlé lokální plánování, např. RRT či PRM. Právě Receveur et al. [7] se zabývá propojením globálního plánování za využití GA s tzv. potential fields, které umožňují rychlé přeplánování v případě potřeby.

## III. Řešení problému

### A. Design

K řešení problému by se dalo přistupovat různými způsoby. Bylo by možné například pozměnit tvoření segmentů z ekvidistančního tak, aby každá překážka byla součástí pouze jednoho segmentu, který sahá od jedné stany dráhy k druhé. V tomto případě by bylo jednoduché daný bod umístit tak, aby trajektorie překážku co nejlépe objela. Tato práce se ovšem zaměřuje na jiný přístup, a tím je přeskakování vybraných segmentů. Tedy výsledná trajektorie není tvořena všemi segmenty, ale pouze určitou množinou. Do optimalizátoru je přidána nová binární proměnná, která nám určí, zda se má bod z daného segmentu využít při vytváření trajektorie, či nikoli.

Tento postup byl zvolen především, jelikož přidání proměnné je méně výpočetně náročné, než tvoření segmentů v závislosti na specifických vlastnostech dráhy, tedy úprava by neměla příliš ovlivnit dobu optimalizace. Navíc je možné tímto přístupem řešit další problém, který je uveden v původní zprávě [1], a tedy, že při zvětšujícím se počtu segmentů se zhoršuje i navržená trajektorie. Při přeskočení tzv. nepotřebných segmentů by mohl být tento problém vyřešen a nebylo by tedy nutné přesně určit jejich celkový počet.

### B. Implementace

Knihovna *ng_trajectory*, která slouží jako podklad, je implementovaná v programovacím jazyce Python3, a tedy i provedená práce je v tomto jazyce. GA byl vytvořen pomocí knihovny *Nevergrad* [8], přesněji funkcí *DoubleFastGADiscreteOnePlusOne*. Ovšem *ng_trajectory* je implementována s použitím starší verze *Nevergrad*, přesněji verze $0.3.0$, která nedovoluje vytvoření pole binárních proměnných. V tomto případě bylo tedy použito pole spojitých proměnných omezených na intervalu $[0, 1]$. Jelikož *Nevergrad* $0.3.0$ inicializuje toto pole proměnných v polovině jejich rozsahu a začínat s malým počtem segmentů by způsobilo mnoho neproveditelných řešení, segmenty s hodnotou $0.5$ a vyšší jsou použity ve výsledném řešení. Tento přístup není ideální, jelikož algoritmus může dlouho upravovat proměnnou bez jakékoli změny v ohodnocení dané kombinace parametrů a to může zhoršit jeho funkci.

Z důvodu uvedeném na konci předchozího odstavce byla celá knihovna *ng_trajectory* upravena tak, aby bylo možné použít nejnovější verzi *Nevergrad* $0.5.0$. V nových verzích proběhly rozsáhlé změny, jak v množství funkcí, tak ve struktuře celé knihovny. Díky tomu je možné vytvořit pole čistě binárních proměnných. Opět byla využita optimalizační funkce *DoubleFastGADiscreteOnePlusOne* a následně byla použita funkce *PortfolioDiscreteOnePlusOne* která podle [9] dosahuje lepších výsledků v kratším čase. Pro porovnání byla následně tato funkce využita i ve starší verzi knihovny.

## IV. Experimenty

### A. Nastavení paramerů

Kvůli výpočetní náročnosti optimalizace bylo pro získání výsledků využit školní server s 32 jádrovým procesorem Intel(R) Xeon(R) Silver 4110 o frekvenci 2.10 GHz a operačním systémem Debian GNU/Linux 11 (bullseye). Všechny experimenty byly provedeny na stejném serveru a se stejným optimalizačním rozpočtem, a tedy doba výpočtu je pro každý přibližně stejná.

