VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

Využití Robotického operačního systému (ROS) pro řízení kolaborativního robota UR3

Utilization of Robotic Operating System (ROS) for control of collaborative robot UR3

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Martin Juříček

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Roman Parák

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Martin Juříček**
Studijní program:	Strojírenství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**Ing. Roman Parák**
Akademický rok:	2019/20

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Využití Robotického operačního systému (ROS) pro řízení kolaborativního robota UR3

Stručná charakteristika problematiky úkolu:

Práce bude zahrnovat rešerši v oblasti kolaborativních robotů a seznámaní se s průmyslovými roboty společnosti Universal Robots, blíž popíše zvoleného robota UR3. Teoretická část práce bude také zahrnovat rešerši frameworku ROS a simulačního prostředí Gazebo.
Předmětem práce bude kofigurace frameworku ROS a návrh řídícího programu pro vybranou laboratorní úlohu. Závěr práce bude věnován implementaci návrhu řídícího programu a ověření funkčnosti vytvořeného řešení.
Práce předpokládá aktivní přístup studenta a nutnost práce v laboratoři.

Cíle bakalářské práce:

– Nastudujte problematiku kolaborativních robotů. Zpracujte přehled aktuálního stavu v dané oblasti.
– Proveďte rešerši průmyslových robotů společnosti Universal Robots a blíže popište zvoleného kolaborativního robota UR3.
– Proveďte rešerši v oblasti využití Robotického operačního systému (ROS) a simulačního prostředí Gazebo.
– Proveďte konfiguraci frameworku ROS a kolaboratívniho robotu UR3.
– Navrhněte řídící program pro vybranou laboratorní úlohu.
– Implementujte návrh řídícího programu.
– Ověřte funkčnost vytvořeného řešení pomocí simulace a na reálnem robotu.

Seznam doporučené literatury:

KOLÍBAL, Zdeněk. Roboty a robotizované výrobní technologie. Brno: Vysoké učení technické v Brně - nakladatelství VUTIUM, 2016. ISBN 978-80-214-4828-5.

ROS.org. ROS.org | Powering the world's robots. [online]. 2.11.2016 [cit. 2016-11-02]. Dostupné z: http://www.ros.org/

THRUN, Sebastian, FOX, Wolfram and Dieter. Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series). Intelligent robotics and autonomous agents. The MIT Press, August 2005.

SICILIANO, Bruno a KHATIB, Oussama, ed. Springer handbook of robotics. 2nd edition. Berlin: Springer, [2016]. ISBN 978-3-319-32550-7.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2019/20

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Cílem bakalářské práce je vytvoření řídícího programu, jeho následné otestování a ověření funkčnosti pro kolaborativního robota UR3 od firmy Universal Robots. Řídící program je napsán v jazyce python a integruje možnosti řízení skrz Robotický operační systém, kdy lze dosahovat definovaného bodu pomocí předem simulovaných trajektorií algoritmů Q-learning, SARSA, Deep Q-learning, Deep SARSA, a nebo za pomocí pouze frameworku MoveIT. V práci je pojednáno průřezem o tématech kolaborativní robotiky, Robotického operačního systému, simulačního prostředí Gazebo, zpětnovazebního a hluboké zpětnovazebního učení. Závěrem je popsán samotný návrh a implementace řídícího programu s dílčími částmi.

ABSTRACT

The aim of the bachelor's thesis is to create a control program, its subsequent testing and verification of functionality for the collaborative robot UR3 from the company Universal Robots. The control program is written in python and integrates control options through the Robotic Operating System, where a defined point can be reached using pre-simulated trajectories of Q-learning, SARSA, Deep Q-learning, Deep SARSA, or using only the MoveIT framework. The thesis deals with a cross-section of the topics of collaborative robotics, Robotic Operating System, Gazebo simulation environment, feedback and deep feedback learning. Finally, the design and implementation of the control program with partial parts is described.

KLÍČOVÁ SLOVA

Kolaborativní robotika, Universal Robots, ROS, Gazebo, zpětnovazební učení, hluboké zpětnovazební učení, hluboká neuronová síť, kobot, UR3, Keras, OpenAI, MoveIT, řízení robotického ramene

KEYWORDS

Collaborative robotics, Universal Robots, ROS, Gazebo, reinforcement learning, deep reinforcement learning, deep neural network, cobot, UR3, Keras, OpenAI, MoveIT, control robotic arm

BIBLIOGRAFICKÁ CITACE

JUŘÍČEK, Martin. *Využití Robotického operačního systému (ROS) pro řízení kolaborativního robota UR3*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Ing. Roman Parák.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce Ing. Romanu Parákovi. Za odborné konzultace, vedení, ochotu, spolupráci a jeho přístup při vytváření této práce. Také bych chtěl poděkovat své rodině, která mě během studia podporovala. V neposlední řadě bych rád poděkoval všem svým přátelům za ochotu a připomínky k pravopisným chybám.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením Ing. Romana Paráka a s použitím literatury uvedené v seznamu literatury.

V Brně dne 22. 5. 2020 ……

Martin Juříček

OBSAH

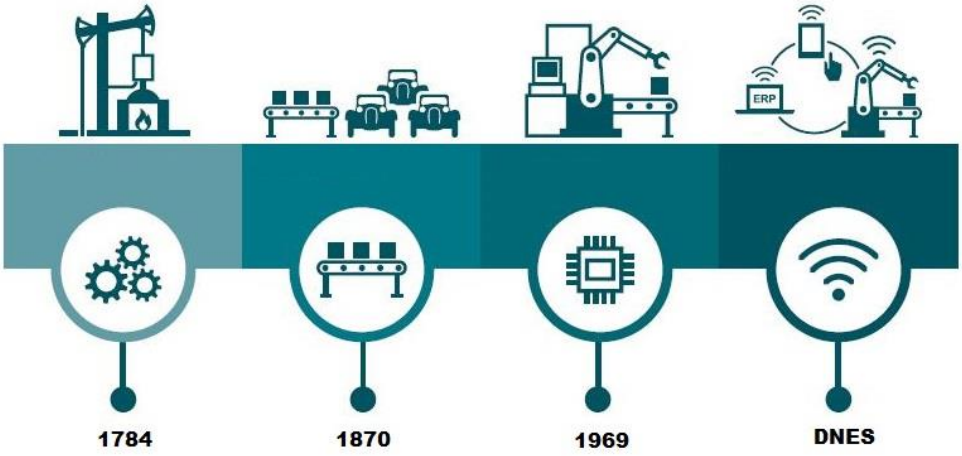
1 ÚVOD

Mezi symboly průmyslu 4.0 se stala kolaborativní robotika a umělá inteligence. Své uplatnění a využití v dnešní době nacházejí tyto obory v širokém spektru průmyslových odvětví. Kolaborativní robotika se stala naprostým trendem a její problematikou se zabývá mnoho menších, ale i větších firem. Cílem této bakalářské práce je návrh a implementace řídícího programu, s využitím frameworku ROS, pro kolaborativního robota UR3 od firmy Universal Robots. Úkolem řídícího programu je ovládání kolaborativního robota pomocí předem zaznamenaného simulovaného učení zpětnovazebních algoritmů, ale také i algoritmů hlubokého zpětnovazebního učení.

Úvodem je popsán současný stav kolaborativní robotiky, přičemž popisuje čtenáři zajímavé aplikace a možnosti programování kolaborativních robotů. Také je zmíněna bezpečnost a pravidla, kterými se řídí výrobci. Dále jsou blíže popsáni kolaborativní roboti od firmy Universal Robots. V této části je především kladen důraz na kolaborativního robota UR3. Další část seznamuje čtenáře s fungováním frameworku ROS a jeho nástroji. Zde je čtenáři přiblížené i simulační prostředí Gazebo a nová generace ROS. V praktické části je popsáno fungování podoboru umělé inteligence, zpětnovazební a hluboké zpětnovazební učení s popisem dosažených výsledků. Následující praktická část popisuje analýzu a využití dosažených výsledků učení a následné aplikaci k řízení fyzického kolaborativního robota UR3.

2 KOLABORATIVNÍ ROBOTIKA

Každá průmyslová revoluce byla odrazem technického pokroku, na počátku to byla mechanizace, následovala elektrifikace a ve třetí průmyslové revoluci byl pokrok definován využitím počítačů (obr. 1). V dnešní době se nacházíme na přelomu období průmyslu 4.0. Hlavním účelem je především vytvoření plně automatizovaných systémů, propojení výrobních, distribučních, obchodních a řídících systémů, kde vše zastřešuje kyberbezpečnost. Toto přelomové období s sebou přináší například využití umělé inteligence, internetu věcí a služeb. Nedílnou součástí je začlenění plánovaní a simulace výroby nebo aplikování pokročilé a kolaborativní robotiky.



Obr. 1: Průmyslové revoluce

Kolaborativní robotika je definována jako komunikace a spolupráce člověka s robotem, kdy robot pomáhá člověku plnit jeho práci. Kolaborativní robot a operátor většinou pracují ve společném prostoru, a proto úkoly, které kolaborativní roboti vykonávají, jsou většinou namáhavé rutinní práce, obtížné operace nebo procesy, které nelze zcela automatizovat. Oproti tomu konvenční průmyslové roboty od lidské obsluhy dělí striktní izolace, která má podobu mechanického oplocení nebo senzorových bariér, kdy při překročení dojde k okamžitému vypnutí robota. Také proto se tito roboti převážně aplikují na úkoly do nebezpečného prostředí nebo tam, kde je potřeba využít maximální rychlosti, přesnosti a manipulaci s těžkými předměty.

2.1 Současný stav a budoucnost kolaborativní robotiky

S pojmem kolaborativní robotiky se pojí také výraz kobot neboli kolaborativní robot. Tito roboti jsou primárně navrženi na spolupráci s člověkem. Tato vlastnost je vykoupena za cenu nižší rychlosti, dosahu a nosnosti oproti konvenčním průmyslovým robotům. V dnešní době se kolaborativní robotika stala celosvětovým trendem hned z několika důvodů:

- Cenová dostupnost, přestože pořizovací náklady jsou cirka 24 000 amerických dolarů, následná průměrná návratnost této investice je šest až dvanáct měsíců.
- Bezpečnost, která je nejpřitažlivější vlastnost kolaborativních robotů, umožňuje to, že mohou pracovat operátoři a koboti bok po boku.
- Flexibilní umístnění a rychlé nastavení, kdy montáž a jeho následné naprogramování trvá obvykle v řádu hodin.
- Jednoduché použití, přičemž i operátoři, kteří s programováním nemají téměř žádné zkušenosti, mohou díky uživatelsky příznivému prostředí jednoduše programovat. [1]

Kolaborativní robotika získala své opodstatnění především v menších a středních podnicích, kde je potřeba kombinovat sílu, přesnost a opakovatelnost robota s lidskou znalostí. Mezi největší a nejznámější výrobce kolaborativních robotů se staly firmy Universal Robots, FANUC a KUKA. Mezi méně známé firmy řadíme zejména AUBO, Techman Robot a Rethink Robotics.

Japonská firma FANUC, která je nejvýznamnější v oblasti průmyslové robotiky, představila svou vizi kolaborativních robotů řadou CR. V této řadě je vlajkovou lodí kobot s označením CR-35iA (obr. 2). Užitečné zatížení tohoto kobota je 4-35 kg, kdy dosah činí 1813 mm s přesností opakovatelnosti polohy ±0,03 mm, přitom se může pohybovat rychlostí až 750 mm/s. Jednou z nepostradatelných výhod je bezpečnostní systém DCS (*Dual Check System*), který drží operátory v bezpečí a současně udržuje prostor robotické buňky na nutné minimum. Tento systém neustále monitoruje polohu, rychlost a hranice bezpečné zóny. Všechny tyto funkce lze částečně nastavit přímo v dotykovém ovladači FANUC iPendant-Touch nebo iHMI. Bezpečnost je také zajištěna za pomoci silových snímačů a měkkému pryžovému obložení. Jistou nevýhodou je, že lze tohoto kobota instalovat pouze v horizontálním směru na podlahu. [2]

Obr. 2: FANUC CR-35Ia [3]

Konceptu propojení autonomního robota a robotického ramene se nejlépe zhostila německá firma KUKA. KMR iiwa je extrémně flexibilní robotický systém, který se skládá z kolaborativního robota LBR iiwa a mobilní autonomní plošiny (obr. 3). Hlavní výhodou je jeho mimořádná autonomie kvůli zabudovaným senzorům a navigačnímu systému KUKA.NavigationSolution, který využívá metodu SLAM (*Simultaneous localization and mapping*). Kola Mecanum umožňují mobilní plošině pohyb všemi směry a také rotaci v rozsahu 360°. Vysoce citlivý kobot lehké konstrukce iiwa dosahuje přesnosti polohování ±0,1 mm s maximální nosností 7-14 kg, dle typu kobota [4]. Mobilní plošina i kobot mají zdroj energie v lithiových bateriích, které je potřeba dobíjet, což vede k jistému úskalí v samotné výdrži činnosti.

Obr. 3: KUKA KMR iiwa [5]

Podle ekonomických odhadů, obchod s kolaborativními roboty z ročního obratu, který činil v roce 2018 710 milionů amerických dolarů, postupně vzroste až do výše 12,3 miliard amerických dolarů v roce 2025 [6]. Budoucnost kolaborativní robotiky už nebude definována pouze robotickými rameny, ale i drony či mobilními roboty, které budou moci operátoři díky gestům, případně hlasovým povelům ovládat.

2.2 Aplikace kolaborativních robotů

Díky bezpečnosti a uživatelsky příznivému ovládání své pole působnosti kolaborativní robotika nenašla jen ve strojním nebo elektrotechnickém průmyslu, ale také ve zdravotnictví nebo potravinářském, chemickém a farmaceutickém průmyslu. Škála využití kobotů je téměř neomezená od ovládání jednoúčelového stroje až po masáž lidského těla. Avšak zatím své uplatnění mají koboti především v jednoduchých monotónních úkolech, jako je bin picking nebo vybrat a umístit (*pick and place*).

Svou nepostradatelnou roli hrají koboti u montážní linky. Kobot může zastávat těžké úkony, jako je například zvedání a umisťování těžkých komponentů. Toto využití má za účel zabránění zranění pracovníka, zvýšení rychlosti a zlepšení kvality daného úkolu. Následně může kobot také zcela sám zastávat obsluhu jednoúčelového stroje. Výhodou je zvětšení propustnosti a zvýšení účinnosti. Další oblastí je kontrola dílů a součástí za pomoci strojového vidění, kde je potřeba udržovat stálou kvalitu produktu. Při nanášení tmelů, lepidel a barev je možné díky použití kobotů zvýšit přesnost a omezit plýtvání [3]. Své nepostradatelné využití nacházejí koboti také při manipulací s deskami plošných spojů či elektronickými součástmi.

Své opodstatnění mají v dnešní době koboti i ve zdravotnictví, kde se především klade důraz na přesnost, preciznost, bezpečnost a hygienu. Příkladem je americká firma ADAMO ROBOT, která se zabývá výzkumem v lékařství. Při řešení problému bolesti zad využila ke svému konceptu aplikaci fyzioterapeutické metody vyvíjení trvalého tlaku na myofasciální spoušťové body a kolaborativního robota. Proces je založen na počítačovém řízení, které ovládá kobota. Kobot následně pomocí zabudovaných kamer u pacienta nalezne doktorem dříve definované body a následně svým koncovým nástrojem působí při trvalém tlaku na určená místa [7]. Švýcarská firma AOT v roce 2019 představila nejdokonalejší osteotomický nástroj, který nese název CARLO (obr. 4). Sofistikovaný software propojený s kolaborativním robotem, na který je přimontován laser. Chirurg má plnou kontrolu nad tímto nástrojem, přičemž díky využití chladného laseru může tento nástroj provádět operace kostí s nevídanou přesností [8].

Obr. 4: AOT CARLO [8]

Zastoupení kolaborativních robotů má i potravinářský průmysl. V potravinářském průmyslu je zvláště důležitá hygiena a bezpečná manipulace produktů. V italské společnosti Cascina Italia, kde musí být denně baleno přes 2,5 milionů vajec, byl použit kobot s pneumaticky ovládanými chapadly, který je zodpovědný za stohování kartonů po deseti vejcích do krabic, kde se následně nachází až 1440 vajec. Tímto způsobem kobot

balí cirka 15 000 vajec za hodinu. Prioritou u řešení tohoto problému byla naprostá bezpečnost manipulace, aby nedošlo k poškození produktů. [9]

Trendem je stále větší aplikace kolaborativních robotů ve farmaceutickém a chemickém průmyslu. V laboratořích je prioritní bezpečnost, přesnost a hygiena. Své využití zde kolaborativní roboti nacházejí především díky vlastnosti možného sdílení pracovního prostoru. V Kodaňské fakultní nemocnici v Gentofte museli pracovníci pracně manipulovat se vzorky při procesu třídění krve. K řešení tohoto problému byli použiti dva kolaborativní roboti. První vyzvedne vzorek a umístí jej ke čtečce čárových kódů (obr. 5). Následně kamera vyfotografuje barvu a po vyhodnocení kobot umístí vzorek podle barvy do jednoho ze čtyř stojanů. Druhý kobot následně vzorky umísťuje do podavače stroje na odstředění a analýzu. Tato aplikace se ukázala jako vhodné řešení, jelikož se zvýšila efektivita práce a pracovníci se mohou zabývat náročnějšími úkoly. [10]

Obr. 5: Kolaborativní robot při manipulaci se vzorkem krve [10]

2.3 Programování kolaborativních robotů

Průmysl 4.0 s sebou také přináší změnu vnímaní uživatelského softwaru. Odbourávají se neestetická, komplikovaná grafická rozhraní a namísto toho se klade čím dál tím více důraz na estetiku a jednoduchost. Proto i firmy, které se zabývají kolaborativní robotikou, se snaží tímto vyjít vstříc uživatelům a vytváří uživatelsky příjemná prostředí pro programování a ovládání kobotů.

Jedním z hlavních aspektů programování kobotů je možnost, kdy operátor ovládá kobota pomocí verbálního nebo neverbálního komunikačního kanálu. Neverbální komunikace probíhá pomocí gest či uživatelského rozhraní, kdežto verbální komunikace probíhá pomocí příkazových (klíčových) slov. Dalším aspektem je optimalizace pohybu kobota. Při programování je potřeba určit, kterým objektům se má kobot vyhnout, a které

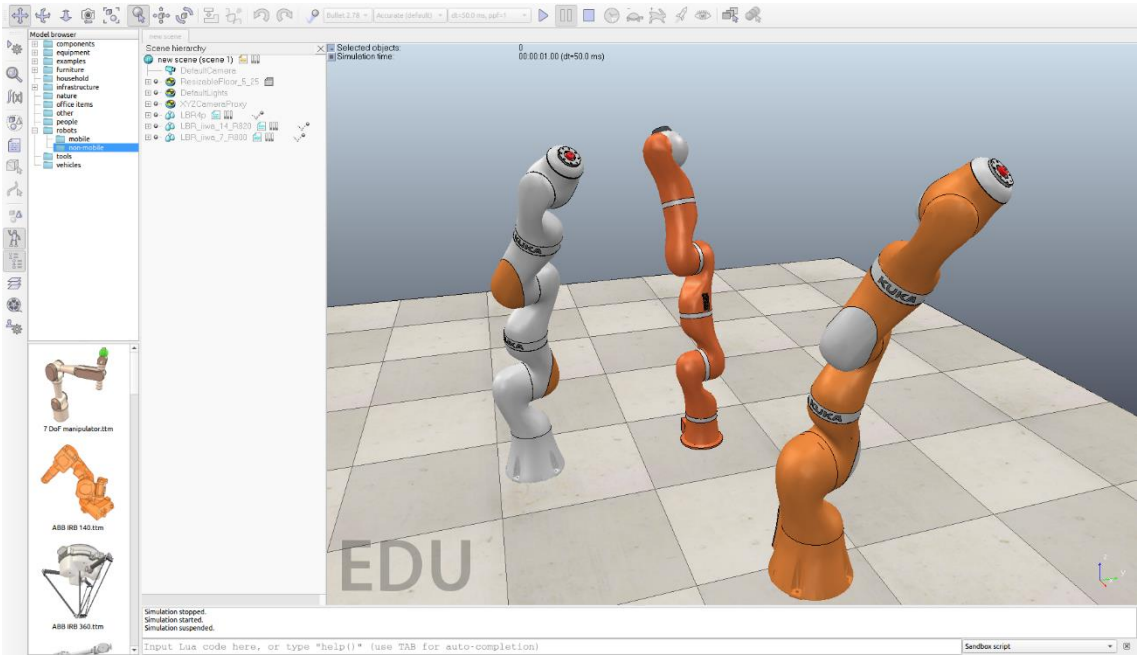
má například uchytit. Optimalizace tedy vede k minimalizaci operátorovy pracovní zátěže, spotřebované energii a času. Posledním aspektem je možnost kobota učit ať už použitím strojového učení, tak fyzického navádění kobota po trase, kterou bude následně kopírovat.

K programování samotných kobotů může docházet hned několika způsoby. První možností je fyzicky či přes ovládací panel, přičemž je kobot veden po zvolené trase a přitom operátor může definovat úkony, které má provést. Kobot zaznamenává tuto trajektorii a sekvenci akcí, kterou následně uloží do paměti, a může již repetitivně vykonávat daný úkol. Programování a řízení kobotů pomocí gest nebo hlasové komunikace většina výrobců neudává do základní výbavy svých produktů, proto je nutné tyto funkce ručně instalovat. Programování gesty vyžaduje použití kvalitní kamery nebo snímače gest, kdy programátor přiřazuje gesty akce kobotu. Hlasové programování nebo řízení má pouze svou aplikaci v místech, kde není příliš vysoký šum. K řešení lze použít Google API nebo Microsoft Speech API. Další možností je programování z pohodlí u počítače pomocí softwaru podporující programování kobotů. V drtivé většině je součásti tohoto softwaru také simulační prostředí, kde může programátor odladit případné kolize. Většina distributorů má vlastní nebo doporučuje software, který podporuje programování daného kobota. [11]

Programování pomocí intuitivního a uživatelsky příznivého ovládacího panelu poskytují téměř všichni výrobci kobotů (obr. 6). Většina těchto ovládacích panelů umožní specifikovat parametry, jako jsou především rychlost a zrychlení, ale také umožní vytvoření vzorů pohybů. Použití ovládacího panelu je velmi jednoduché a intuitivní, avšak jeho možnosti jsou značně omezené.

Obr. 6: FANUC iPendant-Touch [12, 13]

Velmi velké oblibě si přišly programy založené na CAD systému. Nejznámější programy jsou V-REP (obr. 7), Visual Components a RobotStudio od firmy ABB. Tyto programy integrují interaktivní simulační a editační prostředí. Také většinou obsahují funkce, jako je především vizualizace všech chyb nebo funkce zobrazení všech možných řešení, která jsou generována a hodnocena na základě kritérií vybraných uživatelem. Další zajímavější funkcí může být kupříkladu použití VR technologie k absolvování simulace navrhnutého výrobního stanoviště. Tuto funkci obsahuje program Visual Components či RobotStudio a využívá se zejména ke školení operátorů.



Obr. 7: Simulační prostředí V-REP

Kolaborativní roboty lze také programovat výhradně jen pomocí programovacích jazyků. Kuriozitou na poli průmyslových robotických jazyků je to, že každý výrobce si vytvořil vlastní jazyk, což vedlo k jistému bezvládí, a ne příliš uživatelskému komfortu. Například firma Kuka vytvořila programovací jazyk KRL (*Kuka Robot Language*), nebo firma Fanuc používá programovací jazyk Karel a firma Universal Robots URScript. Jistým standardizujícím elementem pro programátory se díky své neuvěřitelné oblibě stal ROS Industrial, který umožňuje programování robotů pomocí jazyku python a C++.

2.4 Bezpečnost kolaborativních robotů

Bezpečnost u kolaborativních robotů je naprostou prioritou, proto se u kobotů aplikuje mnoho různých bezpečnostních funkcí a senzorů. Počínaje instalací měkkého materiálu na klouby a samotné části kobota, až po využití kamer a laserů. Některé bezpečnostní funkce jsou aplikovány tak, aby pouze zpomalily kobota a jiné ho bezprostředně zastaví. Jedním ze základních řešení je kupříkladu využití rychlostních a momentových senzorů (obr. 8). Tyto senzory signalizují sílu působící na paži kobota. Pokud dojde k překročení

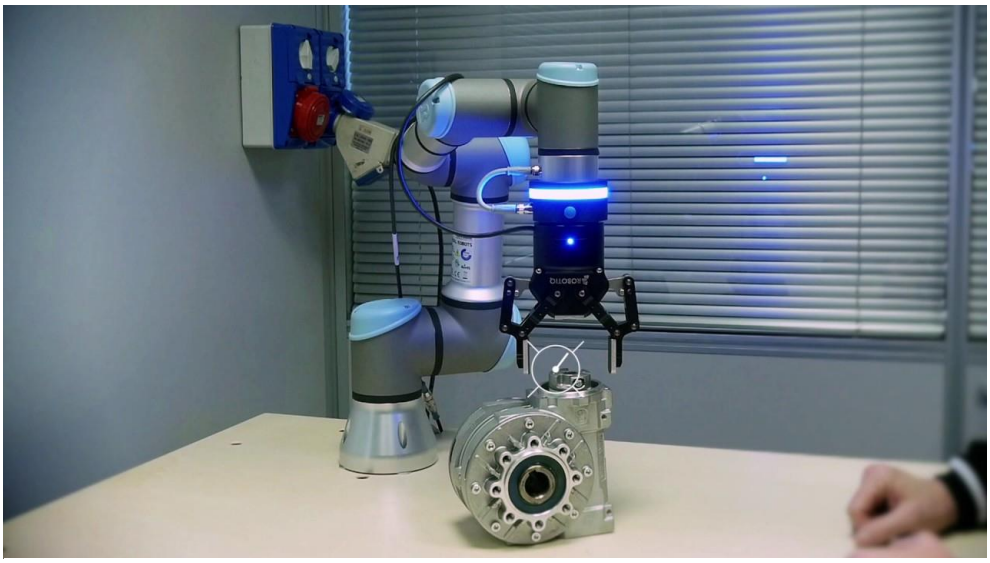
nastavené meze, kobot okamžitě zastaví svůj pohyb. Většinou jsou tyto senzory zabudovány přímo v těle kobota. Dalším ochranným prvkem jsou různé druhy snímačů, mohou být například ultrazvukové, optoelektronické nebo kapacitní. Tato aplikace už závisí na samotném výrobci (obr. 9). V dnešní době je možné využít jako přídavný bezpečnostní prvek kamery s funkcí zpracování obrazu, které mohou člověka detekovat a vyhnout se tak kolizi. [11]

Obr. 8: AIRSKIN pro kolaborativního robota UR10 [14]

Výrobci zabývající se vývojem a výrobou kolaborativních robotů dodržují normu ISO 10218-1 a specifikaci ISO/TS 15066. Aby se zajistilo, že všechny stroje jsou klasifikovány jako kolaborativní a splňují určité bezpečnostní požadavky, byla v roce 2016 představena nová specifikace norem ISO 10218-1 a 10218-2, které byly původně publikovány v roce 2006. Specifikace ISO/TS 15066 není standard, jde nýbrž jen o informace a doporučení pro konstruktéry robotů i robotizovaných pracovišť. Podle specifikace musí mít kolaborativní roboti bezpečnostní prvky, jako je bezpečnostní monitorované zastavení, sledování rychlosti a vzdálenosti, ruční navádění, omezení rychlosti a výkonu. Jedním z podstatných bodů specifikace je ten, že pokud operátor přijde do kontaktu s robotem, nesmí mít za následek bolest nebo zranění.

Zejména funkce využívána na výměnu nástroje, údržbu nebo ergonomii je bezpečnostní monitorované zastavení. Pokud dojde k aktivaci funkce je možné kolaborativního robota ovládat manuálně. Dalším bezpečnostním prvkem je monitorování rychlosti a vzdálenosti. Díky této funkci je možné udržovat dostatečnou vzdálenost pracovního prostoru mezi obsluhou a kobotem. Kobot je řízen tak, aby se pohyboval bezpečnou rychlostí a vyvozoval bezpečnou sílu, pokud by došlo ke kontaktu s obsluhou. [15]

Jedním z rizik možného nebezpečného chování může být kobot manipulující s ostrým nebo nebezpečným nástrojem, jako je například vrtačka. Proto se provádí mnoho progresivního výzkumu za cílem vytvoření absolutní bezpečnosti. Výzkum se zaměřuje především na oblasti využití umělé inteligence, schopnosti plného vnímání pracovního prostoru kobota a simulace pohybů, procesů pomocí VR technologie.

