VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DOKOVACÍ STANICE PRO AUTOMATICKÉ NABÍJENÍ BATERIÍ ROBOTA

DOCKING STATION FOR AUTOMATIC CHARGING OF BATTERIES OF ROBOT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JAN BERAN**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání diplomové práce

Student: **Beran Jan, Bc.**
Program: Informační technologie
Obor: Inteligentní zařízení
Název: **Dokovací stanice pro automatické nabíjení baterií robota**
Docking Station for Automatic Charging of Batteries of Robot
Kategorie: Umělá inteligence
Zadání:

1. Nastudujte metody pro plánování cesty pro roboty. Dále nastudujte principy nabíjení baterií robotů a různé principy dokovacích stanic.
2. Navrhněte systém pro nabíjení baterie na straně robota. Navrhněte i dokovací stanici, ke které se navržený robot připojí a bude se moci nabít.
3. Navrhněte a implementujte program, který umožní robotu sledovat stav jeho baterie, navede ho např. pomocí kamer nebo sonarů k vámi vytvořené dokovací stanici, počká až se baterie nabije a pak robota odpojí.
4. Program otestujte na reálném robotovi a navrhněte případná vylepšení.

Literatura:

- Howie Choset et al., Principles of Robot Motion, 2005, ISN 0-262-03327-5

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 18. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Cílem této práce je vytvořit funkční systém dokování a nabíjení pro robota Trilobot, který je v rámci této práce vylepšen a upraven pro použití v případných dalších projektech. V teoretické části jsou probrány způsoby plánování cesty, typy akumulátorů a způsoby jejich nabíjení, typy dokovacích stanic a samotného způsobu nabíjení. Také je uveden návrh renovace robota, který je poté realizován, spolu s dokovací stanicí. Renovovaný robot má k dispozici funkční ovládací software ve frameworku ROS, který mu umožňuje plnit nejen úkol autonomního dobíjení baterií. V práci je také proveden test celého systému a zhodnoceny výsledky.

Abstract

The aim of this thesis is to create a functional docking and charging system for the Trilobot robot, which is improved and modified for use in future projects as a part of this thesis. The theoretical part discusses the path planning methods, battery types and charging methods, types of docking stations and the charging method itself. Also, a robot refurbishment design is presented, which is then implemented, along with the docking station. The refurbished robot has functional control software in the ROS framework, which allows it to perform not only the task of autonomous battery charging. In this thesis, a test of the whole system is also performed and the results are evaluated.

Klíčová slova

Trilobot, robot, autonomní, nabíjení, dokovací stanice, navigace, plánování cesty, ROS, robot operating system, Raspberry Pi, Arduino.

Keywords

Trilobot, autonomous, robot, charging, docking station, navigation, path planning, ROS, robot operating system, Raspberry Pi, Arduino.

Citace

BERAN, Jan. *Dokovací stanice pro automatické nabíjení baterií robota.* Brno, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Dokovací stanice pro automatické nabíjení baterií robota

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jan Beran
24. května 2022

Poděkování

Tímto děkuji vedoucímu mé práce, Ing. Jaroslavu Rozmanovi, PhD., za ponechání volnosti při realizaci této práce a průběžnou konzultaci.

Obsah

Kapitola 1

Úvod

Autonomní mobilní robotika zažívá v posledních letech nebývalý rozmach. Mnoho firem pracuje na plně autonomních vozidlech, částečně autonomní vozidla (stupně 2 až 3 podle SAE [17]) jsou již několik let k dispozici. Existují dokonce i první pokusy o využití autonomních vozidel v hromadné dopravě [1]. Extrémní nároky na tyto systémy, především co se týče spolehlivosti a bezpečnosti, ovšem kladou před konstruktéry a výzkumníky v oblasti autonomních vozidel obrovské výzvy.

Autonomní vozidla se ovšem již nyní prosazují ve specializovaných nasazeních, jakými jsou třeba pohyb po přesně určených oblastech, například výrobních halách [20], a v omezené míře začínají pronikat i dále [7]. Autonomní vozidla a roboti ovšem slouží i k jiným účelům, než je přeprava zboží a/nebo nákladu. Již přes 20 let existují autonomní vysavače [16] nebo sekačky na trávník.

V posledních letech také zažívá rozkvět odvětví dronů. I přes to, že většina z nich je stále řízena operátorem, již několik let se objevují pokusy využít autonomní drony například k doručování jídla a dalších zásilek, případně jako čistě zábavný produkt.

Kromě civilního použití ale nacházejí technologie autonomního pohybu své uplatnění i v armádě, kde se jejich využití přímo nabízí. Ztráta autonomního vozidla/letounu, jakkoli se může jednat o sofistikovanou technologii v ceně milionů dolarů, je stále mnohem méně bolestná, než ztráta případného řidiče/pilota, který by jinak musel úkol plnit konvenčně řízeným strojem. Díky tomu, že nepilotovaný dron může být mnohem menší, je také daleko hůře odhalitelný prostředky včasné detekce, například radary.

Všechny tyto systémy ovšem mají několik společných znaků. Kromě potřeby kvalitních algoritmů pro jejich řízení také velká většina systémů operuje na elektrickém pohonu. Tato volba je důsledkem předností tohoto typu pohonu, například jeho mnohem lepší škálovatelnosti v porovnání s benzinovým pohonem, menší hlučností a emisemi během provozu. Tato volba ale vytváří i nové výzvy, zejména co se týče napájení a nabíjení. Většina autonomních systémů spoléhá na baterie různé konstrukce a kapacity. Ovšem baterie mají oproti například fosilním palivům daleko nižší hustotu energie, s čímž se pojí i nutnost vozidla častěji nabíjet.

Nabíjení by v ideálním případě mělo také probíhat autonomně, mělo by tedy být plně ve schopnostech robota. A právě tímto problémem se bude zabývat tato diplomová práce. První část se bude skládat z rešerše současných algoritmů pro plánování cesty a představení základních informací o bateriích, nabíjení a různých možnostech jeho realizace, tedy různými typy dokovacícch stanic.

Ve druhé části se bude práce věnovat návrhu a konstrukci dokovací stanice pro robota Trilobot. Vzhledem k tomu, že hardware těchto robotů je již velmi zastaralý, proběhne

jejich renovace a přestavba na úroveň, která umožní nejen využívat dokovací stanici, ale také jejich použití v budoucnu, například ve výuce nebo na propagačních akcích.

Členění práce je následující. Kapitola 2 je rešerší současných algoritmů pro plánování cesty. Budou zde popsány jak základní algoritmy, tak pokročilejší přístupy. V kapitole 3 budou popsány základní typy současných akumulátorů a bude provedeno porovnání. Zároveň budou popsány způsoby, jakými se dané akumulátory nabíjejí. Následující kapitola 4 se bude věnovat nabíjecím stanicím a nabíjení pozemních autonomních robotů. Budou zde představeny jak návrhy z vědeckých prací, tak přístupy použité například ve stanicích pro robotické vysavače. Následně v kapitole 5 bude popsán robot Trilobot a popis jeho renovace. V následující kapitole 6 bude popsán návrh nabíjecí stanice a systému. Kapitola 7 poté detailně popisuje softwarové vybavení Trilobota včetně programu, který využívá k vyhledání dokovací stanice a nabíjení. V kapitole 8 je poté provedeno testování výsledného robota. V závěrečné kapitole 9 jsou poté shrnuty výsledky práce a navrženy možnosti, kam se celý projekt může dále posouvat.

Kapitola 2

Metody plánování cesty

V této kapitole budou probrány jednotlivé metody plánování cesty. Cílem těchto metod a algoritmů na nich založených je zpravidla vytvořit cestu bez kolizí z počátečního bodu q_{start} do cílového bodu q_{goal}, nebo pokrýt celý dostupný prostor (angl. *coverage algorithms/methods*, používané například v robotických vysavačích).

Velmi také záleží, jak je reprezentována mapa, kterou má robot k dispozici:

- Pohyb v neznámém prostředí (nekompletní/žádná mapa). Jedná se o speciální případ, kdy robot ví pouze směr, kterým se nachází cíl, ale nemá k dispozici kompletní mapu prostředí. Tuto si může vytvářet během cesty, nebo pouze reaktivně reagovat na překážky.
- Topologická mapa. Mapa je charakterizována jako graf G = (V, E), kde V je množinou všech „významných míst“, typicky míst, kde robot může změnit směr (křižovatka, zatáčka, ...), E je poté množina hran, která spojuje jednotlivé vrcholy grafu. Existence hrany mezi dvěma vrcholy poté značí, že robot se může mezi těmito vrcholy přemístit zpravidla po přímce, méně často po matematicky popsané křivce (na dané spojnici nestojí překážka).
- Geometrická mapa. Taková mapa se podobá např. plánům budov a zachycuje překážky jako geometrické tvary.
- Mřížková mapa. Prostředí je rozděleno na mřížku, kde je každé buňce přidělena informace, zda je volná, obsazená, případně další informace. Speciálním případem je např. *occupation grid*, mapa vytvářená samotným robotem pomocí senzorů vzdálenosti.