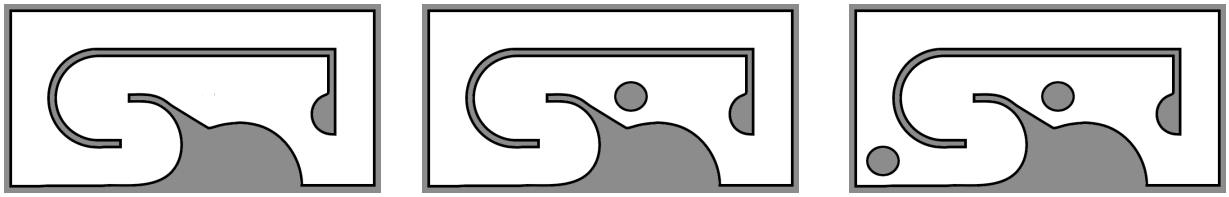
Pro tzv. fitness funkci byly využity parametry autonomního modelu F1/10 [10]. Tento model je často využíván pro testování všech aspektů autonomního řízení, jelikož algoritmy vyvinuté na této platformě jsou aplikovatelné i na reálných automobilech.

Pro testování byla využita závodní dráha ze soutěže F1/10 v Turíně o šířce průjezdu 2 metry znázorněna na obrázku 1. Konfigurace parametrů pro optimalizaci je uvedena v příloze A. Byly zvoleny dva různé počty segmentů, 13 a 25. První hodnota odpovídá nejlepšímu počtu určenému v [1]. Druhá zkoumá chování programu pro vyšší počty segmentů. Díky tomu, že bylo možné pro všechny experimenty vytvořit identické podmínky, lze výsledky porovnat čistě na základě doby potřebné k objetí jednoho kola finální trajektorie.

### B. Výsledky

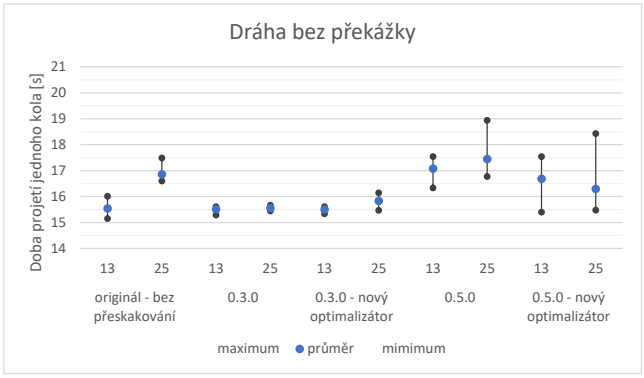
Z provedených experimentů byly vytvořeny 3 grafy, viz obrázek 2. Ty odpovídají výsledkům získaných z příslušných tratí z obrázku 1. Na každém grafu je znázorněn průměrná doba projetí kola, včetně maxima a minima, z 5 proveditelných výsledků testů. Vygenerování několika neproveditelných řešení bylo zaznamenáno pouze v posledních dvou testovacích případech, tedy při použití knihovny *Nevergrad* verze 0.5.0.

Na obrázku 3 jsou znázorněny nejrychlejší trajektorie z provedených testů v porovnání s výsledky z originální knihovny.

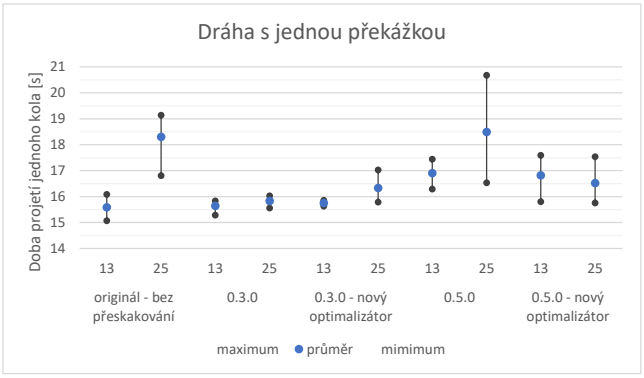


(a) (b) (c)

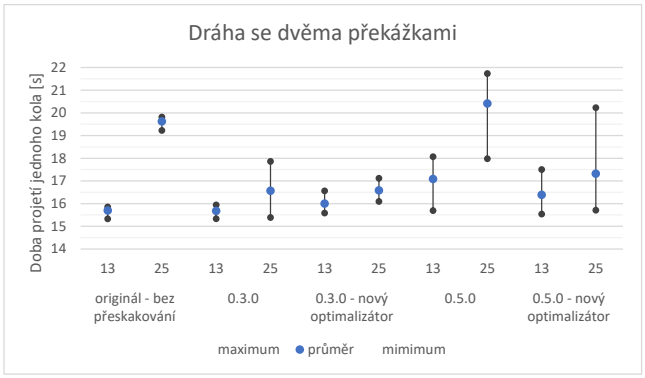
Obrázek 1: Dráhy použité na experimenty: 1a bez překážky, 1b s jednou překážkou a 1c se dvěma překážkami