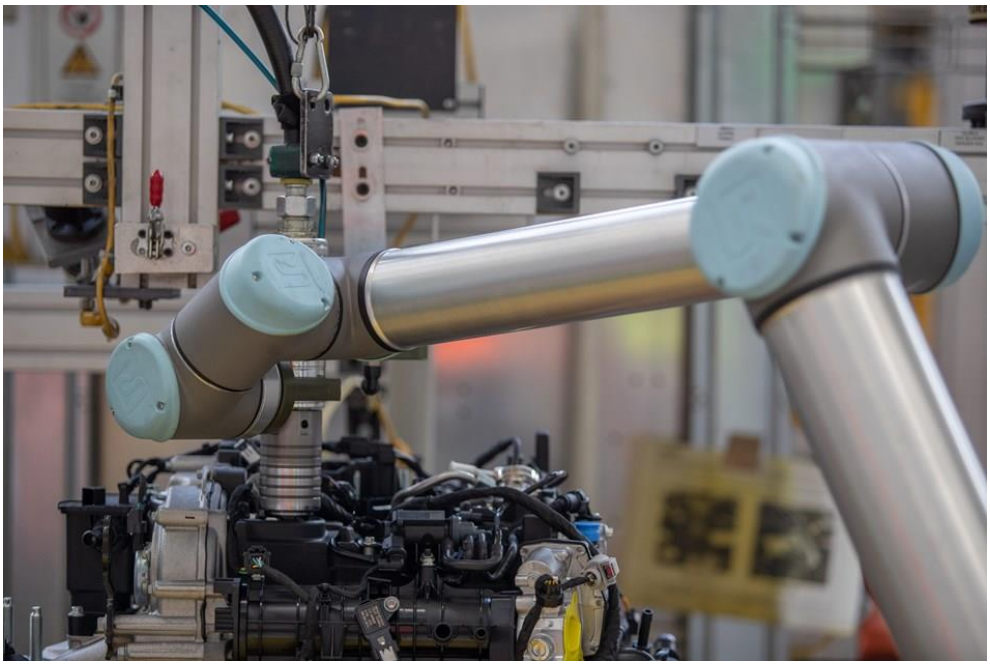


Obr. 9: Bezpečnostní prvek YOUring kolaborativního robota UR3 [16]

3 UNIVERSAL ROBOTS

Zásadní vliv na poli kolaborativní robotiky má dánská firma Universal Robots. Tato firma má své hlavní sídlo ve městě Odense a stala se v dnešní době největším globálním distributorem kobotů. Firma byla založena v roce 2005, přičemž zakladateli jsou Esben Østergaard, Kasper Støy a Kristian Kassow. Na přelomu 20. a 21. století trh ovládaly především velká, těžká a drahá robotická ramena, která si mohly dovolit pořídit jen větší podniky. To vedlo k impulzu vytvoření myšlenky zpřístupnění robotických ramen i do menších podniků. Proto byl v roce 2008 vypuštěn na dánský a německý trh první kobot společnosti s označením UR5. [17]

Své umístění našli kolaborativní roboti i v továrnách velkých gigantů, jako je na kupříkladu Ford Motor Company. Na montážní lince ve městě Craiova v Rumunsku byli koboti UR10 použiti k mazání váčkových hřídelí, plnění olejů do motorů a provádění inspekcí bloku motorů po naplnění provozními kapalinami pomocí UV světla a speciální kamery od firmy Cognex (obr. 10). Cílem nebylo zcela nahrazení operátorů výroby, ale eliminace možných chyb u této repetitivní a náročné práce. Díky rychlé integraci se zvýšila rychlost produkce, efektivita a operátoři výroby se mohou soustředit na klíčové úkoly. [18]



Obr. 10: Kolaborativní robot UR10 při plnění oleje [18]

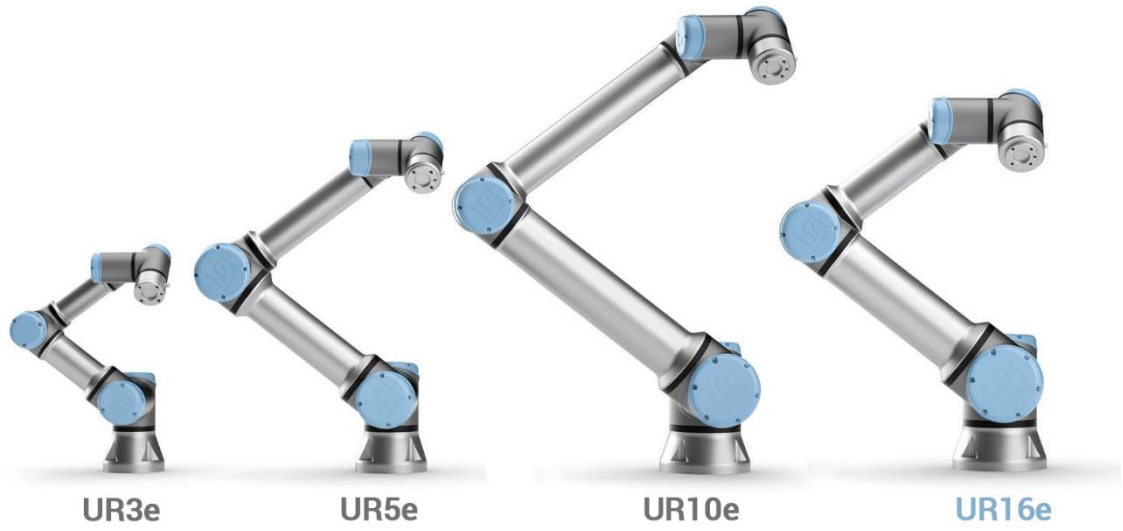
Své nepostradatelné využití mají kolaborativní roboti firmy Universal Robots i v menších podnicích jako je Linaset, sídlící ve městě Budišov nad Budišovkou v České republice. Kobot byl aplikován do výrobní fáze tryskání komponent. Před využitím kobota museli pracovníci výlisky tryskat ručně z obou stran, což vedlo k velmi obtížné práci jak z hlediska času, tak zdrojů. Podmínky pro pracovníky také nebyly příliš příznivé, jelikož se jedná o velmi prašné, hlučné prostředí i fyzicky náročnou práci. Řešením byla

instalace kobota UR5, který má připevněný aplikátor tryskacího media a sdílí pracovní prostor s operátory výroby. Kobot provádí tryskání a operátoři jen manipulují s výlisky. Toto řešení má za následek zvýšení rychlosti procesu, přesnosti a bezpečnosti. [19]

Svou pozici světové špičky v kolaborativní robotice si firma Universal Robots udržuje nízkou cenou kobotů s nízkými náklady na údržbu, kvalitním zpracováním, jednoduchým programování, vysokou bezpečností a flexibilním využitím. Menší podniky především oceňují lehkou konstrukci, jelikož se roboti snadno dají přenášet na různá pracovní místa a přeprogramovat na odlišné úkoly.

3.1 Produkty společnosti

První generace kolaborativních robotů měla označení CB Series, do této série patří koboti UR3, UR5 a UR10. V roce 2018 došlo k pozměnění celé generace kobotů řadou e-Series, přičemž hlavní změnou bylo přidání bezpečnostních funkcí, také došlo k zvětšení přesnosti opakovatelnosti polohy, malé zvětšení hmotnosti, zmenšení hluku a spotřeby elektrické energie [17]. Firma v současné době nabízí celkem 4 typy kolaborativních kobotů, kdy každý kobot má své aplikační opodstatnění. Nejnovějším přírůstkem k nové generaci kolaborativních robotů je kobot UR16e. Firma také nabízí velkou škálu doplňujících funkcí pro koboty od softwaru, až po přídavné ochranné prvky. Tyto produkty jsou výsledkem spolupráce firmy Universal Robots s menšími firmami. Tyto aplikační pomůcky, přístroje nebo programy, jsou označovány jako UR+ Solutions.



Obr. 11: Kolaborativní roboti z řady e-Series [20]

Generace kobotů e-Series (obr. 11) má stupeň krytí 54, tudíž jsou chráněny před nebezpečným dotykem jakoukoliv pomůckou, před vniknutím všech cizích předmětů, prachu částečně a také před stříkající vodou ze všech úhlů [21]. Montáž kobotů je možná jak ve vertikálním směru, tedy na stěnách, tak i v horizontálním směru na podlahu či strop. Všechny koboty lze ovládat pomocí ovládacího panelu s grafickým rozhraním UR

Polyscope. Certifikace kolaborativního chování je zajištěno dle normy EN ISO 10218-1. [22]

1) UR16e

- užitečné zatížení 16 kg
- maximální rychlost 1000 mm/s
- dosah 900 mm
- opakovatelnost pohybu ±0,05 mm
- pracovní rozsah všemi klouby 360°
- hlučnost maximálně 65 dB
- hmotnost samotného kobota 33,1 kg

2) UR10e

- užitečné zatížení 10 kg
- maximální rychlost 1000 mm/s
- opakovatelnost pohybu ±0,05 mm
- dosah 1300 mm
- pracovní rozsah všemi klouby 360°
- hlučnost maximálně 65 dB
- hmotnost samotného kobota 33,5 kg

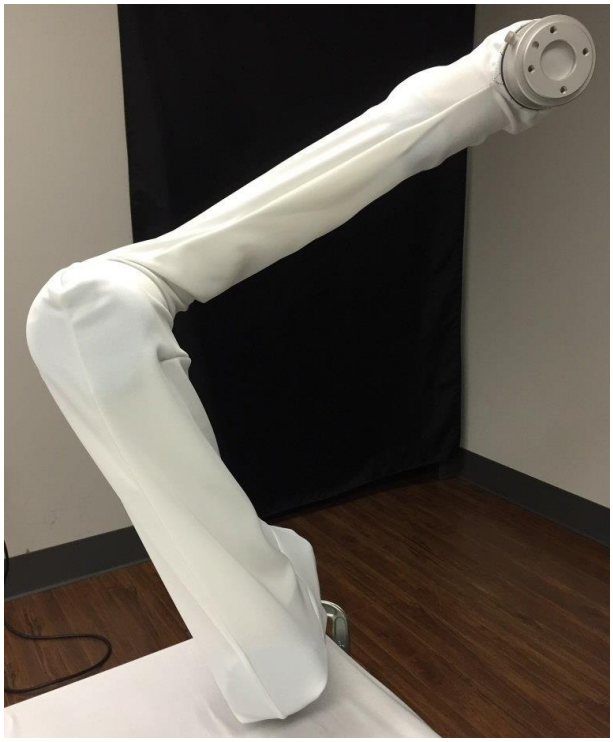
3) UR5e

- užitečné zatížení 5 kg
- maximální rychlost 1000 mm/s
- opakovatelnost pohybu ±0,03 mm
- dosah 850 mm
- pracovní rozsah všemi klouby 360°
- hlučnost maximálně 65 dB
- hmotnost samotného kobota 20,6 kg

4) UR3e

- užitečné zatížení 3 kg
- maximální rychlost 1000 mm/s
- opakovatelnost pohybu ±0,03 mm
- dosah 500 mm
- pracovní rozsah všemi klouby 360°
- hlučnost maximálně 60 dB
- hmotnost samotného kobota 11,2 kg [22]

UR+ Solutions obsahuje doplňky, jako je například řízený klimatický robotický plášť od firmy Robosuit (obr.12), který umožňuje používat kobota ve výrobních podmínkách, které nesplňují provozní požadavky zejména z důvodů velmi vysoké teploty nebo vlhkosti [23]. Dalším doplňkem může být AIRSKIN od firmy Blue Danube Robotics, který pokrývá koboty UR16e, UR10 a UR5 měkkými podložkami a senzory citlivými na tlak. Při kolizi je do řídicí jednotky vyslán signál, aby se aktivovala funkce bezpečného zastavení [24]. Softwarovým doplňkem je kupříkladu cloudově založený program na návrh robotických buněk, Vention's 3D MachineBuilder. Tento program umožňuje uživatelům vytvořit přímo na míru robotické buňky určené pro koboty firmy Universal Robots [25].



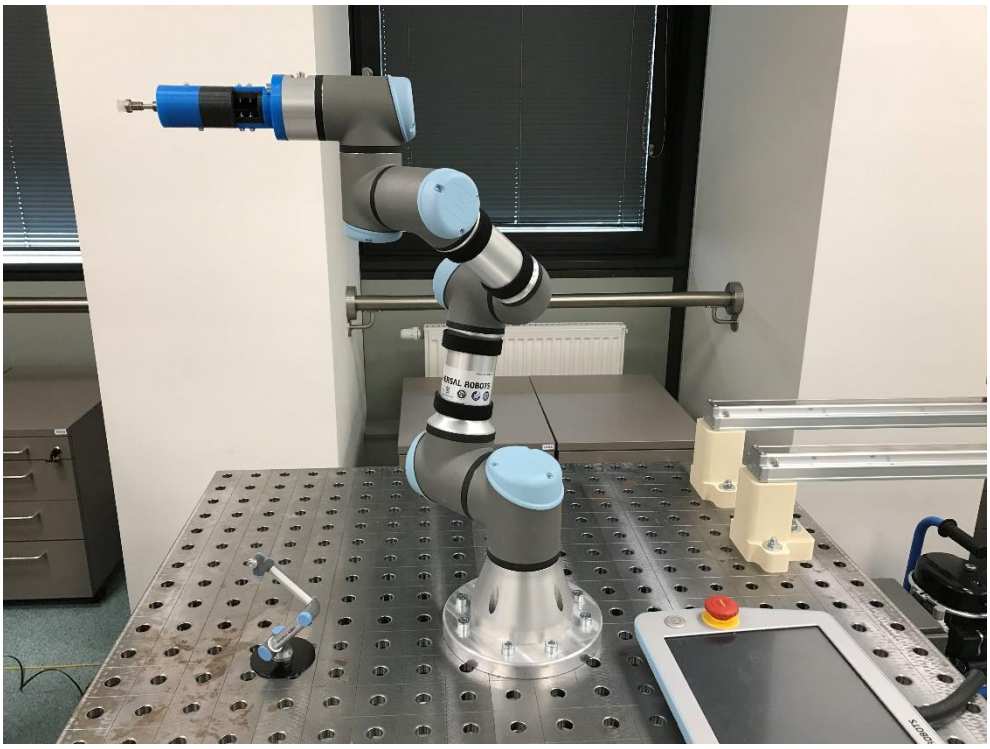
Obr. 12: Robotický plášť pro kolaborativního robota UR5 od firmy Robosuit [26]

3.2 Kolaborativní robot UR3

Ultralehký kolaborativní robot s označením UR3 z generace CB byl vydán na trh v roce 2015. Využitím hliníku a polypropylenu je hmotnost pouhých 11 kg, která nemá téměř žádnou konkurenci. Tento šestiosý kobot má dosah až 500 mm, přitom se může otáčet všemi klouby o 360°. Tato vlastnost je jednou z hlavních výhod oproti konkurenčním kolaborativním robotům. Užitečné zatížení kobota jsou 3 kg s přesností opakovatelnosti polohy ±0,1 mm. Kobot může pracovat v rozsahu 0-50 °C, přičemž při využití stálé vysoké rychlosti, která je 1000 mm/s, se povolená maximální okolní teplota snižuje. Uživatelé mohou instalovat robota jak v horizontálním směru na strop nebo podlahu, tak i ve vertikálním směru na stěnu. Další vlastností, kterou uživatelé a operátoři uvítají, je nízká hlučnost, která se pohybuje cirka kolem hodnoty 70 dB. Stupeň krytí u kobota UR3

je 64, tudíž podle standardu je zajištěno krytí před nebezpečným dotykem jakoukoliv pomůckou, vniknutím prachu a také je chráněno proti stříkající vodě ze všech úhlů [21]. Bezpečné kolaborativní fungování zaštiťuje certifikace TÜV NORD. Hodnoceno a testováno UR3 je dle normy EN ISO 13849:2008 PL d. Ke kobotu je možné se připojit přes 2 digitální vstupy, 2 digitální výstupy a 2 analogové vstupy. Aby mohlo dojít k fungování a programování robota, je potřebný kontrolér a ovládací panel v podobě 12“ dotykového displeje, kde je instalováno grafické uživatelské rozhraní UR Polyscope. Komunikace je možná přes průmyslové sběrnice Modbus TCP, Profinet a EthernetIP nebo komunikace v počítačové síti přes sadu protokolů TCP/IP. [27]

Kolaborativní robot UR3 patří mezi nejlepší ve své třídě (obr. 13). Přímý konkurent, co se týče konstrukce a vlastností, je kobot s označením i3 od firmy AUBO. Rozdíl mezi koboty je například v maximální využitelné rychlosti, kterou má kobot UR3 o 900 mm/s nižší, přitom je nižší i maximální dosah kobota od firmy Universal Robots o 125 mm. Oba mají stejnou maximální nosnou hmotnost, která nabývá hodnoty 3 kg. Výhodou kobota UR3 je nižší samotná hmotnost o 4,5 kg oproti kobotu i3 a také možnost otáčet se všemi klouby o 360° [28]. Dalšími konkurenty v oblasti kolaborativních robotů s malou nosností je kobot Panda Franka Emika, Fanuc CR-4iA nebo TM5-900 od firmy Techman Robot.



Obr. 13: Kolaborativní robot UR3 v laboratoři

4 ROBOTICKÝ OPERAČNÍ SYSTÉM

Nejpopulárnější framework na poli robotiky se stal jednoznačně ROS (*Robot Operating System*), v českém překladu Robotický operační systém. ROS není framework ani operační systém v tradičním smyslu pojetí, nýbrž se jedná o meta operační systém (*meta operating system*), který poskytuje výše strukturovanou komunikační vrstvu nad hostitelským operačním systémem heterogenního výpočetního clusteru. Tak, jako konvenční operační systém, meta operační systém ROS poskytuje velkou škálu služeb, jako je hardware abstraction lawyer, nízko úrovňové řízení zařízení (*low-level device control*), také poskytuje funkce jako zasílaní zpráv (*message-passing*) mezi procesy a správce balíků (*package management*). Jednou z hlavních částí jsou nástroje a knihovny pro psaní a spouštění kódu, ačkoliv je možné integrovat ROS s kódem v reálném čase, nejedná se o real-time framework [29]. Jelikož robotika je stále dynamicky se rozvíjející vědní obor, co se týče rozsahu a zaměření, vývoj a psaní programů pro roboty může být obtížné z důvodů odlišnosti hardwarového vybavení. Tento rys vede na komplikované repetitivní využití již napsaných programů. Proto se tvůrci ROS snažili tuto záležitost odstranit a hlavním cílem bylo vytvoření systému, který lze integrovat na velmi širokou škálu aplikací a výzkumu v oblasti robotiky.

ROS je obrovský projekt, který se začal formovat na univerzitě ve Stanfordu v USA. Na Stanfordské univerzitě se už kolem roku 2000 tvořila různá úsilí k vytvoření vlastních prototypů flexibilních dynamických systémů určených pro použití v robotice. V roce 2007 Willow Garage, vizionářský robotický inkubátor, poskytl značné zdroje na rozšíření těchto konceptů. Mnoho vědců a odborníků z oblasti robotiky přispěli svým časem a znalostmi k základním nápadům vytvoření ROS i k základním softwarovým balíčkům. Tento software byl vyvíjen pod permisivní licencí BSD (*Berkeley Software Distribution*) a postupně se stal široce používanou platformou v robotické výzkumné komunitě a následně i průmyslu. ROS takový, jaký známe dnes, spatřil světlo světa až v roce 2009 pod označením ROS 0.4 mango tango. [30]

4.1 (Ne)průmyslové využití

ROS je ve velké míře využíván především výzkumnou komunitou, přitom tuto technologii lze použít v nejrůznějších oblastech počínaje vesmírným průmyslem, až po strojní průmysl. Například NASA provozuje ROS ve vesmíru na robotu pod označením R2 (Robonaut 2) na mezinárodní vesmírné stanici. R2 je humanoidní robot, který se může se svými robotickými rameny pohybovat až rychlostí 2000 mm/s, přitom jeho maximální nosnost je 18 kg. Tento sofistikovaný robot integruje jak dotykové senzory na špičkách prstů, tak dalších cirka 350 senzorů, které jsou ovládané pomocí 38 PPC (*PowerPC*) mikroprocesorů. Princip fungování R2 je navržen tak, aby se v prvé řadě nastavily úkoly, které uživatel chce provést, a poté robot autonomně s periodickými kontrolami stavu dané úkoly provádí. [31]

Další oblastí je autonomní řízení osobních automobilů, které se v posledních letech stává naprostým trendem. Přední automobilky jako je kupříkladu BMW, Nissan a Tesla investují a podporují firmy, které se touto problematikou zabývají. Část firem postavila své řešení na ROS z důvodů potřebných nástrojů pro snadný přístup k datům ze senzorů, také jejich následnému zpracování a generování vhodné zpětné vazby pro motory a další akční členy. Jednou z nevýhod je, že programy musí být vytvořeny naprosto správně, aby nedošlo k výpadku softwarových komponent, které mají na starost veškerou koordinaci mezi procesy. Další nevýhodou je, že samotný ROS není bezpečný, proto většina firem musí své programy zaštítit kyberbezpečností. [32]

Své využití najde ROS i u létajících zařízení, jako jsou autonomní drony (obr. 14). Jedná se o využití dronů s kamerami ať už pro účely vedení revize ve skladu, nebo přepravu součástek. Příkladem může být dron, který integruje kameru, přičemž ta při pohybu snímá QR kódy na výrobcích v regálech a následně se data posílají do skladové databáze. Využitím ROS přináší své úskalí v podobě možnosti ovládání pouze jednoho dronu.

Obr. 14: ROS drones [33]

Zásadní intervencí do oblasti průmyslové robotiky bylo vytvoření projektu ROS Industrial. Tento otevřený softwarový (*open-source*) projekt byl zkonstruován za účelem využití pokročilého robotického softwaru ROS v průmyslu. Jedním ze základních kamenů tohoto projektu je poskytnout uživatelům jistou standardizaci ovladačů pro roboty, odbourat proprietární systémy, a naopak vytvořit co největší komunitu vývojářů, kteří mohou řešit problémy společně. Díky této komunitě se zrychlil proces testování, následného odhalení a řešení problémů, což má za následek vznik většího počtu inovací na poli průmyslové robotiky. Podporovatelé projektu jsou například technická univerzita Delft, nadnárodní korporace ABB a Microsoft, automobilky BMW a Volvo Group a firmy Universal Robots nebo Wolf Robotics. ROS Industrial v dnešní době lze díky své rozmanitosti využít na různé aplikace řízení robotů s jejich koncovými nástroji. [34]

4.2 Distribuce Robotického operačního systému

Aby byla zaručena aktuálnost ROS k hostitelským operačním systémům, je potřeba vytvářet a udržovat distribuce ROS (obr. 15). Distribuce jsou verze balíků ROS, které jsou primárně zaměřeny na kompatibilitu s verzemi operačních systémů Linux a v nejnovější distribuci i k Windows. Účelem distribucí je vytvořit vývojářům stabilní systém, na který mohou stavět svá řešení [35]. Box Turtle byla první oficiální vydanou distribucí ROS v roce 2010. Tato distribuce byla především navržena na operační systém Linux Ubuntu Hardy, ale také byla možná instalace na Linux Red Hat, Debian a Mac OS X s jistými omezeními. Distribuce Box Turtle při svém vydání měla již poměrně stabilní základnu robotických knihoven, nástrojů a ovladačů [36]. Jednou z nejpoužívanějších a nejstabilnějších distribucí je stále Kinetic Kame, která se udržuje od roku 2016. Distribuce Kinetic Kame je především kompatibilitou cílená na Linux Ubuntu Willy a Xenial, přitom je také možné instalovat na operační systém Linux Debian, Gentoo a Mac OS X za cenu omezené stability a kompatibility [37]. Jelikož Linux Ubuntu Xenial přestane být ve výhledové době aktualizován a podporován, postupně vývojáři přecházejí na nejnovější stabilní verzi distribuce, která nese označení Melodic Morenia. Distribuce byla vydána v roce 2018 a je podporována především pro operační systém Linux Ubuntu Artful a Bionic, Debian Stretch, ale také pro Windows 10. Distribuci je možné instalovat i na operační systém Arch Linux, Gentoo a Mac OS X [38].

Obr. 15: Plakáty ROS distribucí [35]

Linuxové distribuce založené na Arch Linux (obr. 16) se jeví jako velmi schůdná cesta k vhodnému preferovanému hostitelskému operačnímu systému. Oproti ostatním distribucím naskýtá odlehčený a jednoduše přizpůsobitelný, tudíž i výkonnější systém. Instalace a následné použití ROS distribuce Melodic Morenia z ArchWiki bylo testováno na linuxové distribuci Manjaro Gnome. Výsledkem testování je potvrzení faktu, že hlavní výhodou je možnost upravit si systém specifickým preferencím, a to bez redundantních součástí, avšak handicapem je stále menší komunita uživatelů, což vede na neudržování aktuálních balíků ROS, které jsou poměrně nestabilní.

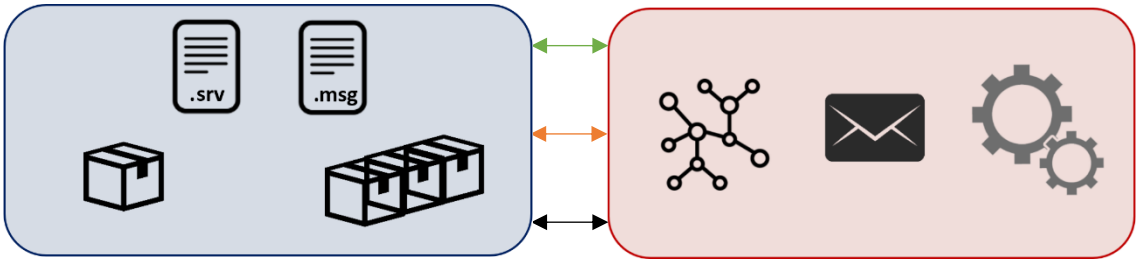
Obr. 16: Logo Arch linux [39]

4.3 Filesystem Level

Tak jako operační systém i ROS má svou strukturu, kterou lze rozdělit do 3 úrovní. ROS ve svém vývoji prodělal i změnu samotných úrovní a v nové, zatím používané podobě, úroveň souborového systému (*filesystem level*) spravuje:

- Balíky (*packages*) tvoří samotný základ veškeré organizace softwarového rozhraní v ROS, jedná se o nejelementárnější položku, se kterou lze interagovat. Balík integruje především ROS runtime procesy (*processes*), také označovány jako uzly (*nodes*), ROS závislé knihovny (*dependent library*), datové sady (*datasets*) a konfigurační soubory (*configuration files*).
- Metabalíky (*metapackages*) představují objemnější speciální balík, který agreguje více vzájemně propojených balíků.
- Manifest balíku (*package manifests*) je soubor, který je uložen ve formě *jmenobaliku.xml*, přičemž popisuje základní metadata o balíku, jako je kupříkladu popis, licence, autor, URL (*Uniform Resource Locator*), v neposlední řadě build, run, test závislosti a pokud je potřeba, také metainformace o exportu.
- Uložiště (*repositories*) s příchodem webových služeb, jako je zejména GitHub nebo GitLab, které umožňují verzování projektů, i ROS přispěl k tomuto uživatelskému komfortu a vývojáři mohou pomocí VCS (*Version Control System*) sdílet a vydávat verze skrz nástroj *catkin bloom*. Omezením uložišť je, že lze integrovat pouze jeden balík.
- Typ zpráv (*message types*) představuje popis definující datovou strukturu zprávy posílané v ROS, která je uložená v podadresáři balíku.

- Typ služeb (*service types*) představuje popis definující datovou strukturu zprávy typu požadavek a odpověď v ROS, která je uložená v podadresáři balíku. [40]



Obr. 17: Úroveň souborového systému úzce souvisí s grafovou výpočtovou úrovní

4.4 Computation graph Level

Hlavní úrovní je grafová výpočtová úroveň *(computation graph level*), která zpracovává veškerá data v peer-to-peer síti procesů a stará se o správný chod ROS. Základní jednotkou této sítě jsou:

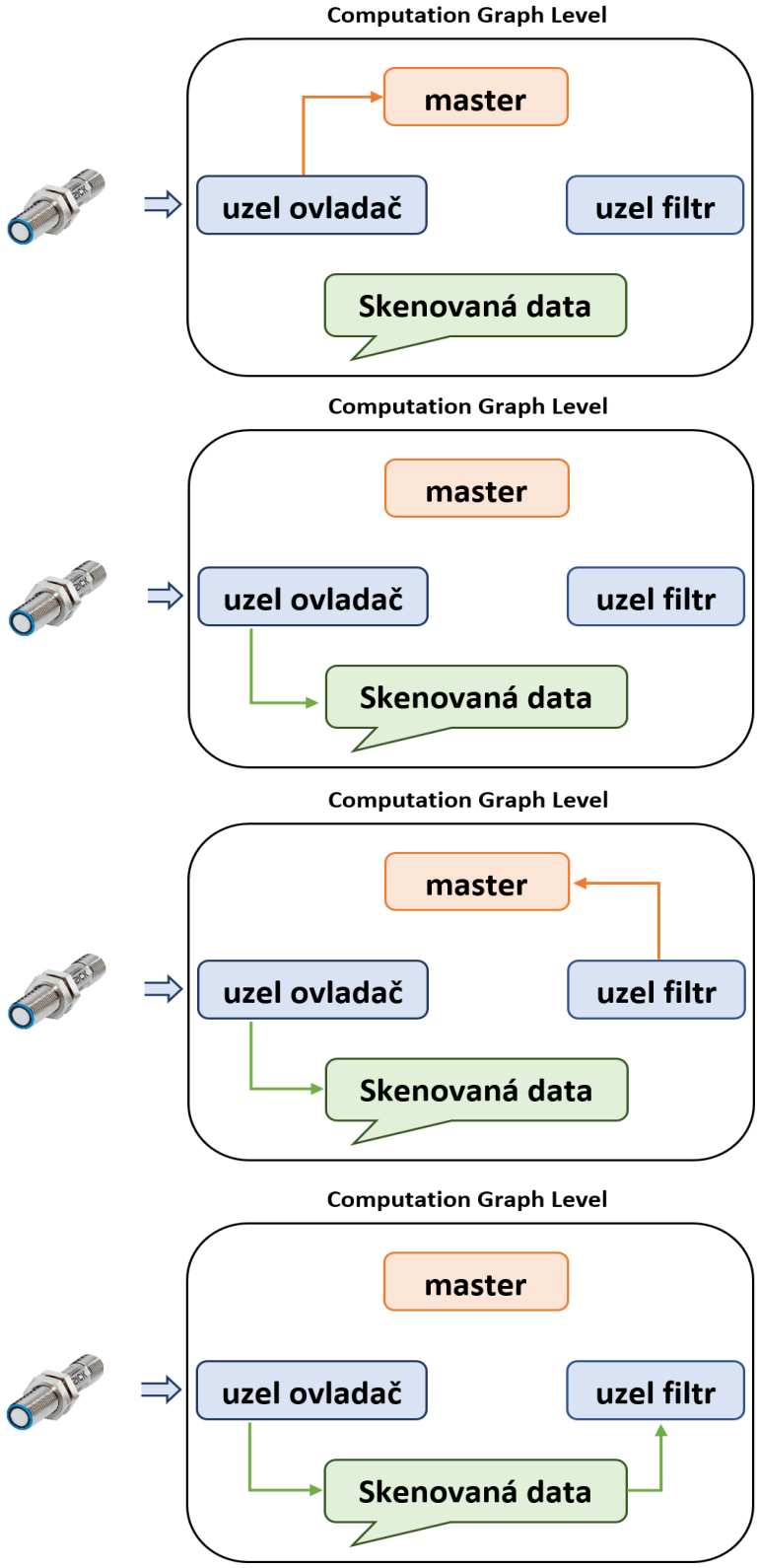
- Uzly (*nodes*) jsou procesy, které provádějí výpočty, přičemž spolu mohou vzájemně komunikovat pomocí témat, služeb a parametru serveru. Tato architektura s sebou přináší hned několik vlastností, jako je například odolnost proti chybám, kde jsou uzly izolovány a lze je spouštět, restartovat nebo ukončovat v jakémkoliv pořadí, aniž by došlo k celému výpadku systému. Další vlastností je menší složitost oproti monolitickým systémům (*monolithic systems*). Zdrojové jméno v grafu (*graph resource name*) určuje všem spuštěným uzlům jedinečné označení v celém systému, kdy každý uzel má také označený svůj typ, který zjednodušuje proces odkazování. Příkladem použití uzlů může být robotický kontrolní systém dronu, který používá jeden uzel k obstarávání informací ze senzorů, další uzel ovládá DC motory, v neposlední řadě další uzel obstarává lokalizaci a poslední se stará o plánování trasy. Uzly je možné programovat skrz uživatelské knihovny rocpp (*C++ library for ROS*) nebo rospy (*python library for ROS*). [41]
- Zprávy (*messages*) umožňují komunikaci mezi uzly, přitom mají striktně danou datovou strukturu. Zprávy mohou obsahovat libovolně vnořené struktury a pole s datovými typy od integerů až po konstanty. Tyto zprávy jsou ukládané v podadresáři balíku v souborech s příponou .msg, což jsou jednoduché textové soubory pro specifikaci datové struktury zprávy. [40]
- Master zajišťuje registraci a vyhledávání uzlů v grafové výpočtové úrovni. Hlavní rolí master je primárně umožnit jednotlivým uzlům najít jeden druhého, přitom ukládá informace o registraci témat a služeb jednotlivých uzlů. Uzly tedy komunikují s master, kdy vždy nahlásí své registrační informace a následně mohou publikovat či přijímat informace od jiných uzlů podle

potřeby. Master poskytuje pouze vyhledání informace, přitom může provést zpětná volání do uzlů, pokud se registrační informace změní, což umožňuje dynamické chování. Další funkcí je umožnit parametr serveru. [42]