Robot přitom může v jednu chvíli používat více map, například topologickou mapu pro globální plánování a mřížkovou mapu pro lokální plánování.

2.1 Náhodný pohyb

Tato metoda je ze všech nejjednodušší, byla (a stále ještě v menší míře je) používána například v některých jednoduchých robotických vysavačích. Principem této metody je, že robot se pohybuje po přímce, dokud není nucen svůj kurz změnit. V takové situaci se otočí o náhodný úhel a dál bude pokračovat opět po přímce. Po určité době pokryje celý dostupný prostor natolik hustě, že se dá v některých případech označit za kompletně pokrytý.

Z teoretického pohledu se jedná o absolutně reaktivní způsob řízení, kdy si robot nedrží žádnou informaci o stavu svého okolí.

Výhody tohoto přístupu jsou zjevné, algoritmus náhodného pohybu je natolik jednoduchý, že lze implementovat i čistě pomocí elektronických součástek bez použití mikrokontroleru [31]. Robot nemusí mít k dispozici mapu a nevadí mu ani dynamické prostředí.

Nevýhody jsou též zjevné. Použití tohoto přístupu pro nalezení cesty mezi dvěma body je vyloučené. Pro pokrytí nějaké plochy je tento algoritmus použitelný za předpokladu, že není nutné celou plochu skutečně pokrýt, ale můžeme se spokojit s „rozumným“, byť neúplným, pokrytím.

2.2 Prohledávání stavového prostoru

Pokud je mapa reprezentována topologicky (případně mřížkou), lze použít i algoritmy známé z prohledávání stavového prostoru (BFS, A*, backtracking, ...) . Pokud je navíc známa délka každé hrany (tedy vzdálenost mezi dvěma vrcholy, které hrana spojuje), lze použít i Dijkstrův algoritmus k nalezení nejkratší cesty.

Topologická reprezentace mapy je přitom grafem G = (V, E), kde V je množinou všech „významných míst“, typicky míst, kde robot může změnit směr (křižovatka, zatáčka, ...), E je poté množina hran, která spojuje jednotlivé vrcholy grafu. Existence hrany mezi dvěma vrcholy poté značí, že robot se může mezi těmito vrcholy přemístit (nestojí mu v cestě překážka).

Formálněji, jednotlivé vrcholy $v \in V$ mohou být charakterizovány jako body v prostoru, pro 2D prostor tedy jako dvojice souřadnic (x, y). Hrany $e \in E$ se poté mohou reprezentovat jako dvojice vrcholů: $e = (v_1, v_2), v1, v2 \in V$.

Může se ovšem stát, že cíl (případně ani start) nejsou vrcholy topologické mapy. V takovém případě musí robot naplánovat trasu, po které se dostane do nějakého z vrcholů mapy a poté trasu, po které sjede z mapy a dojede k samotnému cíli. K tomu je třeba použít jiných algoritmů, například bug algoritmy pro pohyb v neznámém prostředí.

2.3 Bug algoritmy

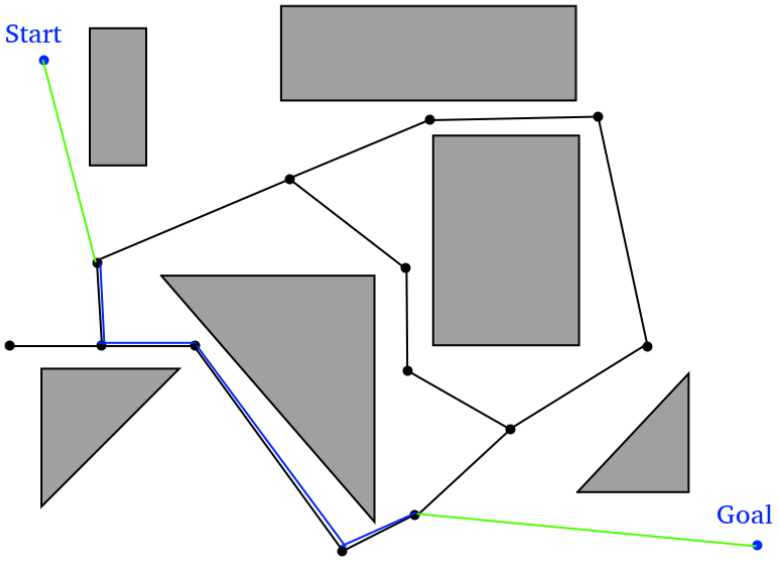
Bug algortimy patří k jedněm z nejstarších algoritmů pro pohyb ve zcela neznámém prostředí. Jsou nesmírně jednoduché a přitom garantují nalezení cesty, pokud taková existuje [11]. Níže budou prezentovány algoritmy Bug1, Bug2 a TangentBug tak, je je ve své knize prezentuje H. Choset [11]. Není-li uvedeno jinak, všechny informace o algoritmech byly čerpány z této knihy.

Bug1 a Bug2

Tyto dva algoritmy jsou určeny pro pohyb v neznámém prostředí a kladou velmi malé požadavky na senzorické vybavení robota; pro jejich použití je třeba, aby byl robot schopen detekovat náraz (případně vzdálenost od překážky) a určit svou polohu v prostoru a tedy i spojnici své polohy a cíle.

Pro oba algoritmy platí, že pokud nenarazí na překážku, pohybují se nejkratší cestou k cíli. Rozdíl nastává až ve chvíli, kdy robot na překážku narazí.

V případě algoritmu Bug1 robot nejprve označí bod nárazu například jako H nebo q^H (z angl. *hit point*, bod nárazu) a začne objíždět překážku, dokud se nedostane zpět do bodu H. Při cestě okolo překážky přitom monitoruje průběžnou vzdálenost k cíli a průběžně aktualizuje bod L, případně q^L (z angl. *leave point*), tedy bod na obvodu překážky, který je nejblíže cíli. Ve chvíli, kdy robot dosáhne opět bodu H, se přesune zpět do bodu L kratší



Obrázek 2.1: Ukázka topologické mapy (černé body a hrany mezi nimi) promítnuté do geometrické mapy. Topologická mapa přitom mohla být získána například pomocí PRM 2.7. Modře jsou označeny start a cíl. Zelené úsečky znázorňují trasu, kterou musel robot projet, než se „napojil“ na topologickou mapu, modré linky poté znázorňují trasu podle topologické mapy. Obrázek převzat z [11] a upraven.

ze dvou možných cest a z tohoto bodu pokračuje dále k cíli. Pokud by směr k cíli vedl do překážky místo od ní, neexistuje cesta, která by vedla k cíli.

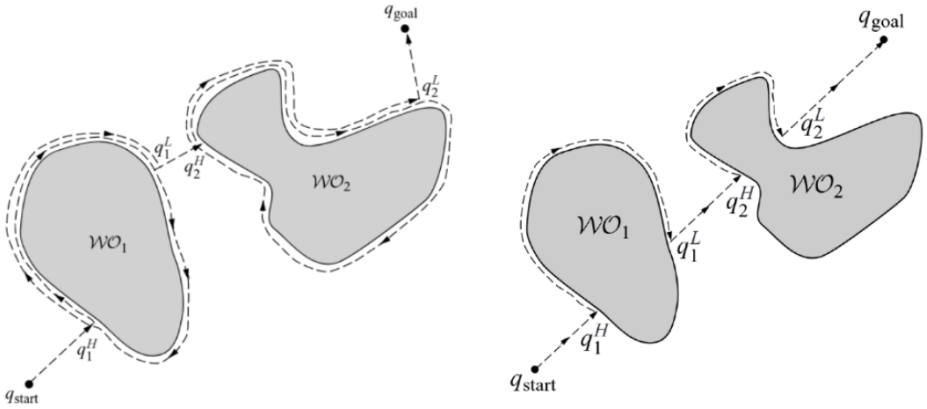
V případě algoritmu Bug2 robot také začne od bodu H objíždět překážku. Pokud ovšem narazí na spojnici startu a cíle a zbývající vzdálenost k cíli je menší než z bodu H, robot rovnou opouští překážku a vydává se k cíli. Pokud se dostane zpět do bodu H, cíl je nedosažitelný. Ukázky algoritmů jsou na Obrázku 2.2. Ač je Bug2 většinu času efektivnější, zvláště u členitých překážek může vykazovat výrazně horší chování, jak je znázorněno na Obrázku 2.3.

Tangent Bug

Tangent Bug je vylepšením algoritmu Bug2. Na rozdíl od předchozích algoritmů ke své činnosti potřebuje senzor vzdálenosti s nekonečným[1] dosahem, který snímá celé 360° okolí robota. Na Obrázku 2.4 vlevo jsou jednotlivá měření znázorněna jako paprsky. V určitých bodech dochází ke skokové změně hodnoty. Toto nastává v situaci, kdy skončí překážka nebo je jedna překážka zastíněna jinou. Tyto body nespojitosti si robot uloží do seznamu O, jednotlivé body budou označeny jako O_i. Toto je znázorněno na Obrázku 2.4 vpravo.

