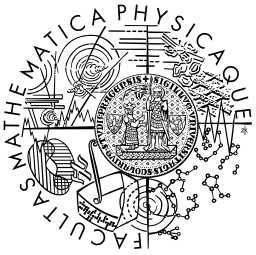

MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Marek Bečvář

Evoluce robotů v simulovaném fyzikálním prostředí

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. František Mráz, CSc.

Studijní program: Informatika

Studijní obor: Informatika se specializací Umělá inteligence

Praha 2023

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Poděkování.

Název práce: Evoluce robotů v simulovaném fyzikálním prostředí

Autor: Marek Bečvář

Katedra: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. František Mráz, CSc., Katedra softwaru a výuky informatiky

Abstrakt: Abstrakt.

Klíčová slova: klíčová slova

Title: Evolution of robots in a simulated physical environment

Author: Marek Bečvář

Department: Department of software and computer science education

Supervisor: RNDr. František Mráz, CSc., Department of software and computer science education

Abstract: Abstract.

Keywords: key words

Obsah

Úvod

Následuje několik ukázkových kapitol, které doporučují, jak by se měla bakalářská práce sázet. Primárně popisují použití TEXové šablony, ale obecné rady poslouží dobře i uživatelům jiných systémů.

1. Základní pojmy

V této kapitole vysvětlíme a rozebereme důležité pojmy, se kterými se v dalším popisu práce budeme setkávat. Znalost těchto pojmů je potřebná pro pochopení důvodů zvolení daných vybraných technologií a pro pochopení základního rozboru implementace řešení, kterou popíšeme v následujících kapitolách.

V této kapitole nejdříve vysvětlíme základní teorii ohledně evolučních algoritmů a ukážeme si již existující knihovny pracující nebo pomáhající pracovat s genetickými algoritmy. Poté se podíváme na základ neuronových sítí a popíšeme pokročilejší evoluční algoritmy sloužící přímo k vývoji těchto sítí. Dále popíšeme simulátory prostředí a fyzikální simulátory, které využijeme pro simulaci při vyvíjení našich robotů.

1.1 Evoluční algoritmy

TODO: POPIS EVOLUČNÍ ALGORITMY

1.1.1 Existující implementace

Pro vývoj řízení robotů budeme využívat evoluční algoritmy. Naše knihovna tedy bude muset implementovat množství alespoň základních genetických operátorů, používaných při vývoji jedinců a co nejlehčeji umožnit jejich konfiguraci před spouštěním jednotlivých experimentů. Naším cílem je co možná nejvíce zpřístupnit knihovnu, která má být výsledkem této práce, aby uživatel se základní znalostí genetických algoritmů a programovacího jazyka byl schopný pochopit běh algoritmu a v případě potřeby mohl jednoduše provádět zásahy do jeho běhu. Není pro nás tedy nutné najít tu nejefektivnější knihovnu, nýbrž tu, která přinese výhody jako přehlednost a snaha úpravy algoritmů, bez větších obtíží s implementací a pochopením knihovny.

Dále ukážeme pár možností knihoven, implementující nebo usnadňující implementaci genetických algoritmů nebo jejich částí.

- **DEAP**
  DEAP (*Distributed Evolutionary Algorithms in Python*) je open-source Python knihovna pro rychlou tvorbu a prototypování evolučních algoritmů, která se snaží jejich tvorbu zjednodušit pomocí přímočarého postupu, podobného pseudokódu, který je se základní znalostí knihovny poměrně jednoduchý na porozumění Knihovna je tvořena ze dvou hlavních struktur `creator`, který slouží k vytváření genetických informací jedinců z libovolných datových struktur a `toolbox`, který je seznamem nástrojů (genetických operátorů), které mohou být použité při sestavování evolučního algoritmu. Dalšími menšími strukturami jsou `algorithms` obsahující 4 základní typy algoritmů a `tools` implementující další základní operátory (kusy operátorů), které je posléze možné přidávat do `toolbox`. Pomocí těchto základních stavebních bloků mohou uživatelé poměrně jednoduše začít tvořit skoro libovolné evoluční algoritmy (Fortin a kol., 2012). Následuje ukázka kódu tvorby základních částí evolučního algoritmu pro One Max problém, popsaná v oficiální dokumentaci knihovny DEAP (DEAP). V problému One