(a)




(b)



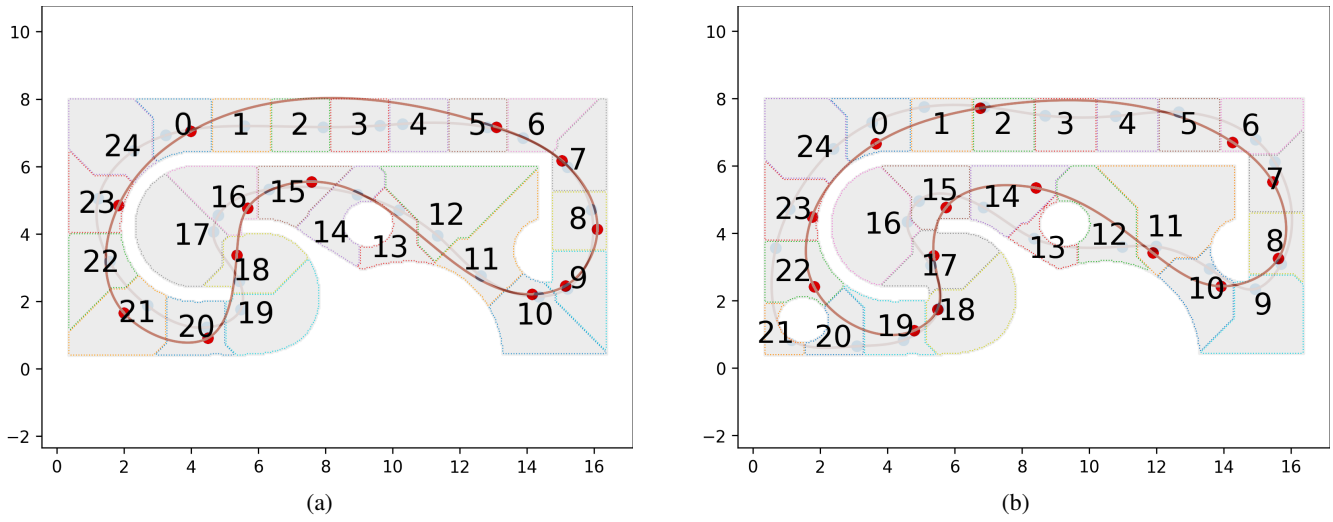

(c)

Obrázek 2: Výsledky experimentů: 2a bez překážky, 2b s jednou překážkou a 2c se dvěma překážkami

### *C. Diskuze*

Z výsledků je vidět, že se potvrdila myšlenka, že v případě, kdy je překážka vně segmentu je možné umístit daný bod tak, aby trajektorie byla stále optimální i bez potřeby přeskakování. Ovšem pokud nastane stav, kdy je segment umístěn např. jako na obrázku 3a, čas projetí dráhou se značně zvýší. Po implementaci přeskakování segmentů se průměrná doba kola snížila přibližné o 3 $s$. V závodech na čas se počítá každá milisekunda, a proto je toto zrychleni v relativním pohledu opravdu velké. I přes to, že by funkce *PortfolioDiscreteOnePlusOne* měla vykazovat lepší výsledky, při použití s verzí 0.3.0 se neosvědčila.

Po využití *Nevergrad* verze 0.5.0 došlo k výraznému zhoršení výsledků. I pro optimální počet segmentů 13 byla doba kola o více jak sekundu větší. Tato verze je poměrně nová a také se velmi liší od verze 0.3.0, a proto je možné, že provedené změny ovlivnili kompatibilitu s *ng_trajctory*. Po změně optimalizační funkce se výsledky více přiblížili optimálnímu řešení, minimum z provedených testů odpovídalo hledané trajektorii, ovšem výsledky z 5 testů se od sebe velmi lišily a průměrná doba kola byla stále o sekundu delší. Lepších výsledků bychom mohli dosáhnout po zvětšení optimalizačního rozpočtu, což ale znamená i delší dobu optimalizace.




Obrázek 3: Znázornění nejrychlejší trajektorie při přeskakování segmentů pro 25 bodů. V pozadí je viditelná nejrychlejší trajektorie z původní knihovny. 3a doba průjezdu 15.56 $s$ versus 16.81 $s$ a 3b doba průjezdu 15.39 $s$ versus 19.23 $s$.

