Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Reinforcement learning na platformě LEGO MindStorms

2020

Gabriela Hrubá

Vedoucí práce:
Mgr. Tomáš Urbanec

Studijní obor: Aplikovaná informatika,
prezenční forma

Bibliografické údaje

Autor:	Gabriela Hrubá
Název práce:	Reinforcement learning na platformě LEGO MindStorms
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2020
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Tomáš Urbanec
Počet stran:	46
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Gabriela Hrubá
Title:	Reinforcement learning implemented with LEGO Mindstorms
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2020
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Tomáš Urbanec
Page count:	46
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Tématem této práce je nastudování metod reinforcement learning a poté implementování vybrané metody při stavbě robota ze sady Lego Mindstorms. Sestavený robot má za cíl naučit se pohybu vpřed za pomoci motorizovaného ramene. K implementaci reinforcement learning je vybrán algoritmus Q-learning. Následně je postaven robot využívající složitějšího pohybového systému. Pohybové systémy robotů jsou poté srovnány.

Synopsis

The topic of this thesis is the study of reinforcement learning and implementation of a chosen method during the construction of a robot from the Lego Mindstorms set. The robot's goal is to learn how to move forward using motorized arm. Q-learning algorithm is used to implement reinforcement learning. After that a second robot using more complex system of movement is built. The systems of movement of the robots are then compared.

Klíčová slova: reinforcement learning; Q-learning; robotika

Keywords: reinforcement learning; Q-learning; robotics

Děkuji vedoucímu bakalářské práce Mgr. Tomáši Urbancovi za trpělivý přístup a přínosné rady. Dále děkuji Mgr. Tomáši Mikulovi za podporu a doporučení v náročných situacích.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracovala samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

1 Úvod

Reinforcement learning je fascinující oblastí strojového učení, která v poslední době nabývá na popularitě. Největším úspěchem této oblasti je program *Alpha Go* [1], který v roce 2017 porazil několikanásobného mistra světa v čínské hře Go [2]. Hlavní inspirací přístupu reinforcement learning je přirozený proces učení u lidí a zvířat, který se skládá z vykonávání různých akcí a získávání zpětných vazeb ze svého okolí. Je to zajímavá metoda, která se inspiruje z přirozeného života. Tento fenomén můžeme také zaznamenat například u výpočetního modelu umělých neuronových sítí.

Vzhledem k postupnému nárůstu výpočetních schopností dnešních počítačů získává oblast robotiky nové příležitosti a možnosti. Aplikace reinforcement learning algoritmů v oblasti robotiky je běžná a často přichází se zajímavými a nepředvídatelnými výsledky [3]. Hlavní výhodou použití reinforcement learning algoritmů v oblasti robotiky je volnost ve fázi učení a absence konkrétního vzoru chování, které by měl robot napodobovat. Robotovi poskytujeme jen zpětnou vazbu, jakožto reakci na jeho provedené akce. Tento přístup pak umožňuje nalezení optimálního řešení, které není zcela typické a očekávané.

Sada Lego Mindstorms je určena především k výuce základů robotiky ve vzdělávání. Ve výchozím stavu neobsahuje žádné prostředky pro aplikaci algoritmů strojového učení, ale s použitím operačního systému ev3dev [4] je možné k této sadě přistupovat jako k plně programovatelné robotické sadě. Hlavním omezením této sady je nízký výpočetní výkon, který je však dostatečný pro méně náročné algoritmy strojového učení. Ke konstrukci robotů pro tuto bakalářskou práci jsem měla dostupné sady Lego Mindstorms EV3 Core Set a Lego Mindstorms EV3 Expansion Set.

Cílem této práce je postavit robota ze sady Lego Mindstorms a následně implementovat vybraný algoritmus z oblasti reinforcement learning. Použitím vhodně zvoleného algoritmu usiluji o naučení robota pohybu vpřed pomocí odrážení motorizovaným ramenem. Poté jsem se pokusila postavit druhého robota ze sady Lego Mindstorms, který používá k pohybu vpřed složitější pohybový systém.

2 Reinforcement learning

Reinforcement learning je jedním ze tří hlavních přístupů strojového učení. Jeho hlavní myšlenka je značně inspirována přirozeným procesem učení u lidí i zvířat. Člověk během učení interaguje s okolním prostředím a pozoruje vzniklé změny. Tyto změny vnímá jako zpětnou vazbu a na základě těchto změn pochopí, jaký vliv měla na okolní prostředí jeho akce. Z tohoto poté vyvodí, zda byla úspěšná či nikoliv. Úspěšnost dané akce člověk hodnotí vzhledem ke stanovenému cíli, kterého se snaží dosáhnout. Často se může stát, že získaná zpětná vazba působí na člověka negativně, například způsobí pocit bolesti. I pomocí této negativní zpětné vazby se člověk učí a intuitivně ví, že danou akci se bude snažit neopakovat.

V této kapitole popíšu základní principy přístupu reinforcement learning, které jsem čerpala z knihy Reinforcement learning: An introduction [1].

2.1 Základní myšlenka

Reinforcement learning je založen na velice podobném přístupu jako je výše zmíněný lidský přístup k učení. Program, který se nachází na pozici žáka, se nazývá agent. Jeho jediný úkol je naučit se dosáhnout určitého předem stanoveného cíle. Agent se nachází v prostředí, se kterým může interagovat pomocí specifikovaných akcí. Má k dispozici množinu možných stavů, ve kterých se může vůči prostředí nacházet, a akcí, které může v daném stavu konat. Stav obsahuje informace o tom, v jaké situaci se aktuálně agent nachází vůči okolnímu prostředí. Agent není dopředu informován, která akce je v daném stavu vhodná a proces učení je založen pouze na interakci agenta s prostředím. Zpočátku může být tato interakce čistě náhodná. Agent dostává po vykonání akce zpětnou vazbu ve formě odměny, která reflektuje úspěšnost dané akce v aktuálním stavu. Vybraná akce může ovlivnit nejen agenta, ale i stav prostředí a úspěšnost následujících akcí. Úkolem agenta je pro každý stav najít jedinou vhodnou akci, která maximalizuje výslednou odměnu. Hodnota dané odměny reprezentuje úspěšnost dané akce vzhledem k celkovému cíli, kterého se snaží agent dosáhnout.

Hlavní přístup reinforcement learning algoritmů čerpá z procesu lidského učení, a proto představuje přínosnou reprezentaci přirozeného učení v oblasti informatiky. Z tohoto důvodu jsou tyto algoritmy využívány i ve výzkumu v oblasti neurovědy a psychologie [1].

2.2 Markovův rozhodovací proces

Pro formalizaci principů reinforcement learning se používá *Markovův rozhodovací proces*. Tento matematický rámec formálně definuje interakce mezi agentem a okolním prostředím vzhledem ke stavům, akcím a odměnám. Markovův rozhodovací proces se skládá z množiny stavů S, množiny akcí A, přechodové funkce T a okamžité odměny R [5].

Množina stavů S je definována jako konečná množina $\{s_1, ..., s_n\}$ obsahující všechny možné stavy, do kterých se agent může dostat. Každý stav určuje unikátní postavení agenta vůči okolnímu prostředí a vůči robotovi samotnému. Toto postavení je určeno vlastnostmi ovlivňující daný proces učení a řešení konkrétního problému.

Množina akcí A je konečná množina $\{a_1, ..., a_k\}$. Provedení dané akce a ve stavu s může vést k přechodu do jiného stavu s' z množiny S.

Přechodová funkce T, definována jako $T : S \times A \times S \to [0, 1]$, určuje míru pravděpodobnosti, že při provedení akce a se ze stavu s přejde do stavu s'. Pro vyjádření této pravděpodobnosti se používá zápis $T(s, a, s')$ a jinými slovy určuje, do jakých stavů s' je možné se dostat ze stavu s při vykonání akce a. Pokud daný stav s neumožňuje provedení akce a, pak platí $T(s, a, s') = 0$ pro všechny stavy

s' z množiny S.

Agentovi je v pravidelných intervalech poskytována číselná hodnota odměny. Okamžitá odměna R (anglicky reward function) reprezentuje aktuální míru získané odměny z provedení dané akce a v aktuálním stavu s. Tato odměna určuje úspěšné a neúspěšné akce v daném stavu a pomáhá s vyvažováním zvolené strategie. Okamžitou odměnu lze reprezentovat funkcí $R : S \times A \times S \to \mathbb{R}$.

Očekávaná odměna (anglicky value function) předpovídá dlouhodobou míru odměny. Při zvolení akce v určitém stavu se očekávaná odměna rovná součtu všech předpokládaných odměn obdržených od této akce až do dosažení konečného stavu. Vyvažuje důležitost aktuální a dlouhodobé odměny. Může totiž nastat situace, kdy akce, která generuje malou okamžitou odměnu, je ve výsledku nejlepší vzhledem k dosažení celkového cíle. Například při hře šachu může nastat situace, kdy obětování pěšce, což generuje malou okamžitou odměnu, může vést k jasnému vítězství celé hry a vysoké očekávané odměně. Obvykle strategie upřednostňují u výběru akce vyšší očekávanou odměnu, namísto okamžité odměny. Pomocí okamžité odměny ale odhadujeme hodnotu očekávané odměny. Z těchto důvodů jsou obě hodnoty klíčové pro správný proces učení.

Definice 1 (Markovův rozhodovací proces [5])
Markovův rozhodovací proces je čtveřice $\langle S, A, T, R \rangle$ kde S je konečná množina stavů, A je konečná množina akcí, T reprezentuje přechodovou funkci a R reprezentuje funkci definující okamžitou odměnu.

Strategie π (policy) definuje vztah mezi daným stavem a výslednou akcí. Určuje, jakou akci agent provede, pokud se bude nacházet v určeném stavu, a tím definuje hlavní rozhodovací proces agenta při řešení daného problému. Jinými slovy strategie řídí celkové chování robota. Reinforcement learning algoritmy dělíme do dvou skupin na základě jejich přístupu k hledání optimální strategie, více v sekci 2.3.

Strategie může být stochastická, nebo deterministická. Stochastická strategie, narozdíl od deterministické, nemusí vždy při provedení akce a ve stavu s přejít do stejného stavu s'. Stochastickou strategii lze reprezentovat funkcí $\pi : S \times A \to [0, 1]$, která určuje míru pravděpodobnosti provedení akce a ve stavu s. Deterministickou strategii pak reprezentujeme funkcí $\pi : S \to A$, která ke každému stavu vrátí nejlepší akci k vykonání.

2.3 Učení on-policy a off-policy

Během procesu učení se reinforcement learning algoritmy obecně řídí danou strategií, která určuje výběr vykonávaných akcí v daném stavu. Cílem těchto algoritmů je nalezení optimální strategie, která bude stavům přiřazovat nejvhodnější akce, které co nejvíce splňují dosažení daného cíle. Existují ale dva odlišné přístupy k nalezení optimálních strategií [1].

On-policy metoda se snaží o vylepšení a upravení strategie π, kterou již ak-

tivně používá během svého procesu učení. Zatímco *off-policy* metoda sice během procesu učení používá strategii π, ale snaží se vylepšovat a upravovat výslednou optimální strategii π^*, která se liší od aktuálně používané strategie. V případě off-policy je strategie π určena pouze na generovaní trénovacích situací a po ukončení fáze učení už není používána, namísto ní je následována strategie π^*. Kdežto u on-policy přístupu slouží strategie π nejen ke generování trénovacích situací během fáze učení, ale i k nalezení výsledné strategie, podle které se naučený robot bude řídit.

On-policy metoda je sice jednodušší implementačně i výpočetně, ale neučí se s cílem nalezení optimální strategie, nýbrž pouze téměř optimální strategie. Překážkou je totiž potřeba prozkoumávat různé nevyzkoušené situace a zároveň nutnost hledání a vyhodnocování optimálních akcí. Tím pádem je strategie π velice limitována, protože musí vyvažovat poměr zkoumání a užívání (2.6).

Off-policy metody jsou složitější z pohledu notace i výpočtu, ale jsou schopny nalézt optimální strategie i ve složitějších situacích. Jejich hlavní výhodou je totiž oddělení trénovací strategie π od výsledné optimální strategie π^*. Díky tomu může algoritmus během fáze učení více prozkoumávat neobjevené kombinace stavů a akcí, aniž by to negativně ovlivnilo výsledek optimální strategie.

2.4 Modelové a bezmodelové algoritmy

Reinforcement learning dělí algoritmy podle způsobu nalezení optimální strategie na *modelové* (anglicky *model-based*) a *bezmodelové* (anglicky *model-free*). Výběr vhodné skupiny algoritmů závisí na konkrétní řešené situaci a na tom, zda máme k dispozici model Markovova rozhodovacího procesu. Modelové algoritmy získají optimální strategii pomocí *plánování*, zatímco bezmodelové algoritmy aplikují přístup *zkoušení*.

2.4.1 Modelové Algoritmy

Modelové algoritmy používají k výpočtu model Markovova rozhodovacího procesu, který reprezentuje model daného prostředí. Model Markovova rozhodovacího procesu může být odhadnut během výpočtu nebo sestrojen před spuštěním algoritmu. Tento model poskytuje agentovi povědomí o tom, jaký vliv mají jeho akce na prostředí, a do jaké míry prostředí ovlivňují. Při zvolení akce v určitém stavu může model předpovědět následující stav i odměnu a lze jej použít k sestavení vyhovující strategie učení. Taková strategie vznikne z modelu na základě plánování, protože model umožňuje předvídat změny a reakce prostředí bez potřeby vykonání dané akce.

V praxi je často nutné model sestrojit, aby se dal implementovat modelový algoritmus. Od modelu prostředí je vyžadována vysoká přesnost, která je důležitou podmínkou při správném fungování modelového algoritmu. Tato podmínka může mít za následek výpočetně náročné sestrojování modelu prostředí, které je v určitých situacích lepší neprovádět a přiklonit se k použití bezmodelového algoritmu.