- Parametr serveru (*parameter server*) je sdílený, mnohotvárný slovník, který je přístupný prostřednictvím síťových rozhraní API. Je součástí a pracuje uvnitř master, přitom umožňuje, aby byla data uložena klíčem v centrálním umístění. Uzly používají tento server k ukládání a načítání parametrů v runtime, přičemž je globálně viditelný, aby mohly nástroje zkontrolovat stav konfigurace systému a případně jej poupravit. [43]
- Témata (*topics*) je označení, které se používá k identifikaci obsahu ve zprávě a umožňuje směrování zpráv přes transportní systém s anonymní sémantikou vydavatel/odběratel. Funkce téma se dá popsat následovně, pokud uzel publikuje a odešle zprávu s daným tématem, následně uzel, který má zájem o data, se stane odběratelem. Může existovat současně více uzlů odběratelů nebo vydavatelů jednoho téma. Cílem této funkce je jednoznačnost. ROS v současné době podporuje přenos zpráv na bázi TCP/IP neboli TCPROS a UDP neboli UDPROS. TCPROS je výchozí přenos používaný v ROS, jelikož UDPROS se vyznačuje ztrátovostí přenosu a nízkou latencí, tudíž je vhodnější pro úkoly, jako je teleoperace. [44]
- Služby (*services*) jsou definovány dvojicí speciálních zpráv, které zprostředkovávají požadavek/odpověď. Funkce služby funguje na principu, kdy uzel poskytuje službu a klientský uzel ji může využít. Uzel, který chce využít službu, odešle zprávu s požadavkem a čeká na odpověď. Pokud dojde ke spojení, klient může navázat trvalé připojení k uzlu, který poskytuje službu, což zprostředkuje vyšší výkon, avšak za cenu menší odolnosti vůči změnám uzlu poskytovatele služeb. Služby jsou definovány pomocí textových souborů, které mají příponu .srv. [45]
- Bags jsou formátem pro ukládání a přehrávání dat ze zpráv. Bags slouží k ukládání dat, například ze senzorů, které lze obtížně sbírat, ale jsou nezbytné pro vývoj a testování algoritmů. Nástrojem rosbag, lze vytvářet bags, které mohou odebírat jedno nebo více témat a následně ukládat data ze zpráv. [46]

Demonstrovat činnost této komplexní úrovně ROS lze kupříkladu na ultrazvukovém senzoru (obr. 18). V prvé řadě ROS spustí uzel ovladač, který obstarává daný typ ultrazvukového senzoru. Uzel ovladač oznámí master, že bude publikovat skrz zprávy data ze senzoru na téma skenována data. Uzel ovladač publikuje tyto data, ale nikdo je neodebírá, tudíž v systému nejsou skutečně nikam posílána. Naprogramovaný další uzel filtr, který má za úkol tato data zpracovávat, oznámí master, že se přihlásí k odběru téma skenovaná data a následně začnou spolu uzly automaticky komunikovat. Uzel ovladač publikuje data a uzel filtr je odebírá pomocí zpráv s daným tématem. Je možné také přidávat další uzly, které budou obstarávat úplně stejnou funkci, akorát budou předdefinované na jiné téma. Další uzel koordinátor potřebuje již zpracovaná data od uzlu

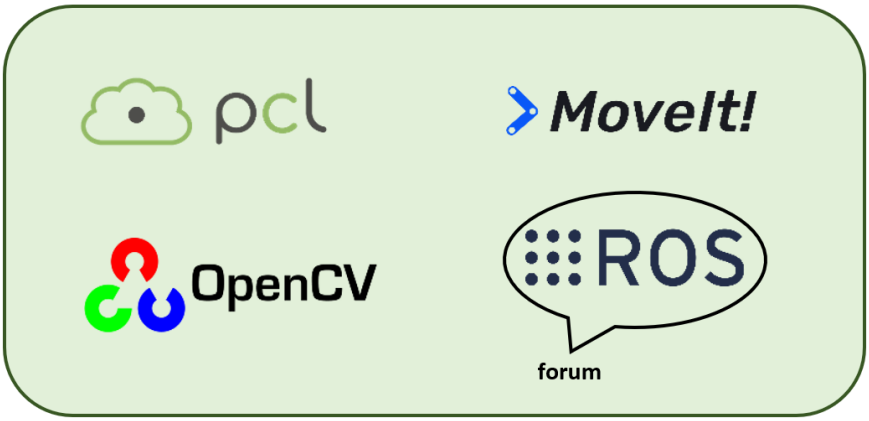
filtr, tak oznámí master požadavek služby od uzlu filtr a následně čeká na odpověď. Uzel filtr, který poskytuje danou službu, následně pomocí master naváže spojení a poskytne službu uzlu koordinátor. [40]



Obr. 18: Demonstrace činnosti grafové výpočtové úrovně

4.5 Community Level

Jednou z nezbytných úrovní je úroveň komunity (*community level*), která vytváří ROS, tak úspěšný framework v oblasti robotiky (obr. 19). Díky silné komunitě vývojářů lze již integrovat do ROS spoustu přídavných nástrojů a knihoven, které umožňují větší programátorský komfort a potřebné prostředky k řešení daných problémů. Například OpenCV, jedna z nejznámějších knihoven, která především slouží jako nástroj k řešení problémů strojového vidění nebo knihovna PCL (*Point Cloud Library*), která se využívá především ke zpracování 3D obrazu. Dále to jsou komunikační kanály a fóra, kde jsou popsány veškeré funkce a problémy, které mohou jednotliví vývojáři řešit společně. Příkladem je zejména ROS Wiki a ROS Answers. [40, 47]



Obr. 19: ROS úroveň komunity

4.6 Platforma MoveIT

Platforma MoveIT je jedním ze základních kamenů řízení robotických ramen pomocí ROS. Platforma disponuje širokou škálou zásuvných modulů, které se prolínají a umožňují uživatelům přizpůsobit si je přímo na míru jejich řešení. Platforma poskytuje především funkci plánování pohybu (řešení inverzní kinematiky) pomocí plánovacích algoritmů. Další nepostradatelnou funkcí platformy je kupříkladu detekce kolizí nebo vyhýbání se překážkám. Platforma integruje také řídící funkci hardwarových řadičů. Často se stává, že k provedení pohybu lze použít více řadičů, přednost dostane ten řadič, který je označen jako preferovaný nebo výchozí. Tato funkce má také manažera, který koordinuje přepínání mezi řadiči. Geometrie, kinematika a další informace o robotu jsou popsány v souborech URDF[1]. Platforma načte URDF soubory, vytvoří stavový prostor

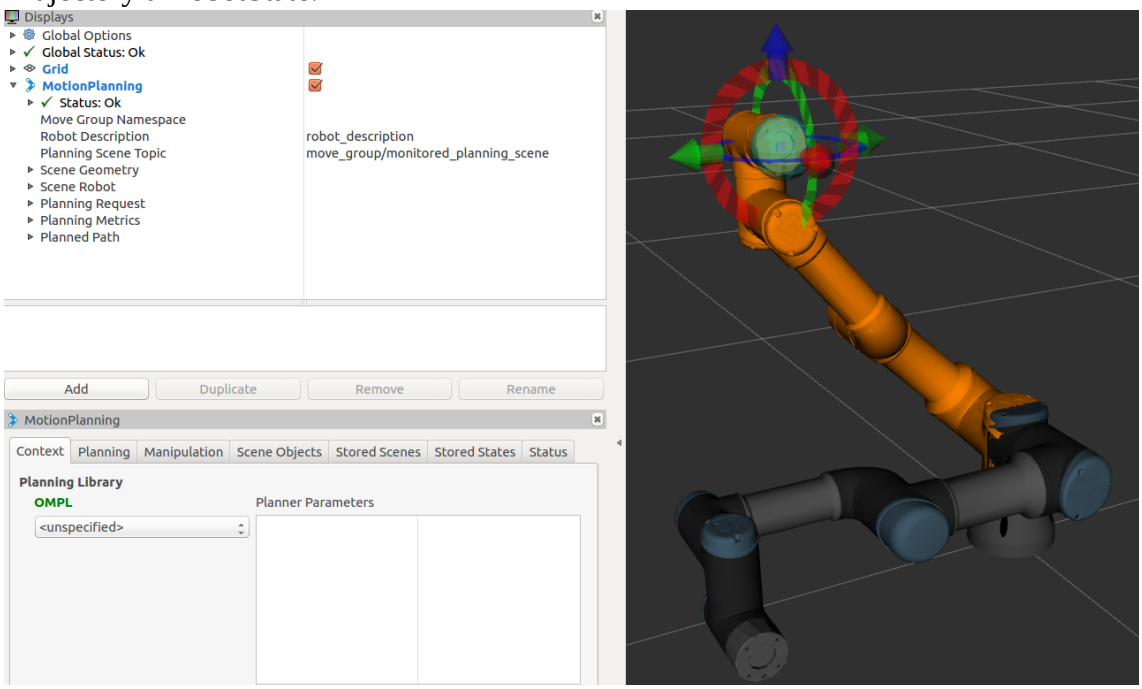
[1] URDF (*Unified Robot Description Format*) je speciální formát XML pro reprezentaci modelu robota [49].
XML (*eXtensible Markup Language*) je značkovací jazyk, který naskýtá především snadnou konverzi do jiných formátů a vysokým informačním obsahem.

robota, zavolá plánovací plugin a následně provede řízení pohybů [48]. Hlavní částí je plugin, který obsahuje knihovny s plánovacími algoritmy, nejpoužívanější knihovny jsou:

- OMPL (*Open Motion Planning Library*) je open-source knihovna pro plánování pohybu, která se skládá z mnoha nejmodernějších algoritmů založených na vzorkování. MoveIT integruje tuto knihovnu jako výchozí sadu plánovacích algoritmů. OMPL knihovna optimalizovaná pro MoveIT obsahuje algoritmy, jako je kupříkladu RRT*, PRM*, BFMT, FMT a SPARS. [50]
- STOMP (*Stochastic Trajectory Optimization for Motion Planning*) je plánovací knihovna založená na optimalizačním algoritmu PI^2. Hlavním rysem tohoto algoritmu je plánování hladké trajektorie. [51]

4.7 Rviz

Základním a nepostradatelným nástrojem ROS je Rviz (*ROS visualization*). Jedná se o nástroj pro 3D vizualizaci, který umožňuje zobrazovat pohyb a řídit roboty (obr. 20). Rviz v první řadě obsahuje vizualizační okno, kde může uživatel zobrazovat a ovládat své roboty, objekty nebo při běhu programu překážky, ať už předem vymodelované nebo zaznamenané ze senzorů, či kamery. Hlavním oknem, které Rviz integruje je Displays. To umožňuje spravovat, nastavovat a zobrazovat důležité funkce robotů nebo samotných senzorů. Příkladem může být FluidPressure, Temperature, LaserScan nebo DepthCloud, do tohoto okna je možné importovat další různé funkce z používaných platforem. Například z platformy MoveIT lze importovat funkce MotionPlanning, PlanningScene, Trajectory a RobotState.



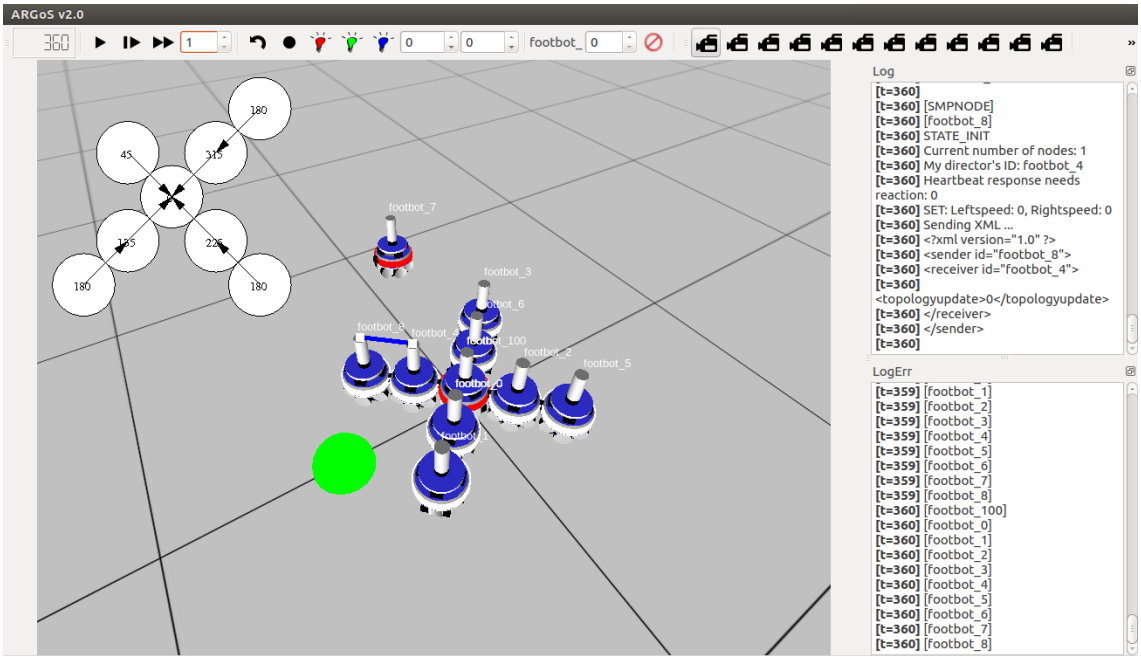
Obr. 20: Rviz

4.8 Simulační prostředí Gazebo

Nezbytným nástrojem pro vývojáře v oblasti robotiky je simulační prostředí, kde mohou otestovat své programy, použité algoritmy, návrhy robotů a provádět regresivní testování. Výchozím simulačním prostředím ROS se stalo Gazebo, jelikož tento projekt začala od roku 2012 spravovat organizace OSRF (*Open Source Robotics Foundation*), která také spravuje framework ROS. Vývoj tohoto simulačního prostředí, které se díky ROS stalo především robotické simulační prostředí, začal v roce 2002 na univerzitě v Jižní Kalifornii. Gazebo integruje výkonné fyzikální enginy ODE (*Open Dynamics Engine*), Bullet, Simbody, DART (*Dynamic Animation and Robotics Toolkit*). Uživatelsky příjemný plášť tomu dodává 3D engine OGRE (*Object-Oriented Graphics Rendering Engine*), který poskytuje realistické vykreslování prostředí, včetně vysoce kvalitních textur, stínů a osvětlení. Jednou z nejvíce atraktivních funkcí, které simulační prostředí Gazebo podporuje, je možnost použití CloudSim. CloudSim slouží ke spouštění a využívání simulačního prostředí Gazebo skrz službu Amazon Web Services, která poskytuje cloud computing platformy. Další součásti CloudSim je také služba GzWeb (*Web client for Gazebo*), která umožňuje interakci a ovládání simulačního prostředí prostřednictvím webového prohlížeče. Masivní obliba frameworku ROS také ovlivnila simulační prostředí Gazebo, díky velké komunitě uživatelů je dostupných již mnoho vytvořených modelů robotů, prostředí a různých objektů [52]. Uživatelé mohou také vytvářet vlastní modely, které jsou popsány pomocí SDF (*Simulation Description Format*) formátu. Tento formát, je speciální případ XML formátu, který byl původně vyvinut jako součást simulačního prostředí Gazebo, přičemž je navržen primárně na aplikace robotů. V průběhu let odlaďování se formát SDF stal stabilním a schopným popsat všechny aspekty robotů, statických a dynamických objektů i fyziky [53].

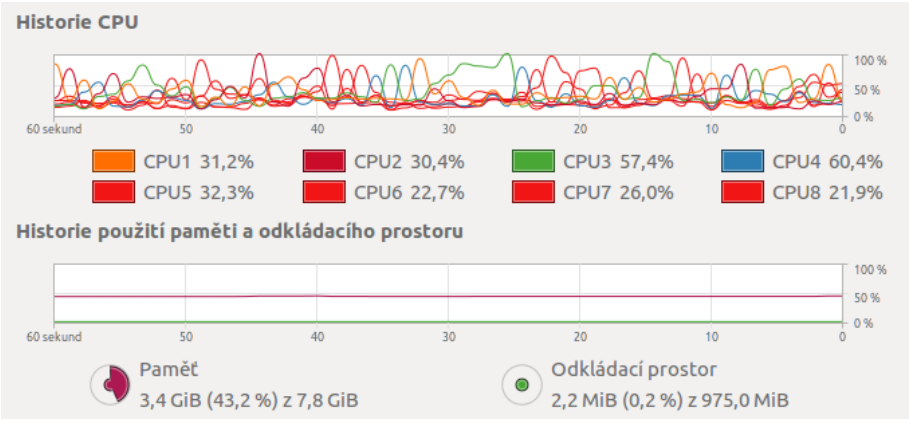
Přímým konkurentem na poli simulačních prostředí využívaných uživateli ROS je V-REP. Simulační prostřední V-REP bylo vyvinuto společností Coppelia Robotics, která má základnu v městě Zurich ve Švýcarsku. V-REP umožňuje také jako simulační prostředí Gazebo multiplatformní instalaci, přitom obsahuje stejné fyzikální enginy ODE i Bullet, avšak na místo Slimbody a DART intergruje fyzikální engine Vortex. Hlavní nevýhodou V-REP oproti Gazebo je horší integrace s ROS z důvodů rapidně menšího množství pluginů a knihoven. Předností V-REP je mnohem uživatelsky příznivější a přirozenější modelování, kdy změny může uživatel provádět i během simulace. Oproti tomu Gazebo vyžaduje k modelování znalost formátu SDF a případně také URDF, aby byla zaručena správná synchronizace a funkčnost. Flexibilnější simulační prostředí pro programátory je jednoznačně V-REP, díky skriptům integrovaných u dostupných modelů robotů, možnosti využití pluginů, ROS nebo samostatných programů, které se připojují přes RemoteAPI. U simulačního prostředí Gazebo lze programovat pouze přes poměrně kompilované C++ pluginy, nebo prostřednictvím ROS. Komplikací ve V-REP je ukládání vytvořených scén a simulací, kdy veškeré změny musí být provedeny v programu, a následně uloženy ve speciálním formátu, který má příponu .ttt. [54]

Specifickým simulačním prostředím, které především dominuje na poli rojové robotiky (*swarm robotics*) je ARGoS. Toto simulační prostředí spočívá v jednoduchosti, jelikož je navrženo v prvé řadě na využití výpočetního výkonu a neklade důraz na grafickou stránku (obr. 21). Simulační prostředí lze programovat pomocí ROS, Lua skriptů nebo C++. Tak jako Gazebo, scény v ARGoS jsou uloženy v souborech formátu XML, což umožňuje lepší editaci. [54, 55]

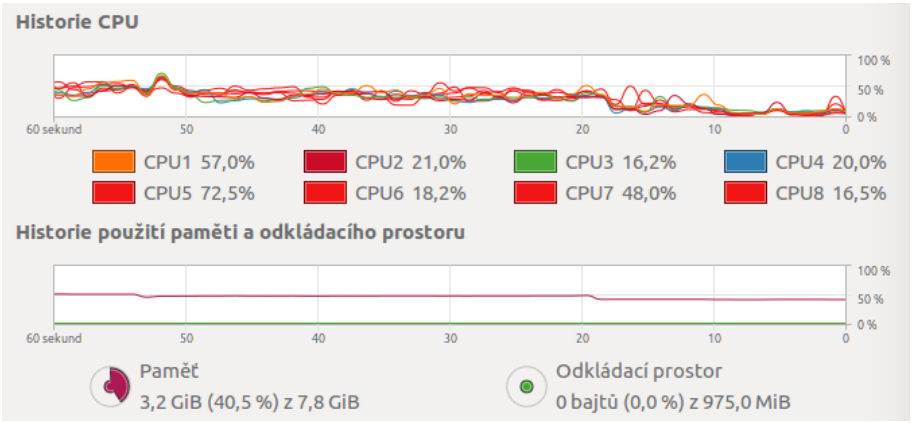


Obr. 21: Simulační prostředí ARGOS [56]

V souladu s testy provedenými na MacBook Pro s 4x Intel Core i7 2,2 Gz, 8 GB RAM a Intel HD Graphics 6000 [56] a na PC s 8,2 GB RAM paměti, CPU AMD Ryzen 5 1500X a s grafickou kartou GeForce GTX 1050 Ti, vykazují to, že simulační prostředí V-REP v porovnání s Gazebo vyžaduje větší CPU a GPU zátěž, a také větší část RAM paměti (obr. 22, obr. 23, obr. 24, obr. 25).



Obr. 22: V-REP test využití paměti RAM a zatížení CPU



Obr. 23: Gazebo test využití paměti RAM a zatížení CPU

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.130                Driver Version: 384.130                   |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 105...  Off  | 00000000:09:00.0  On |                  N/A |
| 45%   37C    P0   ERR! /  75W |    380MiB /  4038MiB |     54%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1150      G   /usr/lib/xorg/Xorg                           217MiB |
|    0      2332      G   compiz                                       156MiB |
|    0      3239      G   /usr/lib/firefox/firefox                       1MiB |
|    0     10515      G   /home/juricekm/V-REPES/vrep/vrep               3MiB |
+-----------------------------------------------------------------------------+
```

Obr. 24: V-REP maximální zatížení grafické karty při testu

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.130                Driver Version: 384.130                   |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 105...  Off  | 00000000:09:00.0  On |                  N/A |
| 45%   31C    P0   ERR! /  75W |    664MiB /  4038MiB |     44%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1150      G   /usr/lib/xorg/Xorg                           392MiB |
|    0      2332      G   compiz                                       161MiB |
|    0      3239      G   /usr/lib/firefox/firefox                       1MiB |
|    0     14119      G   ...quest-channel-token=8206874707129597620    42MiB |
|    0     22975      G   gzserver                                       3MiB |
|    0     22983      G   gzclient                                      60MiB |
+-----------------------------------------------------------------------------+
```

Obr. 25: Gazebo maximální zatížení grafické karty při testu

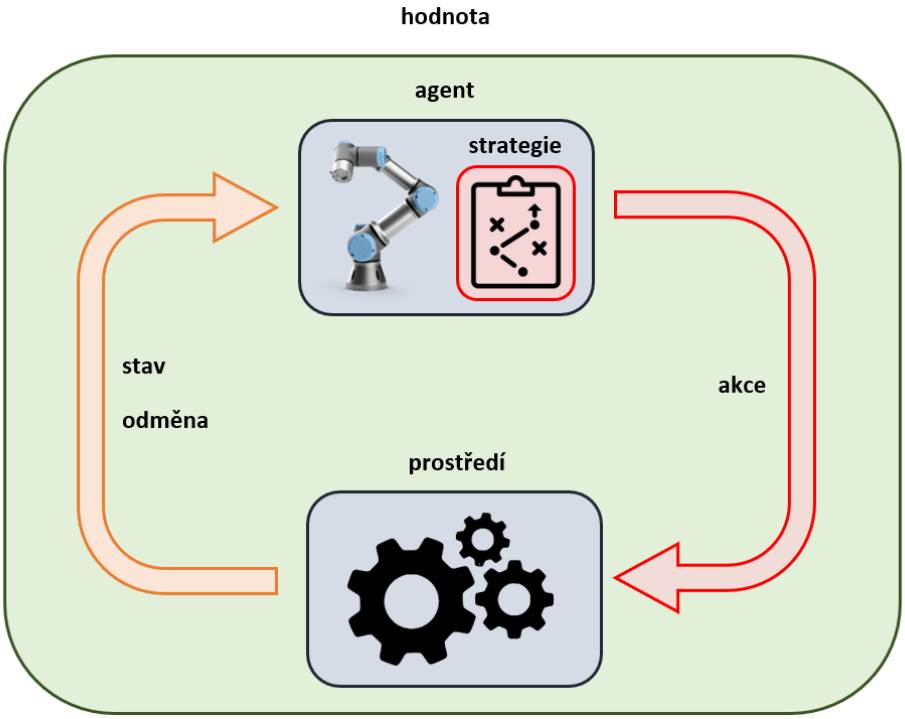
4.9 Robotický operační systém 2

Nová generace ROS 2, která byla představena v roce 2017, poskytuje pro vývojáře zcela nové rozměry využití, oproti první generaci. Zásadní vylepšení přišlo v podobě možnosti řízení a využití více robotů najednou, následně poskytnutí robotické bezpečnosti i dlouho očekávaného real-time řízení. Další inovací je multiplatformní podpora pro Linux Ubuntu, OS X a Windows 10. Nepostradatelnou změnou, která vede k většímu vývojářskému a uživatelskému komfortu je využití aktuálních verzí programovacích jazyků python 3.5 a C++ 11, 14, 17 [57]. Jistou komplikací u první generace je to, že pokud vývojáři či uživatelé chtějí použít softwarové platformy pro strojové učení jako je například TensorFlow, PyTorch, Keras nebo OpenAI, musí instalovat k původní verzi programovacího jazyku ROS z python 2 na aktuálnější, tedy python 3.5 a výš. Tato instalace má v sobě schované další problémy v podobě knihoven a nástrojů podporující python 2, což vede na neúplnou synchronizaci. Svou nevýhodu má zatím ROS 2 v poměrně malé komunitě s čímž souvisí také menší množství dostupných balíků, přitom ne všechny jsou zcela stabilní. Jedná se takřka o novinku a své oblibě si přichází především u vývojářů, kteří mají zkušenosti s ROS a zabývají se autonomním řízením nebo využitím umělé inteligence v robotice.

5 REINFORCEMENT LEARNING

Mezi nástroje průmyslu 4.0 patří také použití umělé inteligence, ať už v rámci robotiky, tak zpracování obrovského množství dat z logistických, výrobních nebo distribučních systémů. Podstatou umělé inteligence je napodobení lidské mysli ve sféře učení nebo uvažování při řešení spletitých problémů s využitím výpočetní techniky. Umělá inteligence je komplexní obor na poli informatiky, který s sebou přináší vědní oborové oblasti, jako je například vytěžování údajů (*data mining*), genetické algoritmy (*genetic algorithms*) či strojové učení (*machine learning*). Tyto oblasti se postupně rozvíjejí a každá oblast má své specifické opodstatnění a využití.

V oblasti robotiky svůj oprávněný vývoj našly algoritmy zpětnovazebního učení (*reinforcement learning*) a jejich rozšířená forma hluboké zpětnovazební učení (*deep reinforcement learning*). Algoritmy zpětnovazebního učení používají metodu, při které se softwarový agent učí interakcí s prostředím, přičemž prostředí sleduje výsledky tohoto působení s cílem dosažení maximalizace funkce kumulativní odměny. Tento návrh téměř věrně imituje lidskou metodu učení pokus/omyl, kdy následné rozhodnutí závisí na vyhodnocení zpětné vazby (obr. 26). Potíž aplikace algoritmů zpětnovazebního učení v robotice může nastat při testování a provádění procesu učení v reálném prostředí, což může vést na nebezpečné situace, jak pro samotného robota, tak i pro člověka. Aby se těmto situacím dalo předejít, vývojáři používají simulační prostředí, jako je kupříkladu Gazebo nebo V-REP, kde se snaží o co nejvěrnější modelování reálného prostředí a testování simulace.



Obr. 26: Princip zpětnovazebního učení

Základními prvky algoritmů zpětnovazebního učení jsou agent a prostředí (*environment*). Dalšími prvky jsou odměna (*reward*), hodnota funkce (*value function*), strategie (*policy*) případně i model prostředí (*model of environment*).