Max máme populaci jedinců tvořených z jedniček a nul a chceme vyvinout jedince, který má na všech pozicích jen samé jedničky.

```
import random

from deap import base
from deap import creator
from deap import tools
```

Tvorba vlastních tříd fitness funkce a individuí pomocí `creator`.

```
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)
```

Šablonu vytvořených individuí můžeme dále použít při tvorbě populace následujícím stylem.

```
toolbox = base.Toolbox()

toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat,
                 creator.Individual, toolbox.attr_bool, 100)
toolbox.register("population", tools.initRepeat,
                 list, toolbox.individual)
```

Zde jsme vytvořili dvě inicializační funkce `individual()` a `population()`, které když zavoláme, vytvoří novou instanci individua nebo populace.

V tomto problému je evaluační funkce jednoduchá, vytvoříme ji následovně.

```
def evalOneMax(individual):
    return sum(individual)
```

Pro zvolení genetických operátorů je musíme registrovat v `toolbox`.

```
toolbox.register("evaluate", evalOneMax)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)
```

Nyní je vše připravené a můžeme začít s evolucí populace. Nejdřív populaci vygenerujeme a poté populaci vyhodnotíme.

```
pop = toolbox.population(n=300)
fitnesses = list(map(toolbox.evaluate, pop))
for ind, fit in zip(pop, fitnesses):
    ind.fitness.values = fit

# Extracting all the fitnesses of
fits = [ind.fitness.values[0] for ind in pop]
```

Následně začneme evoluci. Naši jedinci jsou tvořeni ze 100 čísel 0/1. Evoluce poběží tak dlouho, dokud nebude existovat jedinec z populace, který má všech 100 čísel nastavených na 1, nebo evoluce doběhne určitého počtu generací.

```
# Number of generations
g = 0
# Begin the evolution
while max(fits) < 100 and g < 1000:
    g = g + 1
```

Nejdříve na základě fitness vybereme určité jedince jako rodiče další generace.

```
    # Select the next generation individuals
    offspring = toolbox.select(pop, len(pop))
    # Clone the selected individuals
    offspring = list(map(toolbox.clone, offspring))
```

Následně na vybrané jedince aplikujeme operátory křížení a mutace.

```
    # Apply crossover and mutation on the offspring
    for child1, child2 in zip(offspring[::2], offspring[1::2]):
        if random.random() < CXPB:
            toolbox.mate(child1, child2)
            del child1.fitness.values
            del child2.fitness.values

    for mutant in offspring:
        if random.random() < MUTPB:
            toolbox.mutate(mutant)
            del mutant.fitness.values
```

`CXPB` a `MUTPB` jsou pravděpodobnosti aplikování operátorů křížení a mutace. Klíčové slovo `del` udělá hodnotu fitness daného jedince neplatnou.

Dále necháme zopakovat evaluaci celé populace jedinců a proces se může opakovat.

Dle zdrojů popisující porovnání několika Python modulů, snažící se ulehčit práci s evolučními algoritmy, je DEAP nejefektivnější, tedy že tvoří nejkratší kód, v porovnání počtu řádků potřebných pro tvorbu algoritmu řešící One Max problém z ukázky (Fortin a kol., 2012).

- **Inspyred**
  Inspyred poskytuje většinu z nejpoužívanějších evolučních algoritmů a dalších přírodou inspirovaných algoritmů (simulace reálných biologických systémů - př. optimalizace mravenčí kolonií) v jazyce Python. Knihovna je již předpřipravená s funkčním řešením, ve formě jednotlivých komponentů (Python funkcí), které si uživatel může sám upravovat, nebo je úplně nahradit za vlastnoručně vytvořené řešení v podobě vlastních funkcí. Uživatel pak

při tvorbě algoritmu definuje několik struktur, které ovlivňují jak celý vývoj probíhá. Těmito strukturami jsou - struktury specifické k danému řešenému problému `generator` (jak jsou generována řešení = jedinci) a `evaluator` (definice fitness funkce)). A struktury specifické k danému evolučnímu algoritmu - `observer` (jak uživatel monitoruje evoluci), `terminator` (definuje pravidla pro konec evoluce), `selector` (kteří jedinci se mají stát rodiči), `variator` (jak jsou potomci vytvořeni z aktuálních jedinců), `replacer` (volí kteří jedinci mají přežít do další generace), `migrator` (jak se přenáší jedinci mezi různými populacemi/generacemi) a `archiver` (jak jsou jedinci ukládání mimo stávající populaci). Libovolný vybraný z těchto komponentů pak může být nahrazen odpovídající vlastní implementací (Tonda, 2020).