2.4.2 Bezmodelové Algoritmy

K implementaci bezmodelových algoritmů není zapotřebí znát model Markovova rozhodovacího procesu. Bezmodelové algoritmy nacházejí strategii pomocí zkoušení, což se může jevit jako přímý opak plánování u modelových algoritmů. Bezmodelové algoritmy nemají k dispozici informace o tom, jak bude prostředí reagovat na vybrané akce. Tyto informace získávají jen pomocí vykonávání akcí a sledování změny prostředí. Při testování vybrané strategie se pomocí přístupu zkoušení a omylů hodnotí úspěšnost dané strategie a následně může být pozměněna. Tento přístup zkoušení je méně komplikovaný, než plánování u modelových algoritmů. Při větším počtu akcí nebo stavů ale může být vhodnější modelový algoritmus, který nepotřebuje zkoušet všechny akce k nalezení správné strategie.

2.5 Porovnání s jinými přístupy strojového učení

V oblasti strojového učení existují tři hlavní přístupy k učení [1]. *Supervised learning* (učení s učitelem), *unsupervised learning* (učení bez učitele) a reinforcement learning. Každý z těchto přístupů je vhodný na jiné typy problémů. Daný problém ale nemusí spadat pouze do jediné kategorie problémů. Proto je výběr přístupu k řešení daného problému jedním z klíčových rozhodnutí.

2.5.1 Supervised learning

Supervised learning (učení s učitelem) je aktuálně nejpoužívanější přístup v oblasti strojového učení. Přístup má na začátku k dispozici trénovací kolekci dat doplněnou o správné výsledky. Na této kolekci natrénuje svou strategii, kterou poté použije k vyhodnocení kolekce neoznačených dat. Supervised learning by mělo být schopné správně klasifikovat i data, která nikdy nevidělo, ale která jsou podobná datům z trénovací kolekce dat.

V případě interaktivních a komplexnějších problémů může být reinforcement learning účinnější přístup, protože je velice obtížné vytvořit testovací kolekci dat, která není nadměrně veliká a zároveň obsahuje všechna data reprezentující různorodost celé kolekce dat. V případě reinforcement learningu se agent učí na základě svých vlastních zkušeností a není zapotřebí mu poskytovat dodatečné kolekce dat.

2.5.2 Unsupervised learning

Unsupervised learning (učení bez učitele) pracuje pouze s kolekcí neoznačených dat [6]. Jeho cílem je najít skrytou spojitost nebo strukturu v datech. Často se používá k rozdělení dat do skupin podle právě takto zjištěných spojitostí (anglicky *clustering*). Tyto vstupní kolekce dat jsou většinou velké nebo komplexní, a proto je jednodušší mezi nimi hledat vzájemné vztahy, než pracovat s každým datovým záznamem zvlášť. Tento přístup nemusí být ideální, pokud pro

řešení daného problému potřebujeme nějaké konkrétní chování nebo určité výsledky, protože hlavním výstupem tohoto přístupu jsou pouze nalezené skryté spojitosti.

2.6 Dilema zkoumání a užívání

Jak již bylo zmíněno v sekci 2.1, agent se pro každý stav snaží najít jedinou akci, která generuje maximální odměnu v daném stavu. Takovou akci pak preferuje nad ostatními tak, aby dosáhl co nejvyšší výsledné celkové odměny. Toto chování se nazývá princip *užívání* (anglicky *exploitation*) a vypovídá o tom, že agent preferuje takové akce, které už v minulosti provedl, a ví že budou úspěšné. Aby ale agent takové akce pro každý stav vůbec nalezl, musí nejprve upřednostňovat princip *zkoumání* (anglicky *exploration*). Během tohoto procesu agent hledá a vykonává akce, které ještě v daném stavu nevykonal, a tím pádem neví, jak velkou odměnu budou generovat [1]. Hledání správného vyvážení mezi principem zkoumání a užíváni se nazývá *dilema zkoumání a úžívání* (anglicky *Exploration-Exploitation Dilemma*).

Při konkrétní implementaci je potřeba danou akci vykonat vícekrát, aby agent určil odpovídající hodnotu odměny a úspěšnost této akce. Proto přístup zkoumání často jen vybírá náhodnou akci ze všech dostupných. Tato situace vytváří problém při výběru správného poměru zkoumání a užívání. Agent nikdy nebude úspěšný, pokud bude jen zkoumat anebo naopak jen užívat. Je potřeba najít správný poměr tak, aby agent nacházel vyhovující akce a současně preferoval výhodné akce před náhodně vybranými. Doposud nebyla nalezena jednoznačná odpověď na ideální poměr zkoumání a užívání. Obecně se k tomuto problému přistupuje tak, že na začátku procesu učení se upřednostňuje zkoumání akcí, které je ale postupně nahrazováno užíváním již vyzkoušených akcí s vysokou odměnou. Tento problém je jedním z klíčových problémů reinforcement learning přístupu a není potřeba jej řešit u jiných přístupů strojového učení.

3 Q-learning algoritmus

K realizaci cíle mé bakalářské práce jsem si zvolila algoritmus Q-learning. Je to bezmodelový algoritmus využívající Q funkci (1) a tabulku Q hodnot reprezentující úspěšnost akce v daném stavu. Vyniká jednoduchostí výpočtu při menším počtu stavů a akcí, což z něj dělá vhodný algoritmus k realizaci učení robota složeného ze sady Lego Mindstorms. V této kapitole jsem čerpala z knih Reinforcement learning : An introduction [1] a Reinforcement learning : State-of-the-Art [5].

3.1 Základní popis algoritmu

Jelikož je Q-learning bezmodelový algoritmus 2.4.2, je zapotřebí zkoumat odezvy prostředí pomocí zvolených průzkumných akcí. Což nás přivádí zpět k dilematu

zkoumání a užívání (viz sekce 2.6). V mém případě jsem si zvolila strategii ϵ-*greedy*, která funguje na základě náhodného výběru a je popsána v sekci 3.2.

Algoritmus 1 Q-learning [5]

Require: discount factor γ, learning rate α
  inicializace $Q(s, a) = 0, \forall s \in S, \forall a \in A$
  for each epizoda **do**
    inicializace počátečního stavu na stav s_t
    repeat
      výběr akce $a_t \in A(s_t)$ na základě použité strategie a hodnoty Q
      provedení akce a_t
      zaznamenání nového stavu s_{t+1} a získané odměny r_t
      $Q(s_t, a_t) = Q(s_t, a_t) + \alpha\Big(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big)$
      $s_t := s_{t+1}$
    until s_{t+1} je cílový stav nebo počet iterací dosáhne určeného počtu iterací

Během každé učící epizody (viz pseudokód algoritmu 1) probíhá cyklus vykonávání akcí a změny stavů, který skončí tehdy, jakmile se agent dostane do konečného stavu nebo dosáhne daného počtu iterací.

Agent vybere následující akci z aktuálního stavu s_t na základě zvolené strategie pro výběr akcí (3.2) a vybranou akci pak provede. Poté agent, spolu se získanou odměnou r_t, zaznamená informaci o novém stavu s_{t+1}, do kterého přešel pomocí akce a_t. Následně aktualizuje hodnotu Q pro daný stav s_t a akci a_t podle vzorce 1. Dále nastavíme hodnotu aktuálního stavu na stav s_{t+1} a cyklus opakujeme.

Funkce Q obecně definovaná jako $Q : S \times A \to \mathbb{R}$ přiřazuje dvojici $stav \times akce$ Q hodnotu reprezentující úspěšnost akce v daném stavu. Dané Q hodnoty jsou ukládány do předem určené tabulky, která je na začátku výpočtu inicializována s hodnotami 0.

Konstanta α reprezentuje *míru učení* (anglicky *learning rate*), která určuje, do jaké míry se hodnoty v průběhu epizody aktualizují a jak moc se staré hodnoty přepisují novými. Je doporučené míru učení v průběhu epizody postupně snižovat.

Konstanta γ reprezentuje *diskontní faktor* (anglicky *discount factor*), udává se v intervalu $\langle 0, 1)$ a určuje důležitost dlouhodobých odměn. U problémů, kde je přesně určený konec epizody, není diskontní faktor potřeba [5].

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\Big(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big) \tag{1}$$

Rovnice Q funkce 1 reprezentuje aktualizaci Q hodnoty pro dvojici $(stav, akce)$. Agent přejde ze stavu s_t do stavu s_{t+1} pomocí zvolené akce a_t. Za tento přechod obdrží odměnu r_t. K aktuální Q hodnotě je následně přičten součet získané odměny s rozdílem očekávané Q hodnoty stavu s_{t+1} a aktuální Q hodnoty.

Q-learning je off-policy algoritmus (2.3), což znamená, že i když se řídí strategií π, tak usiluje o nalezení optimální strategie π^*. V případě algoritmu Q-learning je Q hodnota aktualizována s předpokladem, že následná vybraná akce ve stavu s_{t+1} bude akce a. Tento přístup reprezentuje výslednou strategii π^*. Ale při skutečném výběru prováděné akce je možné, že na základě použité strategie pro výběr následující akce, nastane situace náhodného výběru, čímž se nedodrží předpoklad provedení akce a podle výsledné strategie. Tento přístup reprezentuje trénovací strategii π.

Výraz $\max_a Q(s_{t+1}, a)$ z rovnice 1 reprezentuje výběr maximální možné Q hodnoty z následujícího stavu, respektive slouží k nalezení akce, která z následujícího stavu generuje největší Q hodnotu. U příbuzného on-policy algoritmu SARSA je tato část nahrazena částí $Q(s_{t+1}, a_{t+1})$ reprezentující výběr nejvyšší Q-hodnoty z následujícího stavu pomocí dané strategie π.

3.2 Strategie ϵ-greedy

Strategie ϵ-greedy je jednou ze strategií pro výběr následující akce [5], která řeší vyvážení míry zkoumání a užívání (více v sekci 2.6). Tato metoda je založena na stochastickém náhodném výběru a jedná se o jednu z primárně používaných strategií. Mějme konstantu ϵ v intervalu $\langle 0, 1\rangle$, která určuje míru náhodnosti při výběru akce. Před samotným výběrem akce je nejprve vygenerováno náhodné číslo z intervalu $\langle 0, 1\rangle$. Pokud je číslo menší, než konstanta ϵ, je provedena náhodně vybraná akce. V opačném případě je provedena akce s nejvyšší Q hodnotou v daném stavu. Pravděpodobnost výběru náhodné akce je tím pádem určena hodnotou konstanty ϵ a pravděpodobnost výběru nejvhodnější nalezené akce $(1 - \epsilon)$. V mém případě jsem nastavila počáteční hodnotu ϵ na 1.0, což zpočátku zaručí zcela náhodný výběr akcí. Tuto konstantu opakovaně zmenšuji o hodnotu 0.1 po 15 iteracích cyklu provádějícím akce tak, aby se postupně mířilo k výběru nejvhodnějších akcí. Hodnotu této konstanty jsem vybrala tak, aby co nejlépe vyvažovala délku a dosaženou míru učení.

3.3 Zhodnocení algoritmu Q-learning

Q-learning provádí fázi udělování odměn po každém kroku (anglicky *time step*), narozdíl od algoritmů, které vykonávají fázi rozdělování odměn až na konci každé epizody (například algoritmus Monte Carlo) [1]. Tyto odměny jsou určeny součtem okamžité odměny s očekávanou odměnou následujícího stavu [5]. Pro některé případy problémů jsou charakteristické velmi dlouhé epizody běhu, a v takovém případě by proces udělování odměn na konci každé epizody byl nepraktický a zpomaloval by proces učení. Tento přístup se nazývá *temporal difference learning* (TD) a je to obecný mechanizmus použitý v řadě dalších bezmodelových algoritmů jako například TD(0) anebo SARSA [5].

Nevýhodou algoritmu Q-learning je nutná práce s tabulkou Q hodnot. Během procesu učení je tabulka pravidelně procházena za účelem nalezení nejvyšší

Q hodnoty pro daný stav. Velikost této tabulky je tedy závislá na počtu stavů i na počtu akcí. Při libovolném složitějším problému se tato tabulka stane výpočetní překážkou méně výkonným systémům, jako je například Lego Mindstorms. S tímto problémem jsem se setkala při konstrukci druhého robota (viz 6.4). Řešením může být přístup *Approximate Q-learning*, který provede aproximaci Q funkce [7].

4 Konstrukce robota ze sady Lego Mindstorms

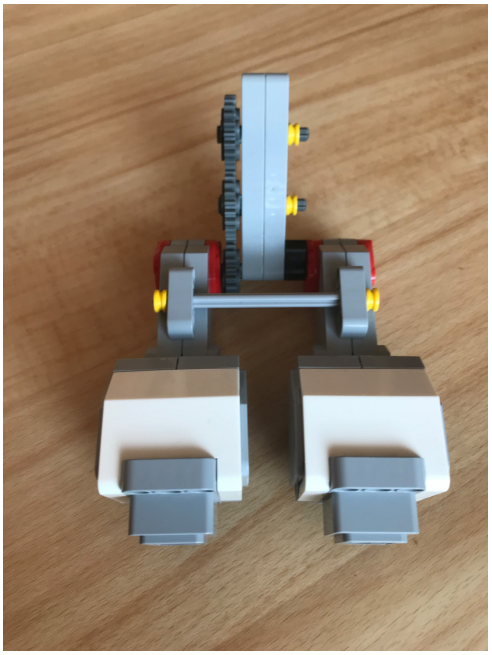
Po nastudování teorie reinforcement learning jsem začala se stavbou robota. Konstrukci robota jsem si rozdělila do tří fází. V první fázi jsem postavila hlavní kostru robota. V druhé fázi jsem naprogramovala jednoduchý program testující správnou funkčnost a pohyb robota. A ve třetí fázi jsem připojila ke kostře robota senzory a implementovala detekci pohybu vpřed. Použité dílky, zmíněné v této kapitole, jsou obsaženy spolu s obrázky v tabulce 1.