Samotný algoritmus funguje tak, že se robot nejprve začne pohybovat směrem k cíli po nejkratší cestě do chvíle, dokud nezaregistruje překážku na spojnici startu a cíle. To znamená, že spojnice start-cíl musí protínat jeden z intervalů kontinuity (intervaly mezi dvěma body O_i, O_j, mezi kterými funkce vzdálenosti nevykazuje skokovou změnu).

[1] Je možné použít i senzor s konečným rozsahem. V takovém případě bude za nekonečno považován maximální dosah senzoru.



Obrázek 2.2: Ukázka algoritmů Bug1 (vlevo) a Bug2 (vpravo). Na první pohled se jeví, že algoritmus Bug2 je efektivnější, ovšem například v případě 2.3 dochází k tomu, že Bug2 zbytečně objíždí část překážky vícekrát. Obrázek přejat z [11].

Nejprve se překážka jeví jako jediný bod nespojitosti O_i, který se vzápětí rozdělí na dva body nespojitosti a vznikne tak drobný interval kontinuity. Nyní se robot začne pohybovat k jednomu z bodů nespojitosti. Bod vybírá s cílem minimalizovat nějakou heuristickou funkci, například $h(x, O_i) = d(x, O_i) + d(O_i, goal)$, kde x je současnou polohou robota, O_i je konkrétní bod nespojitosti a $d(A, B)$ je funkcí vzdálenosti mezi body libovolnými body body A a B, v tomto případě O_i a $goal$. Body nespojitosti jsou přitom průběžně aktualizovány a mění se s každým časovým okamžikem, směr je tedy neustále přepočítáván, jak je vidět na Obrázku 2.5.

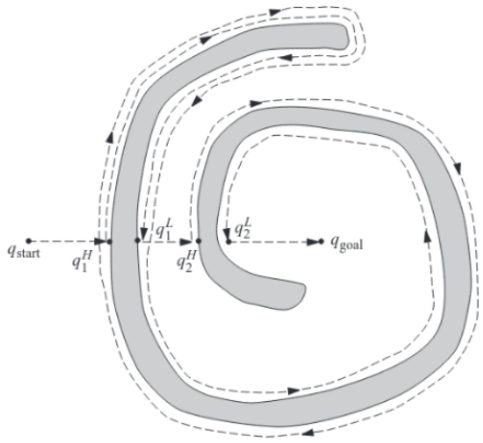
Může se ovšem stát, že heuristická funkce již dále nepůjde minimalizovat – to nastává v okamžiku, kdy se robot dostane do lokálního minima (typickým příkladem jsou překážky ve tvaru podkovy). V tuto chvíli najde bod M, tedy bod na obvodu překážky, který má nejmenší vzdálenost k cíli. Tuto vzdálenost si označí jako $d_{followed}$. Následně pokračuje v pohybu směrem, kterým se pohyboval doposud, přičemž sleduje hranici překážky. Během pohybu přitom aktualizuje dvě hodnoty: již zmíněnou $d_{followed}$ a zároveň hodnotu d_{reach}. $d_{followed}$ je přitom nejmenší vzdálenost mezi dosud prozkoumanou částí obvodu překážky a cílem, d_{reach} je vzdálenost mezi cílem a nejbližším bodem na části obvodu překážky, kterou robot momentálně pozoruje.

Sledování obvodu překážky ukončí ve chvíli, kdy $d_{reach} < d_{followed}$. V tu chvíli se vrací do módu sledování přímky k cíli.

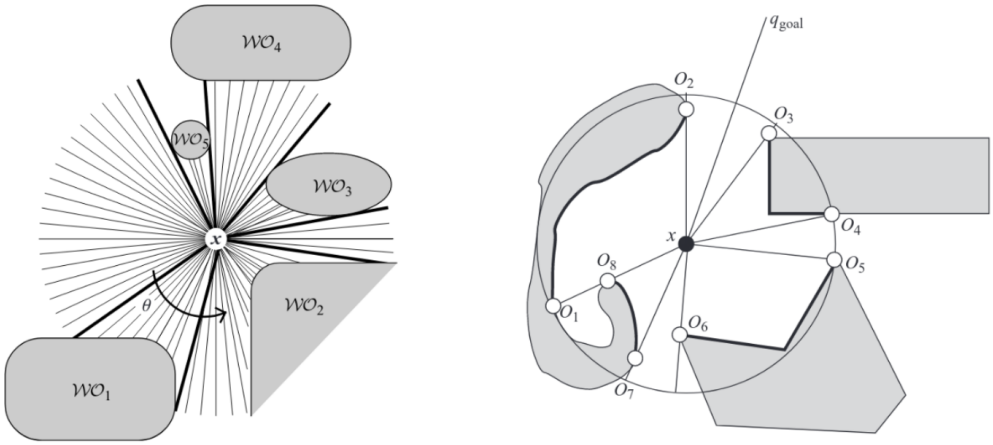
2.4 Plánování cesty na principu potenciálového pole

Následující metody budou založeny na existenci tzv. *potenciálového pole*, které je definované *potenciálovou funkcí*, u které se předpokládá, že je diferencovatelná. Tato funkce poté přiřazuje každému bodu v prostoru nějakou reálnou hodnotu q. Dále s v těchto algoritmech pracuje s pojmem *gradient*. Neformálně řečeno, gradient v určitém bodě prostoru je vektorem, který určuje směr maximálního růstu potenciálové funkce. Pro formálnější definici gradientu doporučuji přílohu C.5 z [11].

Idea algoritmů je poté taková, že potenciálová funkce představuje energii a její gradient poté působící sílu. Robot je poté kladně nabitá částice, která se pohybuje k záporně



Obrázek 2.3: Ve většině případů platí, že algoritmus Bug2 je efektivnější, ovšem například v tomto případě dochází k tomu, že Bug2 zbytečně objíždí část překážky vícekrát. Obrázek přejat z [11].



Obrázek 2.4: Ukázka bodu nespojitosti a intervalů kontinuity (například tučně vyznačený interval mezi body O_3 a O_4. Obrázek přejat z [11].

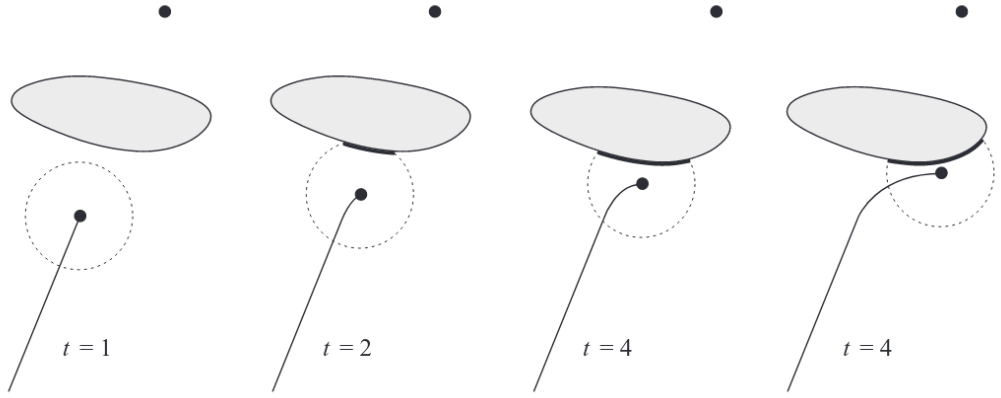
nabitému cíli. Překážky jsou přitom také považované za kladně nabité, což by v ideálním případě mělo způsobit, že se jim robot bude snažit vyhýbat. Ilustrace výše popsaného je k dispozici na Obrázku 2.6.

Nejjednodušší potenciálovou funkcí je přitom prostý součet přitažlivých a odpudivých potenciálů v daném místě:

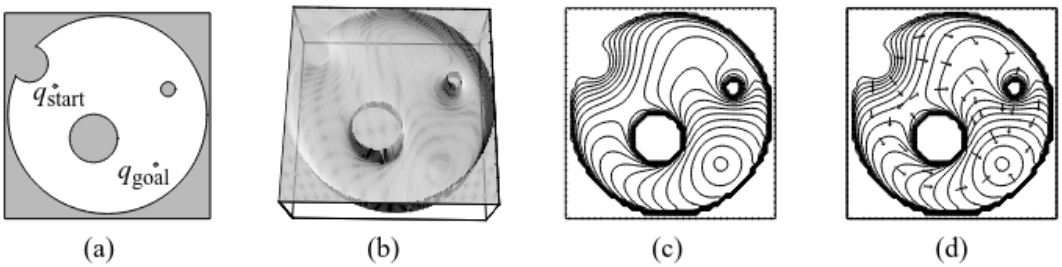
$$U(q) = U_{att}(q) + U_{rep}(q) \tag{2.1}$$

kde U(q) je celková hodnota potenciálové funkce v bodě q, $U_{att}(q)$ je přitažlivá složka v tomto poli (daná zpravidla pouze cílem) a $U_{rep}(u)$ je odpudivá složka, daná překážkami v prostředí.