1.2 Neuronové sítě

1.3 NEAT

1.4 HyperNEAT

1.5 Simulované prostředí

Jelikož chceme vyvíjet řízení robotů založených na korektních fyzikálních pravidlech a interakcích, je pro tuto práci důležité vybrat vhodný simulátor prostředí. Přáli bychom si mít možnost jednoduše konfigurovat co nejvíce vlastností prostředí a zároveň mít co nejlehčí přístup k morfologii simulovaných robotů. Zároveň chceme, abychom měli možnost do morfologie robotů nějakým stylem zasahovat i v průběhu vývoje a aktivně ji za běhu měnit. Jelikož plánujeme v prostředí provádět experimenty s různými typy robotů, používající různé styly pohybu (typy motorů, kloubů, tvarů končetin, atd.), je potřebné, aby fyzikální simulátor (*fyzikální řešič=solver*) byl schopný simulovat i složitější typy robotů. Takovými mohou být právě třeba kráčející roboti, neboli roboti používající k pohybu končetiny připomínající nohy, na rozdíl od jednodušších typů robotů, kteří se mohou pohybovat pomocí kol, jejichž simulace bývá mnohdy jednodušší. Stejně tak jak potřebujeme umožnit složitost robotů, protože nebudeme mít možnost vlastnoručně kontrolovat každý parametr, který bude při vývoji robotům přiřazen, potřebujeme zajistit, aby fyzikální simulátor zvládal libovolné rozsahy parametrů a simulace zůstal pro tyto parametry stabilní. Zároveň chceme, aby simulátor v prostředí byl deterministický, což umožní, že předváděné experimenty můžeme dle potřeby opakovat a výsledky tak náležitě prezentovat. Evoluční algoritmy jsou velmi lehce paralelizovatelné a tedy pro urychlení procesu vývoje a experimentů nám bude výhodné, pokud by simulace zvládala paralelní běh na více vláknech (více simulací, každá na vlastním vlákně). V posledním řadě pro lehčí integraci do vlastního modulu bude užitečné, aby modul spravující zvolený simulátor byl open-source, což nám dá volnost v případě, že si budeme chtít chování systémů v prostředí nějak vlastnoručně upravit.

Při hledání simulátorů prostředí, které by vyhovovali našim požadavkům a umožňovali kontrolu a ovládání prostředí skrz zvolený jazyk Python, jsme narazili na několik možností.

- **Gazebo**
  Gazebo je sada open-source víceplatformní knihoven pro vývoj, výzkum a aplikaci robotů, původně založená v roce 2002. Umožňuje kompletní kontrolu nad simulací dynamického 3D prostředí s více agenty a generování dat ze simulovaných senzorů. Fyzikálně korektní interakce v prostředí pak od začátku projektu zajišťuje známý fyzikální simulátor ODE 1.5.1, nad kterým Gazebo tvoří abstraktní vrstvu, umožňující snazší tvorbu simulovaných objektů různých druhů. V dnešní době je stále výchozím fyzikálním simulátorem ODE, nicméně uživatel již může vybrat celkem ze čtyř různých simulátorů - Bullet 1.5.1, Simbody, Dart 1.5.1 a ODE. Uživatel s knihovnou pracuje skrz grafické rozhraní založené na knihovně Open Scene Graph používající OpenGL, nebo skrz příkazovou řádku. Prostředí a roboti mohou být tvořené buď skrz grafické prostředí, nebo v textovém formátu XML. Limitací Gazebo je pak neschopnost rozdělit simulace mezi vícero vláken kvůli vnitřní architektuře spojené s fyzikální simulací (Koenig a Howard, 2004).