- Agent je model, také často označován jako entita, která provádí akce.
- Prostředí je scénář, který má podobu markovova rozhodovacího procesu (*markov decision process*), ten poskytuje informace agentovi v podobě stavu (*state*) a odměny.
- Strategie je jádro agenta, které definuje akce založené na stavech zprostředkované prostředím a obecně má stochastický charakter.
- Odměna je posílána prostředím agentovi, zatímco agent má za cíl maximalizovat tuto odměnu, kterou dostává při svém učení. Agent může ovlivnit přímo odměnu jen svými akcemi, tudíž odměna závisí také na aktuálním stavu prostředí.
- Hodnota funkce určuje dlouhodobé cíle, tedy označuje v delším časovém horizontu vhodnost stavů, které budou pravděpodobně následovat.
- U některých algoritmů zpětnovazebního učení se může objevit i prvek model prostředí. Model prostředí zastává funkci plánovače, kdy rozhoduje o průběhu akce zvážením možných budoucích situací, než budou skutečně provedeny. [58]

5.1 Markov decision process

Rozhodnutí v algoritmech zpětnovazebního učení má podobu stochastického procesu, tedy markovova rozhodovacího procesu (MDP). Charakterizovat tento proces lze pomocí matematické závislosti na funkci přechodu o aktuálním stavu v systému a aktuální akci. Pokud se tedy proces nachází pokaždé v daném časovém okamžiku v určitém stavu s a agent má možnost zvolit akci a, která je v daném stavu dostupná, proces následně reaguje oceněním agenta užitkem $R_a(s, s')$ a náhodným přesunutím do nového stavu s'. Pravděpodobnost výběru nového stavu s' úzce závisí na vybrané akci a a předchozím stavu s, přitom jsou podmíněně závislé na všech minulých stavech s a akcích a. Jednou z charakteristik tohoto procesu je skutečnost, že historie všech minulých stavů s a akcí a není udržována v paměti. MDP v algoritmech zpětnovazebního učení umožňuje matematicky popsat rozhodování v situacích, kde se jedná o náhodu a také situaci, kterou má agent neúplně pod kontrolou [59]. MDP lze rozdělit na 6 částí:

- konečná množina stavů S
- konečná množina akcí A, kdy agent může vykonat akci, která je ve všech stavech stejná [59]
- diskontní faktor γ snižuje kompenzaci, která je přidělena agentovi v průběhu času a chování ve stavech

- pravděpodobnost přechodu $P_a(s, s')$ je numerické vyjádření pohybu agenta z jednoho stavu s v čase t do jiného stavu s' při akci a v čase $t + 1$
- pravidla určující odměnu $R_a(s, s')$ je okamžitý užitek, který je předán jako informace agentovi, kdy dojde k dosažení přechodu ze stavu s do stavu s' s pravděpodobností přechodu $P_a(s, s')$
- nalezení strategie π pro agenta je hlavním úkolem MDP, přičemž cílem je určit strategii, která bude maximalizovat kumulativní funkci náhodných odměn

V algoritmech zpětnovazebního učení je nutné najít optimální strategii, jak je definováno v MDP, proto optimální strategie π, která má podobu:

$$\pi(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P_a(s, s')V(s') \tag{1}$$

Získání odměny podle určení stavu s a optimální akce a, definované pro stav V, určuje Bellmanův princip optimality dle vztahu (2). Tohoto principu se přitom využívá i u algoritmů zpětnovazebního učení (3). U zpětnovazebního učení je místo stavu využíváno pole $Q(s, a)$, které se označuje jako Q-hodnota (*Q-value*). Q-hodnoty je využíváno především k aktualizaci procesu na základě předešlé zkušenosti. [60, 61]

$$V(s) = R(s) + \gamma \max_{a} \sum_{s'} P_a(s, s')V(s') \tag{2}$$

$$Q(s, a) = R(s) + \gamma \sum_{s'} P_a(s, s')V(s') \tag{3}$$

5.2 Q-learning

Nejreprezentativnějším algoritmem zpětnovazebního učení je Q-learning, který se stal základem pro mnoho dalších algoritmů. Některé algoritmy jsou jen částečnými modifikacemi pro danou aplikaci, jako je například EMA Q-learning nebo QV-learning. Q-learning je algoritmus bez strategie (*off-policy*) a modelu (*model-free*) [59]. Tyto vlastnosti umožňují oddělení učící se strategie od akční strategie a řešení problémů se stochastickými přechody a odměnami bez vnějších úprav. Algoritmus je založen na použití Q-hodnoty, tedy pokud se agent nachází v daném stavu s, Q-hodnota určí, jaká bude odměna, pokud agent zvolí akci a. Jednou z charakteristik algoritmu Q-learning je nepotřebnost znát model prostředí. Je však důležité, aby agent měl možnost prozkoumat co nejvíce stavů, a poté jsou získávány informace o tom, jak vypadá prostředí. Algoritmus má následující podobu:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R + \gamma \max_{a} Q(s', a') - Q(s, a)\right] \tag{4}$$

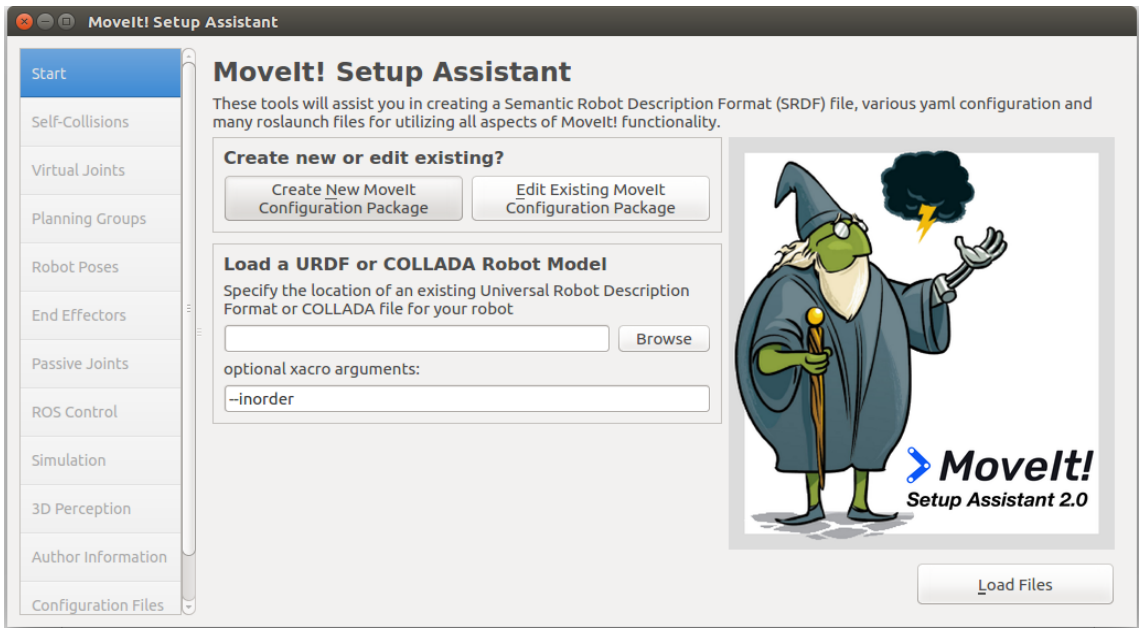
kde α je hyperparametr s označením míra učení (*learning rate*), přitom nabývá hodnoty v intervalu $\langle 0{,}1 \rangle$. Tento hyperparametr určuje, v jaké míře jsou staré informace

potlačovány novými. Nulová hodnota způsobí to, že se agent nenaučí nic, jelikož spoléhá především na předchozí informace. Naopak hodnota rovna 1 určí, že agent bude pouze prozkoumávat. Dalším vystupujícím hyperparametrem v algoritmu je γ s označením diskontní faktor (*discount factor*), který určuje prioritu budoucích odměn. Hodnota rovna 0 u tohoto hyperparametru způsobí, že agent se bude pouze snažit o maximalizaci aktuálních odměn, avšak hodnota rovna 1 má za následek, že agent bude usilovat o dlouhodobou vysokou odměnu. Hodnota γ by se měla pohybovat v intervalu $\langle 0,1 \rangle$, přitom pokud dojde k překročení hodnoty 1, mohou akční hodnoty divergovat. [60, 62]

Demonstrovat chování tohoto algoritmu můžeme ve velmi zjednodušené formě. Na začátku učení se inicializuje Q-hodnota na hodnotu 0, tudíž neproběhly doposud žádné akce v určitých stavech. Následně agent v prvním stavu s zvolí se stochastickou pravděpodobností akci a, kdy následně dostává odměnu R, poté se aktualizuje Q-hodnota a uloží se Q-tabulka[2]. Takto se repetitivně chová agent až do té doby, dokud nebude nalezena nejoptimálnější cesta s největší možnou odměnou.

5.2.1 Návrh a implementace algoritmu Q-learning

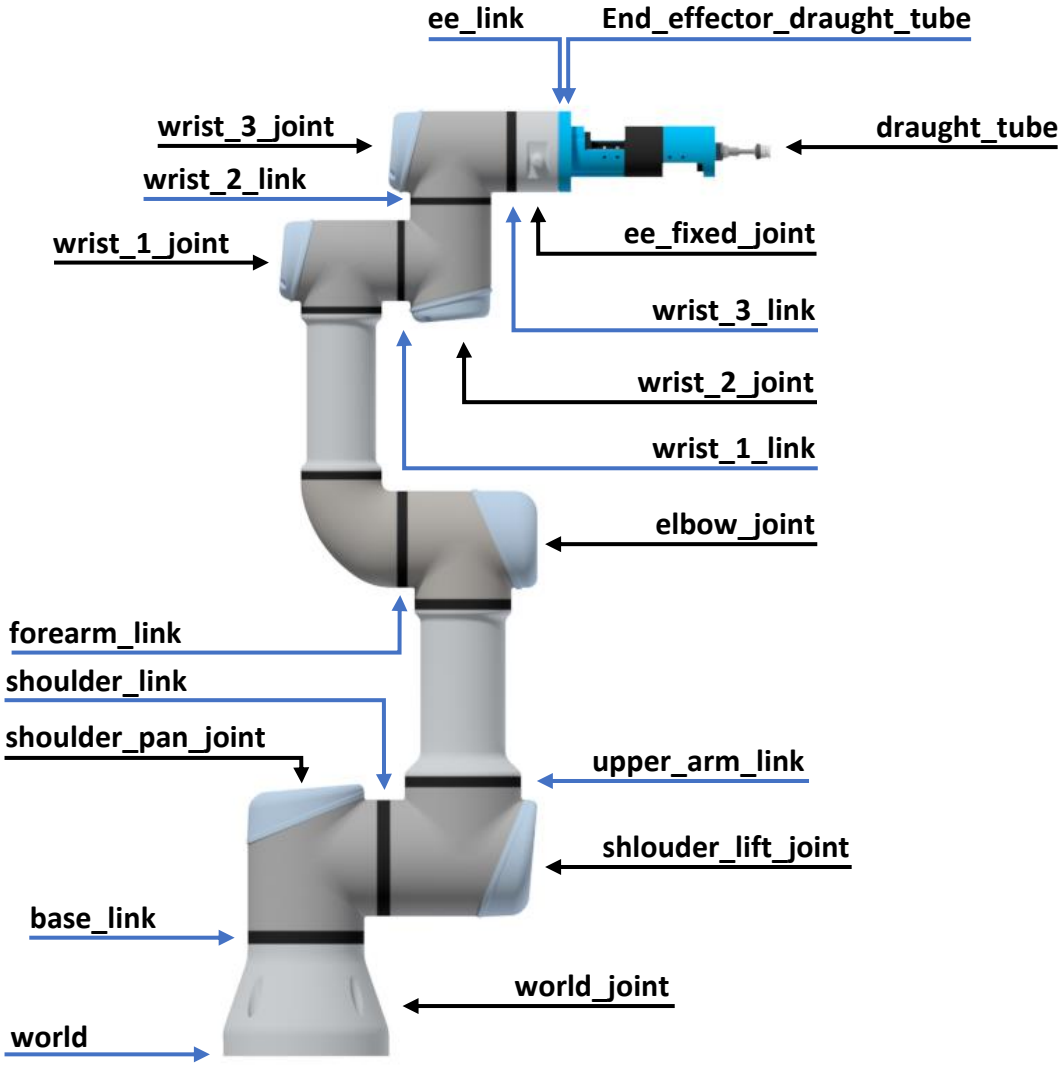
Cílem v této praktické části je aplikace zpětnovazebních algoritmů Q-learning k učení kolaborativního robota UR3 s koncovým nástrojem, přičemž záměrem je školení kobota k dosahování definovaného bodu v rozsahu jeho pohybu. K realizaci je použita distribuce frameworku ROS Kinetic Kane s hostitelským operačním systémem Ubuntu Xenial, simulační prostředí Gazebo, platforma MoveIT, knihovna k vykreslování grafů Plotly a knihovna OpenAI Gym.



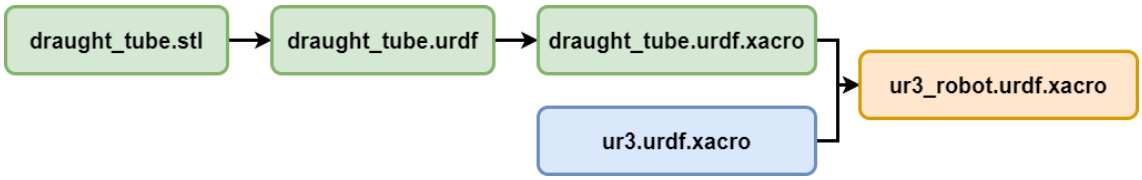
Obr. 27: Úvodní okno MoveIT Setup Assistant

[2] Q-tabulka je vyhledávací tabulka díky, které lze provést výpočet očekávané maximální odměny za akci v daném stavu [63].

K samotnému provedení je nutné vytvoření balíku ROS. Tento Balík bude integrovat všechny potřebné soubory, od skriptů ke spouštění modelu v simulačním prostředí, přes plánovací skripty, až po skripty s aplikací zpětnovazebních algoritmů. ROS je otevřený systém, který nabízí možnost vytvoření celého balíku takzvaně od nuly, avšak díky platformě MoveIT lze využít asistenta k tvorbě základu tohoto balíku. Asistent nese název MoveIT Setup Assistant (obr. 27). Jedná se o grafické rozhraní sloužící ke konfiguraci robotů.



Obr. 28: Architektura UR3 se savkou



Obr. 29: Posloupnost odkazujících se souborů k propojení kobota UR3 se savkou

Při použití asistenta je nutné načíst urdf.xacro[3] soubor modelu kolaborativního robota UR3 se savkou. Napojení modelu koncového nástroje v podobě savky k modelu kobota, lze nakonfigurováním nového *ur3_robot.urdf.xacro* souboru. Tento soubor bude obsahovat odkazy na *ur3.urdf.xacro* soubor modelu kobota z metabalíku *universal_robots* a na soubor savky *draught_tube.urdf.xacro* (viz obr. 28, obr. 29). V souboru savky je nutné určit nadřazený odkaz (*parent link*) kobota, na který se připojí jako nový kloub (*joint*) model koncového efektoru. V neposlední řadě je také potřebné ve skriptu *draught_tube.urdf* změnit měřítko, aby model savky odpovídal modelu kobota. Soubor *ur3.urdf.xacro* je komplexně propojen s odkazy na jiné soubory z metabalíku universal_robots, tudíž k tomu, aby po vytvoření balíku došlo ke spuštění, je nutné tento metabalík uložit v hostitelském operačním systému.

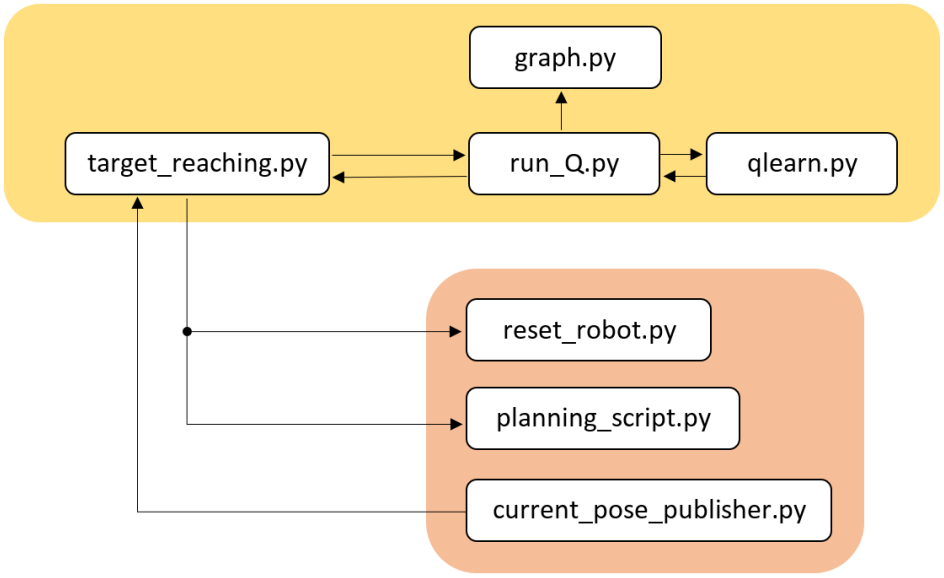
Pomocí asistenta je následně nakonfigurována hustota vzorkování vlastní kolize, aby nedocházelo k samotnému poškozování robota. Výchozí hodnota je 10 000 kontrol, přičemž čím je vyšší hustota, tím je vyžadován delší výpočetní čas. Pokud je hustota příliš nízká může dojít k vlastní kolizi částí robota. Dalším bodem konfigurace je definování virtuálních kloubů, které se používají k připojení a určení robota vůči virtuálnímu světu. Pro model kobota UR3 je určen pouze jeden virtuální kloub. Následně je nutné definovat plánovací skupinu, která slouží jako sémantický popis různých částí robota, jako je například tělo robota a koncový efektor. Dále je nutné také určit kinematického řešitele (*kinematic solver*), kde je možné i spravovat bližší specifikace, jako je kupříkladu počet pokusů řešení nebo časový limit, který má výchozí hodnotu 0,005 s. Při konfiguraci v asistentu je možné specifikovat další vlastnosti, zejména se jedná o základní pozice robota, bližší konfigurace koncového nástroje či ROS kontroléry. K tomu, aby šlo úspěšně vytvořit balík nebo editovat již stávající balík, jsou vyžadovány specifikace o autorovi, které se následně uloží do manifestu balíku. Po generování a vytvoření je nutné následně provést úpravy asistentem vytvořeného balíku, především vytvořením spouštěcího souboru simulačního prostředí Gazebo *rl.launch*. Tento soubor bude integrovat základní parametry k vytváření virtuálního prostředí, kontroléry parametr serveru, v neposlední řadě model kobota UR3 s koncovým efektorem a specifikaci uzlu pro python skript *reset_robot.py*. Další úpravou je vytvoření spouštěcího skriptu využívající funkce frameworku MoveIT *moveit_planning.launch* pro vykonávání a plánování trasy v simulaci, přičemž skript integruje spouštěcí soubory *planning_context.launch*, *move_group.launch* a specifikací uzlů pro skripty *current_pose_publisher.py* a *planning_script.py*.

Při návrhu pohybu simulačního modelu kobota UR3 se savkou v simulačním prostředí Gazebo je uvažováno využití frameworku MoveIT k plánování trasy a řešení inverzní kinematiky pomocí plánovacích algoritmů z knihovny OMPL. Nejedná se tedy o případ učení dosahováním polohy robotického ramene bez využití inverzní kinematiky. Základním kamenem k aplikaci zpětnovazebních algoritmů je použití sady nástrojů, která slouží k trénování agentů a vytváření prostředí OpenAI Gym. Aby bylo možné použít

[3] Xacro (*XML Macros*) je jazyk XML maker, přičemž díky tomuto formátu lze psát kratší a čitelnější soubory XML pomocí maker [64].

OpenAI Gym, je nutné řešit problém v podobě potřebné aktualizace verze programovacího jazyka python instalováním k implicitně integrované verzi programovacího jazyka ROS python 2 verzi python 3 a výš. Tato instalace vede na vytvoření oddělených skriptů (znázorněno na obr. 30). Skripty, které využívají knihovny OpenAI Gym jsou spouštěny v ROS ve verzi python 3, přičemž v architektuře řešení se jedná o skripty:

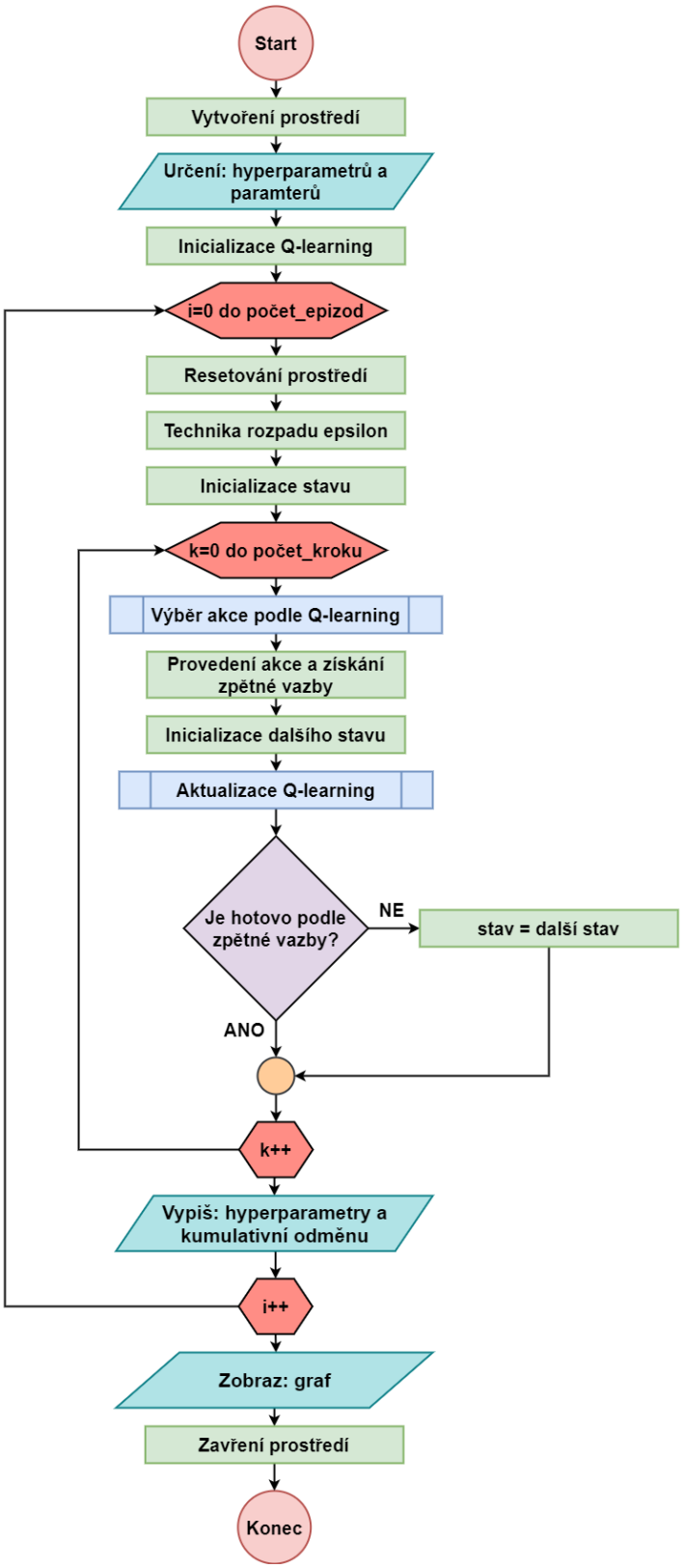
- *run_Q.py*
- *qlearn.py*
- *target_reaching.py*
- *graph.py* – skript sloužící k vykreslení grafů využívající knihovnu Plotly



Obr. 30: Architektura python skriptů Q-learning

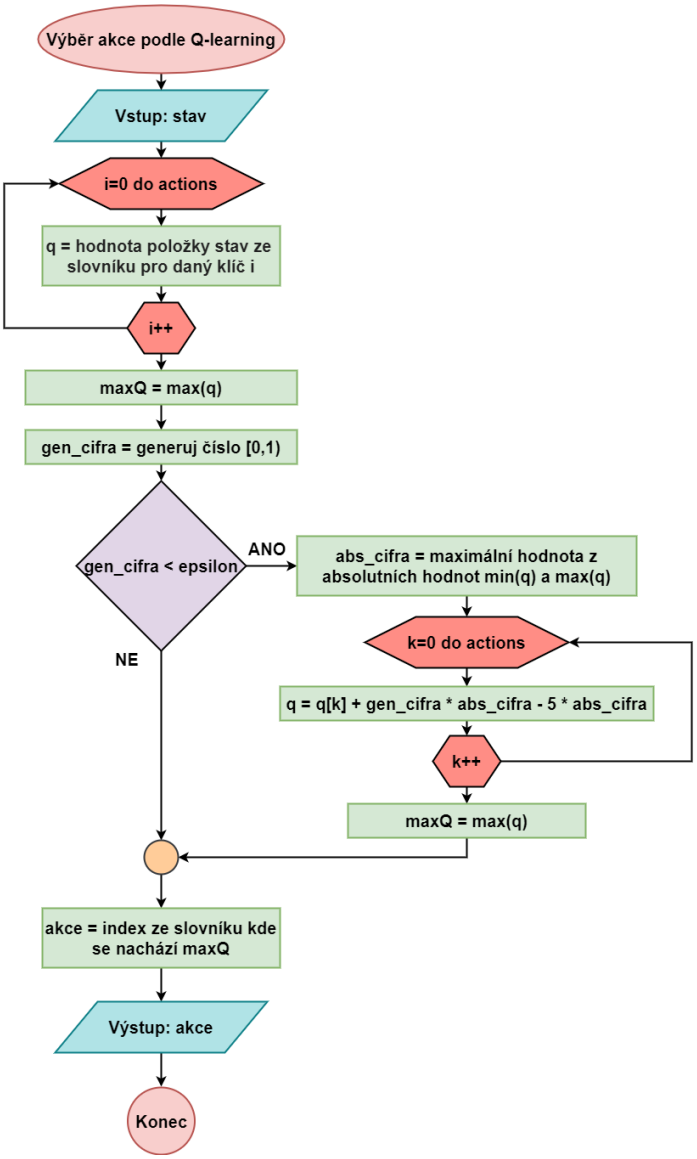
Implementace zpětnovazebního algoritmu Q-learning k učení kobota UR3 pro dosažení určené polohy je komplexní problém od nastavení hyperparametrů, až po řešení pohybu modelu kobota UR3 se savkou. Jádrem implementace je skript *qlearn.py*, který integruje aplikaci samotného algoritmu Q-learning a skript *run_Q.py* (obr. 31). Ten vytváří, inicializuje a řídí prostředí, definuje simulaci a také inicializuje hyperparametry pro algoritmus Q-learning. K testování je nezbytné vhodně definovat maximální počet kroků v epizodě, také celkový počet epizod a velikost jednoho kroku definovaného v *planning_script.py*, který určuje směr pohybu kobota. Pokud je definován příliš malý počet epizod a malá velikost jednoho kroku, agent nemusí najít vhodnou cestu k určenému cíli. Pokud je definován velký počet kroků a malá velikost jednoho kroku, agent bude hledat cestu, která nebude příliš optimální z uživatelského hlediska. Dalším úskalím je definování celkového počtu epizod, kdy může nastat problém při zvolení příliš nízkého nebo vysokého počtu. Pokud je zvoleno příliš málo epizod, výsledkem nemusí být vykazování známek učení a pokud bude zvoleno příliš mnoho, může být nalezena nejoptimálnější cesta, avšak za cenu zdlouhavého času simulace. V každé iteraci je zvolena akce, a to udělej krok ve směru x, y, z, kdy je následně přijímána zpětná vazba

tzv. *observation*, která vrací informaci o dalším stavu s obdrženou odměnou. Dalším problémem je definování vhodné hodnoty hyperparametrů alfa, epsilon a gama, přičemž i malé změny hodnoty mohou vést k podstatným změnám učení.



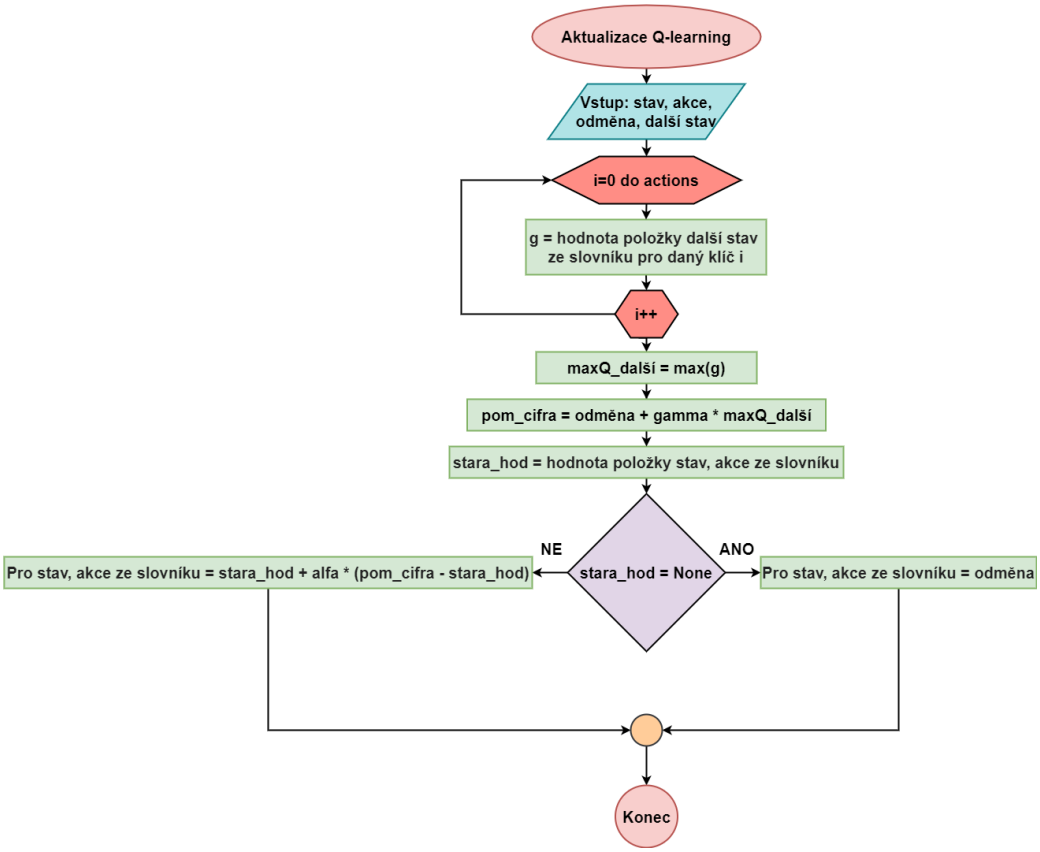
Obr. 31: Vývojový diagram skriptu *run_Q.py*

- Míra učení alfa. Při určení vhodné hodnoty hyperparametru je nutné dbát na rozsah ⟨0,1⟩. Tedy pokud bude hodnota příliš nízká, nebude tak znatelný progres učení, či může dojít ke stagnaci. Pokud bude hodnota příliš vysoká, může nastat konvergence řešení.
- Diskontní faktor gama. Pokud se definovaná hodnota blíží k 0, agent bude brát v úvahu pouze současné odměny, zatímco hodnota blížící se k 1 definuje, že agent bude usilovat o dlouhodobě vysoké odměny.
- Průzkumná konstanta epsilon je použita k náhodnému rozhodování. Této vlastnosti se využívá především v počátcích učení. Pokud bude konstanta nastavena například na hodnotu 0,8, bude prováděno právě 80 % stochastických akcí. K řešení aplikace je zvolena technika rozpadu epsilon pomocí proměnné *epsilon_discount*, tudíž v průběhu učení se bude hodnota epsilon snižovat.



Obr. 32: Vývojový diagram funkce *chooseAction_Q*

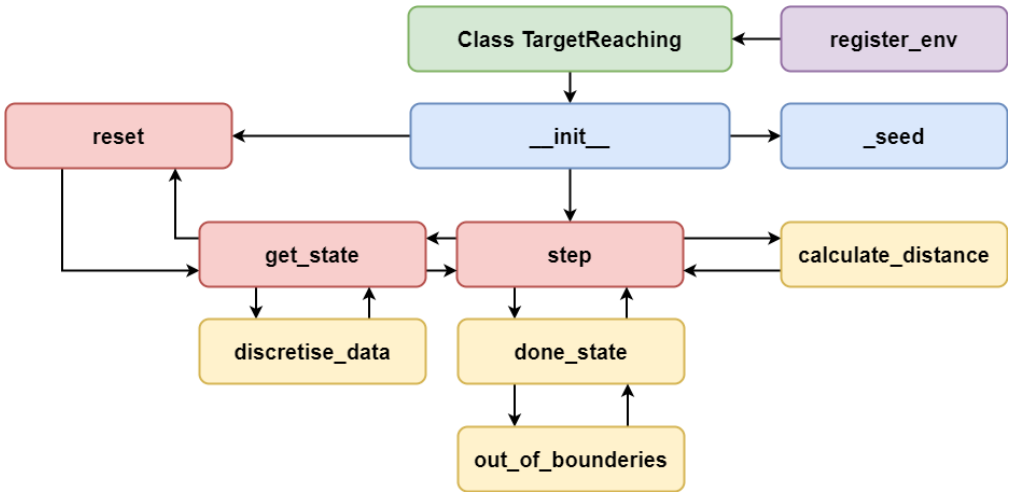
S definováním hyperparametrů, které jsou vstupními parametry ve skriptu *qlearn.py* třídy *Qlearn*, je nutné také definovat parametr *actions*, který nabývá hodnoty v intervalu ⟨0,6⟩ datové typu integer. Tím je deklarováno, že výstupem bude zvolení akce ve směru x, y, z v daném rozmezí. Poté, co se resetuje *observation*, je následně definován stav a pak je zvolena akce podle stavu ve funkci *chooseAction_Q* (obr. 32). V dalším kroku je určen stav pro agenta a poté je proveden výpočet Q-learning algoritmu ve funkci *learn_Q* (obr. 33) ve skriptu *qlearn.py*. Vstupními parametry jsou stav, další stav, akce a odměna.