## V. Závěr

V této práci byla představena úprava již existujícího algoritmu [1], která umožňuje jeho využití i v případě, kdy se na dráze vyskytují statické překážky. Tato úprava spočívá v možnosti přeskakovat jednotlivé segmenty, tedy vyřadit body příslušné přeskočeným segmentům z tvorby finální trajektorie.

Podařilo se výrazně zkrátit dobu projetí kola v případě, že je dráha rozdělena na více segmentů, než je předpokládané optimum. Celkový čas byl kratší až o 3 $s$, což je ve světě závodních aut značný rozdíl.

Budoucí práce se bude více zabývat vylepšení *ng_trajectory* tak, aby za použití *Nevergrad* verze 0.5.0 vykazovala lepší výsledky, než nyní. V této práci už byla tato myšlenka načata, ovšem výsledky nebyly postačující a problém vyžaduje podrobnější prozkoumání.

## Reference


[1] Klapálek, J., Novák, A., Sojka, M., & Hanzálek, Z. (2021). Car Racing Line Optimization with Genetic Algorithm using Approximate Homeomorphism [Conference paper]. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 601–607. https://doi.org/10.1109/IROS51168.2021.9636503

[2] A. Meyer-Baese and V. Schmid, "Chapter 5 - Genetic Algorithms," in *Pattern Recognition and Signal Analysis in Medical Imaging* (Second Edition) (A. Meyer-Baese and V. Schmid, eds.), pp. 135–149, Oxford: Academic Press, Jan. 2014.

[3] T. W. Gamelin and R. E. Greene, *Introduction to topology*. Courier Corporation, 1999. ISBN: 978-0486406800.

[4] F. Braghin, F. Cheli, S. Melzi, and E. Sabbioni, "Race driver model," *Computers & Structures*, vol. 86, pp. 1503–1516, July 2008.

[5] L. Cardamone, D. Loiacono, P. L. Lanzi and A. P. Bardelli, "Searching for the optimal racing line using genetic algorithms,"Proceedings of the *2010 IEEE Conference on Computational Intelligence and Games*, 2010, pp. 388-394, doi: 10.1109/ITW.2010.5593330.

[6] Dana Vrajitoru. 2019. Trajectory optimization for car races using genetic algorithms. *In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '19)*. Association for Computing Machinery, New York, NY, USA, 85–86. https://doi.org/10.1145/3319619.3326792

[7] Receveur, JB., Victor, S. & Melchior, P. Autonomous car decision making and trajectory tracking based on genetic algorithms and fractional potential fields. *Intel Serv Robotics 13*, 315–330 (2020). https://doi.org/10.1007/s11370-020-00314-x

[8] J. Rapin and O. Teytaud, *"Nevergrad - A gradient-free optimiza-tion platform."* https://GitHub.com/FacebookResearch/Nevergrad, 2018. [cit. květen 2022]

[9] J. Rapin and O. Teytaud, *"Examples of benchmarks" in Nevergrad documentation* https://facebookresearch.github.io/nevergrad/benchmarks.html#discrete, 2018. : 21, 2022. [cit. květen 2022]

[10] "F1tenth", [online] Available: https://f1tenth.org.

# Příloha A
## Nastavení parametrů pro experimenty


```
{
    "_version": 3,
    "_comment": "Dummy torino configuration for nevergrad trajectory.",
    "loops": 5,
    "variate": "groups",
    "interpolator": "cubic_spline",
    "segmentator": "flood_fill",
    "selector": "uniform",
    "groups": [13, 25],
    "penalizer": "centerline",
    "penalizer_init": {
        "method": "max"
    },
    "cascade": [
        {
            "algorithm": "matryoshka",
            "budget": 10000,
            "layers": 5,
            "criterion": "profile",
            "criterion_args": {
                "overlap": 100
            }
        }
    ],
    "start_points": "start_points.npy",
    "valid_points": "valid_points.npy",
    "prefix": "map-torino",
    "plot": true,
    "plot_mapping": true,
    "plot_args": [
        {
            "_figure": {
                "function": "axis",
                "_args": [ "equal" ]
            },
            "trackPlot": [ "@track" ]
        },
        {
            "pointsPlot": {
                "_args": [ "@result" ]
            },
            "pointsScatter": {
                "_args": [ "@rcandidate" ]
            }
        }
    ]
}
```