Níže budou představeny dva algoritmy, které umožní použití výše zmíněné potenciálové funkce. Prvním z nich je brushfire algoritmus, který vyplňuje dostupný prostor vzdálenostmi k nejbližší překážce. Výsledné hodnoty v jednotlivých buňkách se používají pro výpočet



Obrázek 2.5: Ukázka objíždění překážky algoritmem Tangent Bug. Obrázek přejat z [11].



Obrázek 2.6: Idea potenciálového pole. (a) mapa prostředí s šedými překážkami a bílým volným prostředím. (b) Vizualizace potenciálové funkce, všimněte si prohlubně v místě, kde se nachází cíl. (c) Jednotlivé vrstevnice potenciálové funkce, Cílová prohlubeň je zde lépe viditelná. (d) Šipky znázorňují opačný směr než gradient, tedy směr k cíli. Obrázek přejat z [11].

odpudivé části funkce $U(q)$ z 2.1. Druhým algoritmem je záplavový algoritmus, který se naopak používá k získání druhé, přitažlivé složky funkce $U(q)$.

U následujících dvou algoritmů se bude používat termín „sousedství“ dvou buněk. V základu existují dva typy sousedství/okolí: 4-okolí a 8-okolí. V případě 4-okolí se za sousední buňky považují ty v základních směrech (nahoru, dolů, doprava, doleva), v případě 8-okolí se přidávají ještě buňky, které leží na diagonále od centrální.

Brushfire algoritmus

Vstupem algoritmu je mapa prostředí rozdělená na buňky, kde se prázdným buňkám přiřadila hodnota 0 a buňkách, které jsou (byť třeba jen částečně) obsazeny překážkou hodnota 1. Výstupem je poté mapa prostředí, kde se v každé volné buňce nachází vzdálenost k nejbližší překážce.

Algoritmus poté pracuje tak, že v prvním kroku přiřadí všem buňkám, jejichž hodnota je nulová, které sousedí s buňkou s hodnotou 1, novou hodnotu 2. Další kroky pokračují tak, že se všem nulovým buňkám, které sousedí s buňkami o hodnotě i, přiřadí hodnota $i + 1$. Toto se opakuje, dokud nejsou zaplněny všechny buňky.

Záplavový algoritmus

Záplavový algoritmus má také na vstupu mapu prostředí, rozdělenou do buněk, kde prázdná buňka má hodnotu 0, obsazená poté hodnotu 1.

Nyní se ovšem označí hodnotou 2 cílová buňka. Následně se všem sousedním buňkám, jejichž hodnota je nulová, přiřadí hodnota 3. V dalším kroku se všem nulovým buňkám, které sousedí s buňkami s hodnotou 3, přiřadí hodnota 4. Další kroky opět pokračují tak, že se všem nulovým buňkám, které sousedí s buňkami o hodnotě i, přiřadí hodnota $i + 1$. Toto se opakuje, dokud nejsou zaplněny všechny buňky.

Kombinací hodnot ze záplavového a brushfire algoritmu lze zkonstruovat plánovač, který bude vytvářet cestu do cíle a zároveň bude udržovat rozumnou vzdálenost od překážek.

2.5 Roadmapy

Roadmapy jsou podmnožinou topologických map, které se snaží obsáhnout „hlavní trasy“ v daném prostředí. Cílem roadmap přitom není umožnit robotovi dostat se z libovolného místa do libovolného jiného místa, ale poskytnout strukturu, která jednoduše a přitom efektivně popisuje možnosti pohybu mezi jednotlivými lokacemi dané mapy.

Funkci roadmap jde přirovnat například k železniční síti. Vlaky umožňují rychlou přepravu mezi významnými místy (větší města), ovšem do menších měst nebo konkrétních lokalit v daných městech je již třeba použít jiný způsob dopravy. Roadmapy stejně tak umožňují robotovi se rychle a bezpečně pohybovat mezi jednotlivými lokacemi na mapě, ovšem do konkrétních bodů se již robot musí dostat jinak. Situaci ilustruje například Obrázek 2.1.

Roadmapa pak může být popsána jako graf G = (V, E), kde V je množinou všech „významných míst“, typicky míst, kde robot může změnit směr (křižovatka, zatáčka, ...), E je poté množina hran, která spojuje jednotlivé vrcholy grafu. Existence hrany mezi dvěma vrcholy poté značí, že robot se může mezi těmito vrcholy přemístit (nestojí mu v cestě překážka). Formálněji, jednotlivé vrcholy $v \in V$ přitom mohou být charakterizovány jako body v prostoru, pro 2D prostor tedy jako dvojice souřadnic (x, y). Hrany $e \in E$ se poté mohou reprezentovat jako dvojice vrcholů: $e = (v_1, v_2), v1, v2 \in V$.

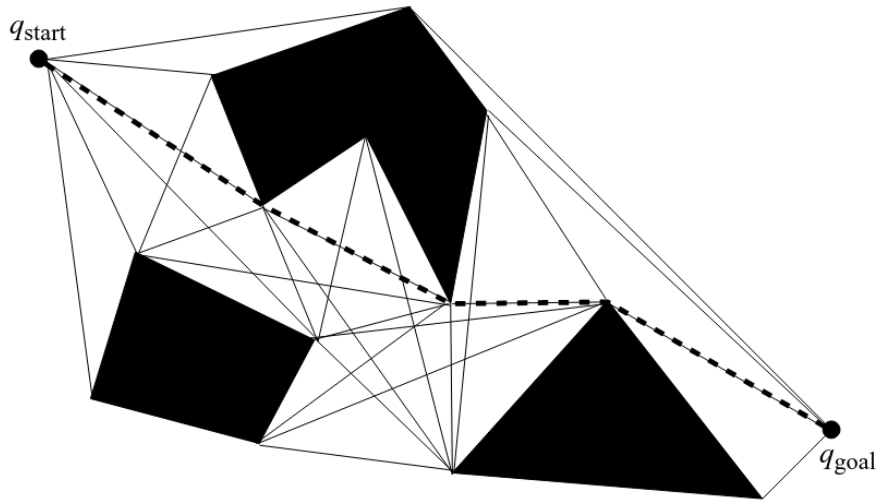
Logicky z jejího určení plynou i další požadavky na roadmapu. Pro každou dvojici bodů q_{start} a q_{goal}, které se nenacházejí uvnitř překážek a nacházejí se uvnitř daného prostředí:

- Musí existovat cesta z q_{start}, kterou se dá napojit do nějakého bodu v, který je jedním z uzlů roadmapy.
- Musí existovat cesta, kterou se lze z nějakého bodu roadmapy u dostat do bodu q_{goal}.
- Mezi body u a v musí v roadmapě existovat cesta.

Roadmapy přitom mohou být různé. V této práci budou představeny dva typy: graf viditelnosti a Voroného diagram, který lze také použít jako roadmpau.

Graf viditelnosti

Graf viditelnosti je nejjednodušší verzí tzv. mapy viditelnosti. V základní podobě se jedná o graf, jehož vrcholy jsou tvořeny: startem, cílem a všemi vrcholy překážek v prostředí. Hrany grafu poté spojují ty vrcholy, které jsou přímo viditelné. Přímo viditelné jsou poté takové vrcholy, které jdou spojit přímkou. Ukázka grafu viditelnosti je na Obrázku 2.7.



Obrázek 2.7: Ukázka grafu viditelnosti. Obrázek přejat z [11].

V [11] se dále diskutují způsoby, jak celý graf upravit tak, aby bylo nalezení cesty efektivnější, jelikož ve své základní podobě obsahuje graf mnoho zbytečných hran. Detaily lze najít tamtéž.

Voroného diagram

Voroného[2] diagram rozděluje rovinu do oblastí, které jsou dány předem určenými body. Jednotlivé oblasti přitom vždy náleží některému z bodů a platí, že body v dané oblasti jsou nejblíže tomuto danému bodu (ilustrace na Obrázku 2.8). V kontextu plánování cesty jsou přitom zajímavé hranice těchto oblastí, což jsou body, které mají od dvou nejbližších bodů stejnou vzdálenost. Jelikož ale překážky nebývají jen samotnými body, ale složitějšími geometrickými oblastmi, je definice Voroného diagramu lehce upravena: hranice jednotlivých oblastí jsou ty body, které mají stejnou vzdálenost od dvou nejbližších překážek. Takovému diagramu se říká generalizovaný Voroného diagram.