Obr. 33: Vývojový diagram funkce *learn_Q*

Primární částí je skript *target_reaching.py*, přičemž hlavním úkolem tohoto skriptu je propojení jádra implementace se simulací pohybu modelu kobota v simulačním prostředí Gazebo. K dosažení správné aplikace zpětnovazebních algoritmů je nutná integrace struktury s funkcemi *reset*, *step* a *get_state* (obr. 34).

- *reset* funkce specifikuje a spouští resetování simulačního prostředí s kobotem ve skriptu *reset_robot.py*.
- *step* funkce slouží k publikování akcí pomocí uzlů do skriptu *planning_script.py*, přitom v této funkci je spravován stav, odměna či status dosaženo/nedosaženo, které jsou následně využívány ve skriptu *run_Q.py*.
- *get_state* funkce určuje aktuální polohu díky odebírání hodnot z uzlu ze skriptu *current_pose_publisher.py*, kdy lze následně díky těmto datům určovat stav i odměny.



Obr. 34: Struktura skriptu *target_reaching.py*

Struktura návrhu implementace řešení rozděluje na skripty spouštěné ve verzi python 3 a python 2. Toto nezbytné dělení je vyžadováno také z důvodů využívání knihoven z platformy MoveIT, které jsou kompatibilní s verzi programovacího jazyka python 2. V architektuře se jedná o skripty:

- *current_pose_publisher.py* – skript, který slouží pro publikování aktuální pozice pomocí uzlu v ROS.
- *planning_script.py* – skript, ve kterém jsou odebírány data z uzlu k určení směru x, y, z pohybu kobota v simulačním prostředí, načež ve skriptu dojde k plánování a vykonání trasy. Možných pohybů je 6, tedy v směru ±x, ±y a ±z.
- *reset_robot.py* – skript, ve kterém dochází k pozastavení fyziky, následnému resetování simulačního světa a model kobota se inicializuje do původní polohy.

K samotnému spuštění programu dojde použitím následujících příkazů. Prvním příkazem je nutné zavolat spouštěcí soubor simulačního prostředí v prvním terminálu. Po napsání příkazu se spustí program Gazebo s modelem kolaborativního robota UR3 s koncovým nástrojem. V druhém terminálu je nutné zavolat spouštěcí soubor, který následně spouští skripty potřebné k plánování a řízení pohybu kobota v simulačním prostředí. V poslední řadě už stačí napsat příkaz ve třetím terminálu, který spustí samotný proces učení, přičemž pohyb lze sledovat v programu Gazebo.

```
user@user-pc:~$ roslaunch testos rl.launch

user@user-pc:~$ roslaunch testos moveit_planning.launch

user@user-pc:~$ rosrun testos run_Q.py
```

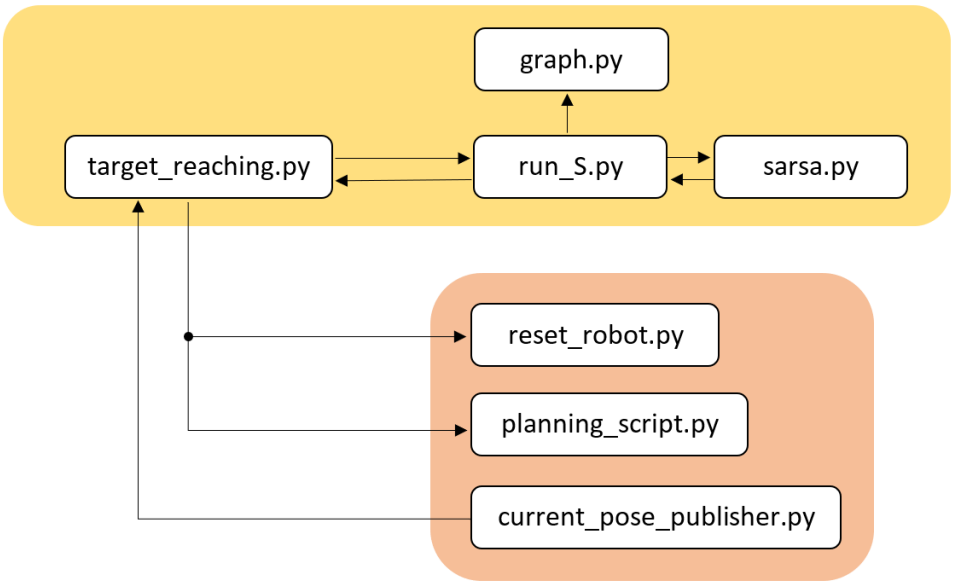
5.3 SARSA

Algoritmus SARSA (*State-Action-Reward-State-Action*) je velmi podobný algoritmu Q-learning, kdy se jedná o algoritmus se strategií (*on-policy*) a bez modelu (*model-free*). Tyto vlastnosti umožňují určení jednotné strategie a řešení problémů se stochastickými přechody a odměnami bez vnějších úprav. SARSA oproti algoritmu Q-learning ke dvojici stav, akce s odměnou určí dvojici nový stav, nová akce. Pokud se tedy agent nachází ve stavu s, přičemž provede akci a vedoucí k odměně R. Poté přesune do nového stavu $s´$ a provede novou akci $a´$, za kterou dostane odměnu, přitom se následně aktualizuje Q-hodnota dle vztahu:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma Q(s´,a´) - Q(s,a)] \tag{5}$$

5.3.1 Návrh a implementace algoritmu SARSA

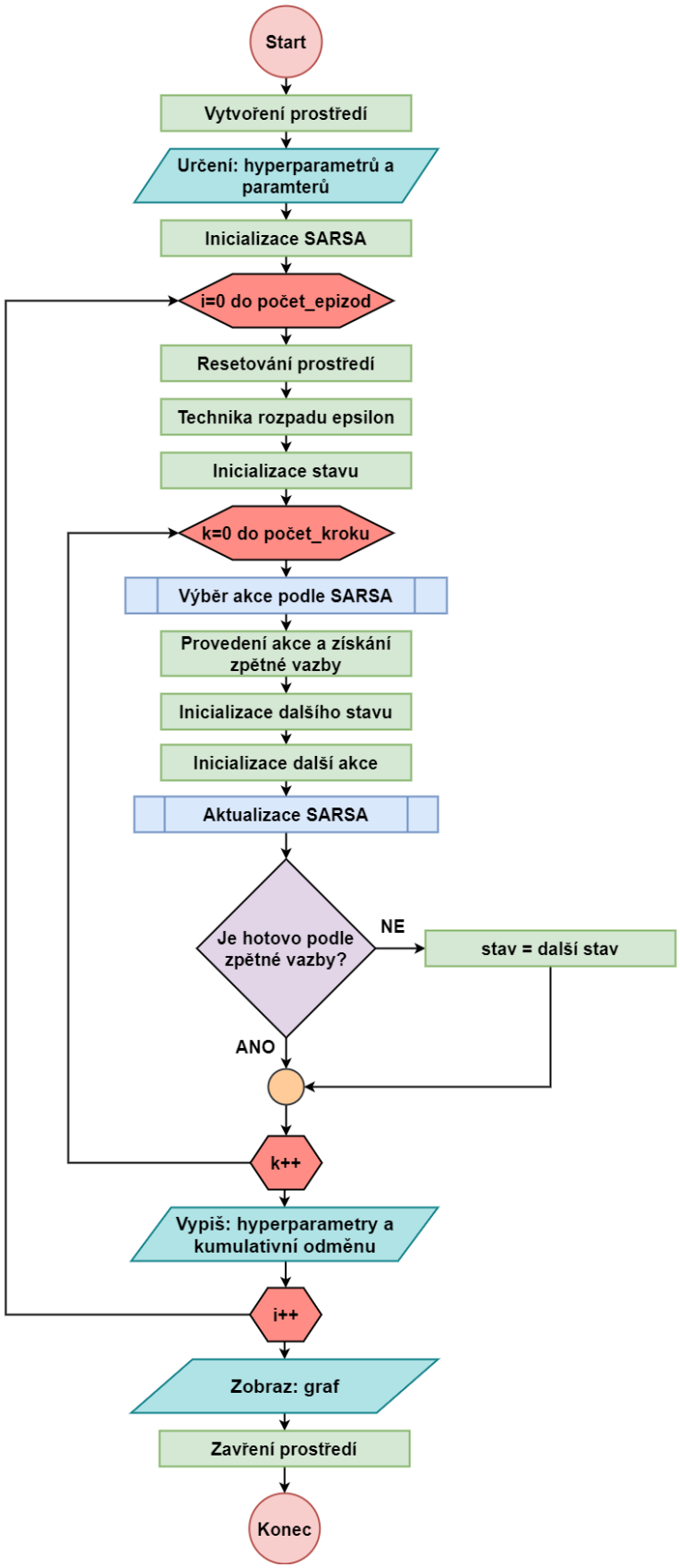
Záměrem v této praktické části, je implementace k algoritmu Q-learning algoritmus SARSA. Jelikož se jedná pouze o změnu použití algoritmu, nebyly provedeny žádné významné změny oproti implementaci v kapitole 5.2.1, a bylo tedy využito stejného úkolu učení ve stejném balíku, avšak byla částečně pozměněna architektura python skriptů (viz obr. 35).



Obr. 35: Architektura python skriptů SARSA

Cílem je učení modelu kolaborativního robota UR3 s koncovým efektorem k dosahování definovaného bodu, což vede k zachování skriptů, které využívají knihoven frameworku MoveIT a také primárního skriptu propojující jádro implementace algoritmu se skripty obsluhující simulaci v simulačním prostředí. Změnou tedy bylo vytvoření nového skriptu *sarsa.py* obsahující aplikaci samotného algoritmu SARSA spolu se skriptem *run_S.py* (obr. 36), který vytváří, inicializuje a řídí prostředí, definuje samotnou simulaci a také inicializuje hyperparametry pro algoritmus SARSA. Problémy, které byly nutné řešit při implementaci a testování algoritmu Q-learning, jako je například vhodné

zvolení hyperparametrů nebo maximálního počtu kroků v epizodě přetrvává i při implementaci algoritmu SARSA.



Obr. 36: Vývojový diagram *run_S.py*

Zásadní intervence přichází teprve ve skriptu *run_S.py*, a to v podobě vstupního parametru funkce *learn_S* (obr. 37) ve skriptu *sarsa.py*. Vstupními parametry jsou stav, akce, odměna, další stav a další akce. V každé iteraci je zvolena akce udělej krok ve směru x, y, z, přitom následně je přijímána zpětná vazba *observation*, která vrací informaci o dalším stavu s obdrženou odměnou. K tomu, aby bylo možné vložit vstupní parametr další akce do funkce *learn_S*, je nutné určit další stav a poté také určit další akci zavoláním funkce *choose_action_S,* která má stejnou podobu jak funkce *choose_action_Q* ve skriptu *qlearn.py*. Poté, co jsou určeny všechny potřebné parametry k zavolání funkce *learn_S* ze skriptu *sarsa.py*, je následně proveden výpočet algoritmu.

Obr. 37: Vývojový diagram funkce *learn_S*

K samotnému spuštění programu učení je nutné přistupovat jako v kapitole 5.2.1, kdy je nutné otevřít 3 terminály a v každém spustit následující příkazy.

```
user@user-pc:~$ roslaunch testos rl.launch

user@user-pc:~$ roslaunch testos moveit_planning.launch

user@user-pc:~$ rosrun testos run_S.py
```

5.4 Umělá neuronová síť

Zásadní rozvoj v oblasti umělé inteligence umožnily umělé neuronové sítě (*artificial neural network*). Svým návrhem má umělá neuronová síť základ v biologickém výpočetním modelu lidského mozku, kde jsou signály přenášeny skrz síť neuronů. Jejich aplikace k řešení reálných problémů v posledních letech značně vzrostla a své využití nacházejí v medicíně, telekomunikaci nebo při těžení dat či strojovém vidění.

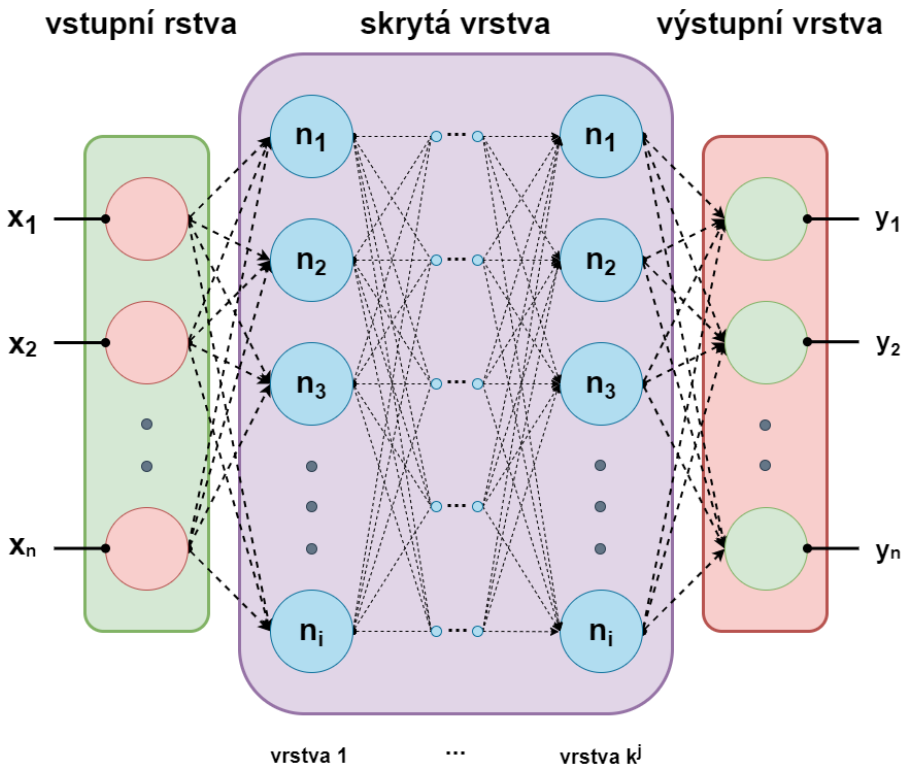
Základní fungování umělé neuronové sítě je inspirováno biologickou neuronovou sítí. V biologické neuronové síti mozku přijímají dendrity neuronu vstupní signály, které jsou následně zpracovávány v samotném těle buňky. Dále jsou zpracované signály přenášeny podél axiomu až na výstup k axionálnímu zakončení, kde vzniká spojení mezi dalším neuronem. Spojení mezi neurony (synapse) tedy umožňuje přenášet signály z jednoho neuronu na druhý, přičemž neurony jsou schopné signály zpracovávat a rozesílat. V umělé neuronové síti se v prvé řadě transformují vstupní data použitím nelineární funkce na vážený součet vstupů. Tato transformace je definována jako neurální vrstva (*neural network*) s funkcí neurální jednotky (*neural unit*) [65]. Výstupy z jedné vrstvy jsou následně využívány jako vstup do další vrstvy, kdy se za pomoci opakovaných transformací kombinují do konečné vrstvy, která vytvoří předpověď. Trénink neuronové sítě spočívá v učení z poskytnutých dat, kdy se za pomocí změn hmotností (*weight*) a parametrů sítě učí tak, aby se minimalizoval rozdíl mezi predikcemi a požadovanými hodnotami [66]. Postup zpracování signálu může být matematicky vyjádřeno jako:

$$y(x) = f\left(\sum_{i=1}^{n} w_i\, x_i\right) \tag{6}$$

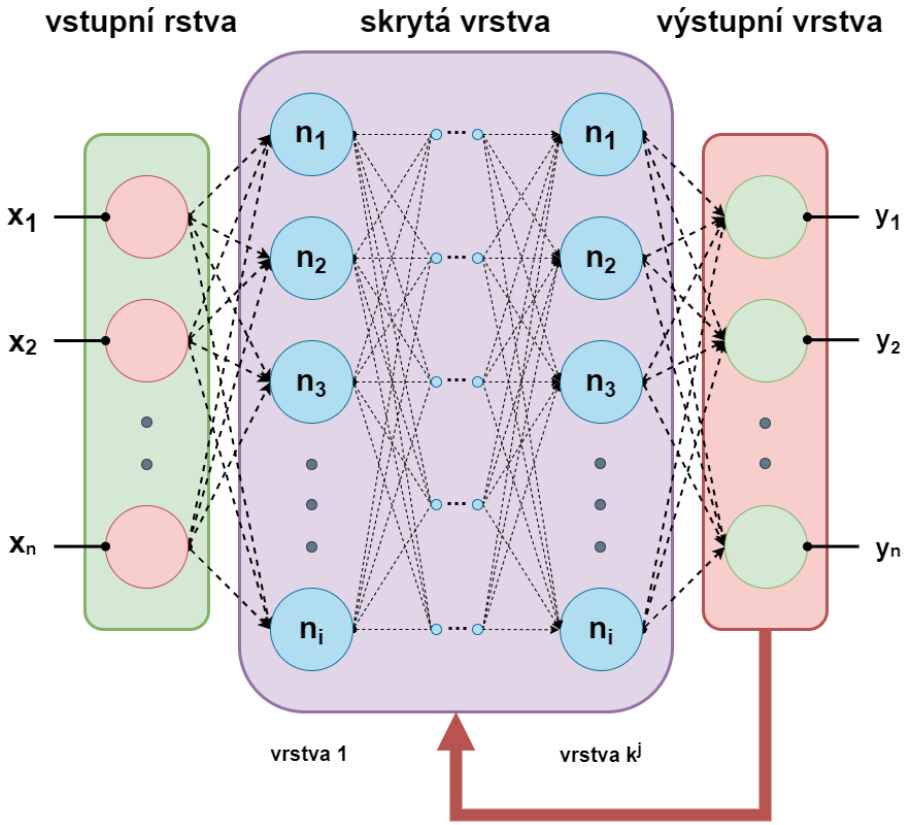
- y výstupní signál
- f aktivační funkce[4]
- x vstupní proměnná
- w hmotnost přiřazená ke každé vstupní proměnné [65]

Neuronová síť vzniká tehdy, pokud je propojeno vstupy a výstupy více neuronů, a je často rozdělena do tří vrstev (vstupní, skryté, výstupní). Pokud je v topologii sítě alespoň jedna skrytá vrstva, jedná se o hlubokou neuronovou síť. Mohou, ale také existovat neuronové sítě, které jsou rozdělené pouze na vstupní a výstupní vrstvu. Neuronové sítě lze také rozdělit na rekurentní (*reccurent neural network*) a dopřednou neuronovou síť (*feedforward neural network*). Dopředná neuronová síť (znázorněno na obr. 38) přenáší informace pouze z jedné vrstvy do další, nýbrž rekurentní neuronová síť (znázorněno na obr. 39) využívá paměti nebo smyčky zpětné vazby. [66]

[4] Aktivační funkce aplikují transformaci na vážená vstupní data (násobení matice mezi vstupními daty a váhami). Funkce může být lineární nebo nelineární [66].



Obr. 38: Schéma dopředné neuronové sítě



Obr. 39: Schéma rekurentní neuronové sítě

5.5 Deep Q-learning

Algoritmus Deep Q-learning neboli také DQN (*Deep Q-network*) kombinuje algoritmus zpětnovazebního učení Q-learning a hluboké neuronové sítě (obr. 40). Toto spojení vzniklo především za účelem redukce stavového prostoru, jelikož algoritmus Q-learning vytváří a následně ukládá stavy do Q-tabulky. Avšak při využití algoritmu Q-learning a hlubokých neuronových sítí jsou aproximovány Q-hodnoty, a tudíž nevzniká Q-tabulka. Své vysoké popularity nabyl tento algoritmus, kdy po 38 dnech trénování hry Atari 2600, byl schopen dosáhnout úrovně lidského hráče. Algoritmus vykazoval silnou schopnost rozhodování, a dokonce v některých hracích částech převyšoval výsledky člověka. Tak jako algoritmus Q-learning jedná se o algoritmus bez strategie a bez modelu. [67]

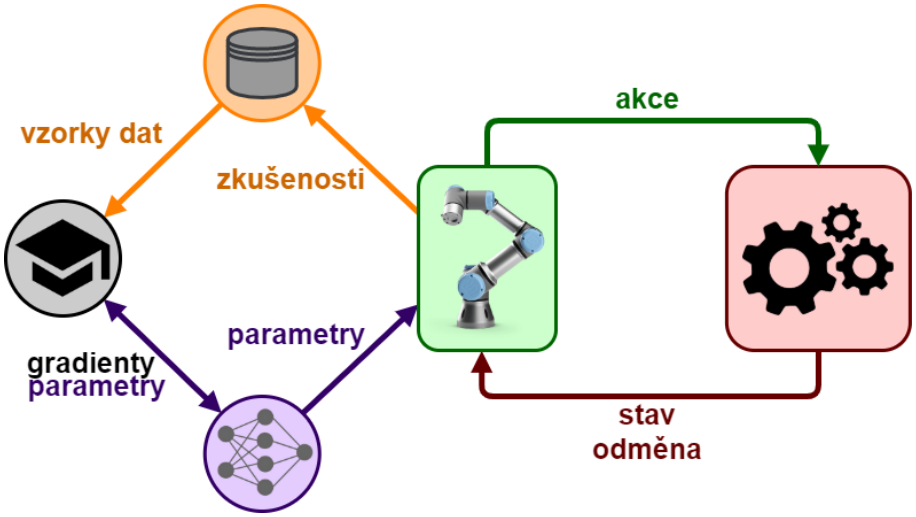
V algoritmu vyvstávají základní dvě techniky, využití cílové sítě (*target network*) a zkušenosti opakování (*experience replay*), k zvýšení rychlosti a výsledků trénování. Cílová síť je separována pro určení Q-hodnot během tréninku a je vždy aktualizována v přesně daném bodu cyklu, kde parametry $Q(s, a, w)$ z predikční sítě jsou zkopírovány do cílové sítě. Tento návrh vede ke stabilnějšímu tréninku, protože udržuje pevnou cílovou funkci [68]. Cílovou síť algoritmu DQN můžeme definovat jako:

$$T_t^{DQN} = R + \gamma \max_{a'} Q(s', a', w_t^-) \tag{7}$$

Kde T_t^{DQN} udává cílovou hodnotu funkce v čase t a w_t^- jsou parametry hmotnosti cílové sítě v čase t. Účelem cílové sítě je snížit korelaci mezi cílovou hodnotou a akční hodnotou, mimo jiné i tak, aby minimalizovala ztrátovou funkce $L(w)$:

$$L(w) = \left(R + \gamma \max_{a'} Q(s', a', w') - Q(s, a, w)\right)^2 \tag{8}$$

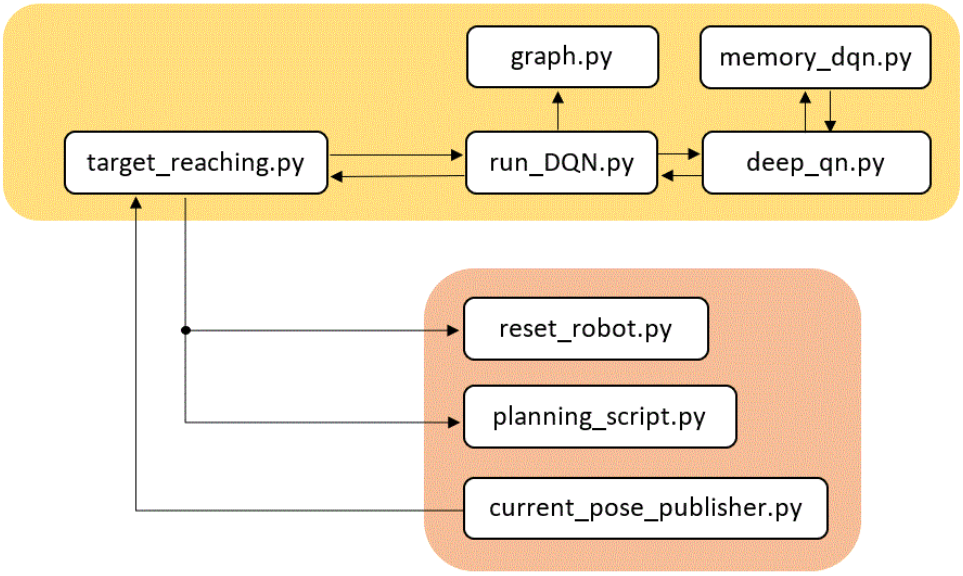
Zkušenosti opakování určují, kdy jsou trénovaná data z algoritmu náhodně vzorkována ze zkušební paměťové banky (*experience memory bank*). Banka výsledky běžícího systému ukládá a snižuje korelaci vzorků dat. [67]



Obr. 40: Schéma hlubokého zpětnovazebního učení

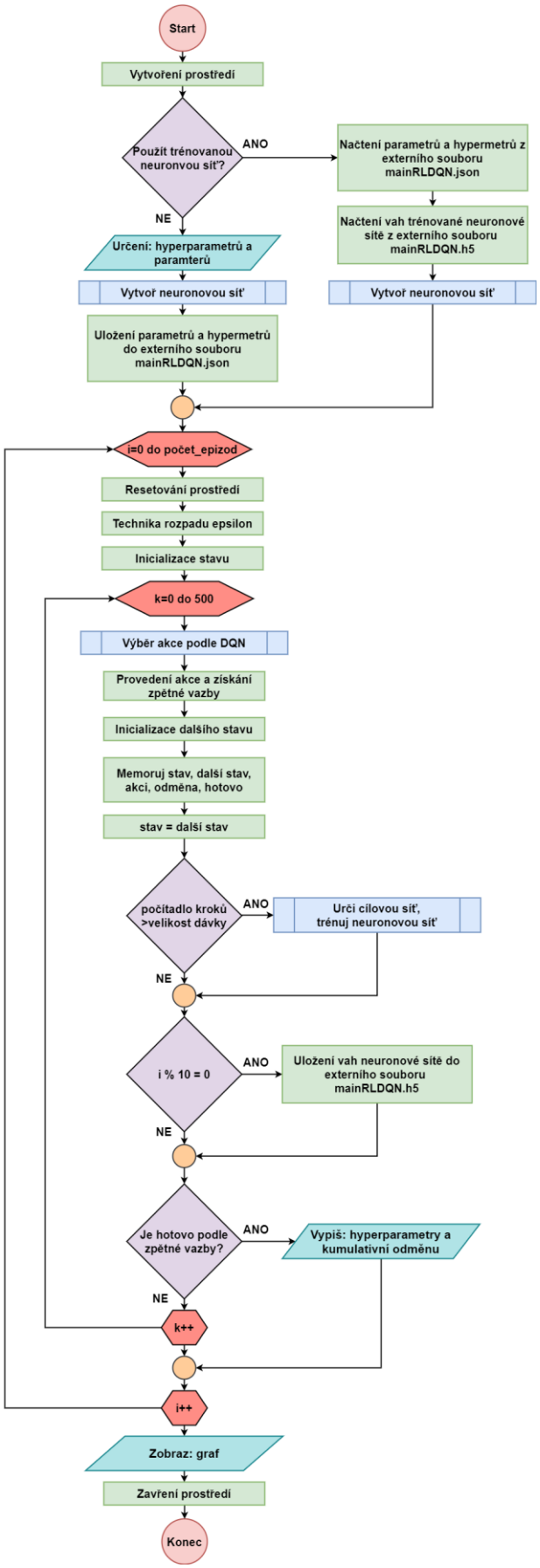
5.5.1 Návrh a implementace algoritmu Deep Q-learning

V této praktické části je popsána aplikace algoritmu Deep Q-learning. Záměrem tedy byla implementace algoritmu k učení modelu kolaborativního robota UR3 s koncovým efektorem k dosahování předem definovaného bodu. Při návrhu implementace bylo využito již vytvořeného balíku a část architektury již naprogramovaného zpětnovazebního algoritmu Q-learning z kapitoly 5.2.1. K řešení aplikace byla provedena částečná změna původního návrhu architektury a přibylo využití knihoven Keras a Tensorflow.