- **Webots**
  Webots je open-source víceplatformní robustní a deterministický robotický simulátor vyvíjený od roku 1998, umožňující programování a testování virtuálních robotů mnoha různých typů a jednoduchou následnou aplikaci softwaru na reálné roboty. Simulátor je možné použít pro simulaci prostředí s vícero agenty najednou s možnostmi lokální i globální komunikace mezi agenty. Výpočty fyzikálních interakcí zajišťuje fyzikální simulátor ODE. Pro vývoj robotů a prostředí je možné využít řady programovacích jazyků a to C, C++, Python, Java, MATLAB nebo ROS (*Robot Operating System*). Prostředí umožňuje práci v grafickém rozhraní a vizualizaci simulací pomocí OpenGL. Knihovna dále nabízí využití připravených modelů robotů, vlastní editor robotů a map a možnosti vložení vlastních robotů z 3D modelovacích softwarů v CAD formátu (Michel, 2004) (Webots).

- **CoppeliaSim**
  CoppeliaSim (kdysi známý pod jménem *V-REP = Virtual Robot Experimentation Platform*) je víceplatformní simulační modul pro vývoj, testování a jednoduchou aplikaci softwaru pro roboty. Dovoluje vývoj ovladačů pomocí 7 různých programovacích jazyků a ulehčuje jejich aplikace v simulovaných a skutečných robotech. Simulaci ovladačů je možno jednoduše rozdistribuovat mezi vícero vláken dokonce vícero strojů, což urychluje vývoj a snižuje nároky na procesor v době simulace. Navíc je možné vyvíjený ovladač nechat v době simulací běžet na vlastním na dálku připojeném robotovi, co dále ulehčuje přenos finální verze ovladačů od vývoje do skutečného světa. Prostředí umožňuje práci s širokou řadou typů objektů, druhů kloubů, senzorů a dalších objektů obvykle používaných při vývojích robotických ovladačů. Obsahuje lehce použitelný editor prostředí a robotů samotných s řadou předem vytvořených modelů, které může uživatel hned využít. Modely zároveň mohou být přidány skrz řadu různých formátů (XML, URDF, SDF). Prostředí podporuje pět různých fyzikálních simulátorů (Bullet, ODE, MuJoCo 1.5.1, Vortex 1.5.1, Newton), mezi kterými si uživatel může vybrat dle potřeb přesnosti (reálnosti), rychlosti a

dalších možností jednotlivých fyzikálních simulátorů (Rohmer a kol., 2013) (Nogueira, 2014).

1.5.1 Fyzikální simulátory

V této podkapitole se podíváme na základní popis a možné výhody a nevýhody jednotlivých fyzikálních simulátorů, na které jsme narazili při hledání simulátorů prostředí.

- **ODE**
  ODE (*Open Dynamics Engine*) je víceplatformní open-source fyzikální simulátor, jehož vývoj začal v roce 2001. Vhodný pro simulaci pevných těles s různými druhy kloubů a pro detekci kolizí. Tvořený pro využití v interaktivních nebo real-time simulacích, upřednostňující rychlost a stabilitu nad fyzikální přesností (Smith a kol., 2007). Potřeba menších simulačních kroků pro stabilitu. Hodí se pro simulaci vozidel, pochodujících robotů a virtuálních prostředí. Široké využití v počítačových hrách a 3D simulačních nástrojích CoppeliaRobotics.

- **Bullet**
  Bullet je open-source fyzikální knihovna, podporující detekci kolizí a simulaci pevných a měkkých těles. Bullet je používán jako fyzikální simulátor pro hry, vizuální efekty a robotiku (Coumans). Byl použit jako hlavní fyzikální simulátor pro simulaci NASA *Tensegrity* robotů (s vlastními úpravami pro simulaci měkkých těles, kvůli nerealistickým metodám řešení simulace provazů) (Izadi a Bezuijen, 2018).

- **Dart**
  Dart (*Dynamic Animation and Robotics Toolkit*) je víceplatformní open-source knihovna pro simulace a animace robotů. Od předchozích se odlišuje stabilitou a přesností, díky zobecněné reprezentaci koordinací pevných těles v simulaci. Na rozdíl od ostatních fyzikálních simulátorů, aby dal vývojáři plnou kontrolu nad simulací, umožňuje Dart plný přístup k interním hodnotám simulace. Zároveň se díky línému vyhodnocování hodí pro vývoj real-time ovladačů pro roboty. (Lee a kol., 2018).