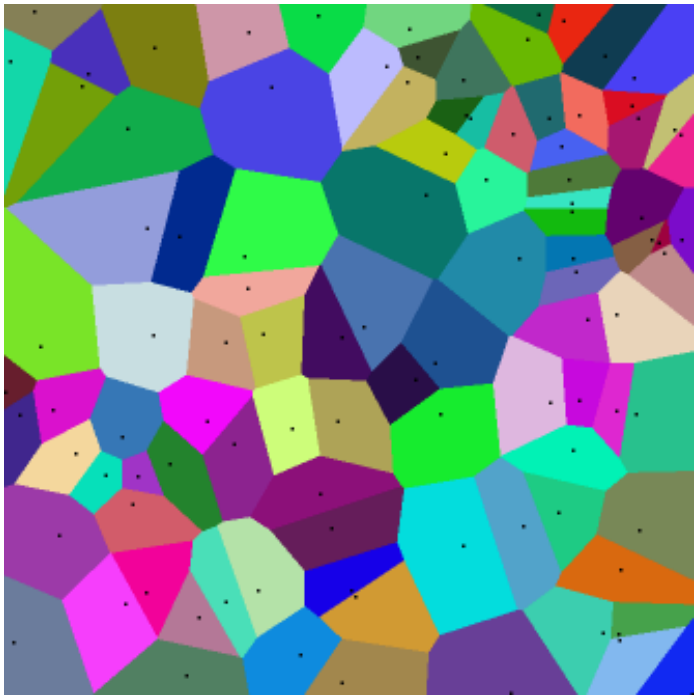
Po takto zkonstruované roadmapě se pohybuje podobně, jak již bylo řečeno; robot se nejdřív napojí na roadmapu tak, že dorazí k nejbližšímu bodu této roadmapy, následně se dostane do blízkosti cíle a z roadmapy sjede.

2.6 Rozdělování prostoru na buňky

Principem této rodiny přístupů je rozdělení volného prostoru do buněk. Buňky si lze představit jako určité oblasti volného prostoru, které mají nějaké vlastnosti (například lichoběžníkovitý tvar, konvexnost...). Pokud spolu buňky sdílí alespoň jednu hranu, nazývají se sousedními buňkami.

Dalším krokem je často konstrukce tzv. grafu sousednosti. Vrcholy tohoto grafu reprezentují jednotlivé buňky, hrany poté sousednost tak, jak je popsaná výše.

[2]Občas se lze setkat i s podobou „Voronoiův“, což je ovšem nesprávně.



Obrázek 2.8: Voroného diagram. Obrázek přejat z `https://commons.wikimedia.org/wiki/File:Coloured_Voronoi_2D.png`.

Lichoběžníková dekompozice

Nejpopulárnější metodou je dekompozice na lichoběžníky. Metoda předpokládá, že překážky mají tvar obecného lichoběžníku. Jednotlivé buňky mají poté tvar konvexních lichoběžníků, vzácně potom mohou být degenerovány na trojúhelníky.

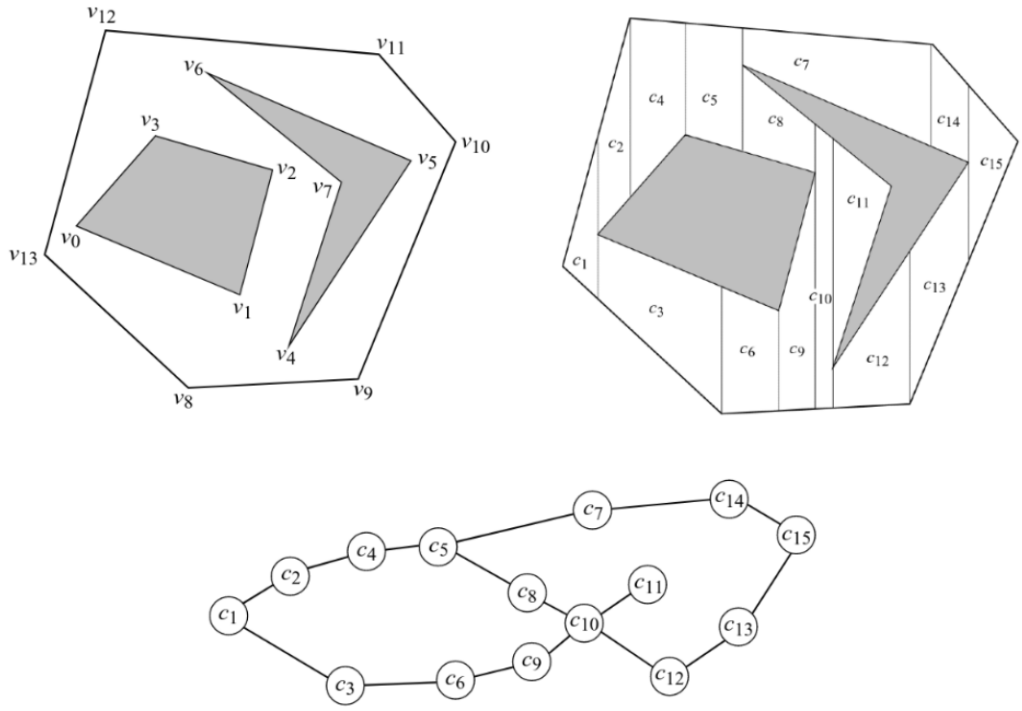
Idea této metody je následující:

1. Vytvoření seznamu V všech vrcholů všech překážek. K těmto vrcholům jsou přidány i vrcholy hranice prostoru (vizte Obrázek 2.9 vlevo). Předpokládá se přitom, že každý vrchol má jedinečnou souřadnici x.

2. Následně se vytvoří tzv. „horní a dolní vertikální prodloužení“. Ve své podstatě se jedná o vertikální úsečky proložené každým z vrcholů $v \in V$. Počátečními body těchto úseček jsou první průsečíky s libovolnou hranou, jak je znázorněno na Obrázku 2.9 vpravo. Tímto vzniknou lichoběžníkové buňky. Samotný vrchol je poté buď krajním bodem (například v_1), v takovém případě má jen horní/dolní prodloužení. Pokud není koncovým bodem, pak ji rozděluje na dvě samostatné úsečky, které budou hranami dvou různých buněk.

3. Následně je vytvořen graf sousednosti dle popisu výše (Obrázek 2.9 dole).

Cesta se v takto vzniklém grafu již hledá snadno. Nejprve se určí, v jaké buňce leží start a cíl a následně se v grafu najde nejkratší cesta. Princip transformace sekvence vrcholů na cestu, kterou může robot projet, je poté takový, že se naleznou středy společné hrany vždy dvou po sobě jdoucích buněk. Tyto středy se poté mohou bezpečně spojit přímkami Po přidání přímky z počátečního bodu do prvního středu a z posledního středu do cíle vznikne samotná cesta.



Obrázek 2.9: Ukázka jednotlivých fází při buňkové dekompozici. Obrázek přejat z [11].

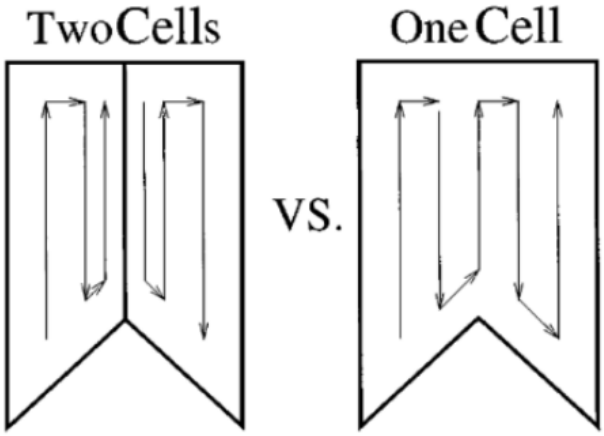
Boustrophedon dekompozice

Buňková dekompozice, na rozdíl od jiný algoritmů, které zde byly dosud představeny, mohou být použity i za účelem tzv. *coverage planning*, plánování pokrytí. Tento úkol řeší například robotické vysavače. Předchozí metoda lze na podobné úlohy použít s tím, že robot postupně pokrývá jednotlivé lichoběžníkové buňky a pomocí grafu sousednosti se ujišťuje, že je pokryt skutečně celý prostor. Bohužel, v některých situacích se robot na hranici buněk chová neoptimálně – například zbytečně vykonává dvě cesty velmi blízko hranicím obou buněk atp. Příklad je na Obrázku 2.10.

Boustrophedon[3] dekompozice je poté jednou z možností, jak optimalizovat lichoběžníkovou metodu [10]. Rozdílem oproti předešlé metodě je to, že nyní jsou při tvorbě vertikálních prodloužení uvažovány pouze ty vrcholy, které mají horní i dolní prodloužení. Jinými slovy, vrchol není krajním bodem vertikálního prodloužení. Následně je opět vytvořen graf sousednosti jako v předchozím případě.