4.1 Základní kostra robota

Při stavbě základní struktury robota jsem začala nejprve konstrukcí pohybového ramene. Jednou z inspirací pro tuto bakalářskou práci bylo následující video [8]. Inspirovala jsem se použitím převodových koleček pro ovládání míry zakřivení ramene [9]. Převodová kolečka jsem připevnila k dvěma spojeným dílkům 4611705 z jedné strany tak, aby se kolečka volně protáčela a zároveň do sebe navzájem zapadala. Motor ovládající rotaci celého ramene jsem pak umístila z opačné strany, než se nachází převodová kolečka. Motor jsem připojila pomocí dvou dílků 4121715 tak, aby rotace motoru otočila celou strukturou ramene a nedocházelo k protáčení. Na straně převodových koleček byl přidán druhý motor ovládající míru zakřivení ramene. Tento motor byl spojen s dílkem 4611705 v rameni skrz jedno převodové kolečko. Pro spojení jsem použila jednu osku, která se při rotaci motoru protáčí v dílku 4611705. Tím pádem rotace motoru otočí převodové kolečko, ale neovlivní rotaci celého ramene.

Obrázek 1 je první verzí poloviny pohybového ramene. U dalších verzí zůstávala základní myšlenka stejná, pouze jsem upravovala vzhledové a konstrukční vady, aby bylo rameno co nejvíce symetrické a konstrukčně pevné. Na opačný konec ramene, než se nacházejí motory, jsem prostrčila osku skrz dílky 4611705 a poslední převodové kolečko. Na oba konce této osky jsem připojila dva dílky 4495412 a mezi ně jsem připevnila kolečko sloužící k odrážení robota tak, aby nebylo umožněno jeho protáčení.

Při konstrukci pro mě byla důležitá robustnost robota [10], proto jsem vytvořila hlavní podpůrný rám pomocí čtyř dílků (4540797), které umožňují pevné vzájemné propojení a navíc společně tvoří pravidelnou obdélníkovou kostru. Na obou stranách jsem připevnila velké motory s již sestaveným ramenem. Na druhém konci rámu jsem přidala dvě středně velká kola, díky kterým se může robot volně pohybovat. Každé kolo je připevněno jednou neutaženou oskou, která

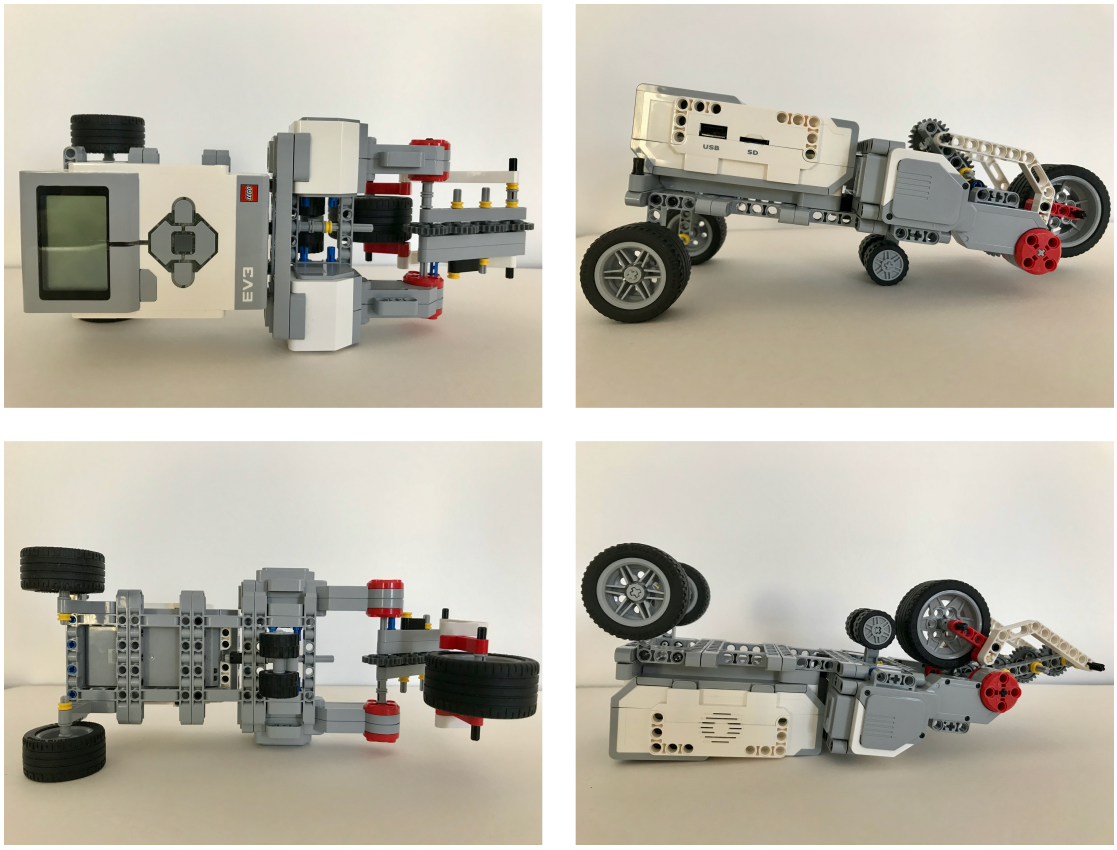


Obrázek 1: První polovina pohybového ramene

umožňuje volné protáčení kol. Obě tyto osky jsem pak spojila pomocí dílku 4513174 díky kterému se kola otáčejí rovnoměrně, což je velice důležité při budoucí funkčnost senzorů. Aby ale druhý konec rámu nedřel o zem, vložila jsem mezi velké motory dvě malá podpůrná kolečka 4639695. Nejprve jsem uvažovala o použití kovové kuličky, která je obsažena v sadě Mindstorms. Kulička by ale nakonec nebyla dostatečně vysoká a navíc bych nikdy nevyužila možnosti pohybu do všech stran, kterého je schopna.

Po připojení pohybového ramene k hlavnímu rámu jsem začala pracovat na umístění programovatelné kostky. Z počátku byl robot navržen jen ze dvou dílků 4540797 a připevnění kostky bylo kvůli nedostatku místa možné pouze na samotné motory. Toto umístění však přineslo mnoho konstrukčních komplikací a zbytečnou hmotnostní zátěž na jednu stranu robota. Při takovém umístění programovatelné kostky by rameno nefungovalo správně, kvůli nerovnoměrnému rozložení váhy. Proto jsem se rozhodla prodloužit hlavní rám o další dva dílky 4540797 a programovatelnou kostku jsem následně mohla umístit na nově vzniklé místo. Při připojování kostky jsem musela brát ohled na pozici motorů a senzorů, aby nenastala situace, kdy by byl blokován jeden ze vstupních portů programovatelné kostky.

Jednu z prvních verzí kostry robota je možné vidět na obrázku 2. V této verzi nejsou například spojeny osky předních kol a robot nemá senzory pro detekci pohybu vpřed.



Obrázek 2: První verze základní kostry robota

4.2 Testování základní kostry robota

Po připojení programovatelné kostky k robotovi bylo potřeba ověřit funkčnost a pohyblivost celé konstrukce. To jsem ověřila pomocí jednoduchého programu, který simuluje pohyb robota naučeného odrážení vpřed.

4.2.1 Připojení robota k počítači

Základní pohybové instrukce jsem ze začátku programovala pomocí Lego Mindstorms aplikace pro iOS zařízení [11], která je s robotem propojena pomocí Bluetooth.

Pro implementaci složitějších instrukcí jsem použila operační systém ev3dev [4]. Ev3dev je linuxový operační systém určený pro Lego Mindstorms platformu. Tento operační systém umožňuje flexibilnější programování robota Lego Mindstorms. Obsahuje výchozí podporu několika populárních programovacích jazyků, ze kterých jsem si pro účely této práce vybrala jazyk Python. Další výhodou Ev3dev je linuxový základ, který poskytuje možnost instalace dodatečných balíčků i základní funkčnost USB a Bluetooth zařízení, jako například USB Wi-Fi karty. Výhodou je jednoduchá instalace, která nijak nezasahuje do originálního systému od firmy Lego. Ev3dev stačí nahrát na formátovanou microSD kartu a tu poté vložit do programovatelné kostky. Pro spuštění originálního Lego systému stačí vyjmout kartu z vypnutého zařízení.

Pro připojení systému ev3dev jsem použila USB Wi-Fi kartu připojenou k programovatelné kostce. Na svém počítači jsem nainstalovala vývojové prostředí Visual Studio Code [12], které disponuje rozšiřující knihovnou ev3dev-browser [13]. Tato knihovna umožňuje nalézt systémy ev3dev přes síť Wi-Fi. Po připojení systému ev3dev k prostředí Visual Studio Code knihovna umožňuje posílaní a vzdálené spouštění programů v systému ev3dev.

Systém ev3dev také umožňuje manuální spuštění programu pomocí programovatelné kostky a může tedy spustit program i bez připojení k počítači. Program je ale nejprve potřeba nahrát do paměti systému ev3dev za pomocí počítače. Po spuštění systému ev3dev zvolíme možnost *File browser* a za malou chvíli se na obrazovce objeví seznam souborů a složek nahraných v paměti systému ev3dev. Poté stačí navigovat k cílenému programu, který chceme spustit a výběr programu potvrdit prostředním tlačítkem. Po potvrzení začne systém vykonávat zvolený program.

Má bakalářská práce je napsána v jazyce Python, což vyžadovalo instalaci aktuální verze tohoto jazyka na systému ev3dev. Tuto aktualizaci jsem provedla přes SSH terminál dostupný díky knihovně ev3dev-browser.

Pro samotné ovládání robota, jeho motorů, senzorů a tlačítek, jsem použila Python3 knihovnu ev3dev2 [14]. Tato knihovna poskytuje základní funkce pro ovládání motorů robota, detekci hodnot z jeho senzorů, vydávání zvuků i výpis přímo na obrazovce programovatelné kostky a další užitečné funkce. Knihovna poskytuje pohodlný přístup k ovládání celého robota. Při implementaci algoritmu Q-learning jsem v této knihovně narazila na závažnou chybu. V ostatních ohle-

dech jsem však byla s touto knihovnou spokojena. Popisu této chyby se věnuji v sekci 5.1.1.

4.2.2 Program testující funkčnost konstrukce

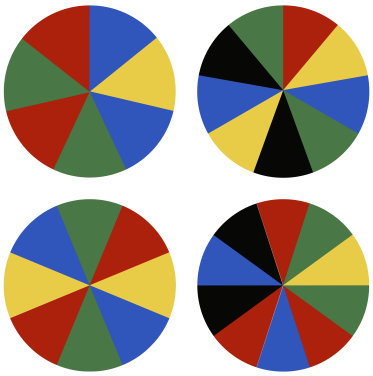
Jakmile jsem měla zprovozněné připojení ev3dev systému k mému vývojovému prostředí a funkční verzi jazyka Python, napsala jsem základní program pro pohyb robota vpřed pomocí odrážejícího se ramene. Program se skládal z jednoho cyklu, během kterého se volala funkce motoru *on_for_degrees* z knihovny ev3dev2. Tato funkce provede rotaci motoru o daný úhel a přijímá dva argumenty, první udává rychlost rotace a druhý reprezentuje úhel vykonávané rotace. Zdrojový kód programu je možné vidět níže (zdrojový kód 1). Výsledný běh tohoto programu je zaznamenán na videu `predpokladany-pohyb.mov` obsaženém na CD.

```
1  d = LargeMotor(OUTPUT_D)
2  a = LargeMotor(OUTPUT_A)
3
4  for x in range(4):
5    a.on_for_degrees(20, 90)
6    d.on_for_degrees(20, 210) #odražení
7    a.on_for_degrees(5, -50)
8    d.on_for_degrees(5, -205) #vrácení ramene
9    a.on_for_degrees(10, -40)
10   d.on_for_degrees(10, -75) #kompenzace nepřesných pohybů
```

Zdrojový kód 1: Základní instrukce pro pohyb robota

S pohybem robota jsem byla spokojena, problém byl v konzistenci jednotlivých pohybů. Knihovna ev3dev2 umožňuje zjistit aktuální pozici motoru. Pozice je reprezentována celočíselnou hodnotou, reprezentující míru otočení motoru vzhledem k startovací pozici. Začala jsem analyzovat tyto hodnoty, abych si ověřila konzistenci navazujících posunů. Zjistila jsem, že pozice motorů nejsou po provedení pohybu stejné, ale hodnoty kolísají. Po provedení jedné iterace cyklu by se motor měl nacházet v původní pozici, ve které se nacházel před provedením dané iterace. Ve skutečnosti vznikala odchylka od této původní pozice, která s postupným počtem iterací neustále narůstala a ovlivňovala konzistenci pohybu. Takové odchylky jsou kritické pro jakékoliv metody strojového učení, protože mohou narušit proces učení a zkreslovat potřebná data.

Funkce *on_for_degrees* otočí motorem o zadaný úhel a tím pádem je výsledek vždy jiný, pokud je počáteční pozice jen o jeden stupeň posunutá. Takové posunutí o jeden stupeň se děje hlavně kvůli vůli motorů, ke kterým je připojené dlouhé a těžké rameno. Tuto kolísavost hodnot jsem vyřešila používáním funkce motoru *on_to_position*. Motor se namísto rotace o daný úhel otočí na zadanou pozici, která je reprezentována celočíselnou hodnotou. Tento způsob pohybu je spolehlivý, protože není závislý na počáteční pozici motoru.