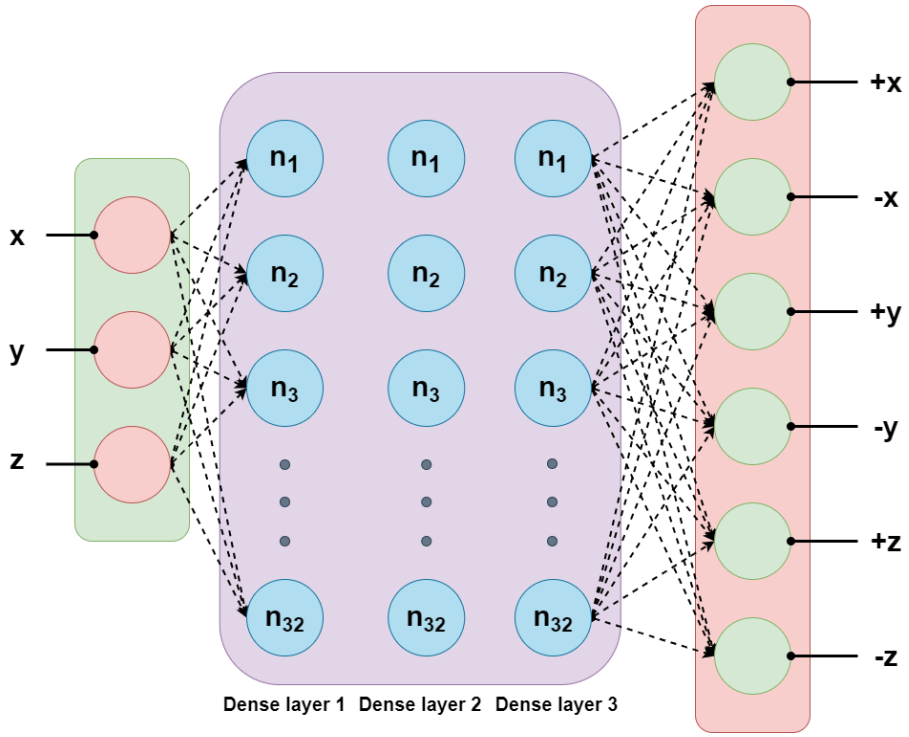
Obr. 41: Architektura python skriptů DQN

Návrh řešení zachovává skripty, které využívají knihovny frameworku MoveIT a také primárního skriptu propojující jádro implementace algoritmu se skripty obsluhující simulaci v simulačním prostředí (obr. 41). Zásadní změnou bylo vytvoření zcela nového skriptu *run_DQN.py* (obr. 42), který vytváří, inicializuje a řídí prostředí, definuje samotnou simulaci a také inicializuje hyperparametry, přičemž je ukládá spolu s dalšími parametry pro možné opakování učení. Dalším přidaným skriptem je *deep_qn.py*, který představuje samotné jádro algoritmu. V tomto skriptu je definována architektura neuronové sítě spolu s funkcemi ukládaní a načítání vah Q-sítě. Další nedílnou částí tohoto skriptu jsou funkce, které umožňují výpočet ze vstupních dat, výstup podle algoritmu Deep Q-learning. Posledním přidaným skriptem je *memory_dqn.py*. V tomto skriptu je obstaráváno ukládání vystupujících prvků v algoritmu při každé iteraci. Otázky, které bylo nutné řešit při aplikaci algoritmu Q-learning, jako je vhodné zvolení hyperparametrů nebo správné zvolení maximálního počtu kroků v epizodě, přetrvává i při implementaci algoritmu Deep Q-learning. Avšak u aplikace algoritmů hlubokého zpětnovazebního učení vyvstávají i problémy ve vhodném zvolení například optimalizátoru (Adam, Adagrad, Adatela a jiné) nebo aktivační funkce (ReLu, Sigmoid, Hyperbolic tangent).

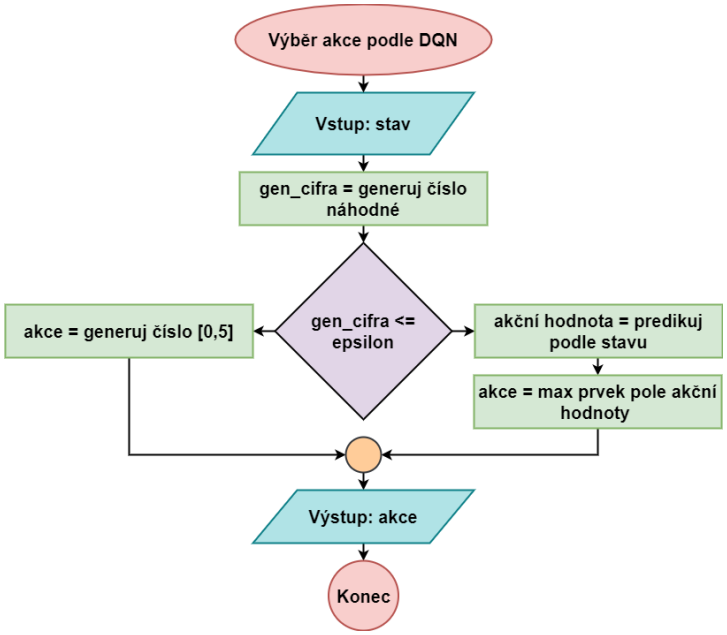


Obr. 42: Vývojový diagram *run_DQN.py*

Architektura vytvořené neuronové sítě obsahuje 3 plně propojené vrstvy (*Dense layer*), kde je v každé 32 neurálních jednotek s aktivační funkcí ReLu (*Rectifier Linear unit*). V této architektuře (obr. 43) je spojen každý neuron v jedné vrstvě s každým neuronem v jiné vrstvě. ReLu byla zvolena, jelikož je nejčastěji používanou aktivační funkci při návrhu hlubokých neuronových sítích. Jako optimalizátor byl použit optimalizační algoritmus Adam se ztrátovou funkci MSE (*Mean Squared Error*), který vykazoval pro daný úkol nejlepší výsledky.

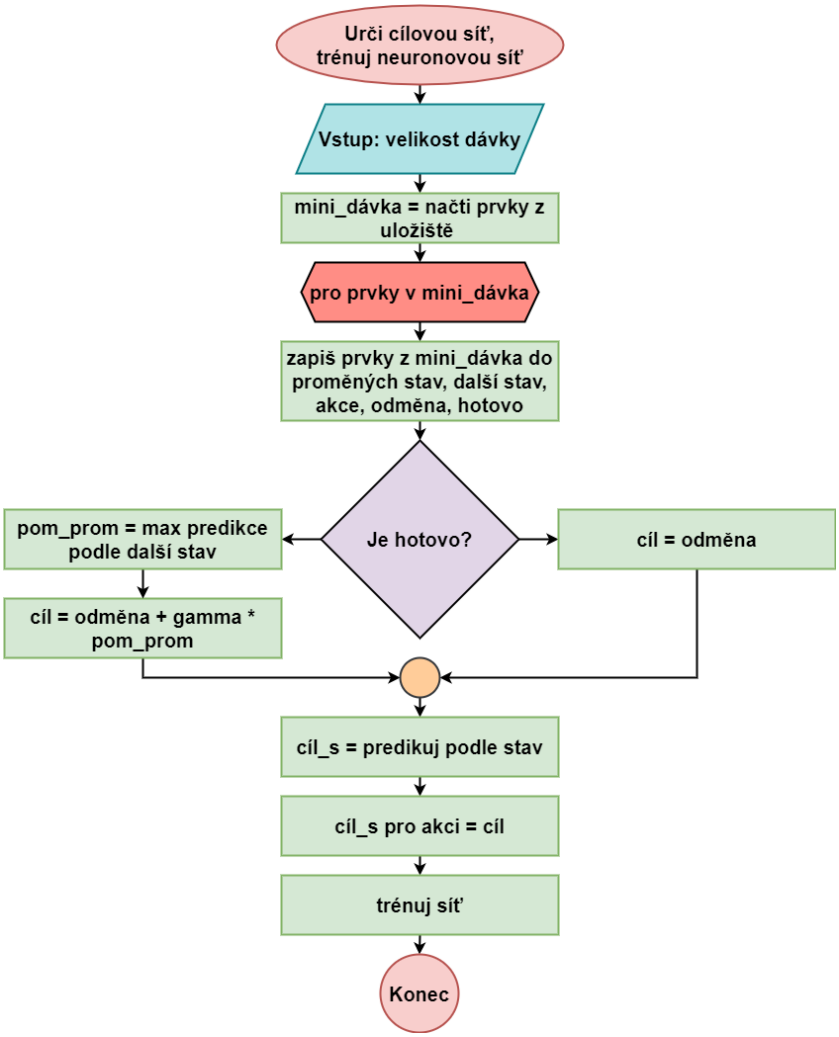


Obr. 43: Návrh architektury neuronové sítě



Obr. 44: Vývojový diagram funkce *chooseAction_DQN*

Další zásadní funkcí ve skriptu *deep_qn.py* je *chooseAction_DQN* (obr. 44), která má vstupní parametr stav a je zvolena daná akce. V poslední řadě svou zásadní úlohu ve skriptu má funkce *learn_DQN*, která má vstupní parametr velikost dávky. V této funkci je proveden výpočet cílové sítě a následné provedení trénování dle obr. 45.



Obr. 45: Vývojový diagram funkce *learn_DQN*

Další intervencí do původní aplikace je přidání skriptů *memory_dqn.py*. V tomto skriptu se ukládají parametry stav, další stav, akce, odměna, hotovo každé iterace. Prvky jsou ukládány do seznamů, kdy se po zavolání vrátí jako další seznam slovníků, přičemž každý klíč odpovídá danému prvku.

Proces učení algoritmu Deep Q-learning se spustí při stejném přístupu, jak v kapitole 5.2.1, kdy je nutné otevřít 3 terminály a v každém spustit následující příkazy.

```
user@user-pc:~$ roslaunch testos rl.launch

user@user-pc:~$ roslaunch testos moveit_planning.launch

user@user-pc:~$ rosrun testos run_DQN.py
```

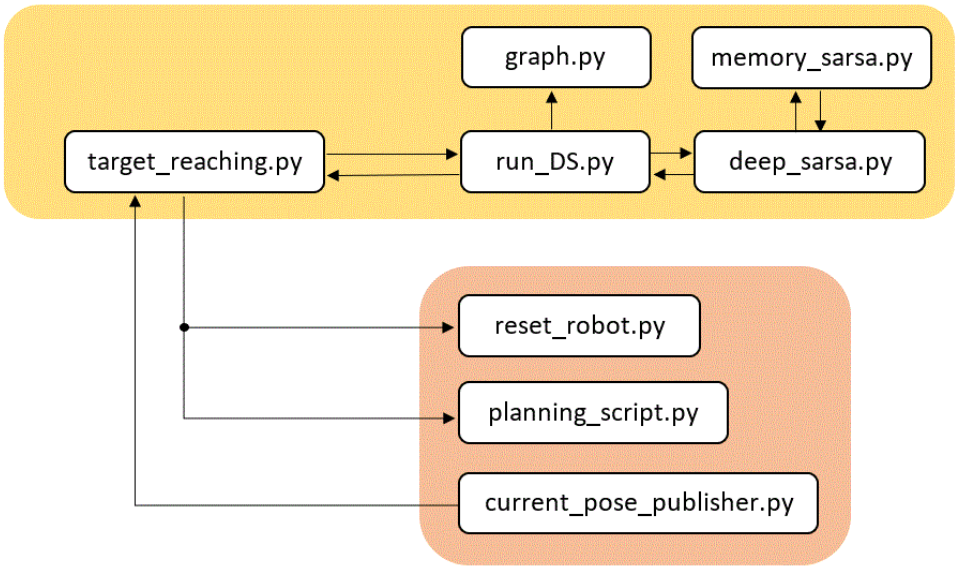
5.6 Deep SARSA

Algoritmus Deep SARSA má úzkou podobnost k algoritmu Deep Q-learning, avšak jedná se o algoritmus se strategií (*on-policy*) a bez modelu (*model-free*). Základ vychází z algoritmu SARSA a je rozšířen o hluboké neuronové sítě. Hlavním rozdílem oproti algoritmu Deep Q-learning je využití k učení i další akce, kdy ztrátovou funkci algoritmu Deep SARSA lze definovat jako:

$$L(w) = (R + \gamma\, Q(s', a', w') - Q(s, a, w))^2 \tag{9}$$

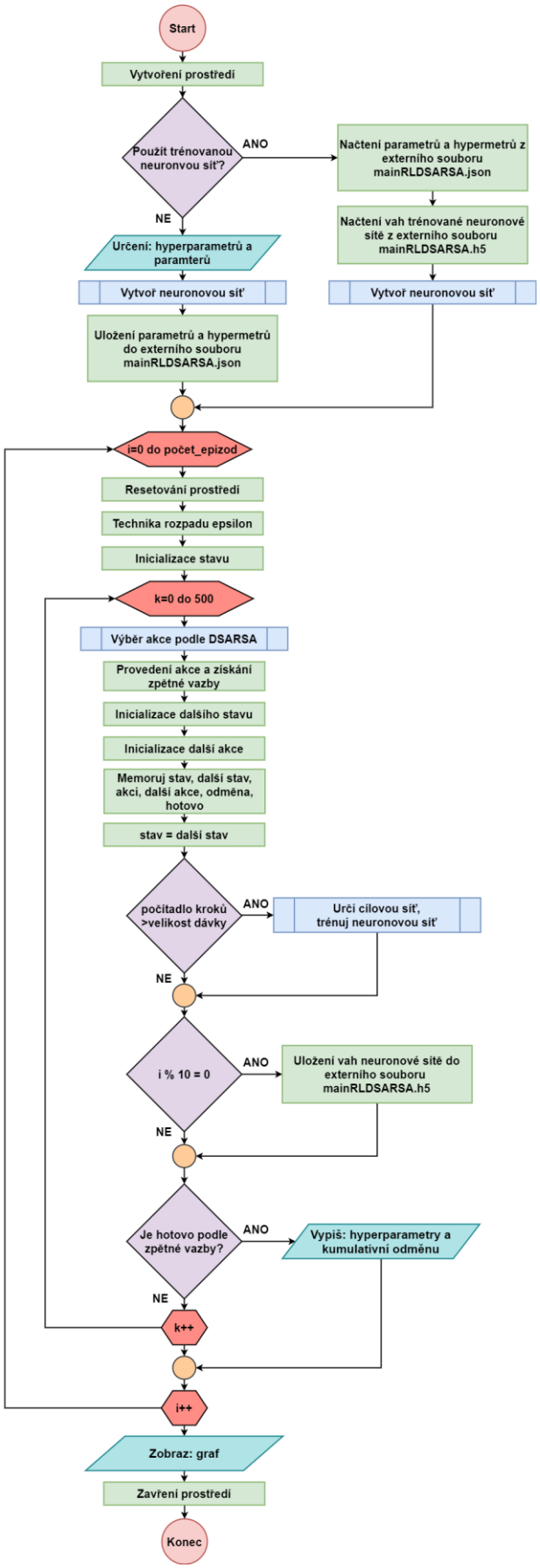
5.6.1 Návrh a implementace algoritmu Deep SARSA

K implementaci algoritmu Deep Q-learning, byla provedena i aplikace algoritmu Deep SARSA. Jelikož se jedná pouze o změnu použití algoritmu, nebyly provedeny žádné zásadní změny oproti implementaci v kapitole 5.5.1, a tedy bylo využito stejného úkolu učení ve stejném balíku, avšak byla částečně pozměněna architektura python skriptů (viz obr. 46).



Obr. 46: Architektura python skriptů Deep SARSA

Hlavní změna byla provedena ve skriptu *run_DS.py*, kdy byl přidán výpočet zvolení další akce (vývojový diagram na obr. 47). Tato změna také vedla k uzpůsobení samotné funkce *learn_DSARSA* (určení cílové sítě a následnému trénování) ve skriptu *deep_sarsa.py*. Vstupními parametry funkce *learn_DSARSA* zůstává velikost dávky, avšak při výpočtu cílové sítě je využito i parametru další akce. Tuto změnu lze vidět na vývojovém diagramu na obr. 48.



Obr. 47: Vývojový diagram *run_DS.py*

Obr. 48: Vývojový diagram funkce *learn_DSARSA*

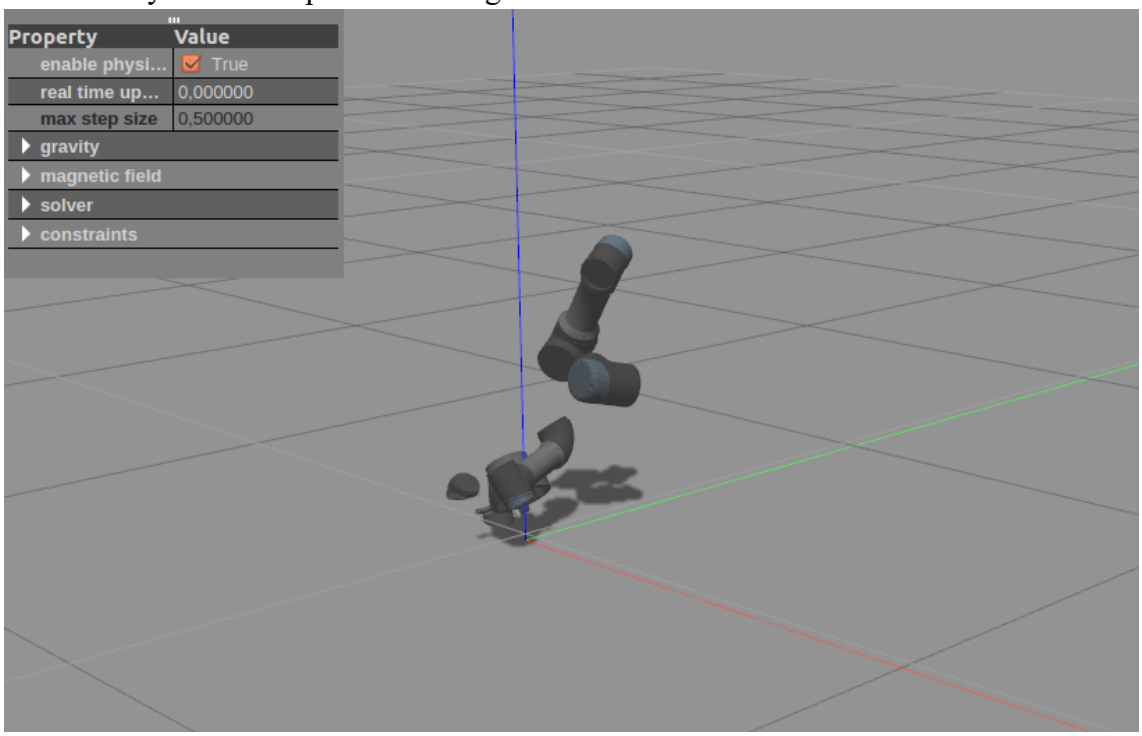
Nedílnou součástí změn bylo i uzpůsobení skriptu *memory_sarsa.py*, kdy se k ukládaným parametrům stav, další stav, akce, odměna, hotovo každé iterace přidá i parametr další akce. Architektura hluboké neuronové sítě zůstala stejná i samotné její nastavení.

Algoritmus učení Deep SARSA lze spustit, jak v kapitole 5.2.1, kdy je nutné otevřít 3 terminály a v každém spustit následující příkazy.

```
user@user-pc:~$ roslaunch testos rl.launch

user@user-pc:~$ roslaunch testos moveit_planning.launch

user@user-pc:~$ rosrun testos run_DS.py
```

5.7 Výsledky testování

K tomu, aby bylo možné v jistém časovém horizontu získat výsledky, lze nastavením parametrů fyziky v simulačním prostředí Gazebo docílit nepatrného urychlení času simulace. Jedná se především o parametry *real time factor*[5] a *max step size*[6]. Nevýhodou je, že pokud se nastaví parametr *max step size* na příliš vysokou hodnotu a není k dispozici dostatečně velký výpočetní výkon, fyzikální engine kolabuje a dojde k deformaci modelu v simulačním prostředí, a tím pádem i k pádu celého programu, se kterým souvisí i činnost dalších skriptů (obr. 49). Další nevýhoda, která byla vypozorována při testování, je kladení velkého požadavku na paměť RAM u zpětnovazebních algoritmů (viz obr. 50 obr. 51). Po příliš dlouhé simulaci došlo k zamrznutí celého hostitelského operačního systému. Poslední nevýhodou je nemožnost nastavit v simulačním prostředí Gazebo rozložení výkonů mezi procesorem a grafickou kartou.



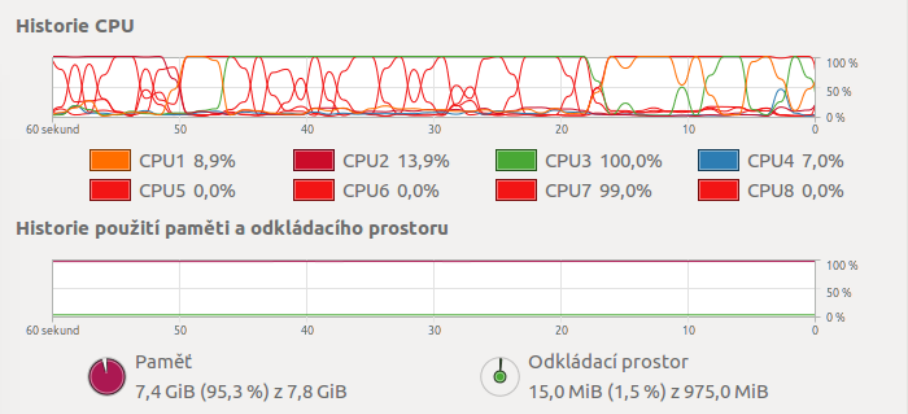
Obr. 49: Gazebo při pádu ODE enginu

```
[spawn_gazebo_model-4] process has finished cleanly
log file: /home/juricekm/.ros/log/f51d343e-5a7f-11ea-b2c3-7085c23c13af/spawn_gaz
ebo_model-4*.log
terminate called after throwing an instance of 'std::runtime_error'
  what():  Time is out of dual 32-bit range
Aborted (core dumped)
[gazebo-2] process has died [pid 10944, exit code 134, cmd /opt/ros/kinetic/lib/
gazebo_ros/gzserver -e ode worlds/empty.world __name:=gazebo __log:=/home/jurice
km/.ros/log/f51d343e-5a7f-11ea-b2c3-7085c23c13af/gazebo-2.log].
log file: /home/juricekm/.ros/log/f51d343e-5a7f-11ea-b2c3-7085c23c13af/gazebo-2*
.log
```

Obr. 50: Výpis chybové hlášky při přeplněné RAM paměti

[5] *Real time factor* definuje, jak se simulace bude lišit od reálného času [69].

[6] *Max step size* je maximální velikost časového kroku u výpočtu ve fyzikálním enginu, kterého lze během simulace získat [69].



Obr. 51: Stav RAM paměti po delší době simulace

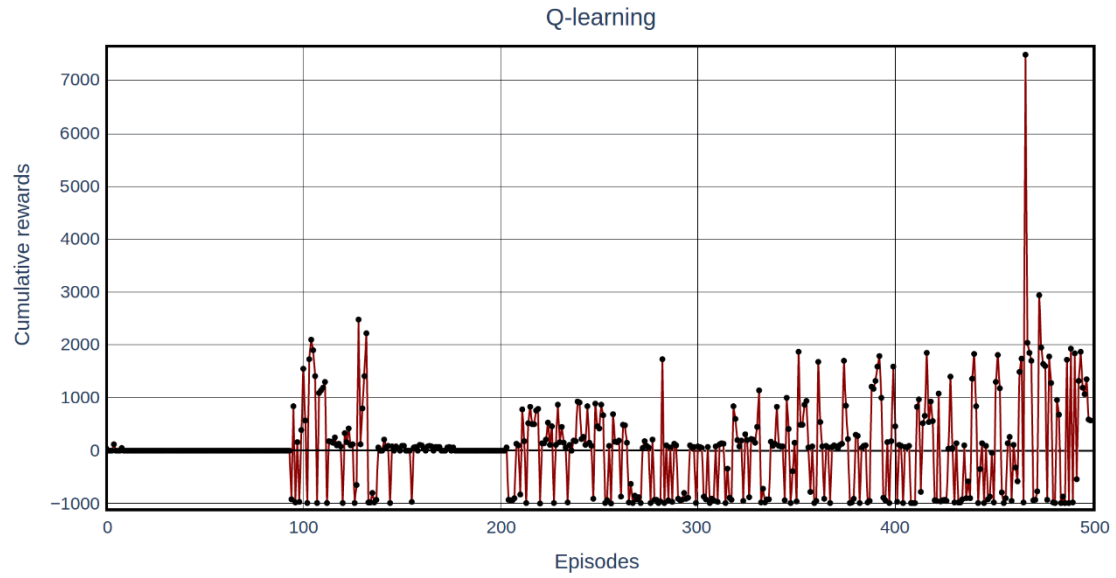
Aby bylo možné analyzovat proces učení daných algoritmů, je využit skript *graph.py*, který vykresluje kumulativní odměny k jednotlivým epizodám. Dalším krokem je určení parametrů. V prvé řadě je definován požadovaný bod, kdy je následně určena velikost jednoho kroku pohybu pro plánování a samotnou simulaci pohybu. Dalším nutným specifikovaným parametrem je počet kroků v epizodě, přičemž tento parametr úzce souvisí s velikostí kroku pohybu. V neposlední řadě je nutné definovat hyperparametry a celkový počet epizod. Pro algoritmy hlubokého zpětnovazebního učení je nutné určit i parametry jako velikost paměti, velikost dávky, začátek učení. V poslední řadě je specifikována funkce odměn.

V řešení aplikace je použit návrh obdržení odměny ze vzdálenosti od požadovaného cíle se záměrem co nejmenší komplexnosti získávání odměn. Agent se může pohybovat v daném rozmezí od cíle, avšak pokud se dostane za definované hranice, získá velmi nízkou odměnu a je proveden reset. Pokud se však agent bude pohybovat blíže k cíli, tím větší odměnu obdrží, kdežto pokud se bude oddalovat od cíle, může se dostat do oblasti, kdy neobdrží žádnou odměnu. Aby bylo možné jednoznačně určit, že se agent dostal do požadovaného cíle a splnil úkol, obdrží velmi vysokou hodnotu odměny.

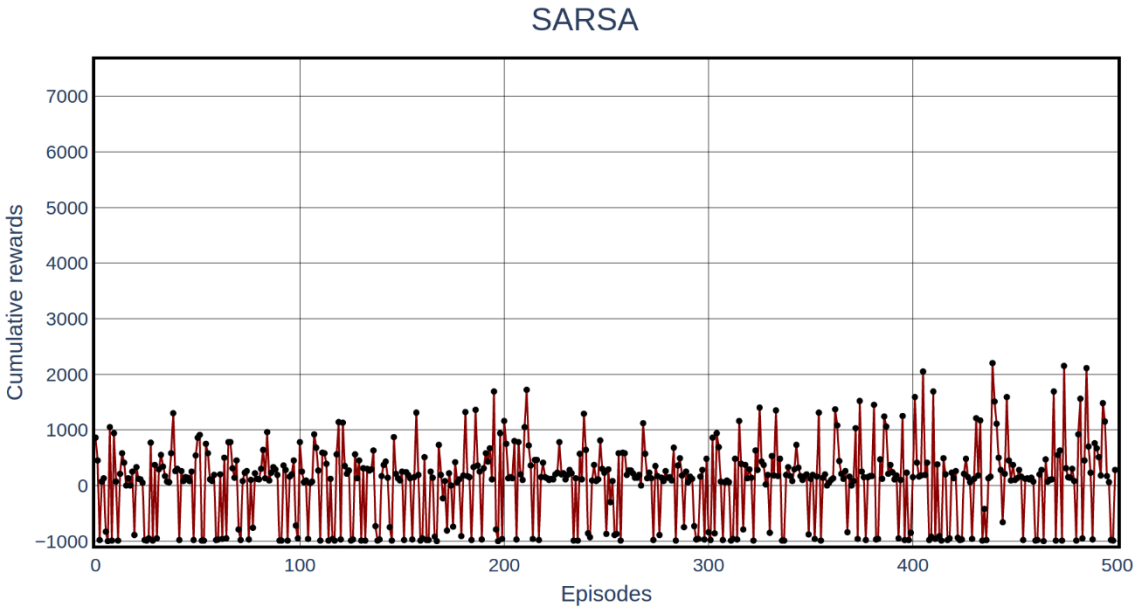
V prvé řadě bylo provedeno testování za účelem vhodného určení hyperparametrů alfa, gama a epsilon. Dle teorie a výsledku z testování jsou zvoleny k výslednému testování hodnoty hyperparametrů:

- Q-learning, SARSA (alfa = 0,15, gama = 0,85, epsilon = 0,9, epsilon_discount = 0,999)
- Deep Q-learning, Deep SARSA (alfa = 0,01, gama = 0,95, epsilon = 0,9, epsilon_discount = 0,999)

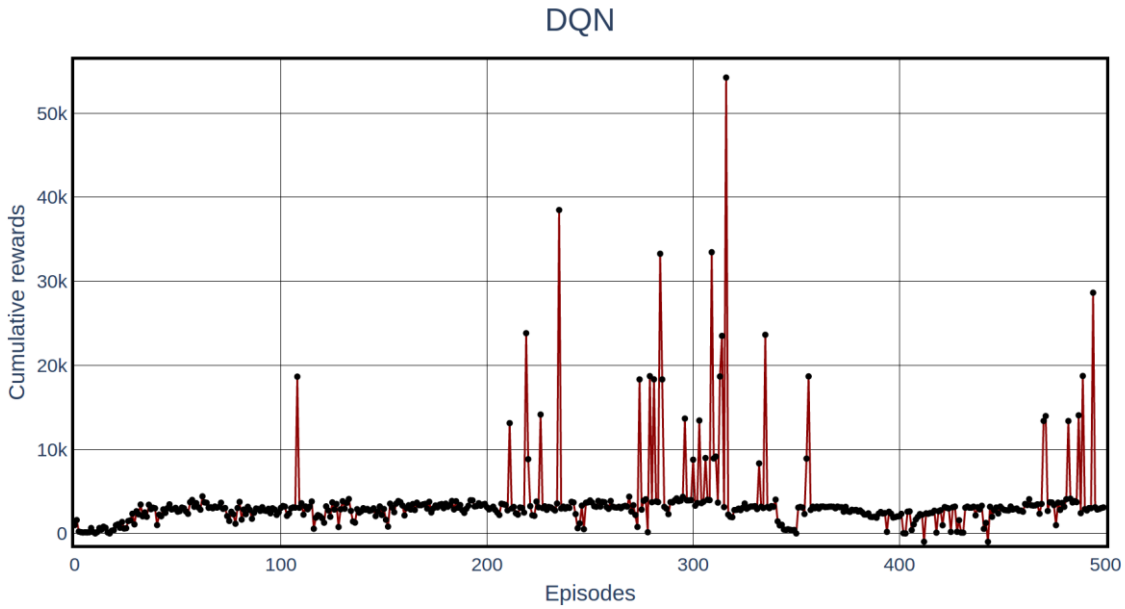
Aby bylo možné určit a porovnat výsledky učení, je zvoleno 50 kroků v epizodě a 500 epizod, přičemž velikost jednoho kroku pohybu je zvolena 0,005 a jednotná poloha cílového bodu.