Plánování pokrytí poté funguje tím způsobem, že plánovač naplánuje trasu všemi vrcholy grafu, například pomocí *depth first search*, prohledávání do hloubky. Ve chvíli, kdy je známe pořadí, v jakém budou buňky navštíveny, plánovač vytvoří konkrétní trasu pro každou buňku. Tato trasa přitom vypadá podobně jako brázdy na poli (ilustrace na Obrázku 2.10).

[3]Z řeckého *boustrophedon*, což přibližně znamená „orání volem“.



Obrázek 2.10: Vizualizace neefektivity lichoběžníkové metody (vlevo) oproti boustrophedon metodě dekompozice. Obrázek přejat z [10].

2.7 Vzorkovací algoritmy

Vzorkovací (v češtině také někdy pravděpodobnostní) algoritmy fungují na principu náhodného vzorkování z množiny všech možných konfigurací robota a zjišťování, zda jsou tyto konfigurace bezkolizní. V případě, že je mobilní robot uvažován jako bod, konfigurací tohoto robota se v takovém případě rozumí poloha bodu, který ho reprezentuje.

Pravděpodobnostní roadmpay – PRM

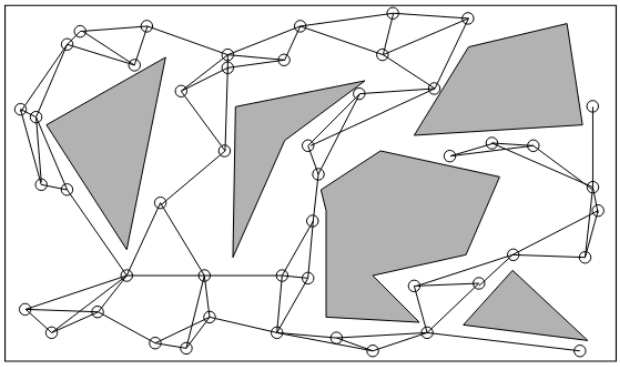
Roadmapy již byly popsány v 2.5. Ve stručnosti, roadmapa je grafem, který umožňuje jednoduché cestování mezi sekcemi mapy, přičemž cesta z počátečního bodu do bodu na roadmapě a obráceně, při „sjíždění“ z roadmapy, musí být vytvořeny jinak. Uvnitř roadmapy se ale cesta hledá jednoduše jako nejkratší cesta grafem.

Metoda PRM má přitom dvě fáze:

- Fáze učení. V této fázi se ustavuje roadmapa. Nejprve je navzorkován určitý počet konfigurací, u kterých je ověřeno, že jsou bezkolizní. Následně se tyto konfigurace (v případě 2D robota jsou konfiguracemi body na ploše, v případě 3D pak body v prostoru) spojovány s cílem vytvořit roadmapu. Způsob spojování se přitom může lišit. Lze například vybrat dva libovolné body a pokusit se je spojit – nejčastěji přímkou – tak, aby jejich spojnice byla bezkolizní. Body se také mohou vybírat na základě vzdálenosti atp. Ukázka možného výsledku je na Obrázku 2.11
- Fáze dotazování. Jakmile je jednou roadmapa ustanovena, je již možné se nad ní dotazovat stejně jako nad normální roadmapou. Mohou přitom vznikat snahy připojit počáteční a cílový bod do roadmapy jako nově navzorkované body. Pokud se ovšem toto nepodaří (start/cíl nelze připojit přímkou k žádnému bodu roadmapy), platí, že je třeba zkonstruovat cestu na roadmapu jiným způsobem.

PRM je přitom spíše celá rodina různých metod a algoritmů, jelikož lze měnit velké množství parametrů (způsob vzorkování a propojování vzorků, limit pro velikost roadmapy...).

Výhodou PRM je například to, že díky vzorkování je možné řešit i úlohy, které by byly neřešitelné prostřednictvím konvenčních algoritmů. Nevýhodou je většinou horší řešení dané náhodným charakterem vzorků.



Obrázek 2.11: Ukázka výsledné pravděpodobnostní roadmapy. Na první pohled je patrná neoptimalita. Obrázek přejat z [11].

2.8 Další možné přístupy

Prozatím zde byly popsány pouze konvenční algoritmy plánování cesty. Existují ovšem i další přístupy, založené např. na umělé inteligenci nebo inteligenci davu/hejna [6]. Pro příklad zde bude představeno plánování cesty založené na optimalizaci mravenčí kolonií (ACO, *ant colony optimalization*).

Na úvod zde bude stručně představen algoritmus ACO. Algoritmus předpokládá, že mapa prostředí je uložena ve formě grafu. Jednotliví agenti, v kontextu algoritmu ACO nazývaní mravenci, se následně z počátečního uzlu grafu přemisťují do dalších uzlů, dokud nedojdou do cílového uzlu. Pravděpodobnost přesunu mravence a z uzlu i do uzlu j za předpokladu, že mezi těmito dvěma uzly existuje hrana, je dána 2.2. Pokud hrana mezi těmito uzly neexistuje, je pravděpodobnost přesunu přirozeně nulová.

$$p_{ij}^{a} = \frac{(\tau_{ij}^{a})^{\alpha}(\eta_{ij}^{a})^{\beta}}{\sum_{k \in N}((\tau_{ik}^{a})^{\alpha}(\eta_{ik}^{a})^{\beta})} \tag{2.2}$$

V 2.2 je τ_{ij}^{a} úrovní feromonů na trase mezi i a j pro mravence a, η_{ij}^{a} reprezentuje heuristickou informaci („kvalitu“ dané volby na základě dostupných informací). Koeficienty α, β jsou váhovými koeficienty, které určují vliv jednotlivých složek. Množina N je poté množinou všech sousedních uzlů.

Poté, co všichni mravenci dojdou do cílového stavu, jsou následně upraveny úrovně feromonů na cestách. Nejprve dojde k jejich částečného vypaření s koeficientem ρ 2.3, poté výpočtu nových úrovní feromonů 2.4.

$$\tau_{ij} = (1 - \rho)\tau_{ij} \tag{2.3}$$

$$\tau_{ij} = \tau_{ij} + \sum_{a=0}^{A-1} \Delta\tau_{ij}^{a} \tag{2.4}$$

$$\Delta\tau_{ij}^{a} = \frac{1}{C^{a}} \tag{2.5}$$

Číslo A udává počet mravenců, $\Delta\tau_{ij}^{a}$ je poté změnou úrovně feromonu na hraně mezi vrcholy i a j, která je způsobena mravencem a.

Následně se celý proces opakuje v dostatečném počtu iterací. Populace mravenců má poté tendenci konvergovat k takové cestě, která není nutně nejkratší možná, ovšem velmi se jí blíží. Algoritmus je vhodný i pro dynamické prostředí, kde při přidání překážky jsou mravenci schopní nalézt alternativní trasu. Předpokladem je ovšem resetování úrovní feromonů v okolí překážky.

Kapitola 3

Akumulátory a jejich nabíjení

Akumulátory slouží jako přenosné nebo záložní zdroje energie. Tato zařízení jsou schopna měnit elektrickou energii na energii chemických vazeb a později ji opět uvolnit, čímž je i vymezen rozdíl oproti kondenzátorům, které fungují čistě na fyzikálním principu a nedochází k žádným chemickým změnám na jejich elektrodách. Akumulátory lze opakovaně nabíjet, čímž se liší od tzv. primárních článků, které jsou pouze na jedno použití.

Členění kapitoly je následující. Nejprve se v sekci 3.1 popíší různé technologie akumulátorů a následně se v sekci 3.2 vysvětlí principy jejich nabíjení.

3.1 Rozdělení

Akumulátory mohou mít různý chemický princip, na kterém jsou založeny. V této sekci budou probrány ty nejpoužívanější, konkrétně Ni-Cd, Ni-MH, Li-Ion a Li-Pol technologie.

Ni-Cd

Ni-Cd akumulátory mají zápornou elektrodu tvořenou kadmiem a kladnou elektrodu tvořenou oxid-hydroxidem niklitým – NiO(OH). Elektrolyt tvoří zásaditá látka; zpravidla vodný roztok hydroxidu draselného (KOH). Elektrody přitom mohou být volně zaplavené, pro osobní použití se vyrábějí i akumulátory s hermetizovanými články. Napětí jednoho článku je přitom 1.2 V.

Nikl-kadmiové akumulátory se v dnešní době již příliš nepoužívají a jsou nahrazovány Ni-MH a Li-Ion bateriemi. Jedním z hlavním důvodů úpadku tohoto typu akumulátorů je především přítomnost kadmia jakožto záporné elektrody. Tento prvek je přitom toxický a baterie musí být odborně likvidovány, jinak by hrozila rizika nejen pro jedince, ale případně i pro přírodu. Dalšími nevýhodami tohoto typu akumulátorů jsou paměťový efekt nebo poměrně vysoké samovybíjení.