Obrázek 3: První návrh barevného kruhu

4.3 Detekce pohybu robota pomocí senzorů

Po vyzkoušení základního pohybu jsem začala s implementací senzorů pro detekci pohybu vpřed. Tato detekce bude později sloužit pro kalkulaci okamžité odměny při učení pomocí přístupu reinforcement learning.

4.3.1 Výběr senzorových snímačů

Mým cílem bylo implementovat detekci pohybu tak, aby nebylo potřeba robota nijak restartovat a manuálně posouvat. Z tohoto důvodu nebyl první způsob detekce pohybu pomocí ultrazvukového senzoru 45504 vhodný. Tento senzor má maximální dosah 250 cm. Detekce pohybu fungovala na jednoduchém principu. Robot začínal v pozici co nejblíže měřenému povrchu a pohybem vpřed se od daného povrchu vzdaloval. Při takovém způsobu učení by musel někdo dohlížet, aby se robot nevzdálil na více jak 250 cm. Předtím, než by k tomu došlo, by bylo potřeba robota ručně posunout zpět na začínající pozici a vynulovat hodnoty vzdálenosti, aniž by byly vymazány hodnoty dosavadního učení.

Můj druhý nápad spočíval v detekci změny barev na vnitřní straně kol. Jsou použity dva barevné senzory 45506, každý z nich připevněný ke kostře robota tak, aby detekoval barvy na vnitřní straně předních kol. Pro tento účel jsem navrhla barevný kruh, skládající se z několika barev. Barevné odstíny jsem vybrala co nejvíce podobné klasickým Lego kostkám. U takových barevných odstínů má totiž senzor největší přesnost při detekci barev. První návrh barevného kruhu je na obrázku 3. Kolečka vlevo jsou navržena s menším počtem barev a s většími barevnými výřezy než kolečka vpravo. U koleček vpravo jsem také vyzkoušela jiný počet opakování barev.

4.3.2 Detekce pohybu

Detekci jsem naprogramovala tak, že na začátku iterace jsou uloženy aktuální barvy měřené senzory. Na konci jedné iterace se porovnají barvy měřené na

začátku a barvy měřené na konci. Z tohoto údaje jsem se, pomocí vypočítané vzdálenosti mezi těmito barvami, snažila zjistit směr a délku pohybu.

Tento přístup byl však nespolehlivý. Oba návrhy kruhů na obrázku 3 obsahují opakující se barvy, takže mohla nastat situace, kdy robot provedl pohyb, ale program tento pohyb nezaznamenal, protože identifikoval na konci iterace stejnou barvu, jako na začátku. Rovněž určení směru z informace o počáteční a koncové barvě nebylo úspěšné. V programu jsem pracovala s daty z obou senzorů a spoléhala jsem na to, že se obě kola otáčí stejným směrem, ale ani s těmito informacemi se mi nepodařilo ve všech případech určit správný směr pohybu.

Použila jsem tedy paralelní způsob programování [15], který mi umožnil snímat nejen počáteční a koncové barvy, ale všechny detekované barvy během celého pohybu. Každému senzoru jsem přidělila jedno samostatné vlákno, které ukládá do samostatného seznamu všechny barvy rozpoznané daným senzorem během vykonávání jedné akce. Ukládání jsem optimalizovala tak, že se barva do seznamu neuloží, pokud je stejná jako poslední zapsaná barva. Tím pádem se nezapisují dvě stejné barvy za sebou.

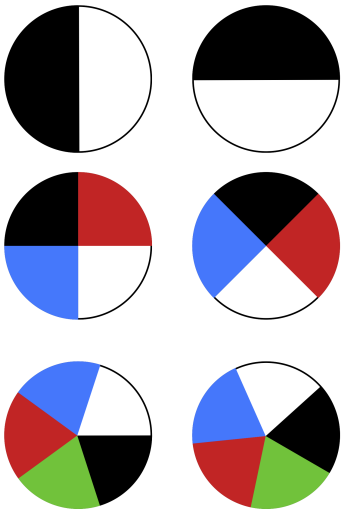
Díky tomuto přístupu jsem mohla naprogramovat jednoduchou funkci pro detekci směru pohybu *decide_direction*. Funkce vrací hodnotu 0, pokud se robot nepohnul, anebo provedl pohyb vzad. Hodnotu 1 vrací v případě, že byl rozpoznán pohyb vpřed. Pokud seznam rozpoznaných barev obsahuje více než jednu barvu, je potřeba rozlišit pohyb vpřed od pohybu vzad. Pro tento účel jsem vytvořila referenční seznamy barev, které znázorňují posloupnost barevného rozložení na každém kole. Pomocí těchto seznamů a seznamů rozpoznaných barev během vykonaného pohybu lze jednoduše určit směr posunu robota porovnáváním daných posloupností barev. Tento přístup také umožňuje i měření délky pohybu na základě délky seznamu rozpoznaných barev.

4.3.3 Problémové rozpoznávání barev

Při aplikaci přístupu zmíněném v sekci 4.3.2 jsem narazila na problém se správným rozpoznáváním snímaných barev. Návrh barevných kruhů 3 obsahuje barvy, které mohou být, například ve špatně osvětlené místnosti, interpretovány jako jiné barevné odstíny. Například přechod mezi barvou žlutou a červenou byl velice často interpretován jako barva hnědá a barva modrá byla zaměňována za černou.

Tento problém jsem se nejprve snažila vyřešit tak, že jsem naprogramovala funkci na vyfiltrování barev *filter_colors*. Tato funkce bere jako vstup seznam všech barev naměřených během pohybu a vrací seznam, který obsahuje jen barvy obsaženy v referenčním seznamu barev. Jinak řečeno, funkce vyfiltruje všechny nesprávně rozpoznané barvy a vrátí seznam obsahující pouze reálně použité barvy. Tento přístup však v určitých případech ovlivnil výsledek detekovaného směru pohybu. Pokud byla ze seznamu vymazána hnědá barva, která reálně reprezentovala barvu červenou, nastala změna pořadí barev v rozpoznaném seznamu a směr pohybu byl nesprávně klasifikován.

Vzhledem k těmto problémům jsem navrhla nové barevné kruhy vyobrazené na obrázku 4. Tento návrh obsahuje odstínově kontrastní barvy. Barvy se v da-



Obrázek 4: Finální návrh barevných kruhů

ném kruhu neopakují a tím pádem jsou i barevné výřezy větší. Nový návrh zcela eliminoval problém špatného rozpoznání barev, a to v libovolných světelných podmínkách. Kola jsem připevnila k robotovi tak, jak je vidět na obrázku. Tím pádem, pokud jeden senzor aktuálně snímá přechod mezi dvěma barvami, druhý senzor snímá jen jednu barvu. Neopakování stejných barev pak výrazně zjednodušilo proces rozhodování o směru provedeného pohybu. Namísto porovnávání celého seznamu rozpoznaných barev s referenčním seznamem stačí porovnat pouze první dvě rozpoznané barvy a jednoznačně z nich usoudit směr pohybu.

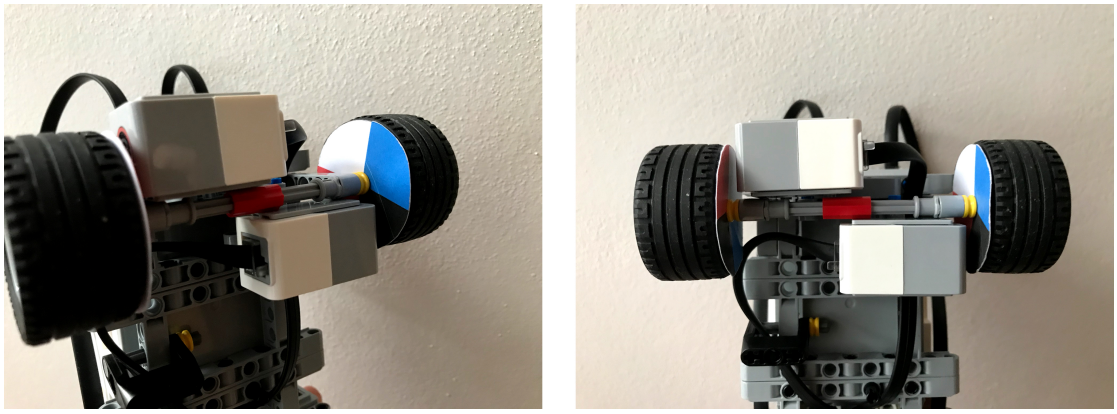
Z návrhu 4 jsem použila rozvržení zobrazené na druhém řádku, protože nabízí největší vyváženost mezi počtem barev a velikostí výřezů. Seznam referenčních barev těchto použitých kruhů je ve zdojovém kódu 2. Výsledné umístění senzorů na kostře robota lze vidět na obrázku 5.

```
1 colors = ["Red", "Black", "Blue", "White"]
2 left_wheel = ["Red", "Black", "Blue", "White"]
3 right_wheel = ["Red", "White", "Blue", "Black"]
```

Zdrojový kód 2: Seznamy referenčních barev pro použité barevné rozložení

4.4 Návod k sestavení robota

Po dokončení celkové konstrukce robota jsem se pokusila vytvořit krok za krokem návod, který by byl podobný běžným Lego návodům. Návod měl sloužit jako pomůcka při konstrukci robota. K návrhu jsem používala program pro tvorbu Lego 3D modelů Bricksmith [16] a webového klienta sloužícího k převodu 3D modelu



Obrázek 5: Výsledné umístění senzorů

do formátu instrukčního návodu Web Lic [17]. Návod jsem nakonec nedokončila, protože klient Web Lic měl problémy s načítáním některých dílků ze sady Lego Mindstorms a neumožňoval použití alternativních dílků. Pokusila jsem se alespoň dokončit 3D model robota v programu Bricksmith, ale i zde jsem narazila na problém s dostupností některých použitých dílků. Z tohoto důvodu jsem jako náhradu návodu podrobně vyfotila finální konstrukci robota a pomocí těchto fotografií je možné robota kdykoliv znovu postavit. Hlavní fotografie jsou na obrázku 6 a 7, zbylé fotografie jsou obsaženy na CD ve složce `photos`.

5 Implementace algoritmu Q-learning

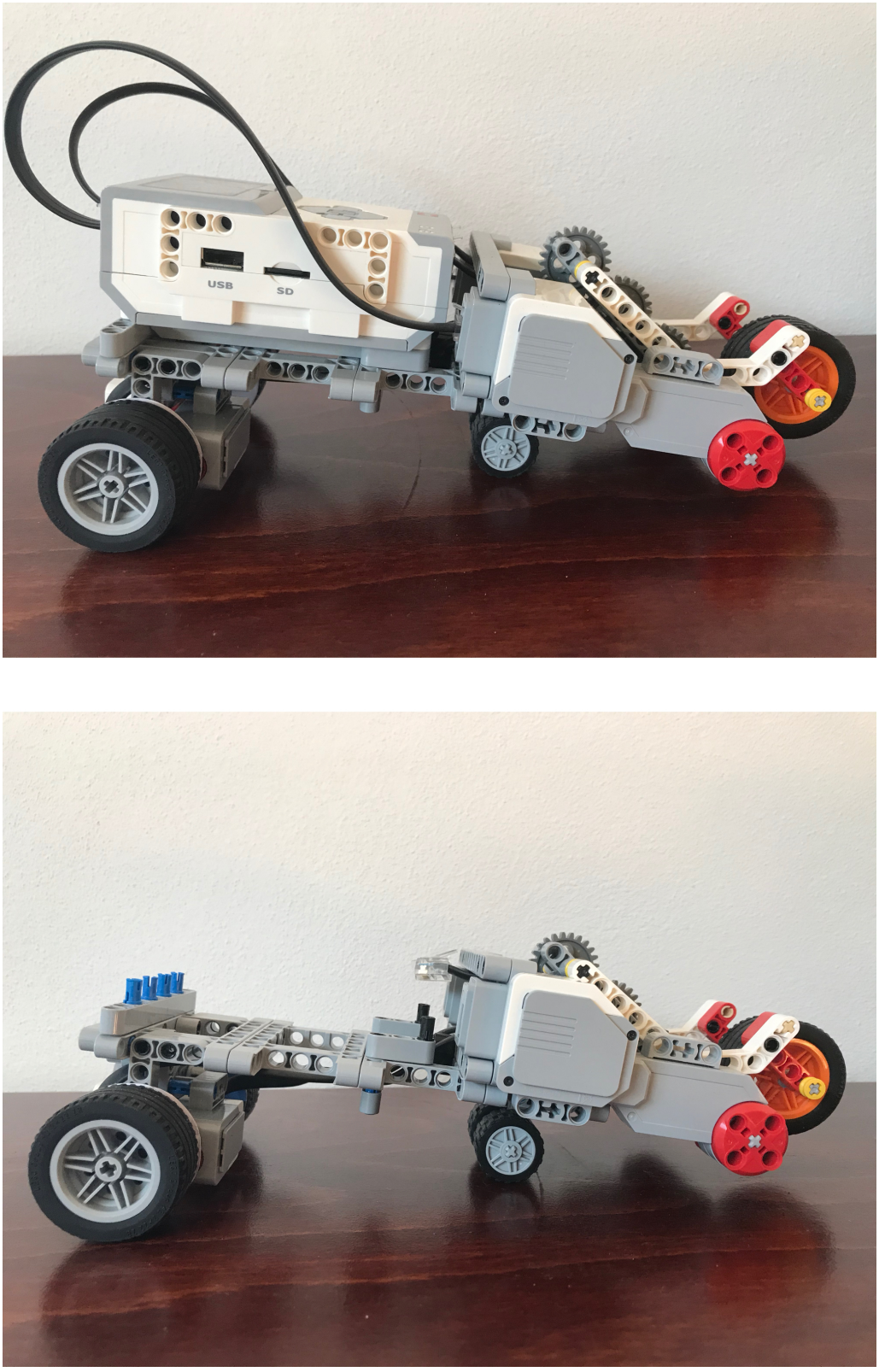
V této sekci se věnuji implementaci algoritmu Q-learning [18]. Pro správné a jednoduché fungovaní algoritmu na platformě Lego Mindstorms jsem vytvořila několik pomocných struktur a procedur. Také bylo potřeba vybrat vhodnou reprezentaci akcí a stavů tak, aby každý stav vyjadřoval unikátní postavení robota a aby reprezentace dané akce byla srozumitelná. Dodatečně jsem přidala možnost ukládání a načítání Q tabulky, což umožňuje přerušení a následné pokračování ve fázi učení bez ztráty dat nebo uložení naučeného chování robota.