- **MuJoCo**
  MuJoCo (*Multi-Joint Dynamics with Contact*) je open-source fyzikální simulátor pro vývoj v oblasti robotiky, biomechaniky a dalších. Často je využíváno pro testování a porovnávání různých metod navrhování robotických systémů jako jsou třeba evoluční algoritmy nebo metody zpětnovazebného učení (Salimans a kol., 2017). V simulacích je pro roboty možné nakonfigurovat využití mnoha druhů aktuátorů, včetně těch simulující práci svalů a k dispozici je i velké množství kloubů. Simulátor zároveň umožňuje velký nárůst v rychlosti běhu simulace za pomoci plné podpory paralelizace na všech dostupných vláknech a stabilitě simulace i při velmi velkých simulačních krocích (**?**). Zároveň nabízí jednoduchý styl, jakým si může uživatel konfigurovat všechny detaily simulace a samotných simulovaných robotů pomocí jednoduchých XML konfiguračních souborů (XML formát modelů

MJCF). V komplexním rozboru řady četně používaných fyzikálních simulátorů byl simulátor MuJoCo hodnocen jako jeden z nejlepších co se týče stability, přesnosti a rychlosti simulací. Další výhodou zlepšující přesnost tohoto simulátoru je, že MuJoCo pro simulaci používá kloubní souřadnicový systém, který předchází narušení fyzikálních pravidel a tedy nepřesností v kloubech (Erez a kol., 2015).

- **Vortex**
  Vortex je uzavřený, komerční fyzikální simulátor určený pro tvorbu reálnému světu odpovídajících simulací. Obsahuje mnoho parametrů, umožňující nastavení reálných fyzikálních parametrů dle potřeb, většinou industriálních a výzkumných aplikací (CoppeliaRobotics) (Yoon a kol., 2023).

2. Specifikace

2.1 Funkční požadavky

3. Implementace

V předchozí kapitole jsme prošli funkční požadavky, očekávané od vyvíjeného souboru programů. Následuje rozbor jednotlivých modulů, které vznikly při vlastní implementaci. Zároveň zde projdeme možné alternativy, které se pro vývoj nabízejí a probereme důvody stojící za zvolením jednotlivých z možností.

Nejprve vysvětlíme volbu programovacího jazyka, ve kterém je celá knihovna vytvořena. Poté projdeme systémy umožňující vývoj ve fyzikálním prostředí a ovládání uživatelem definovaných robotů. Zde představíme i možnosti tvorby vlastních robotů. Dále ukážeme možné varianty modulů umožňující vývoj řízení robotů pomocí genetických algoritmů a popíšeme vlastní implementaci. Následně projdeme všechny části implementace spojující tyto moduly do přístupné rozšiřitelné knihovny. V poslední části představíme implementaci grafického rozhraní, které slouží uživateli, který chce používat knihovnu a provádět experimenty, bez nutnosti využití příkazové řádky.

3.1 Programovací jazyk

Jako programovací jazyk, ve kterém tento projekt bude psán, jsme zvolili **Python**. Cílem projektu je vytvořit platformu, kterou bude uživatel moci použít k vývoji robotů pomocí evolučních algoritmů. Pokud uživatel bude mít potřebu jakkoli připravený proces vývoje měnit, Python lehce umožní nahlédnout do zdrojových kódů vypracované knihovny a provést úpravy dle vlastních potřeb. Zároveň to umožňuje rozšiřování knihovny o nové metody, které bude chtít uživatel zkusit zařadit do již funkčního procesu. Jednoduchá čitelnost Pythonu spojená s rychlostí, jakou mohou být prováděny iterace změn bez potřeby zdlouhavého překladu celé knihovny, se zdají býti dostatečně dobré vlastnosti pro volbu programovacího jazyka pro tento projekt.

3.2 Simulované fyzikální prostředí

Po zhodnocení vypsaných a dalších možností jsme vybrali pro využití v tomto projektu fyzikální simulátor MuJoCo. Na rozdíl od ostatních se zdá býti přístupnější do začátku a zároveň dostatečně robustní a konfigurovatelný tak, aby splnil veškeré požadavky, které od fyzikálního simulátoru máme.