Výhodami jsou ovšem dlouhá životnost z pohledu počtu cyklů.

Ni-MH

Záporná elektroda byla u tohoto typu nahrazena kovovou slitinou, kladná je přitom stejná jako u Ni-Cd akumulátoru. Napětí zůstalo zachováno na hodnotě 1.2 V. Ni-MH akumulátory zažívaly svou popularitu před nástupem Li-Ion baterií, především v první dekádě 21. století, ale používají se i nyní, jako nabíjecí náhrada klasických tužkových baterií. Nižší napětí (1.2 V Ni-MH článku versus 1.5 V u klasické tužkové baterie) může v některých aplikacích

způsobit problémy. Například v nízkopříkonových zařízeních (bezdrátové myši, digitální hodiny...), je ovšem záměna za Ni-MH články možná.

Ni-MH články se používaly i v mobilních telefonech, dnes jsou ovšem vytlačeny Li-Ion/Li-Pol bateriemi a jejich hlavní použití tedy spočívá v náhradě jednorázových článků tam, kde to zařízení umožňují.

Olověné akumulátory

Olověné akumulátory jsou nejstarší technologií nabíjecích článků. Zápornou elektrodu v nabitém stavu tvoří čisté houbovité olovo, kladnou elektrodu tvoří oxid olovičitý PbO_2. Elektrolytem je zde vodný roztok kyseliny sírové o koncentraci kolem 35 % obj. Při vybíjení se poté obě elektrody mění na síran olovnatý $PbSO_4$, při nabíjení probíhá opačná reakce. Existují přitom dva hlavní typy těchto akumulátorů. První typ má elektrolyt volně nalitý mezi elektrodami, druhý typ má elektrolyt např. nasáklý do skelné vaty nebo ztužený na gel, což zvyšuje bezpečnost.

Ač jsou použité suroviny při výrobě baterií poměrně neekologické a nebezpečné, kvůli jejich nízké ceně, jednoduché konstrukci a poměrně vysoké míře recyklovatelnosti jsou stále velmi používané, a to především tam, kde nevadí jejich vysoká hmotnost (spojená s nízkou energetickou hustotou). Typickým případem užití těchto baterií jsou automobily a záložní zdroje energie (UPS).

Li-Ion

Lithium iontové akumulátory (spolu s lithium polymerovými) patří k nejmodernějším a nejpoužívanějším akumulátorům současnosti. V tomto případě je kladná elektroda vyrobena z uhlíku, záporná elektroda poté z oxidu kovu (například oxid kobaltlithný $LiCoO_2$). Elektrolytem je poté sůl lithia (hexafluorofosforečnan lithný $LiPF_6$).

Mezi hlavní přednosti těchto baterií patří především vysoká energetická hustota, možnost různých tvarů baterií a přiměřená životnost. Na druhou stranu, při nesprávném použití můžou tyto baterie začít hořet, v krajním případě i explodovat. Recyklace jakékoli lithiové baterie je v současné době velmi nákladná, rozmachu těchto baterií to ale přesto nezabránilo.

Li-Pol

Lithium polymerový akumulátor vznikl dalším vývojem a evolucí Li-Ion akumulátorů. Kladná elektroda je tvořena opět například oxidem kobaltlithným, záporná poté čistým lithiem nebo jeho sloučeninami s uhlíkem. Elektrolytem je vodivý polymer (polyethyleneoxid).

Oproti Li-Ion akumulátorům má menší riziko exploze při nesprávném zacházení a může dosahovat větší energetické hustoty.

Typ akumulátoru	Ni-Cd	Ni-MH	Olověné	Li-Ion	Li-Pol
Energetická hustota [Wh/kg]	45-80	60-120	30-50	100-160	100-260
Životnost (počet cyklů)	1000-2500	700-1000	300-1000	1000	300*
Napětí jednoho článku [V]	1.2	1.2	2	3.7	3.7
Samovybíjení [% za měsíc]	10-20	30	5	2	0.3

Tabulka 3.1: Přehled základních charakteristik nabíjecích baterií. Data jsou čerpána z [8] a případně aktualizována dle aktuálního stavu. *Vizte poznámku pod čarou.

Přehledové shrnutí

V tabulce 3.1 jsou shrnuty všechny podstatné údaje o výše popsaných typech akumulátorů[1].

3.2 Nabíjení

Obecně vzato neexistuje jeden univerzální způsob, kterým lze nabíjet všechny typy akumulátorů. Pomineme-li rozdílné nominální napětí jednoho článku akumulátoru napříč typy, některé typy baterií dosahují lepších výsledků při rychlejším nabíjení (proud až 1 C u Ni-Cd), u některých akumulátorů je lepší nabíjet je pomaleji.

U Ni-Cd akumulátorů je situace poměrně jednoduchá a existuje několik variant nabíjení. Nejjednodušší nabíječky používají buď nabíjení konstantním proudem, který je roven cca 1/10 C bez ukončení nabíjení nebo s ukončením nabíjení ve chvíli, kdy teplota článku dosáhne určené hodnoty (např. 50 C). Tento systém je sice velmi jednoduchý, ale uživatel musí sám ukončit nabíjení, nebo se smířit s tím, že baterie bude přebíjená (čímž se začne zvedat teplota, která nabíjení ukončí), což snižuje její životnost. Ve druhém zmíněném případě (odpojení po dosažení maximální určené teploty) lze nabíjet baterii i vyššími proudy a zkrátit tak čas nabíjení.

Poněkud méně drastickým způsobem je tzv. „dT/dt“ metoda, kdy se sleduje změna teploty za čas. Při tomto způsobu se nabíjení ukončí při změně 1 C za minutu. Nejlepším způsobem je ovšem sledovat pokles napětí, který se u Ni-Cd (a v menší míře i u Ni-MH) akumulátorů děje po překročení 100% kapacity. Při ukončení nabíjení pomocí poklesu napětí je ale třeba nabíjet vyššími proudy (kolem 0.5 C nebo více) pro zajištění výraznějšího poklesu. Nové, mikroprocesorem řízené nabíječky ovšem jsou schopny detekovat i menší poklesy.

Je třeba mít na paměti, že Ni-Cd akumulátory trpí na tzv. paměťový efekt – pokud nejsou tyto akumulátory před nabitím plně vybity, snižuje se jejich kapacita. Tento fenomén vzniká, když se na záporné elektrodě začnou vytvářet krystaly, které znemožní správné fungování baterie. Lze je rozpustit pomocí pulzního nabíjení (nabíjení pomocí relativně krátkých pulzů relativně vysokým proudem).

Jednotlivé techniky nabíjení přitom jdou kombinovat, nabíjení se poté ukončí po dosažení alespoň jedné ukončovací podmínky (maximální teplota, změna teploty za čas, pokles napětí, ...).

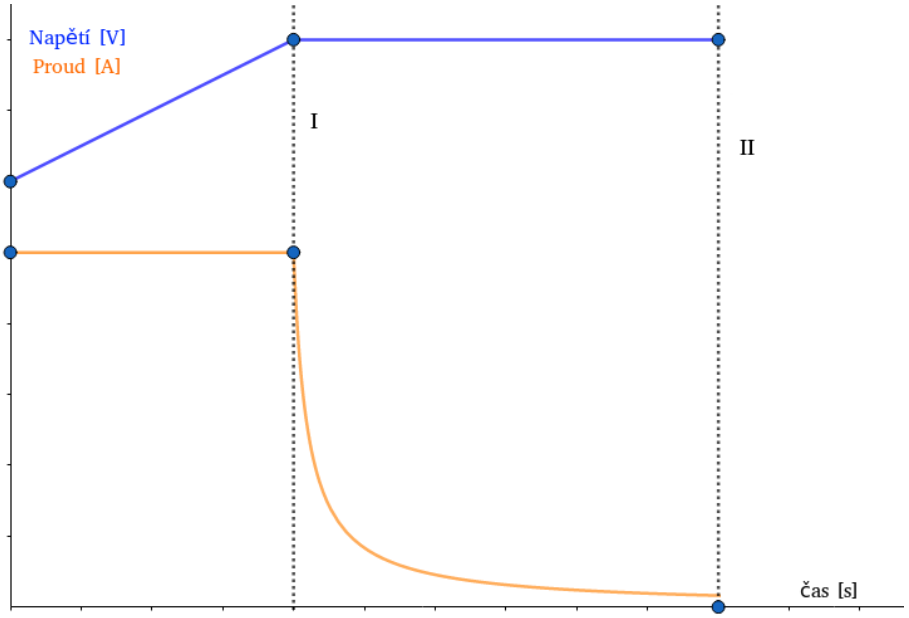
[1] V případě životnosti Li-Pol akumulátorů se údaje z různých zdrojů liší a udávají hodnoty od 40 do 1000 cyklů. Nízké hodnoty jsou pravděpodobně vztaženy ke starším generacím Li-Pol článků, v případě vysokých hodnot se pravděpodobně jedná o záměnu s Li-Ion technologií. Uváděná hodnota 300 je tedy nejčastější hodnotou, která se zmiňuje jako horní hranice životnosti těchto článků při praktickém použití - napájení RC modelů aj.