5.1 Pomocné struktury a funkce

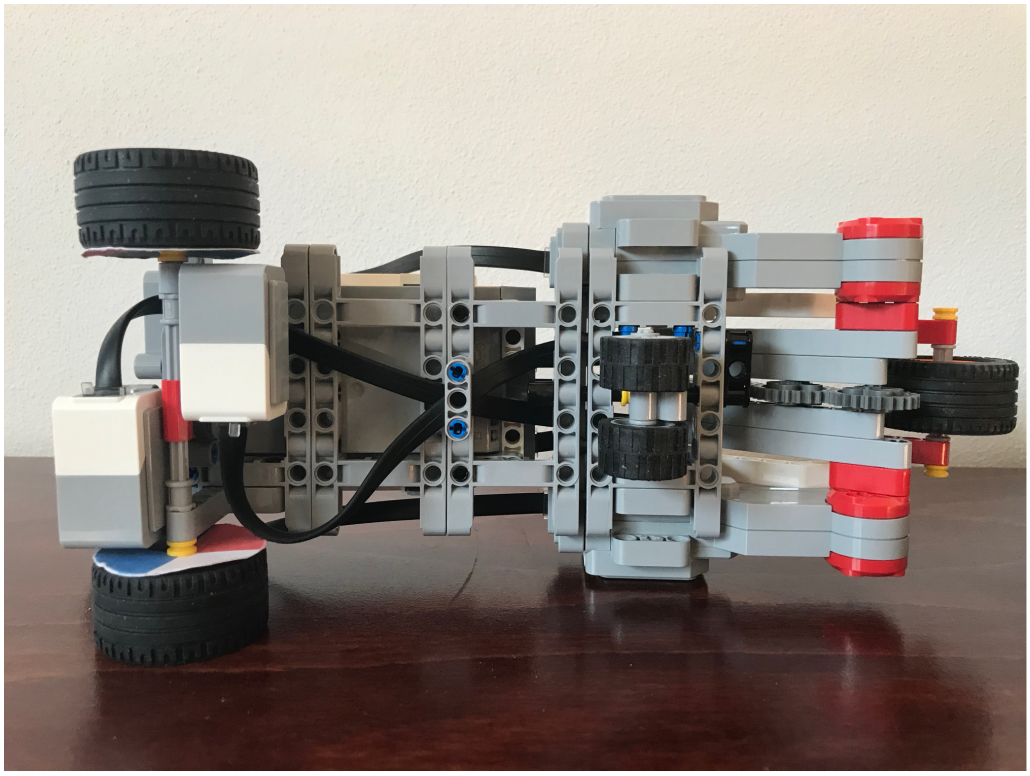
Některé pomocné struktury, jako například referenční seznamy barev, jsou popsány v sekci 4.3.2.

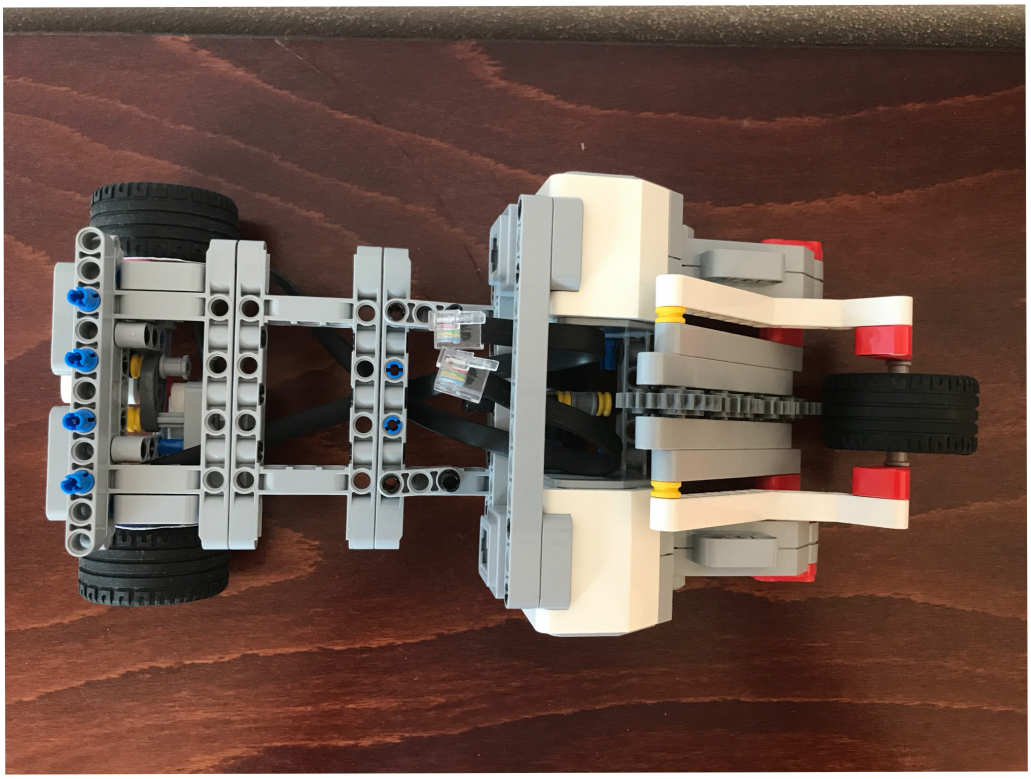
Tabulku Q hodnot jsem inicializovala pomocí Python knihovny NumPy [19] a funkce *numpy.full* [20] na dvojrozměrné pole s hodnotami 0.0, kde řádky reprezentují stavy a sloupce reprezentují akce.

NumPy je rozšířená a populární knihovna zejména v oblasti programování strojového učení v jazyce Python. Tato knihovna přináší vlastní datový typ *NumPy array*, vhodnější pro práci s daty ve formě vektorů a matic [21]. NumPy



Obrázek 6: Výsledná celková konstrukce robota (boční strana)





Obrázek 7: Výsledná celková konstrukce robota (spodní a vrchní strana)

array představuje kolekci datových elementů stejného typu s volitelným počtem dimenzí a často nahrazuje základní datový typ jazyka Python, seznam. Výhoda knihovny NumPy a jejích datových typů je efektivita. Tato knihovna poskytuje řadu funkcí pro práci s datovým typem NumPy array [22], které umožňují jednoduché manipulování s touto datovou strukturou. Kombinace datového typu a optimalizovaných funkcí má za následek rychlejší provedení operací, menší paměťovou zátěž i širší aplikovatelnost v porovnání s výchozími seznamy jazyka Python. Knihovnu NumPy jsem mimo jiné použila, abych se s ní naučila pracovat, a to z důvodu její aktuální důležitosti při programování strojového učení. Rovněž obsahuje funkce pro jednoduchý převod mezi typem NumPy array a souborovým formátem CSV, který jsem využila k exportování a importovaní Q tabulky.

Další důležitou strukturou je matice *next_state_table* sloužící ke zjištění následujícího stavu z aktuálního stavu a vykonávané akce. Řádky této matice reprezentují indexy stavů a sloupce určují indexy vykonávaných akci. Hodnota obsažena v matici je index následujícího stavu. Tuto matici jsem opět reprezentovala pomocí datového typu *NumPy array*.

5.1.1 Problém uváznutí knihovny ev3dev2

Funkce *move_left* a *move_right* sloužící k otočení daného motoru přijímají jako vstupní argument celočíselnou hodnotu, reprezentující pozici, na kterou se má motor otočit. Původní funkce fungovala velice jednoduše a je obsažena ve zdrojovém kódu 3. Obsahuje volání funkce knihovny ev3dev2 *on_to_position* s parametrem celočíselné konstanty určující rychlost pohybu *SPEED* a parametrem *position* určujícím cílovou pozici motoru.

```
1 left_motor = LargeMotor(OUTPUT_D)
2 def move_left(position):
3     left_motor.on_to_position(SPEED, position)
```

Zdrojový kód 3: Původní funkce pohybu levého motoru

Během učení pomocí algoritmu Q-learning jsem ale narazila na problém uváznutí. Při delším běhu programu došlo ke vzájemnému čekání procesů a program musel být manuálně ukončen. Po řadě různých testovacích funkcí jsem došla k závěru, že k uváznutí dochází při volání funkce *on_to_position* z knihovny ev3dev2. V GitHub repozitáři této knihovny jsem nalezla informaci o nevyřešeném problému (anglicky *issue*) uváznutí této funkce [23]. Problém mohl způsobovat stav, kdy se motor má otočit na pozici, ve které se aktuálně nachází. Z tohoto důvodu jsem přidala podmínku `if(left_motor.position == position): return`, která neprovede volaní funkce *on_to_position*, pokud se motor již nachází v cílové pozici. Vyvarování se této situaci ale nevyřešilo problém uváznutí.

Jeden z vývojářů knihovny ev3dev2 v reakci na tento problém vytvořil nový *issue* [24], ve kterém popisuje problém souběhu interní funkce *wait* způsobu-

jící uváznutí funkce *on_to_position*. Vývojáři se pokusili přijít s řešením tohoto problému a částečně jej i vyřešili. Bohužel toto zlepšení nevyřešilo moji situaci, a tak jsem se snažila přijít na vlastní řešení. Analyzovala jsem vnitřní implementaci funkce *on_to_position* (v souboru motor.py na GitHub stránce knihovny ev3dev2) a zaměřila jsem se na důvod používání problémové funkce *wait*. Našla jsem, že ve výchozím nastavení se funkce *on_to_position* volá s nastaveným atributem `block = True`. Pokud je tento atribut nastaven na hodnotu *True*, tak funkce *on_to_position* čeká na dokončení rotace motoru, před pokračováním v běhu programu. Podmínka knihovny ev3dev2 implementující čekání na dokončení pohybu je obsažena ve zdrojovém kódu 4, kde je možné vidět opakované volání problémové funkce *wait*. Rozhodla jsem se tento problém vyřešit nastavením atributu `block = False` a implementací své vlastní funkce *wait_motor_stop(motor)*, která se postará o blokování výpočtu dokud se nedokončí pohyb motoru. Tato funkce je obsažena ve zdrojovém kódu 5.

```
1 if block:
2   self.wait_until('running', timeout=WAIT_RUNNING_TIMEOUT)
3   self.wait_until_not_moving() #funkce cekajici na dokonceni pohybu
```

Zdrojový kód 4: Podmínka blokujícího volání funkce on_to_position

```
1 def wait_motor_stop(motor):
2     pos = motor.position
3     sleep(1)
4     while(motor.position != pos): #dokud se meni pozice motoru
5         pos = motor.position
```

Zdrojový kód 5: Funkce blokující běh programu při pohybu motoru

Pomocí této funkce a nastavení atributu `block = False` jsem vyřešila problém uváznutí funkce *on_to_position*. Mám v plánu informovat vývojáře knihovny ev3dev2 ohledně řešení tohoto problému. Má výsledná funkce pro otočení levého motoru je uvedena ve zdrojovém kódu 6. Velice obdobně vypadá i funkce pro otočení pravého motoru.

```
1 def move_left(position):
2     if(left_motor.position == position):
3         return
4     left_motor.on_to_position(SPEED, position, block=False)
5     wait_motor_stop(left_motor)
```

Zdrojový kód 6: Výsledná funkce pohybu levého motoru

5.2 Reprezentace stavů, akcí a odměn

Důležitým aspektem pro úspěšnou implementaci algoritmu Q-learning je výběr správné reprezentace stavů, akcí a odměn. Pokud by tato reprezentace byla chybná nebo nepřesná, zkomplikovalo by to proces učení, který by nemusel správně fungovat. Stavy, akce i odměny by měly obsahovat co nejpřesnější hodnoty a důležité informace, které pak robot používá při procesu učení [25].

5.2.1 Reprezentace stavů

Stavy jsem se rozhodla reprezentovat uspořádanou dvojicí *(poloha levého motoru, poloha pravého motoru)*. Stav je tím pádem reprezentován jako aktuální pozice motorizovaného ramene. Pro zjednodušení procesu učení a zachování přijatelné velikosti Q tabulky jsem omezila rozsah pohybu motorů na čtyři konkrétní pozice, namísto neomezené možnosti pohybu. Důsledkem tohoto rozhodnutí je i omezení počtu možných akcí, které může robot vykonat k přechodu mezi těmito pozicemi. Tyto vybrané pozice jsou 0, 25, 50 a 125. Byly vybrány tak, aby umožnily robotovi co nejlepší podmínky pro naučení pohybu odražením. Dochází však pouze k menšímu ulehčení procesu učení, protože oba motory mají možnost nacházet se v jedné z těchto pozic a ke správnému odražení robota je zapotřebí najít správné pozice obou motorů.

Množinu stavů jsem reprezentovala jako n-tici obsahující veškeré kombinace uspořádaných dvojic *(poloha levého motoru, poloha pravého motoru)*. Tato n-tice se tedy skládá z 16 dvojic.