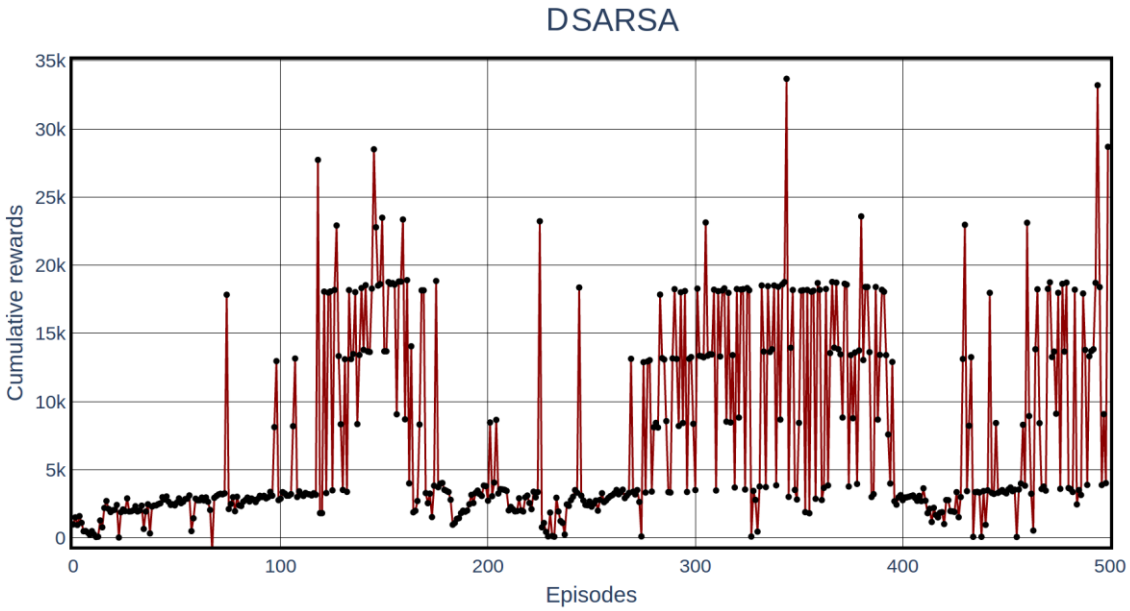
Obr. 52: Výsledky učení algoritmu Q-learning



Obr. 53: Výsledky učení algoritmu SARSA



Obr. 54: Výsledky učení algoritmu Deep Q-learning



Obr. 55: Výsledky učení algoritmu Deep SARSA

Algoritmus Q-learning se podle výsledků učí rychleji a dosáhl také vyšší kumulativní odměny oproti algoritmu SARSA, přitom také dosáhl definovaného cíle (obr. 52). Toto chování je způsobeno učením se bez strategie, což je základní vlastnost algoritmu Q-learning. Během testování bylo vypozorováno, že algoritmus Q-learning vykazoval více prohledávacích pohybů, kdežto algoritmus SARSA (obr. 53) prokazoval hledání optimálnější cesty a postupnější učení, kdy k dosažení definovaného cíle potřeboval více epizod učení.

Ačkoliv z grafů lze vidět, že při běhu programu nebylo nabyto příliš velkého progresu učení v rámci kumulativní odměny, přesto se agent dokázal během učení přibližovat k danému cíli a zlepšovat tak trajektorii. Výsledky testování nepřinesly zcela uspokojení, ale tento fakt byl částečně očekáván, jelikož se jedná o velmi komplexní problém pro algoritmus Q-learning nebo SARSA. Podobné výsledky mohou nastat i při použití těchto algoritmů k učení drona, kde je také možné provádět pohyb ve směru x, y, z [70].

Algoritmus Deep Q-learning (obr. 54) oproti algoritmu Deep SARSA (obr. 55) prokazoval o něco horší výsledky netrénovaného učení v rámci kumulativní odměny. Přesto oba algoritmy nalezly definovaný bod několikrát během daného učení. Hlavní předností těchto algoritmů oproti algoritmům Q-learning a SARSA je v možnosti trénování neuronové sítě. Algoritmy hlubokého zpětnovazebního učení vykazují mnohonásobně lepší výsledky oproti algoritmům zpětnovazebního učení při řešení daného problému, přičemž své využití mohou nacházet i například při řešení problému parkování osobních automobilů nebo rozpoznávaní státních poznávacích značek.

6 ŘÍZENÍ KOLABORATIVNÍHO ROBOTA UR3

V této praktické části je popsáno řízení reálného kobota UR3 pomocí frameworku ROS a jeho následné testování. Aby bylo možné propojit kolaborativního robota s ROS, je nutné na řídící jednotku, v podobě například stolního nebo jednodeskového počítače, instalovat hostitelský operační systém. Pro testování byl použit operační systém Ubuntu Xenial a distribuce ROS Kinetic Kame. V prvé řadě, aby došlo ke komunikaci, je nutné vytvořit catkin[7] pracovní prostor, kde je nutné implikovat metabalík universal_robots s balíkem ur_modern_driver. Balík ur_modern_driver obsahuje vylepšenou verzi ovladače ur_driver z metabalíku universal_robots. Metabalík i balík jsou standardizováni, jelikož spadají pod projekt ROS Industrial. K samotnému propojení je následně využíváno uživatelské rozhraní kobotů od firmy Universal Robots Polyscope a ovladač ur_modern_driver. Ovladač ur_modern_driver komunikuje s kolaborativním robotem skrz ethernet, přičemž komunikace probíhá v počítačové síti sadou protokolů TCP/IP. Propojení závisí na nepřetržité a spolehlivé komunikaci, aby nedocházelo k výpadkům ovladačů a vnitřních komponent ROS. Z tohoto důvodu není možné zcela spolehlivě řídit kolaborativního robota z virtuálního stroje. K samotnému řízení a plánování trasy je využíváno frameworku MoveIT, kdy metabalík universal_robots integruje základní balíky umožňující inicializaci a plánování trasy. K programování a řízení z uživatelského hlediska lze přistupovat pomocí nástroje Rviz nebo editoru zdrojového kódu. Před testováním pohybu je provedena simulace v programu Gazebo.

6.1 Simulace trajektorie pohybu

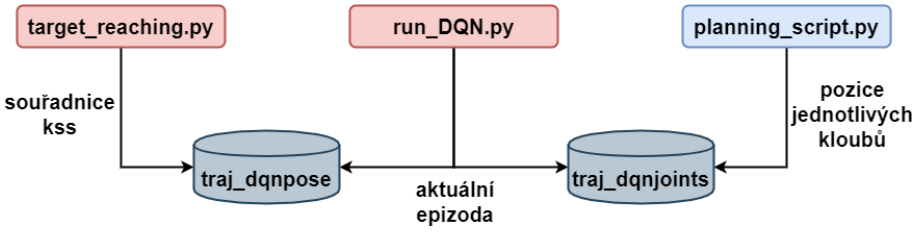
6.1.1 Trajektorie pohybu algoritmů zpětnovazebního učení

Aby bylo možné využít trajektorie z učení algoritmů Q-learning, SARSA, Deep Q-learning a Deep SARSA je nutné provést malé úpravy původních verzí aplikace z kapitoly 5.2.1, 5.3.1, 5.5.1, 5.6.1. Tyto úpravy vedou pouze ke změně v podobě zapisování aktuálních pozic jednotlivých kloubů a souřadnic kartézského souřadnicového systému do externích textových souborů.

Obr. 56: Schéma zápisu dat do externích souborů pro algoritmus Q-learning

[7] Catkin je kolekce CMake maker a přidruženého kódu používaného k vytváření balíčků používaných v ROS [71].

Své využití tyto externí soubory nabydou při analýze a následného zpracování k řízení hmotného kolaborativního robota UR3. Do externích souborů se zapíše vždy na začátku číslo epizody a následně dle kroků v dané epizodě se postupně ukládají jednotlivé souřadnice. Návrh implementace pro algoritmus Q-learning (obr. 56) a Deep Q-learning (obr. 57), je obdobný jak pro algoritmy SARSA a Deep SARSA.



Obr. 57: Schéma zápisu dat do externích souborů pro algoritmus DQN

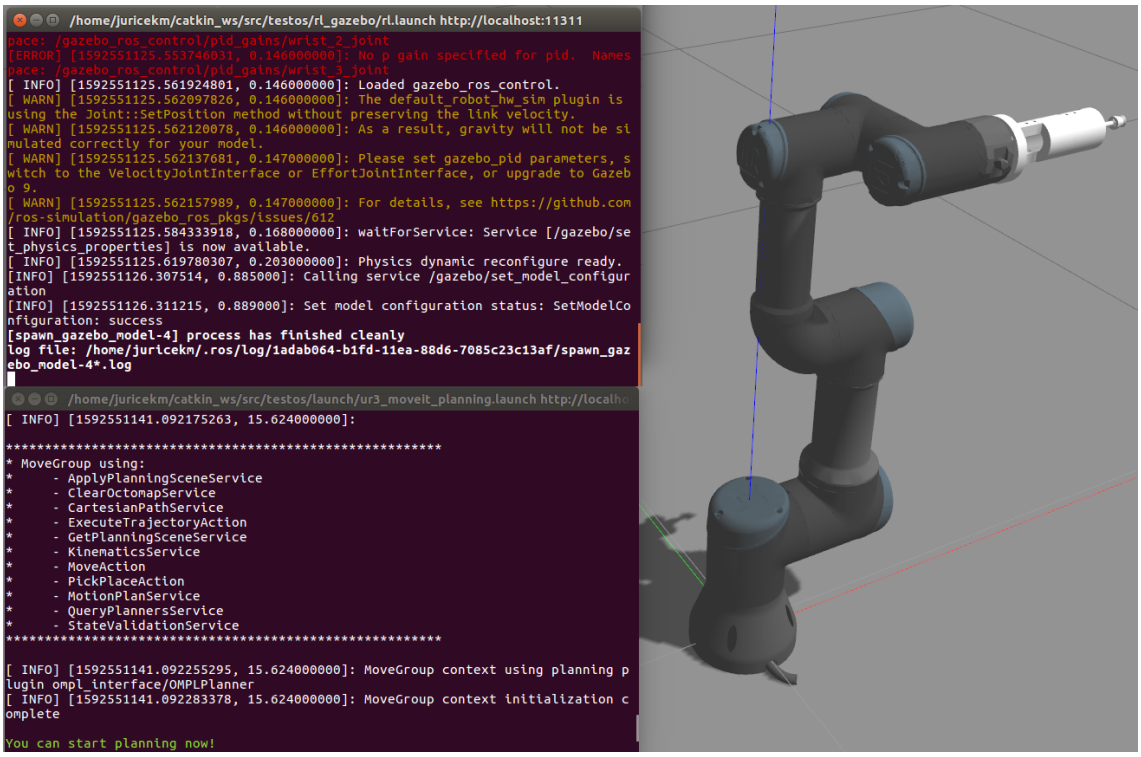
6.1.2 Trajektorie pohybu plánovacích algoritmů z frameworku MoveIT

K simulování pohybu v programu Gazebo je využíváno již naprogramovaných spouštěcích souborů z vytvořeného ROS balíku (kapitola 5.2.1). Řízení simulačního modelu lze pomocí frameworku MoveIT, z nakonfigurovaných souborů, které jsou také integrovány v balíku. Tyto soubory inicializují uzly a umožní plánovaní pohybu. Spuštění modelu kobota UR3 v simulačním prostředí Gazebo (obr. 58), spolu se spouštěcím souborem k řízení pomocí frameworku MoveIT lze napsáním do nových terminálů:

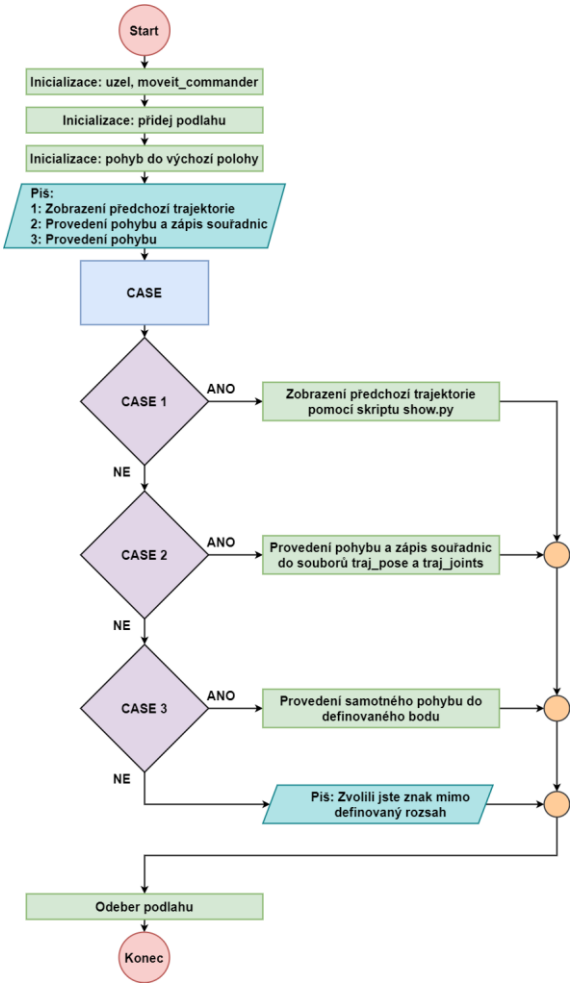
```
user@user-pc:~$ roslaunch testos rl.launch

user@user-pc:~$ roslaunch testos
ur3_moveit_planning.launch sim:=true
```

Skript *s_traj.py* slouží primárně k simulování pohybu v programu Gazebo, kdy při simulaci jsou zapisovány do externích souborů aktuální pozice jednotlivých kloubů a souřadnice kartézského souřadného systému k určenému cíli (obr. 59). Tento skript využívá k pohybu do definovaného bodu již naprogramovaný plánovací algoritmus *RRTstarkConfigDefault* z knihovny OMPL frameworku MoveIT. Mezi další funkce skriptu patří provedení pohybu do daného cíle bez zápisu do externích souborů nebo zobrazení trajektorie pomocí knihovny Plotly. Spuštění skriptu *s_traj.py* lze napsáním do nového terminálu:

```
user@user-pc:~$ rosrun testos s_traj.py
```



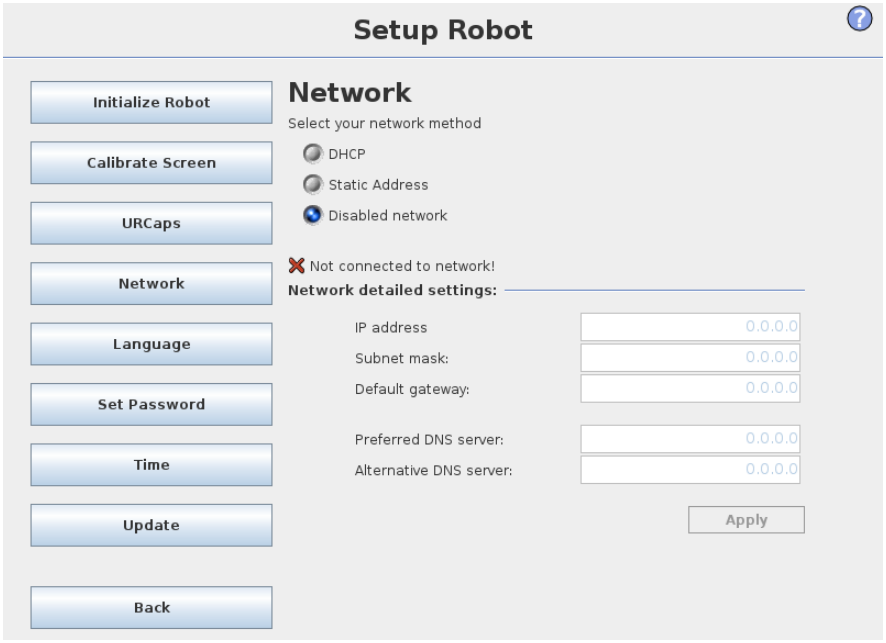
Obr. 58: Zobrazení modelu kobota UR3 v simulačním prostředí Gazebo



Obr. 59: Vývojový diagram skriptu *s_traj.py*

6.2 Propojení reálného robota s ROS

Aby bylo možné propojit kolaborativního robota s ROS, je využito ovladače ur_modern_driver, který je navržen tak, aby transparentně nahradil ovladač ur_driver z metabalíku universal_robots. Ovladač ur_modern_driver integruje již některé vyřešené problémy ovladače ur_driver, a také byla zlepšena použitelnost a kompatibilita. V prvé řadě je spuštěno uživatelské rozhraní Polyscope (obr. 63), kde je následně nastavena IP adresa kolaborativního robota (viz obr. 60). Pokud se jedná o simulaci na řídící jednotce je dosazena adresa 127.0.0.1.



Obr. 60: Nastavení sítě v prostředí Polyscope

Následně je spuštěn ovladač ur_modern_driver spolu se spouštěcím souborem frameworku MoveIT z balíku ur3_moveit_config z metabalíku universal_robots. Spouštění ovladače a řídících soborů dojde napsáním do nových terminálů:

```
user@user-pc:~$ roslaunch ur_modern_driver ur3_bringup.launch robot_ip:=robot_ip_address

user@user-pc:~$ roslaunch ur3_moveit_config ur3_moveit_planning_execution.launch limited:=true
```

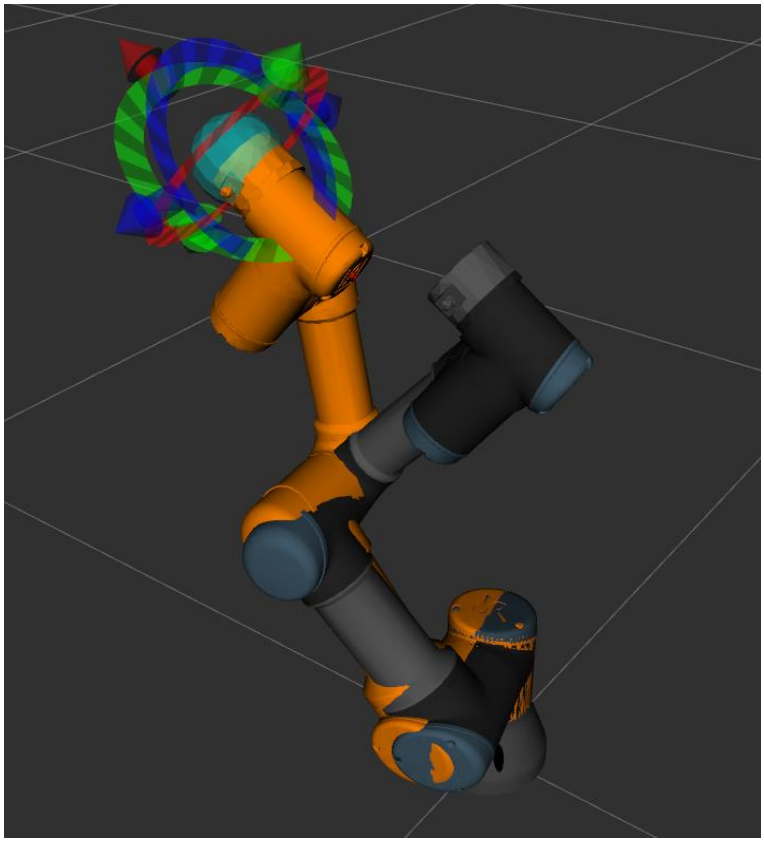
6.3 Testování řízení na kolaborativním robotu UR3

6.3.1 Ovládání kolaborativního robota pomocí Rviz

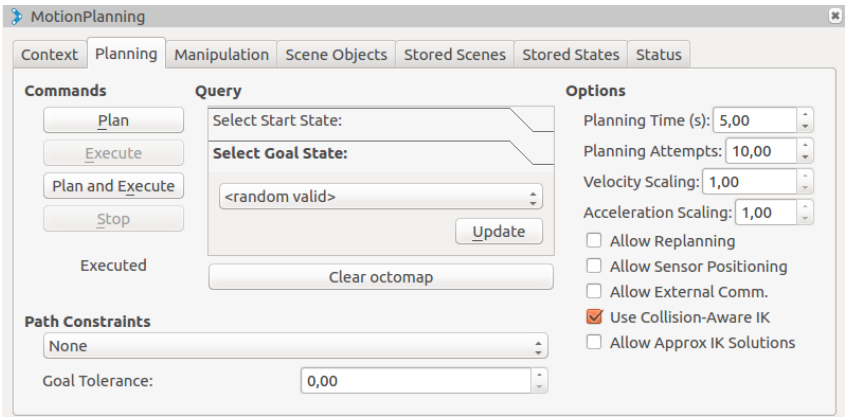
Po propojení kolaborativního robota a ROS, lze také k řízení využít nástroj Rviz. Do tohoto 3D vizualizačního nástroje je možné integrovat díky frameworku MoveIT hned několik základních funkcí, jako je MotionPlanning, PlanningScene, RobotState nebo

Trajectory. Funkce MotionPlanning je základní funkcí umožňující řízení. V této funkci lze nastavit například plánovací čas, rychlost či zrychlení nebo zvolení preferovaného plánovacího algoritmu. S definováním plánovacích algoritmů také souvisí tabulka, kde je možné spravovat parametry algoritmů (obr. 64). Dalšími možnostmi je kupříkladu definování pracovního prostoru nebo specifikace vizualizace samotného modelu. Použití Rviz k řízení umožňuje uživatelsky příjemné programování, v kterém lze velmi jednoduše nastavit základní parametry, následně naplánovat a provést daný pohyb kobota. Nejjednodušší možnost, jak naprogramovat pohyb kobota, je pomocí grafického okna, kde jen stačí zobrazit pomocný model, který je určen k plánování. S tímto modelem lze interagovat díky manipulačnímu nástroji v podobě koule a přidruženým mezikružím s šipkami. Pomocí koule a šipek lze manipulovat modelem ve směru x, y, z, přičemž mezikružím je možné definovat náklon koncového efektoru (viz obr. 61). Dalším krokem je jen stisknutí tlačítka *Plan and Execute* (obr. 62) a kolaborativní robot provede daný pohyb. Ke spuštění nástroje Rviz spolu s modelem kolaborativního robota UR3 a integrovaných funkcí z frameworku MoveIT je využíváno již naprogramovaných spouštěcích souborů v balíku ur3_moveit_config z metabalíku universal_robots. Samotné spuštění bude provedeno po napsání do nového terminálu:

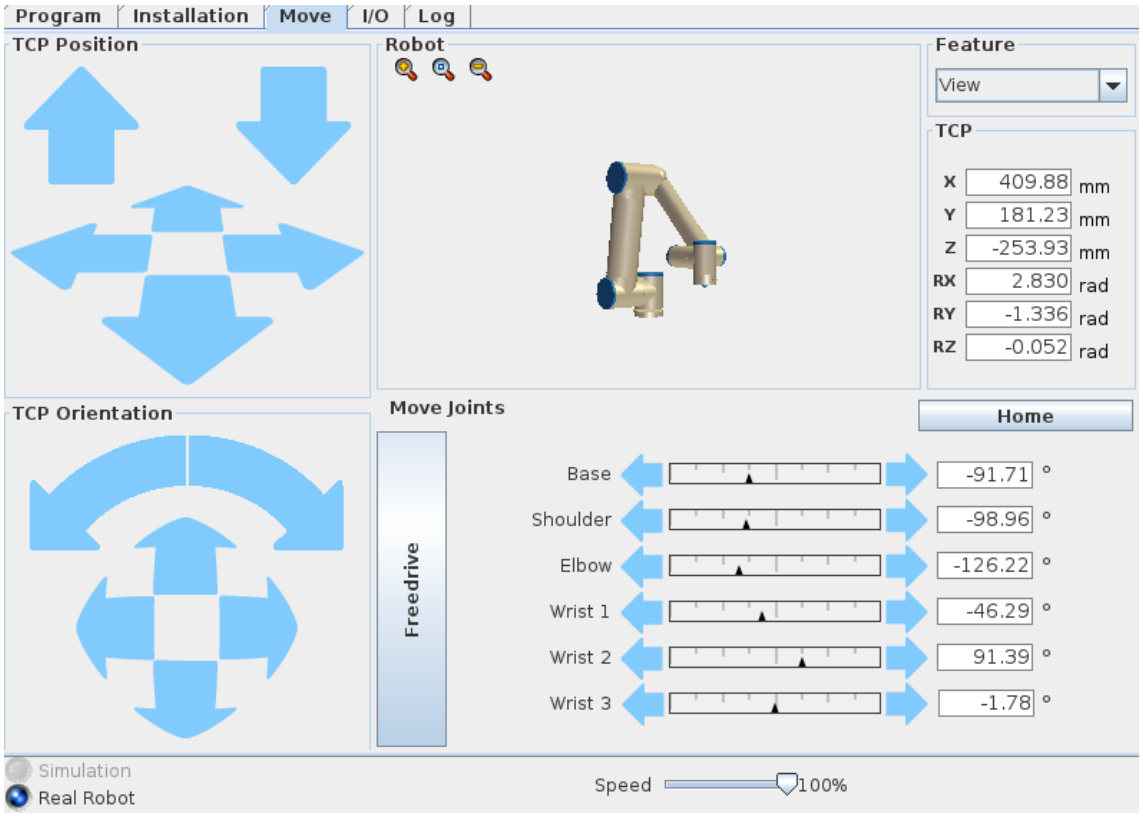
```
user@user-pc:~$ roslaunch ur3_moveit_config
moveit_rviz.launch config:=true
```



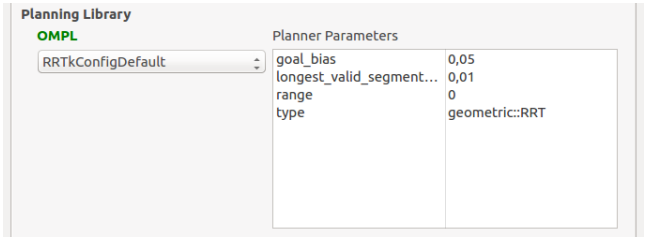
Obr. 61: Grafické okno Rviz



Obr. 62: Řídící funkce MotionPlanning ve Rviz



Obr. 63: Uživatelské rozhraní Polyscope

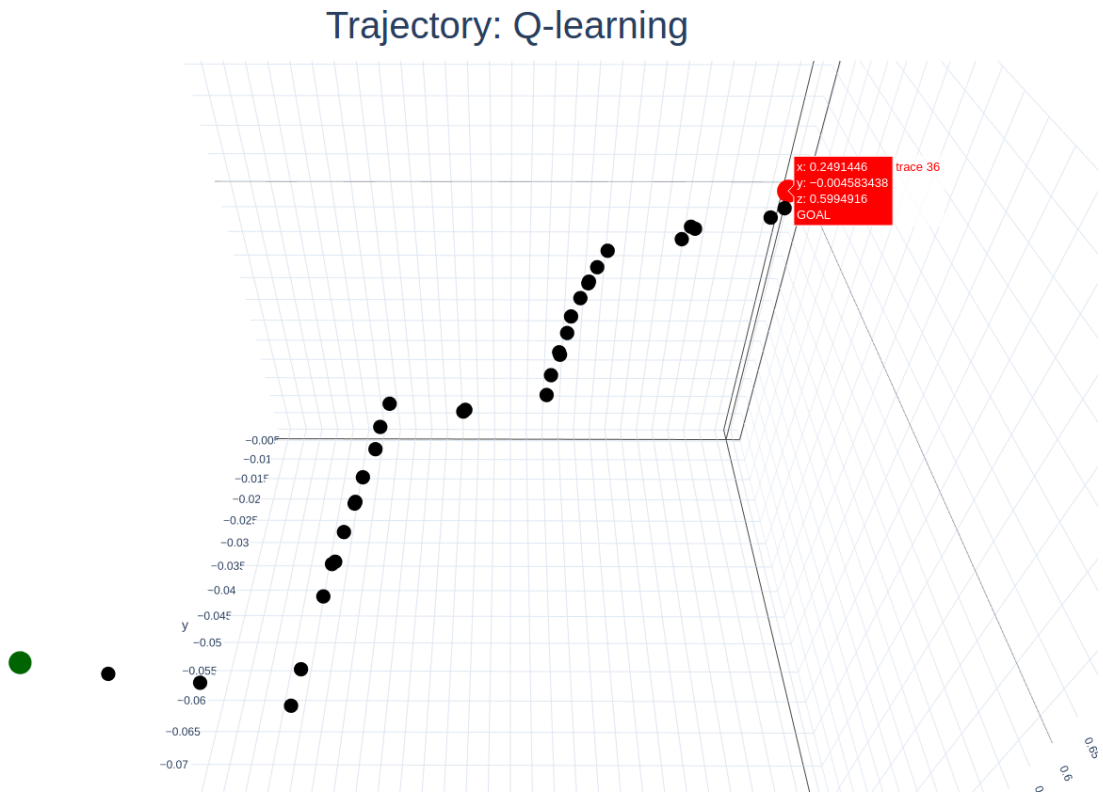


Obr. 64: Záložka s nastavením parametrů plánovacích algoritmů ve Rviz

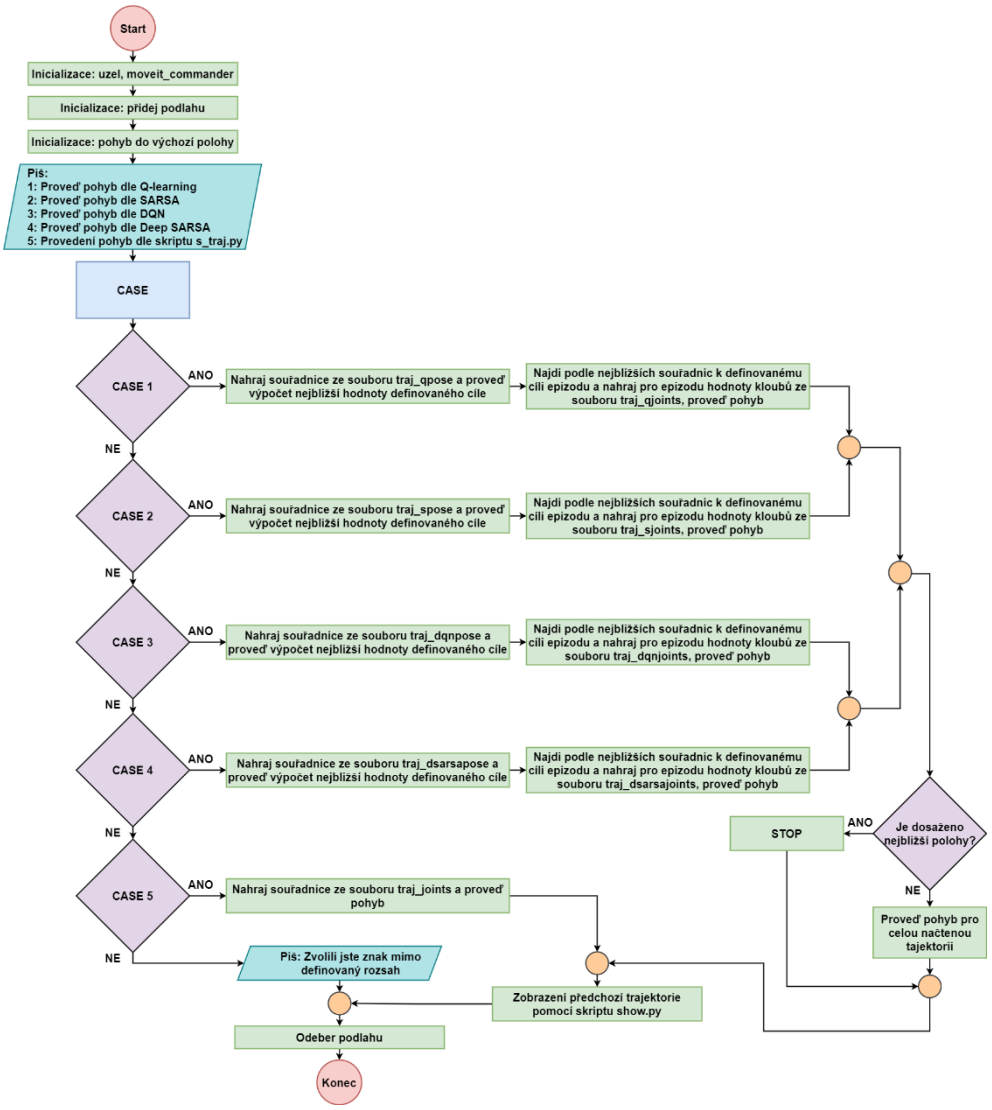
6.3.2 Ovládání kolaborativního robota pomoci řídícího programu