MuJoCo 1.50 je fyzikální simulátor zpřístupněný skrz volně dostupné aplikační rozhraní společnosti **OpenAI** v rámci jejich sady různých prostředí **Gym** (textové hry, jednoduché 2D i plně fyzikálně simulované 3D prostředí, Atari hry aj.) pro vývoj metod zpětnovazebného učení na různých problémech. Toto rozhraní umožňuje uživatelům jednoduchý přístup k datům z poskytnutých prostředí a ovládání prostředím definovaných agentů, pomocí standardizovaných vstupů i výstupů napříč všemi prostředími. Tímto způsobem můžeme velmi lehce ovládat i roboty v prostředích simulátoru MuJoCo. Navíc otevřená vlastnost tohoto aplikačního rozhraní umožňuje úpravu částí procesu tak, aby se lépe hodil při řešení námi zvolených problémů. Přestože je **Gym** převážně používána pro vývoj metod zpětnovazebného učení agentů, nic nám nebrání a je velmi jednoduché namísto

toho využít vlastního agenta, který je vyvíjen pomocí evolučních algoritmů.

3.2.1 Roboti

3.3 Genetické algoritmy

Po porovnání různých možností modulů pro tvorbu a použití evolučních algoritmů v naší knihovně, jsme se rozhodli pro vlastní implementaci jednotlivých částí evolučních algoritmů (genetických operátorů) a jejich propojení mezi sebou. Důvodem je hlavně jednodušší zapojení do zbytku knihovny a snížení nároků na znalosti mnohdy složitých výše popsaných externích knihoven pro uživatele, který by případně mohl chtít si do naší knihovny dopsat vlastní kus evolučního algoritmu. Tímto způsobem, pokud bude chtít něco takového udělat, dojde-li k dodržení zdrojovým kódem stanovených pravidel, vlastní kus kódu (metoda popisující genetický operátor) bude možné hned bez problému využít při dalším vývoji robotů.

3.4 Implementace knihovny

3.5 Grafické rozhraní

4. Experimenty a výsledky

4.1 Vývoj řízení robotů

4.2 Vývoj řízení a morfologie robotů

4.3 Diskuze výsledků

Závěr

Seznam použité literatury

CoppeliaRobotics. https://www.coppeliarobotics.com. URL `https://www.coppeliarobotics.com`. Robot simulation software.

Coumans, E. Bullet real-time physics simulation. URL `https://pybullet.org/wordpress/`. Accessed: March 19, 2023.

DEAP, P. Deap documentation. URL `https://deap.readthedocs.io/`. Accessed: March 19, 2023.

Erez, T., Tassa, Y. a Todorov, E. (2015). Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 4397–4404. IEEE.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A. G., Parizeau, M. a Gagné, C. (2012). Deap: Evolutionary algorithms made easy. *The Journal of Machine Learning Research*, **13**(1), 2171–2175.

Izadi, E. a Bezuijen, A. (2018). Simulating direct shear tests with the bullet physics library: A validation study. *PLOS one*, **13**(4), e0195073.

Koenig, N. a Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE.

Lee, J., X. Grey, M., Ha, S., Kunz, T., Jain, S., Ye, Y., S. Srinivasa, S., Stilman, M. a Karen Liu, C. (2018). Dart: Dynamic animation and robotics toolkit. *The Journal of Open Source Software*, **3**(22), 500.

Michel, O. (2004). Cyberbotics ltd. webots™: professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, **1**(1), 5.

Nogueira, L. (2014). Comparative analysis between gazebo and v-rep robotic simulators. *Seminario Interno de Cognicao Artificial-SICA*, **2014**(5), 2.

Rohmer, E., Singh, S. P. N. a Freese, M. (2013). Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. www.coppeliarobotics.com.

Salimans, T., Ho, J., Chen, X., Sidor, S. a Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Smith, R. a kol. (2007). Open dynamics engine.

Tonda, A. (2020). Inspyred: Bio-inspired algorithms in python. *Genetic Programming and Evolvable Machines*, **21**(1-2), 269–272.

Webots. http://www.cyberbotics.com. URL http://www.cyberbotics.com. Open-source Mobile Robot Simulation Software.

Yoon, J., Son, B. a Lee, D. (2023). Comparative study of physics engines for robot simulation with mechanical interaction. *Applied Sciences*, **13**(2), 680.

A. Přílohy

A.1 První příloha