5.2.2 Reprezentace akcí

Množinu akcí jsem reprezentovala pomocí pojmenované n-tice `namedtuple` [26]. Tento typ je speciální n-tice k jejímž prvkům lze přistupovat pomocí předem určeného názvu, který reprezentuje prvek na dané pozici. Při použití pojmenované n-tice je nejprve potřeba definovat novou třídu pomocí `collections.namedtuple (typename, field_names, ...)`. Tato třída dědí z třídy `tuple` a rozšiřuje ji o možnost pojmenování daných prvků. Argument `typename` určuje název této podtřídy. Argument `field_names` reprezentuje pojmenování prvků ve výsledné n-tici a nejčastěji je zapsán seznamem řetězců. Po definici je možné vytvořit konkrétní instance této třídy, reprezentující n-tice obsahující hodnoty na pojmenovaných pozicích.

Použití pojmenovaných n-tic lze vidět ve zdrojovém kódu 7. Nejprve jsem vytvořila podtřídu *ActionTuple*, která definuje pojmenování prvního prvku *action* a druhého prvku *position*. Poté jsem deklarovala proměnnou *action_space* jako n-tici obsahující instance třídy *ActionTuple*. Prvek *action* obsahuje název funkce pohybující levým nebo pravým motorem a prvek *position* obsahuje pozici na kterou se má motor otočit.

Používání pojmenovaných n-tic mi umožnilo zpřehlednit můj kód. Například provedení první akce z n-tice *action_space*, uložené do proměnné *action_tuple*

pomocí `action_tuple = action_space[0]`, mohu namísto nepřehledného značení `action_tuple[0](action_tuple[1])` provést akci pomocí instrukce `action_tuple.action(action_tuple.position)`.

```
1  import collections
2  ActionTuple = collections.namedtuple('ActionTuple', ['action', '
       position'])
3  action_space = (ActionTuple(action=move_left, position=0),
4              ActionTuple(action=move_left, position=25),
5              ActionTuple(action=move_left, position=50),
6              ActionTuple(action=move_left, position=125),
7              ActionTuple(action=move_right, position=0),
8              ActionTuple(action=move_right, position=25),
9              ActionTuple(action=move_right, position=50),
10             ActionTuple(action=move_right, position=125))
```

Zdrojový kód 7: Použití pojmenované n-tice

5.2.3 Reprezentace odměn

Velikost udělené odměny při detekovaném pohybu vpřed je závislá na počtu rozpoznaných barev během pohybu. Odměna je určena jako součet délek seznamů měřených barev z obou senzorů a tento součet je vydělený dvěma, aby hodnota odměny reprezentovala průměrný počet rozpoznaných barev při vykonaní dané akce v určitém stavu. Barevná kolečka jsou pootočena tak, aby při snímaném přechodu mezi dvěma barvami na jednom senzoru druhý senzor snímal pouze jednu danou barvu. Tím pádem se často stane, že seznamy barev nejsou stejně dlouhé, a proto je vhodné z těchto hodnot uvažovat jejich průměr.

Původně měla funkce *decide_direction* (zmíněná na konci sekce 4.3.2) při detekci pohybu vzad vracet hodnotu -1, znázorňující negativní odměnu. Došla jsem ale k závěru, že pro správný proces učení není potřeba jakkoliv trestat pohyb vzad (více v sekci 5.3).

Vzhledem k tomu, že má robot možnost pohybu vzad, může nastat situace, kdy robot provede pohyb vzad a poté se vrátí zpět na původní pozici. Tento pohyb bude rozpoznán jako pohyb vpřed, avšak neobdrží tak vysokou odměnu jako efektivní pohyb vpřed, protože délka tohoto pohybu nebude velká.

5.3 Neočekávaně efektivní způsob pohybu

Umožnění pohybu vzad bylo ve výsledku velmi důležité při procesu učení. Robot se totiž naučil odrážet neočekávaným, ale velice efektivním způsobem, který vyžaduje možnost pohybu vzad. Robot se nejprve položí na odrážecí rameno, což vyžaduje provedení pohybu vzad, přenese na něj většinu své váhy a poté se s jemným nadskočením velice silně odrazí vpřed. Tento způsob pohybu je zaznamenán na videu `nauceny-pohyb.mov` obsaženém na CD ve složce `videos`.

Styl pohybu byl překvapující, protože je možné, že by většinu lidí při řešení stejného úkolu nenapadl. K porovnání může sloužit zdrojový kód 1 a video `predpokladany-pohyb.mov`, které obsahuje mnou vymyšlený a implementovaný základní způsob pohybu. Tato situace ukazuje výhody a potenciál strojového učení, protože v tomto případě bylo nalezeno efektivnější řešení problému v porovnání s mým vlastním řešením.

5.4 Ukládání Q tabulky

Během procesu učení se aktualizuje a formuje výsledná podoba Q tabulky, která určuje, jakou funkci robot provede v daném stavu. Po dokončení fáze učení robot provádí jen ty akce, které produkují nejlepší výsledek dle Q tabulky. Q tabulka tedy určuje výsledné chování robota, který skončil s fází učení a následuje výslednou optimální strategii π^*.

Rozhodla jsem se implementovat ukládání a načítaní této tabulky [27], které umožňuje uložení neočekávaného nebo překvapivě efektivního chování robota a možnost v libovolnou dobu předvést dané chování načtením této tabulky. Z tohoto důvodu jsem implementovala možnost zastavení robota pomocí delšího podržení tlačítka na ovládací kostce. Pokud program detekuje podržení tlačítka, uloží aktuální Q tabulku a zastaví běh programu. O tuto funkcionalitu se stará funkce *stop_program* a je zobrazena ve zdrojovém kódu 8. Hodnota globální proměnné *STOP_PRESSED* je kontrolována v hlavní funkci programu *learn* a její nastavení na *True* zastaví běh celého programu. Tabulku ukládám pomocí funkce *NumPy.savetxt* [28], která uloží dané jednorozměrné nebo dvourozměrné pole do uvedeného souboru. Argument *Q-table.csv* určuje název výsledného souboru a argument *Q* reprezentuje ukládanou datovou strukturu. Hodnota argumentu *delimeter* určuje zvolený oddělovač hodnot v souboru a argument *fmt* reprezentuje formát ukládaných dat, který je v mém případě desetinný.

Funkce *stop_program* je volána pokud některé z vláken, ukládající měřené barvy na senzorech, rozpozná stisknutí libovolného tlačítka na ovládací kostce. Pokusila jsem se tuto funkcionalitu implementovat pomocí samostatného vlákna, které by poskytovalo nejpřesnější detekci stisknutého tlačítka. Výpočetní výkon ovládací kostky však nebyl dostatečně velký pro přidání dalšího vlákna a běh robota se začal zasekávat a zpožďovat. Z tohoto důvodu je nutné tlačítko déle podržet, protože pro správnou detekci je zapotřebí, aby jedno z daných vláken zkontrolovalo stisk tlačítka pomocí funkce *Button.any* [29].

```
1 def stop_program():
2     global STOP_PRESSED
3     STOP_PRESSED = True
4     np.savetxt('Q-table.csv', Q, delimiter=',', fmt='%i')
```

Zdrojový kód 8: Uložení Q tabulky do souboru a zastavení programu

Následně jsem vytvořila dva nové programy k práci s Q tabulkou. Program *Q_learning_load_learn.py* neinicializuje Q tabulku s hodnotami 0.0 tak, jak to dělá základní program, ale načte tabulku ze souboru *Q-table.csv*. Poté přejde do fáze učení a pokračuje ve vyvažování hodnot načtené Q tabulky. Tento program je užitečný zejména pokud došlo k přerušení předchozí fáze učení a je zájem o její dokončení. Druhý program *Q_learning_load.py* načte tabulku Q hodnot ze souboru *Q-table.csv*. Narozdíl od programu *Q_learning_load_learn.py* ale nepřechází do fáze učení. Tento program má nastavenou konstantu ϵ na hodnotu 0, tím pádem nikdy nedojde k náhodnému výběru následující akce, ale vždy se provede ta akce, která obsahuje v Q tabulce nejvyšší hodnotu. Jinak řečeno, program se řídí podle načtené Q tabulky, která reprezentuje výslednou optimální strategii a neprovádí žádný náhodný výběr následující akce. Pokud se robot naučil zajímavý způsob pohybu a uložili jsme si jeho odpovídající Q tabulku do souboru, můžeme použít tento program k načtení této tabulky a k opakovanému vykonávání daného způsobu pohybu.

6 Robot implementující složitější způsob pohybu

Po dokončení konstrukce prvního robota a úspěšné implementaci algoritmu Q-learning jsem začala s konstrukcí druhého robota. Tento robot implementuje složitější systém pohybu, pomocí kterého se má pohybovat vpřed.

6.1 Konstrukce robota

Jako složitější systém pohybu jsem si vybrala odrážení pomocí dvou otočných ramen, která se nachází na bočních stranách robota. Použité dílky, zmíněné v této kapitole, jsou obsaženy spolu s obrázky v tabulce 1.

Začala jsem základní konstrukcí kostry robota, skládající se zejména ze tří dílků 4539880. Robota jsem stavěla se záměrem co nejmenší celkové konstrukce, vzhledem k počtu použitých dílků. A to kvůli omezení dostupných kostek k této konstrukci a také za účelem snížení hmotnosti celkové konstrukce.

Snížení hmotnosti robota je důležité, protože ramena sloužící k odrážení robota jsou ovládána středními motory, které nemají takovou sílu jako velké motory použité v prvním robotovi. Z důvodu vyvážení hmotnosti robota jsem se pokusila připevnit tyto motory co nejblíže ke středu kostry robota. K motorům jsou připojena ramena, která mají na konci připevněno kolečko. Použité kolečko je ze sady Lego Technic, protože ve vypůjčené sadě Lego Mindstorms nebylo k dispozici žádné menší kolečko s křížkovou osou, která zabraňuje jeho protáčení. Délka těchto ramen je co nejkratší kvůli snížení hmotnosti a omezení poskakování robota při rotaci, ale zároveň je dostatečně dlouhá, aby se při rotaci ramena dotkla země.

Před a za motory sloužící k otáčení ramen jsem připevnila kola na kterých robot stojí a které umožňují pohyb robota. Po následném testování pohybu pouhým odrážením jsem zjistila, že robot stojící pouze na těchto kolech je velice

nestabilní. Proto jsem na boční strany připevnila kolečka 4587275, sloužící k vyvažování rovnováhy robota.

K rozpoznání pohybu vpřed jsem použila ultrazvukový senzor 45504, umístěný na zadní straně robota. Robot se tedy nachází v počáteční pozici co nejblíže měřenému povrchu a pohybem vpřed se od něj vzdaluje. Výsledná konstrukce robota je zobrazena na obrázku 8 a 9.

6.2 Složitější způsob pohybu

Pohyb pomocí odrážení dvou ramen umístěných na boku robota je složitější, protože obsahuje více proměnlivých stavů, ve kterých se může robot nacházet. Při tomto pohybu je, stejně jako u prvního robota, důležitý směr rotace motoru. Směr určuje, zda se robot bude pohybovat vpřed nebo vzad. Zároveň má ale robot možnost zatáčet. Pokud se bude odrážet jedním ramenem více než druhým, ovlivní tím směr vykonávaného pohybu. Při odrážení jedním ramenem robot stojí na místě a provádí rotaci daným směrem. Pro dosažení přímého pohybu vpřed je nutné, aby robot pravidelně střídal odrážení pravým a levým ramenem. Složitost pohybu daného robota tedy není určena počtem ovládacích motorů, ale počtem kontaktů ramen s povrchem, po kterém se robot pohybuje. V tomto případě má robot dva možné kontakty se zemí a tím pádem i možnost zatočení.

6.3 Problém s konzistentním pohybem vpřed

Před implementací algoritmu Q-learning jsem opět napsala jednoduchý program testující základní pohyb robota. Program se skládá z hlavního cyklu ve kterém se volají funkce *on_to_position* z knihovny ev3dev2, provádějící střídavý pohyb obou ramen, a z jednoho vlákna, které během vykonávaného pohybu zaznamenává měřené vzdálenosti z ultrazvukového senzoru.

Při naprogramování pohybu vpřed pomocí konkrétních instrukcí, bez použití jakékoliv formy učení, nastal problém v konzistentním pohybu vpřed. Program testující základní pohyb střídavě otáčí levé a pravé ramena na stejné pozice, což by mělo zaručit přímý konzistentní pohyb. Ve skutečnosti robot prováděl pohyb vpřed, ale po každém odrazu pravým ramenem se pootočil více než po odrazu ramenem levým. To má za následek postupné zatáčení vlevo.

Tento problém způsobuje více faktorů, jako například typ a křivost povrchu na kterém se robot nachází, přiléhavost koleček umístěných na konci ramen, plocha a barva měřeného povrchu nebo rozdílnost síly motorů ovládající ramena. Došla jsem k názoru, že detekce pohybu vpřed pomocí ultrazvukového senzoru nikdy nebude spolehlivá, protože neumožňuje jednoznačnou detekci míry otočení robota vůči měřenému povrchu.

6.3.1 Pohyb po dvojbarevné čáře

Jako řešení této situace mě napadlo umístit ze spodní strany robota barevný senzor. Robot by se mohl orientovat dle dvojbarevné čáry nacházející se na povrchu,

po kterém by se pohyboval. Tato čára by mohla být nápomocná při detekci natočení robota. Detekovaná barva by reprezentovala směr natočení robota. Tento přístup by ale stále nebyl vhodným řešením této situace. Pokud by se robot dostal mimo čáru, tak by bylo potřeba robota ručně vrátit zpět na čáru a zaznamenat tuto interakci ve fázi učení. Také výpočet získané odměny za vykonaný pohyb vpřed by byl komplikovaný. Pouhá detekce pohybu po čáře nedokáže určit délku provedeného pohybu či směr a použití ultrazvukového senzoru k měření délky pohybu není vhodné, protože míra natočení má vliv na měřenou délku.