Řídící program kolaborativního robota UR3 je strukturován do python skriptu *r_traj.py* (obr. 66). Tento skript využívá především knihovny frameworku MoveIT k řízení a plánování trasy, ale je také aplikována knihovna Plotly k vykreslení dané trajektorie po provedení pohybu. Hlavním úkolem tohoto řídícího programu je načíst data v podobě souřadnic z textových souborů, zpracovat a následně provést pohyb po zvolené trajektorii. Skript *r_traj.py* obsahuje funkci, která analyzuje výsledky učení algoritmů jak zpětnovazebního učení, tak i hlubokého zpětnovazebního učení, přičemž funkce následně provede pohyb po trajektorii, která se nejvíce přiblížila k definovanému cíli (obr. 65, obr. 67, obr. 68, obr. 69). Další funkcí, kterou skript *r_traj.py* nabízí, je zpracování již předem simulované trajektorie ze skriptu *s_traj.py*, kde je použito pouze plánovacích algoritmů z frameworku MoveIT. Výsledky testování na reálném robotu lze vidět na videu v příloze, kdy nejoptimálnější cesty bylo nalezeno při testování algoritmu Deep SARSA. Spuštění samotného řídícího programu lze spustit napsáním do nového terminálu:

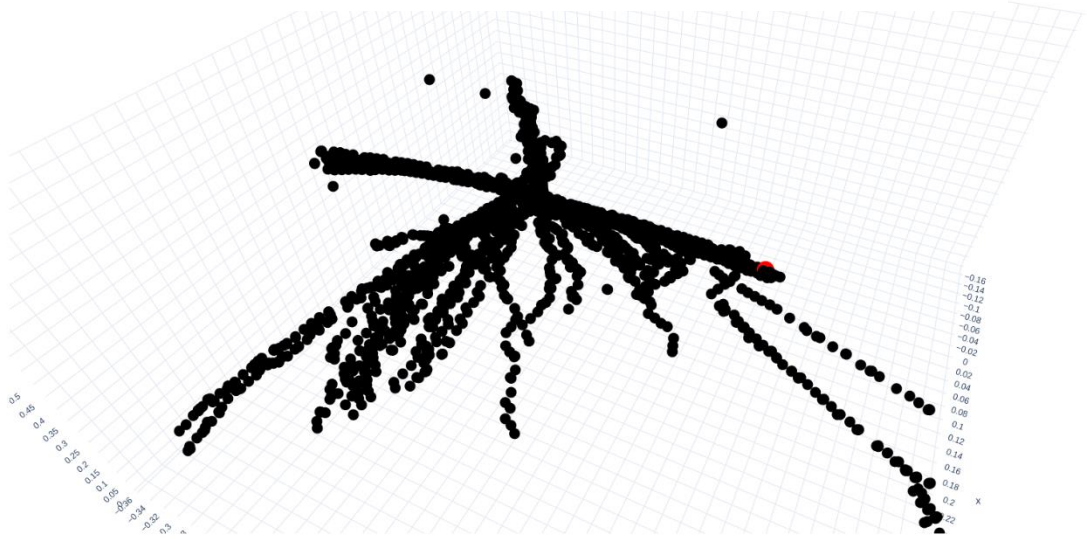
```
user@user-pc:~$ rosrun testos r_traj.py
```



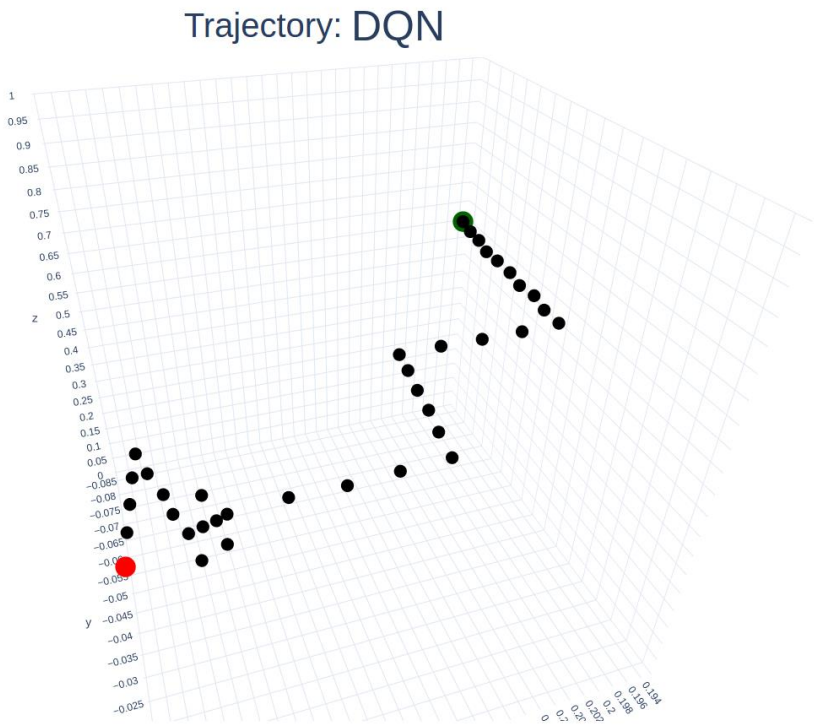
Obr. 65: Zobrazení provedené trajektorie z učení algoritmu Q-learning



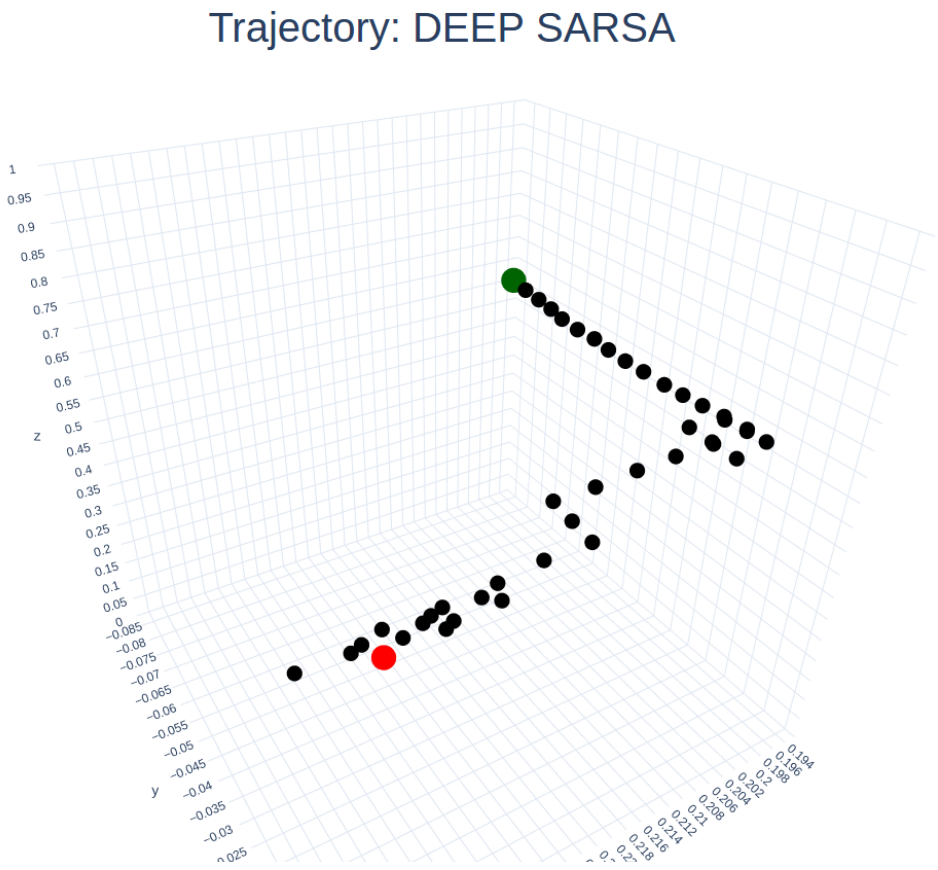
Obr. 66: Vývojový diagram řídícího skriptu *r_traj.py*



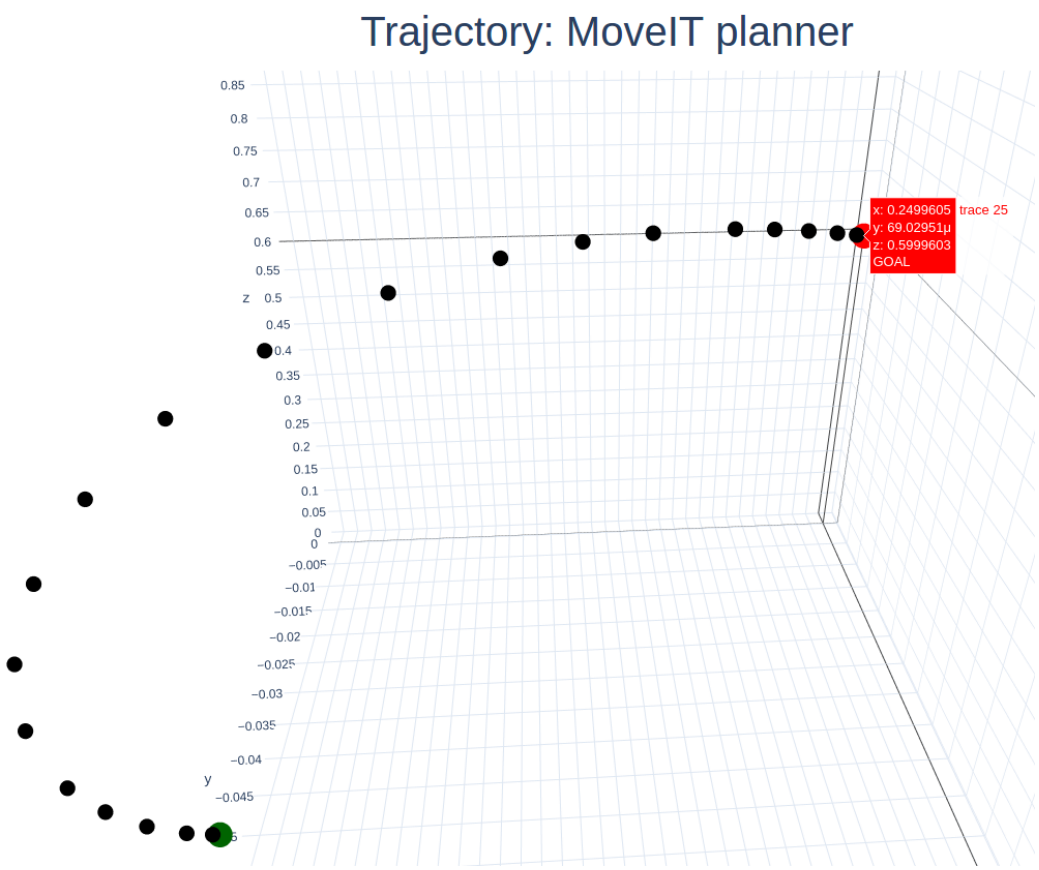
Obr. 67: Algoritmus SARSA zobrazení pohybů provedených při učení



Obr. 68: Zobrazení provedené trajektorie z učení algoritmu Deep Q-learning



Obr. 69: Zobrazení provedené trajektorie z učení algoritmu Deep SARSA



Obr. 70: Zobrazení provedené trajektorie plánovacího algoritmu knihovny OMPL

7 ZÁVĚR

Cílem bakalářské práce bylo seznámení se s problematikou kolaborativních robotů, Robotickým operačním systémem a simulačním prostředím Gazebo. Výsledkem práce bylo vytvoření řídícího programu a jeho následná aplikace pro kolaborativního robota UR3 od firmy Universal Robots. Řídící program naskýtá možnosti ovládání kolaborativního robota do předem definovanému bodu za pomoci již zaznamenaného simulovaného učení zpětnovazebních algoritmů, ale také i algoritmů hlubokého zpětnovazebního učení.

První část práce je věnována současnému stavu kolaborativní robotiky a její možné aplikace v různých odvětvích. V této rešeršní části jsou popsány i možnosti programování a řízení kolaborativních robotů, nedílnou součásti je i obecnější popis souboru doporučeních, kterými by se měli výrobci kolaborativních robotů řídit. Další část rešerše se zabývá kolaborativními roboty od firmy Universal Robots, kdy je specifičtěji popsán kolaborativní robot UR3. Teoretická část o Robotickém operačním systému dává čtenáři nahlédnout pod roušku nejpopulárnějšího frameworku na poli robotiky. Následující část popisuje simulační prostředí Gazebo, kde je porovnáno i k přímému konkurentu v oblasti simulačních prostředí V-REP. Tato rešeršní část je zakončena kratším popisem nové generace Robotického operačního systému.

Druhá část bakalářské práce naskýtá čtenáři část rešeršní a část praktickou, kde je popsáno základní fungování zpětnovazebních algoritmů, blíže algoritmy Q-learning a SARSA s jejich samotnou implementací. V této části je čtenáři popsáno, jaké knihovny byly zvoleny, samotné vytvoření ROS balíku, proces návrhu až po samotné testování učení. Po absolvováni těchto kapitol má čtenář základní informace o fungování zpětnovazebních algoritmů a integraci s Robotickým operačním systémem, kdy je připraven o rozšíření, a to v podobě hlubokého zpětnovazebního učení. Blíže jsou popsány od samotného návrhu až po implementaci algoritmy Deep Q-learning a Deep SARSA. Závěrem je v této části specifikováno samotné testování, možné problémy a výsledky učení daných algoritmů.

Poslední část práce má ryze praktický charakter, kde je popsána konfigurace Robotického operačního systému, propojení s fyzickým kolaborativním robotem UR3 a následná aplikace řídícího programu. Bylo provedeno úspěšné testování na reálném kolaborativním robotu pro algoritmy Q-learning, SARSA, Deep Q-learning, Deep SARSA a také pro skript využívající pouze knihovny frameworku MoveIT. Práci lze rozšířit o přidání algoritmů kupříkladu A3C nebo DDPG. K maximálnímu využití potenciálu již naprogramovaných algoritmů Deep Q-learning nebo Deep SARSA lze jen pozměnit definovaný úkol, a to v podobě dosahování náhodně generujících bodů, čímž by došlo k natrénování neuronové sítě k pohybu do libovolného bodu v pracovním rozsahu daného kolaborativního robota.

8 SEZNAM POUŽITÉ LITERATURY

[1] Industries That Can Use Cobots. *Mobile Automation Industrial Electrical & Automation Specialists* [online]. ©2017 [cit. 2020-03-28]. Dostupné z: https://www.mobileautomation.com.au/industries-using-cobots

[2] FANUC. [offline katalogový list]. *Experience more Collaborative robots for a wide range of applications* [cit. 2020-03-28]. Dostupné z: https://www.fanuc.eu/cz/cs/roboty/stránka-filtru-robotů/spolupracující-roboty/collaborative-cr35ia

[3] RobotWorx - FANUC CR-35iA. In: *RobotWorx: Industrial Robot Automation Integrator* [online]. ©2020 RobotWorx [cit. 2020-03-28]. Dostupné z: https://www.robots.com/robots/fanuc-cr-35ia

[4] KUKA AG [online katalogový list]. *Mobile robotics_KMR iiwa.* ©2017 [cit. 2020-03-11]. Dostupné z: https://www.kuka.com/-/media/kuka-downloads/imported/9cb8e311bfd744b4b0eab25ca883f6d3/kuka_kmriiwa_en.pdf

[5] KUKA KMR iiwa - Special Mention Industry. In: *German Design Award* [online]. [cit. 2020-03-28]. Dostupné z: https://www.german-design-award.com/en/the-winners/gallery/detail/8600-kuka-kmr-iiwa.html

[6] Collaborative Robot Market Size, Growth, Trend and Forecast to 2025. *MarketsandMarkets: Market Research Reports, Marketing Research Company, Business Research by MarketsandMarkets* [online]. ©2020 [cit. 2020-03-11]. Dostupné z: https://www.marketsandmarkets.com/Market-Reports/collaborative-robot-market-194541294.html

[7] The Adamo Robot. *Adamo: Robot Physio* [online]. [cit. 2020-03-28]. Dostupné z: https://adamorobot.com/the-adamo-robot

[8] CARLO® (Cold Ablation Robot-guided Laser Osteotome). *AOT: We know how bone likes to be cut* [online]. ©2016 [cit. 2020-03-11]. Dostupné z: https://aot.swiss/en/carlo

[9] Robotics adding value by picking, packaging and shellin. *Collaborative robotic automation: Cobots from Universal Robots* [online]. ©Universal Robots 2020 [cit. 2020-03-11]. Dostupné z: https://www.universal-robots.com/case-stories/cascina

[10] Cobots work collaborativly at Gentofte hospital lab. *Collaborative robotic automation: Cobots from Universal Robots* [online]. ©Universal Robots 2020 [cit. 11.03.2020]. Dostupné z: https://www.universal-robots.com/case-stories/gentofte-hospital

[11] EL ZAATARI, Shirine, Mohamed MAREI, Weidong LI a Zahid USMAN. Cobot programming for collaborative industrial tasks: An overview. *Robotics and Autonomous Systems* [online]. 2019, 162-180 [cit. 2020-01-24]. DOI: https://doi.org/10.1016/J.ROBOT.2019.03.003. Dostupné z: https://www.sciencedirect.com/science/article/pii/S092188901830602X

[12] Robot Controllers. *FANUC America: Automation Solutions for CNC Systems, Industrial Robotics & ROBOMACHINE* [online]. ©1987 [cit. 2020-03-11]. Dostupné z: https://www.fanucamerica.com/products/robots/controllers

[13] FANUC iPendant touch. *FANUC: The Factory Automation Company* [online]. [cit. 2020-04-05]. Dostupné z: https://www.fanuc.eu/cz/en/robots/accessories/robot-controller-and-connectivity/ipendant-touch

[14] AIRSKIN® for Universal Robots UR10 - Blue Danube Robotics. In: *Home - Blue Danube Robotics* [online]. [cit. 2020-04-05]. Dostupné z: https://www.bluedanuberobotics.com/product/ur10/

[15] ROSENSTRAUCH, Martin J. a Jörg KRÜGER. *Safe human-robot-collaboration-introduction and experiment using ISO/TS 15066* [online]. Nagoya, Japan: IEEE, 2017 [cit. 2020-01-24]. ISBN 978-1-5090-6088-7. Dostupné z: https://ieeexplore.ieee.org/document/7942795

[16] YouRing - COBOTS. In: *COBOTS - handla enkelt kollaborativa robotar och tillbehör online* [online]. [cit. 2020-04-05]. Dostupné z: https://cobots.se/produkt/youring/

[17] Universal Robots. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2020 [cit. 2020-03-11]. Dostupné z: https://en.wikipedia.org/wiki/Universal_Robots

[18] UR10 Cobots Optimize the Assembly Line at Ford Romania. *Collaborative robotic automation: Cobots from Universal Robots* [online]. ©2020 Universal Robots [cit. 2020-03-11]. Dostupné z: https://www.universal-robots.com/case-stories/ford-motor-company/

[19] UR5 cobot solution provides time and energy relief. *Collaborative robotic automation: Cobots from Universal Robots* [online]. ©2020 Universal Robots [cit. 11.03.2020]. Dostupné z: https://www.universal-robots.com/case-stories/linaset/

[20] Read, Compare and Choose: The Cobots Buyers Guide is Out!. In: *Workfloor: Robotics News for the Factory* [online]. [cit. 2020-04-05]. Dostupné z: https://blog.robotiq.com/read-compare-and-choose-the-cobots-buyers-guide-is-out

[21] IP Code. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2020 [cit. 2020-03-11]. Dostupné z: https://en.wikipedia.org/wiki/IP_Code

[22] Universal Robots. [offline katalogový list]. *Technical details e-series* [cit. 2020-03-28]. Dostupné z: https://www.universal-robots.com/products/

[23] UR+ Solutions | Roboworld Climate-Controlled Robosui for UR3, UR5, UR10 and e-Series. *Collaborative robotic automation: Cobots from Universal Robots* [online]. ©2020 Universal Robots [cit. 2020-03-11]. Dostupné z: https://www.universal-robots.com/plus/urplus-components/accessories/climate-controlled-robosuit/

[24] UR+ Solutions | AIRSKIN, PLe certified safety skin for UR5 and UR5e. *Collaborative robotic automation: Cobots from Universal Robots* [online]. ©2020 Universal Robots [cit. 2020-03-11]. Dostupné z: https://www.universal-robots.com/plus/urplus-components/accessories/airskin-ple-certified-safety-skin-for-ur5-and-ur5e/

[25] UR+ Solutions | Vention 3D MachineBuilder™. *Collaborative robotic automation: Cobots from Universal Robots* [online]. ©2020 Universal Robots [cit. 2020-03-11]. Dostupné z: https://www.universal-robots.com/plus/urplus-components/software/3d-machinebuilder-design-and-order-custom-robot-cells/

[26] Robosuit Products - Roboworld. In: *Roboworld* [online]. ©2020 Roboworld and Roboworld Molded Products, LLC. All rights reserved. [cit. 2020-03-11]. Dostupné z: https://roboworld.com/robosuit/products/

[27] Universal Robots [online katalogový list]. *Technical details UR3.* [cit. 2020-03-28]. Dostupné z: https://www.universal-robots.com/media/1801288/eng_199901_ur3_tech_spec_web_a4.pdf

[28] AUBO. [offline katalogový list]. *Aubo. Neúnavný parťák do výroby* [cit. 2020-03-28]. Dostupné z: https://www.aubo.cz/produkty/aubo-i3/

[29] QUIGLEY, Morgan, Ken CONLEY, Brian P. GERKEY, Josh FAUST, Tully FOOTE, Jeremy LEIBS, Rob WHEELER a Andrew Y. NG. *ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software* [online]. 2009, [cit. 2020-01-24]. Dostupné z: https://www.researchgate.net/publication/233881999_ROS_an_open-source_Robot_Operating_System

[30] History. *ROS.org: Powering the world's robots* [online]. [cit. 2020-04-05]. Dostupné z: https://www.ros.org/history/

[31] Robonaut. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2020 [cit. 2020-03-11]. Dostupné z: https://en.wikipedia.org/wiki/Robonaut

[32] How to Start with Self-Driving Cars Using ROS. *ROS For Beginners LEARNING PATH: The Construct* [online]. ©2019 The Construct. All rights reserved. [cit. 2020-03-11]. Dostupné z: https://www.theconstructsim.com/ja/start-self-driving-cars-using-ros/

[33] PX4 Offboard Control Using MAVROS on ROS. 404warehouse. In: *404warehouse: Small Projects Big Ideas in Robotics, New Media and other Projects* [online]. [cit. 2020-04-05]. Dostupné z: https://404warehouse.net/2015/12/20/autopilot-offboard-control-using-mavros-package-on-ros/

[34] Description-ROS-Industrial. *ROS-Industrial* [online]. [cit. 2020-04-05]. Dostupné z: https://rosindustrial.org/about/description

[35] Distributions. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/Distributions

[36] boxturtle. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/boxturtle

[37] kinetic. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/kinetic/Installation

[38] melodic. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/melodic/Installation

[39] Arch Linux - Artwork. In: *Arch Linux* [online]. ©2002 [cit. 2020-03-11]. Dostupné z: https://www.archlinux.org/art/

[40] Concepts. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/ROS/Concepts

[41] Nodes. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/Nodes

[42] Master. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/Master

[43] Parameter Server. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/Parameter Server

[44] Topics. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/Topics

[45] Services. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/Services

[46] Bags. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/Bags

[47] Integration with Other Libraries. *ROS.org: Powering the world's robots* [online]. [cit. 2020-04-05]. Dostupné z: https://www.ros.org/integration/

[48] Plugin Interfaces. *MoveIt Motion Planning Framework: Moving robots into the future* [online]. [cit. 2020-04-05]. Dostupné z: https://moveit.ros.org/documentation/plugins/

[49] urdf. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/urdf

[50] Ioan A. Șucan, Mark Moll, Lydia E. Kavraki, The Open Motion Planning Library, *IEEE Robotics & Automation Magazine*, **19**(4):72–82, December 2012. [cit. 2020-04-01]. Dostupné z: http://ompl.kavrakilab.org

[51] Planners Available in MoveIt. *MoveIt Motion Planning Framework: Moving robots into the future* [online]. [cit. 2020-04-05]. Dostupné z: https://moveit.ros.org/documentation/planners/

[52] Gazebo. *Gazebo* [online]. ©2014 Open Source Robotics Foundation [cit. 2020-03-11]. Dostupné z: http://gazebosim.org

[53] SDFormat. *SDFormat Home* [online]. [cit. 2020-04-05]. ©2019 Open Source Robotics Foundation [cit. 11.03.2020]. Dostupné z: http://sdformat.org

[54] PITONAKOVA, Lenka, Manuel GIULIANI, Anthony PIPE a Alan WINFIELD. *Feature and Performance Comparison of the V-REP, Gazebo and ARGoS Robot Simulators* [online]. Cham: Springer, 2018 [cit. 2020-02-03]. ISSN 978-3-319-96728-8. Dostupné z: https://link.springer.com/chapter/10.1007%2F978-3-319-96728-8_30

[55] ARGoS concepts. *The ARGoS Website* [online]. [cit. 2020-04-05]. Dostupné z: https://www.argos-sim.info/concepts.php

[56] Robots modulares capaces de reorganizarse y autorreparse – La ciencia en el mundo. In: *UNAM* [online]. ©2020 [cit. 2020-03-11]. Dostupné z: http://blogs.ciencia.unam.mx/cienciamundo/2017/09/12/robots-modulares-capaces-de-reorganizarse-y-autorreparse/

[57] ROS 2 vs. ROS 1: Which One Is Better For Me?. *ROS For Beginners LEARNING PATH: The Construct* [online]. ©2019 The Construct. All rights reserved. [cit. 2020-03-11]. Dostupné z: https://www.theconstructsim.com/infographic-ros-1-vs-ros-2-one-better-2/

[58] SUTTON, Richard S. a Andrew G. BARTO. *Reinforcement Learning: An Introduction* [online]. 2. 55 Hayward Street, Cambridge, MA, United States: A Bradford Book, 2018 [cit. 2020-04-05]. ISBN 978-0-262-03924-6. Dostupné z: https://dl.acm.org/doi/book/10.5555/3312046

[59] JANG, Beakcheol, Myeonghwi KIM, Gaspard HARERIMANA a Jong Wook KIM. Q-Learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access* [online]. IEEE, 2019, **7**(7), 133653-133667 [cit. 2020-03-12]. DOI: 10.1109/ACCESS.2019.2941229. ISSN 2169-3536. Dostupné z: https://ieeexplore.ieee.org/document/8836506

[60] Q-learning. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2020 [cit. 2020-05-28]. Dostupné z: https://en.wikipedia.org/wiki/Q-learning

[61] DAŇHELOVÁ, Jana. *Zpětnovazební učení pro řešení herních algoritmů* [online]. Brno, 2018 [cit. 2020-05-28], 48 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Ing. Martin Kolařík Dostupné z: https://www.vutbr.cz/en/students/final-thesis/detail/110275

[62] ZAMORA, Iker, Nestor Gonzalez LOPEZ, Víctor Mayoral VILCHES a Alejandro Hernández CORDERO. Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo. In: *ResearchGate* [online]. Erle Robotics, 2017 [cit. 2020-03-12]. Dostupné z: https://www.researchgate.net/publication/306376848_Extending_the_OpenAI_Gym_for_robotics_a_toolkit_for_reinforcement_learning_using_ROS_and_Gazebo

[63] An introduction to Q-Learning: reinforcement learning. *Learn to code: freeCodeCamp.org* [online]. [cit. 2020-04-05]. Dostupné z: https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/

[64] xacro. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/xacro

[65] ZHANG, Zhongheng. A gentle introduction to artificial neural network. *Ann Transl Med* [online]. 2016, **4**(19), 370 [cit. 2020-05-28]. DOI: 10.21037/atm.2016.06.20. Dostupné z: https://www.researchgate.net/publication/309141452_A_gentle_introduction_to_artificial_neural_networks

[66] Artificial Neural Network | NVIDIA Developer. *NVIDIA Developer* [online]. ©2020 NVIDIA Corporation [cit. 2020-05-28]. Dostupné z: https://developer.nvidia.com/discover/artificial-neural-network

[67] ZHANG, Wenyu, Jingyao GAI, Zhigang ZHANG, Lie TANG, Qingxi LIAO a Youchun DING. Double-DQN based path smoothing and tracking control method for robotic vehicle navigation. *Computers and Electronics in Agriculture* [online]. 2019, **166**(104985) [cit. 2020-05-28]. DOI: https://doi.org/10.1016/j.compag.2019.104985. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0168169919302066

[68] Deep Q-Learning | An Introduction To Deep Reinforcement Learning. *Analytics Vidhya - Learn Machine learning, artificial intelligence, business analytics, data science, big data, data visualizations tools and techniques.* [online]. ©2013 [cit. 2020-05-28]. Dostupné z: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

[69] Tutorial Physics Parameters. *Gazebo* [online]. ©2014 Open Source Robotics Foundation [cit. 2020-03-12]. Dostupné z: http://gazebosim.org/tutorials?tut=physics_params&cat=physics

[70] Using OpenAI with ROS. *ROS For Beginners LEARNING PATH: The Construct* [online]. ©2019 The Construct. All rights reserved. [cit. 2020-03-12]. Dostupné z: https://www.theconstructsim.com/using-openai-ros/

[71] catkin. In: *Documentation - ROS Wiki* [online]. [cit. 2020-04-05]. Dostupné z: http://wiki.ros.org/catkin

9 SEZNAM OBRÁZKŮ

10 SEZNAM PŘÍLOH

PŘÍLOHA A. CD-ROM

Tab. 1: Obsah CD – hlavní adresář („\Bakalářská práce \“)

Adresář	Soubor
\BP_pdf\	2020_BP_JUŘÍČEK_Martin_200543_VUT_FSI_UAI.pdf
\Vizuální_dokumentace\	pracoviste_foto_1.jpg pracoviste_foto_2.jpg pracoviste_foto_3.jpg video_Q-learning.mp4 video_SARSA.mp4 video_DQN.mp4 video_DSARSA.mp4
\ROS_workspace\	testos.filefolder
\Readme\	Readme.txt