6.3.2 Pohyb ve zúžené chodbě

Další možností bylo umístění robota do zúžené chodby, ve které se může pohybovat vpřed a připevnění dalších dvou ultrazvukových senzorů. Tyto senzory by měřily aktuální vzdálenost od bočních stran chodby, což by určovalo míru natočení robota. I v tomto přístupu by ale nastal problém, pokud by se robot otočil o více než 90 stupňů, protože by senzory přestaly snímat vzdálenost od bočních stran chodby a bylo by zapotřebí robota ručně vrátit na původní pozici, a tuto interakci zaznamenat v procesu učení.

6.3.3 Použití gyroskopického senzoru

Nejlepším řešením tohoto problému by bylo použití gyroskopického senzoru 6227055, který je schopen měřit úhel otočení daného robota. Původně to vypadalo, že použití tohoto senzoru by mohlo zcela vyřešit problém detekce natočení robota. Nakonec jsem ale došla k závěru, že i tento senzor není optimálním řešením této situace.

I při fungující detekci natočení robota by bylo nutné spolehlivě detekovat délku provedeného pohybu, která by se však měnila na základě natočení robota. Bylo by velice obtížné určit, zda změna ve vzdálenosti od měřeného povrchu byla způsobena natočením robota (a tím pádem by ve fázi učení neměla být udělena za tuto akci žádná odměna) anebo byla změna způsobena natočením i reálným posunem vpřed (a měla by být udělena odměna za tuto akci). Pokud se robot doopravdy posunul vpřed, tak by zde byl stále nevyřešený problém, jak určit míru udělené odměny za tuto akci. A to z důvodu, že nemůžeme přesně říci, jak moc velký vliv mělo na změnu vzdálenosti natočení robota a jak moc velký vliv měl samotný posun.

Další komplikací při použití tohoto senzoru je jeho nespolehlivost. Tento senzor není doporučený pro dlouhodobé měření [30]. Gyroskopický senzor má často dva hlavní problémy: *drift* a *lag*. Největší problém v této situaci dělá drift, který způsobuje nepřesnosti v měřených hodnotách gyroskopického senzoru [31]. Příkladem je častá situace, kdy je robot zcela nehybný, ale gyroskopický senzor měří nekonzistentní hodnoty, které se i bez pohybu robota mění. Při dlouhodobém měření pomocí tohoto senzoru se začne chyba postupně akumulovat, což má velice negativní důsledky na proces učení. Řešením problému driftu je jedině opakovaná kalibrace gyroskopického senzoru [32], která by ale mohla komplikovat běh

Q-learning algoritmu, zejména pokud by bylo potřeba kalibrovat senzor během fáze učení, což by opět mohlo mít vliv na nekonzistentnost měřených dat.

6.4 Implementace algoritmu Q-learning

Pro správnou implementaci algoritmu Q-learning je zapotřebí implementovat spolehlivou a konzistentní detekci pohybu vpřed. Jak už bylo zmíněno, použití ultrazvukového senzoru k detekci pohybu není spolehlivé. Situace, při kterých senzor měří chybné změny ve vzdálenosti od měřeného povrchu, by mohly mít za následek udělení pozitivní odměny ve fázi učení, aniž by robot doopravdy provedl pohyb vpřed. Také může nastat situace, kdy senzor po provedené rotaci nebude vůbec mířit na měřený povrch a vzdálenosti budou měřeny od jiného objektu.

Pokud by byl robot schopen měřit míru otočení vzhledem k povrchu, od kterého detekuje vzdálenost, například pomocí gyroskopického senzoru, tak by i tak bylo obtížné vhodně implementovat algoritmus Q-learning. Pro správnou funkci algoritmu je nutné přesně reprezentovat všechny stavy, ve kterých se může robot nacházet. Každé natočení robota vzhledem k měřenému povrchu by tedy reprezentovalo jeden daný stav. Míra natočení robota totiž značně ovlivňuje výběr následující akce, a tato informace je ve fázi učení velmi důležitá. Takových stavů by ale bylo zapotřebí větší množství. Následné vyhledávání v Q tabulce by se pro programovatelnou kostku ze sady Lego Mindstorms mohlo stát výpočetně náročnou operací. Zároveň ale není vhodné omezovat reprezentaci natočení robota na určitý počet stavů, jako to bylo provedeno při reprezentaci pozic prvního robota, protože míra natočení v sobě uchovává důležité informace pro výběr vhodné následující akce.

6.5 Zhodnocení robota

Kvůli složitějšímu způsobu pohybu se tento robot může vůči svému okolí nacházet ve více proměnlivých stavech. Při jeho konstrukci a následném uvažování o implementaci algoritmu Q-learning jsem přišla na několik problémů, které kolidují se základním úmyslem použití tohoto robota. Na tomto robotovi, pohybujícím se složitějším způsobem, jsem měla vyzkoušet implementovat algoritmus Q-learning a zkoumat jeho funkčnost. Tento algoritmus ale vyžaduje konzistentní způsob pohybu, přijatelné množství reprezentovaných stavů a spolehlivý způsob k určení míry získané odměny za provedení dané akce. Proto dle mého názoru tento robot ze sady Lego Mindstorms není vhodný pro použití principu reinforcement learning a k implementaci Q-learning algoritmu.

7 Srovnání a zhodnocení robotů

Hlavní rozdíl mezi těmito roboty je jejich způsob pohybu, který má vliv i na stavy, ve kterých se následně nachází. Rovněž používají dva různé přístupy detekce pohybu.

7.1 Způsob pohybu

První robot se pohybuje pomocí odrážení motorizovaným ramenem. Má možnost pohybovat se pouze dvěma směry, vpřed a vzad. Hlavní princip při učení tohoto pohybu je dosáhnutí správné pozice motorizovaného ramene a následné odražení. Tento způsob pohybu není nijak významně ovlivněn okolním prostředím a povrchem, po kterém se robot pohybuje. Robotovi nedělá problém ani křivá podlaha či drobné nerovnosti. Jediným problémem by mohla být kluzká podlaha, od které je obtížné se odrazit.

Druhý robot se pohybuje pomocí dvou menších motorizovaných ramen, která jsou umístěna na stranách robota. Tento robot se může pohybovat více směry než pouze vpřed a vzad, protože je schopen pomocí bočních ramen zatáčet. Hlavní princip pohybu vpřed je správná synchronizace bočních ramen a směr jejich rotace. Zároveň je robot nucen se naučit i princip zatáčení a způsob, jakým lze dosáhnout dlouhodobě rovného pohybu. Tento robot je ovlivněn typem povrchu, po kterém se pohybuje. Důležitá je zejména křivost a přilnavost povrchu. Nerovnosti jsou pro tohoto robota problém.

7.2 Detekce provedeného pohybu

První robot používá k detekci pohybu vpřed barevné označení na dvojici předních koleček a dva barevné senzory. Pomocí snímání posloupnosti barev dokáže určit směr i délku provedeného pohybu. Na základě těchto informací určí výslednou okamžitou odměnu. Díky této detekci není robot omezen délkou uražené vzdálenosti, pokud to umožňují prostory, ve kterých se robot nachází, a může se takto pohybovat neomezeně dlouhou dobu, aniž by bylo potřeba robota někam manuálně posouvat.

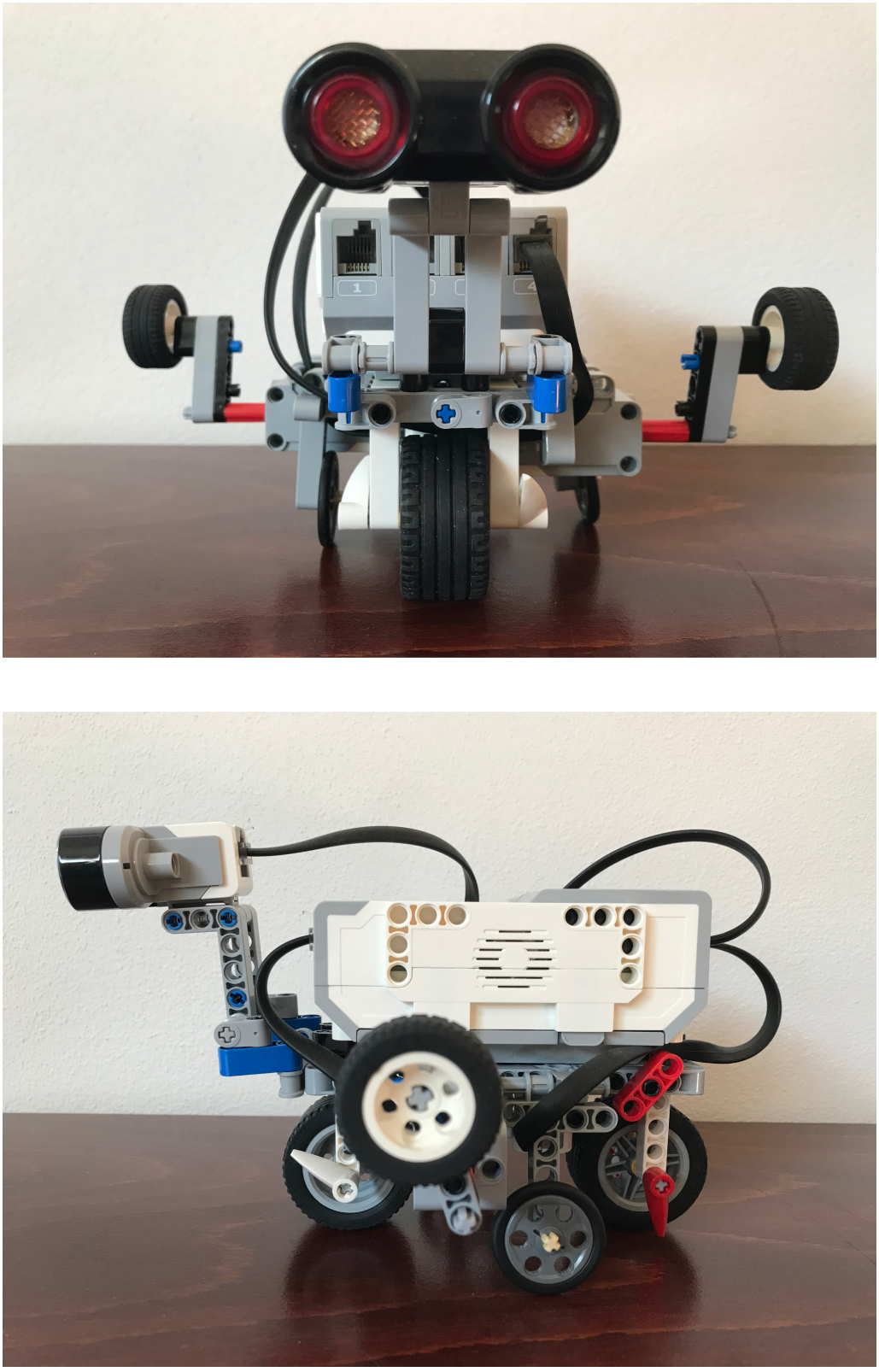
Druhý robot používá k detekci ultrazvukový senzor, který měří vzdálenost od měřeného povrchu. Při provedení přímého pohybu vpřed se tato měřená vzdálenost zvýší a tím pádem dokáže robot určit délku výsledného pohybu. Použití ultrazvukového senzoru k měření vzdálenosti vyžaduje manuální umístění robota zpět na počáteční pozici při dosáhnutí maximální měřené vzdálenosti. Způsob pohybu tohoto robota však umožňuje při prováděném pohybu zatáčet. Proto je nutné použít dodatečné senzory k detekci otočení robota. I tak není detekce provedeného pohybu spolehlivá a konzistentní.

7.3 Zhodnocení robotů

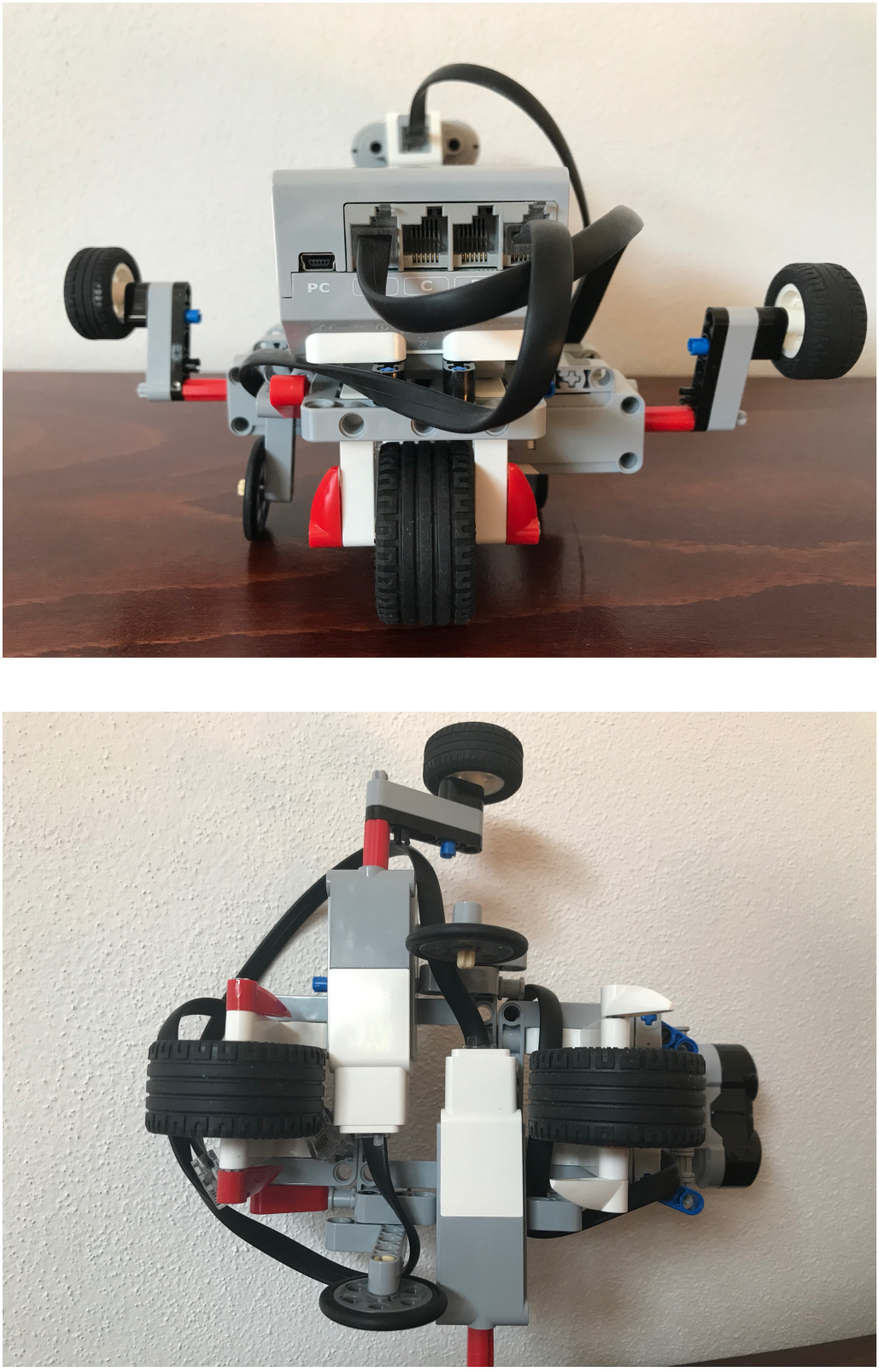
První robot byl po implementaci algoritmu Q-learning velice úspěšný a použitá konstrukce splňuje potřebné požadavky k naučení konzistentního pohybu vpřed. Robot se nakonec dokázal naučit i nadmíru efektivní způsob pohybu, který je zmíněný v sekci 5.3.

Druhý robot nevyhovoval požadavkům k implementaci fungujícího algoritmu Q-learning a nakonec jsem tento algoritmus pro tohoto robota neimplementovala.

Důvody, proč tento robot není vhodný pro implementaci algoritmu Q-learning, jsou zmíněny v sekci 6.4.



Obrázek 8: Výsledná celková konstrukce druhého robota (zadní a boční strana)



Obrázek 9: Výsledná celková konstrukce druhého robota (přední a spodní strana)

Závěr

Výsledkem této práce je robot postavený ze sady Lego Mindstorms, implementující algoritmus Q-learning. Robot se úspěšně naučil pohybovat vpřed pomocí odrážení motorizovaným ramenem. Použitím přístupu reinforcement learning robot neměl přesně stanovený vzor pohybu, který se má naučit. Díky tomuto přístupu se robot naučil neočekávaný a nadmíru efektivní pohyb vpřed. Při tomto pohybu nejprve provede pohyb vzad a přenese svoji váhu na motorizované rameno, pomocí kterého se pak velice silně odrazí vpřed. Tento způsob pohybu znázorňuje výhody přístupu reinforcement learning, při kterém má robot možnost zkoušet i jiné přístupy k řešení, které nejsou zcela typické.

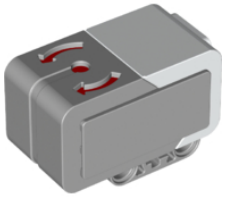
Dále jsem se pokusila postavit druhého robota, který používá k pohybu vpřed složitější pohybový systém. Robot se pohybuje odrážením dvou motorizovaných ramen, která jsou umístěna na stranách robota. Tento pohybový systém ve výsledku nebyl vhodný pro implementaci algoritmu Q-learning, protože daný pohybový systém vytvořil mnoho implementačních překážek. Robot je na rozdíl od prvního robota schopen při provedeném pohybu zatáčet, což velice komplikuje samotný proces učení. Je zapotřebí se naučit nejen samotný pohyb vpřed, ale také princip zatáčení. Další problém vznikl při návrhu detekce pohybu vpřed, jejíž spolehlivost je nezbytná pro zdařilý proces učení. Kvůli výskytu těchto problémů, jsem se nakonec rozhodla neimplementovat algoritmus Q-learning pro tohoto robota.

Conclusions

The result of this thesis is a robot built from the Lego Mindstorms set implementing the Q-learning algorithm. The robot has successfully learned how to move forward by pushing itself with the motorized arm. Using the reinforcement learning approach the robot did not have any certain kind of movement that it should learn. Thanks to this approach the robot was able to learn a very surprising and effective way to move forward. During this movement the robot at first moves backwards and shifts its weight onto the motorized arm and then the robot launches itself forward using the arm. This way of movement illustrates the advantages of reinforcement learning approach because the robot has an opportunity to try different approaches, which are not entirely standard, to solve the problem.

Next I tried to build a second robot which uses more complex system of movement to move forward. The robot moves by rotating two motorized arms which are placed on the sides of the robot. In the end this system of movement was not suitable for the implementation of Q-learning algorithm because the system of movement caused a lot of implementation issues. Unlike the first robot, this robot is able to turn and this ability substantially complicates the process of learning. The robot has to learn not only how to move forward, but also how to turn. Another problem arose during the draft of the forward movement detection. This detection needs to be very accurate and its accuracy is necessary for a successful learning process. Because of these problems I decided not to implement the Q-learning algorithm for this robot.

A Seznam použitých dílků

Číslo dílků	Obrázek	Číslo dílků	Obrázek
4121715		4611705	
4495412		4540797	
4513174		4639695	
45504		45506	
4539880		4587275	
6227055			

Tabulka 1: Seznam dílků [33]

B Obsah přiloženého CD/DVD

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src/
: Kompletní zdrojové kódy této bakalářské práce spolu s naučenou Q tabulkou `Q-table-learned.csv`. Adresář obsahuje všechny soubory, které obsahuje systém ev3dev a jsou na systému ev3dev spustitelné. Soubor `Q-learning-load.py` načítá naučenou Q tabulku `Q-table-learned.csv`, ale tento argument lze změnit.

Navíc CD/DVD obsahuje:

photos/
: Pořízené fotografie dokumentující výslednou konstrukci robota.

videos/
: Videa dokumentující pohyb daného robota a různé fáze procesu učení. Složka obsahuje i video `proces-uceni.mp4` zachycující zrychlený (300%) proces učení.

U veškerých cizích převzatých materiálů obsažených na CD/DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na CD/DVD, je uveden jejich zdroj (např. webová adresa) v bibliografii nebo textu práce.

Literatura

[1] SUTTON, Richard S; BARTO, Andrew G. *Reinforcement learning: An introduction*. Second edition. 2018. Dostupný také z: ⟨http://incompleteideas.net/book/RLbook2020.pdf⟩. ISBN 978-0-262-19398-6.

[2] CONTRIBUTORS, Wikipedia. *Go (game) — Wikipedia, The Free Encyclopedia* [online]. 2020 [cit. 2020-4-8]. Dostupný z: ⟨https://en.wikipedia.org/wiki/Go_(game)⟩.

[3] KOBER, Jens; BAGNELL, J Andrew; PETERS, Jan. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* [online]. 2013, [cit. 2020-4-3]. Dostupný z: ⟨https://www.ias.informatik.tu-darmstadt.de/uploads/Publications/Kober_IJRR_2013.pdf⟩.

[4] *Ev3dev Operating System*. [online]. [cit. 2020-3-10]. Dostupný z: ⟨https://www.ev3dev.org⟩.

[5] WIERING, Marco; OTTERLO, Martijn. *Reinforcement learning: State-of-the-Art*. 2012. Dostupný také z: ⟨http://read.pudn.com/downloads793/ebook/3130123/Reinforcement%20Learning%20State-of-the-Art.pdf⟩. ISBN 978-3-642-27644-6.

[6] MISHRA, Sanatan. *Unsupervised Learning and Data Clustering* [online]. 2017 [cit. 2019-11-10]. Dostupný z: ⟨https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a⟩.

[7] GÉRON, Aurélien. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. 2017. ISBN 978-1491962299.

[8] TIMM, Tyler. *Reinforcement Learning with Lego Mindstorms* [online]. 2016 [cit. 2019-4-3]. Dostupný z: ⟨https://www.youtube.com/watch?v=bVbT9zkPIvs⟩.

[9] *Lego Robotics Mechanisms*. [online]. [cit. 2019-12-10]. Dostupný z: ⟨http://westonk12engineering.org/robotics/pages/library_of_mechanisms.htm⟩.

[10] PAWEL, Kmieć. *The Unofficial LEGO Technic Builder's Guide*. 2013. Dostupný také z: ⟨http://data.pakkau.edu.hk/~lamchitai/ebook/No.Starch.Unofficial%20LEGO.Technic.Builders.Guide.2012.RETAIL.eBook-ZOiDB00K.pdf⟩. ISBN 978-1-59327-434-4.

[11] LEGO. *EV3 Programmer Application*. Dostupný z: ⟨https://apps.apple.com/us/app/ev3-programmer/id1039354955⟩.

[12] MICROSOFT. *Visual Studio Code*. Dostupný z: ⟨https://code.visualstudio.com⟩.

[13] EV3DEV. *ev3dev-browser*. Dostupný z: ⟨https://marketplace.visualstudio.com/items?itemName=ev3dev.ev3dev-browser⟩.

[14] EV3DEV. *Ev3dev Python Bindings* [online]. [cit. 2020-1-10]. Dostupný z: ⟨https://python-ev3dev.readthedocs.io/en/ev3dev-stretch/index.html⟩.

[15] WARD, Nigel. *EV3dev Python Multithreading* [online]. [cit. 2020-4-15]. Dostupný z: ⟨https://sites.google.com/site/ev3devpython/learn_ev3_python/threads⟩.

[16] SMITH, Allen. *Bricksmith* [online]. [cit. 2020-3-15]. Dostupný z: ⟨http://bricksmith.sourceforge.net⟩.

[17] GAGNE, Remi. *Web Lic Instruction Creator* [online]. 2018 [cit. 2020-4-20]. Dostupný z: ⟨http://bugeyedmonkeys.com/lic/about/⟩.

[18] VIOLANTE, Andre. *Simple Reinforcement Learning: Q-learning* [online]. 2019 [cit. 2020-3-6]. Dostupný z: ⟨https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56⟩.

[19] OLIPHANT, Travis E. *A guide to NumPy*. 2006.

[20] *Numpy.full Manual*. [online]. [cit. 2020-3-12]. Dostupný z: ⟨https://numpy.org/doc/stable/reference/generated/numpy.full.html⟩.

[21] *Introduction to Numpy Array*. [online]. [cit. 2020-3-12]. Dostupný z: ⟨https://www.sharpsightlabs.com/blog/numpy-array-python/⟩.

[22] WIKIPEDIE. *NumPy — Wikipedie: Otevřená encyklopedie* [online]. 2018 [cit. 2020-3-12]. Dostupný z: ⟨https://cs.wikipedia.org/wiki/NumPy⟩.

[23] EV3DEV. *Problems with on_to_position* [online]. [cit. 2020-2-20]. Dostupný z: ⟨https://github.com/ev3dev/ev3dev-lang-python/issues/608⟩.

[24] EV3DEV. *Race condition in "wait()"* [online]. [cit. 2020-2-20]. Dostupný z: ⟨https://github.com/ev3dev/ev3dev-lang-python/issues/583⟩.

[25] A line follower robot implementation using Lego's Mindstorms Kit and Q-Learning. *Acta Universitaria* [online]. 2012, [cit. 2019-12-20]. Dostupný z: ⟨https://www.redalyc.org/articulo.oa?id=41623190016⟩. ISSN 0188-6266.

[26] *Python Collections Named Tuples*. [online]. [cit. 2020-3-12]. Dostupný z: ⟨https://docs.python.org/3/library/collections.html#collections.namedtuple⟩.

[27] BROWNLEE, Jason. *How to Save a NumPy Array* [online]. 2019 [cit. 2020-3-12]. Dostupný z: ⟨https://machinelearningmastery.com/how-to-save-a-numpy-array-to-file-for-machine-learning/⟩.

[28] *Numpy.savetxt Manual*. [online]. [cit. 2020-3-12]. Dostupný z: ⟨https://numpy.org/doc/stable/reference/generated/numpy.savetxt.html⟩.

[29] EV3DEV. *Button Class*. Dostupný z: ⟨https://ev3dev-lang.readthedocs.io/projects/python-ev3dev/en/stable/button.html⟩.

[30] BUILDERDUDE35. *The Truth about the EV3 Gyro Sensor* [online]. 2015 [cit. 2020-4-3]. Dostupný z: ⟨https://www.youtube.com/watch?v=2vbNTyLkhHY⟩.

[31] SESHAN, Arvind; SESHAN, Sanjay. *Gyro Sensor Drift* [online]. [cit. 2020-4-3]. Dostupný z: ⟨http://ev3lessons.com/en/ProgrammingLessons/advanced/GyroDrift.pdf⟩.

[32] *Gyro Sensor Angle Detection*. [online]. [cit. 2020-4-3]. Dostupný z: ⟨https://makecode.mindstorms.com/reference/sensors/gyro/angle⟩.

[33] *Brickset: Your Lego set guide*. Dostupný z: ⟨https://brickset.com⟩.