Ni-MH akumulátory jsou podobné Ni-Cd akumulátorům. Jejich pokles napětí ovšem není tak výrazný a proto je třeba kombinovat více nabíjecích technik pro zajištění bezpečnosti a spolehlivosti celého procesu.

Olověné a lithiové baterie je poté nejlepší nabíjet metodou CC/CV. Tato metoda je dvoufázová; nejprve je akumulátor nabíjen konstantním proudem při zvyšujícím se napětí, dokud toto napětí nedosáhne zvolené hodnoty (*constant current*, CC). Poté přechází nabíjecí cyklus do druhé fáze, ve které je napětí fixováno na dané hodnotě a nabíjecí proud postupně klesá (*constant voltage*, CV). Nabíjení je ukončeno, když nabíjecí proud dosáhne cca 0.01 C.

Pro olověnou baterii se doporučuje v první fázi nabíjet proudem 0.1-0.3 C/h a jako hraniční napětí 2.3-2.45 V na jeden článek. Nižší hodnoty prodlužují životnost baterie, ale prodlužují dobu nabíjení. Vyšší hodnoty zrychlují nabíjení a baterie se chová konzistentněji z pohledu kapacity, na druhou stranu je baterií vyvíjeno větší množství plynu a je třeba doplňovat vodu do elektrolytu. Zároveň se tím snižuje životnost baterie.

Li-Ion a Li-Pol baterie se zpravidla nabíjejí v první fázi proudem do 0.8 C/h a druhá fáze nastává po dosažení 4.2 V. Pokud je třeba baterii dobít rychle, zpravidla lze přistoupit k nabíjení proudem 1 C/h, některé baterie snášejí i 2 C/h, což je zpravidla bezpečné maximum.



Obrázek 3.1: Ilustrační graf nabíjení metodou CC/CV. Od počátku grafu do svislé tečkované přímky označené jako I probíhá fáze CC (z angl. *constant current*), poté do přímky II probíhá fáze CV (*constant voltage*). Následně je již proud tak malý, že je nabíjení ukončeno. Osy grafu jsou úmyslně nepopsány, vzhledem k tomu, že konkrétní hodnoty se liší podle typu baterie, požadavků na rychlost nabíjení atp.

Kapitola 4

Nabíjecí stanice pro autonomní roboty a jejich navádění k nim

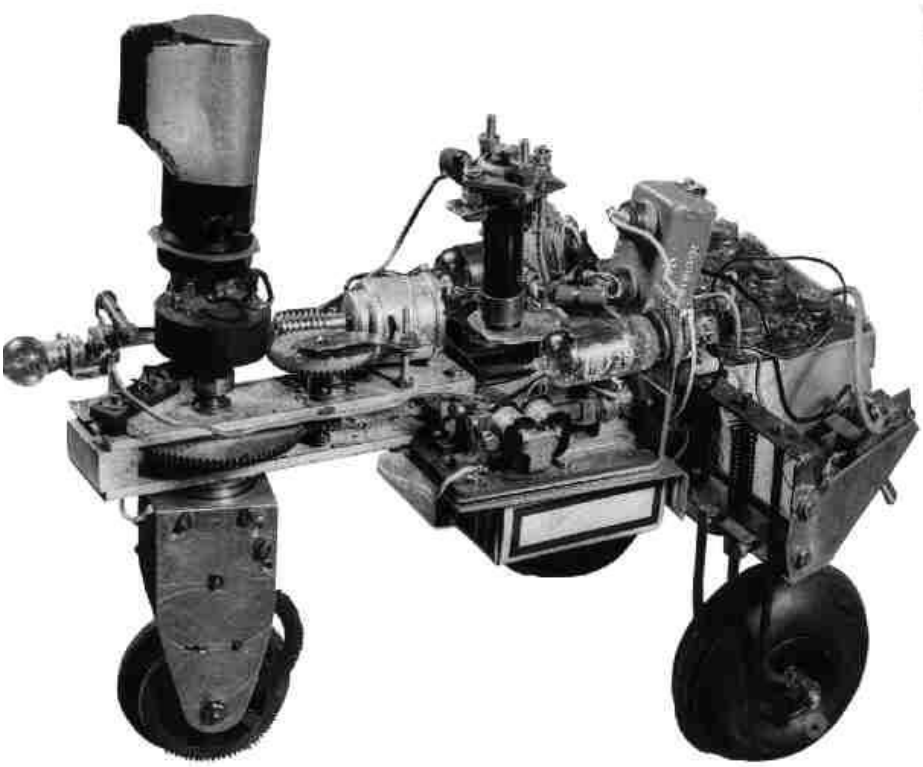
V této kapitole budou popsány konstrukce některých dokovacích stanic jak z výzkumných prací, tak komerčně používané. Budou zde také diskutovány principy navádění autonomních robotů k těmto stanicím.

Prvním robotem, který se byl schopný autonomně nabíjet, byl robot Elsie 4.1. Spolu s totožným robotem Elmerem (který ale dle [29] nebyl vybaven funkcí automatického nabíjení), byl sestaven ve 40. letech kanadským neurofyziologem a kybernetikem W. G. Walterem [31]. Překvapivé je, že cílem nebylo demonstrovat pokroky v oblasti robotiky, ale skutečnost, že bohatá spojení mezi relativně malým počtem nervových buněk mohou vyústit v poměrně komplexní chování [31] – tito roboti tedy byli řízení prvotními formami neuronových sítí, a to dříve, než byly umělé neuronové sítě vůbec formalizovány.

V současné době existuje zde několik různých přístupů tvorby dokovacích stanic. Některé stanice používají k navigaci robota aktivního vysílání [18], [9], zatímco jiné fungují pouze na principu pasivního navádění – na jejich povrchu je umístěná značka (angl. *tag*) 4.1, případně jen oblast nezvyklé barvy. Ve skladových halách a jiných prostorách, kde se roboti pohybují po předem určených trasách, je navigace k dokovací stanici řešena např. podle čar na podlaze a pro finální přiblížení a dokování slouží senzory vzdálenosti (ať už infračervené nebo např. sonary). Samotná dokovací stanice tedy nemusí disponovat žádným systémem, který by robotovi pomohl s naváděním.

Liší se i samotné způsoby dokování. Velmi častým přístupem jsou napájecí konektory ve formě pásů na zdi nebo jiném vertikálním objektu a robot přitom pouze najede k danému objektu svými styčnými nabíjecími plochami (prosté kusy kovu) [15]. V [28] je navržen poměrně komplexní design nabíjecího konektoru, který je schopen kompenzovat poměrně velké nepřesnosti v přiblížení a navigaci. V [18] je k dokování využito pružinového drapáku, který využívá vlastní pohyb robota ke svému uzavření a fixaci robota při nabíjení.

Níže budou probrány jak různé přístupy k samotnému dokování, tak různé způsoby navigace robota ke stanici. I když obě sekce budou vycházet často ze stejných prací, nebudou tyto dvě oblasti probírány kombinovaně, jelikož různé principy navigace lze použít pro různé způsoby dokování a vice versa. Práce se přitom zabývá pouze dokováním suchozemských robotů ve 2D prostoru – nebudou tedy probírány principy dokování vodních ani vzdušných robotů, i když zde jistě existuje mnoho různých zajímavých řešení.



Obrázek 4.1: Jeden z dvojice robotů W. G. Waltera, dobová fotografie.

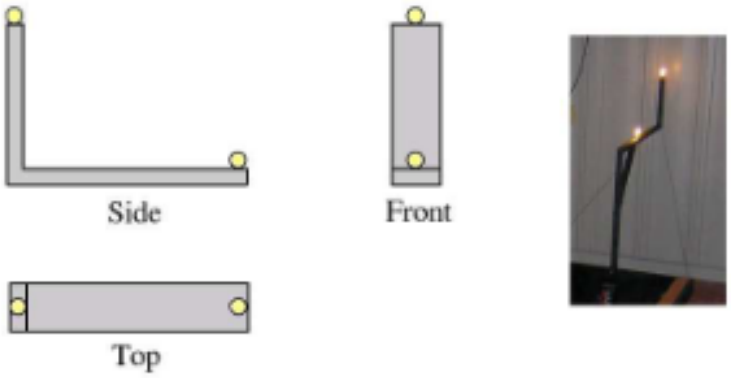
4.1 Principy navádění k dokovací stanici

Prvním výzkumníkem, který řešil problém navigace k dokovací stanici, byl již zmíněný Grey Walter [31]. Ten k navádění používal blíže nespecifikovaný způsob, kdy jeho robot Elsie byl naváděný světlem – neobsahoval žádný kód, celý řídící obvod byl modelem několika nervových buněk, které byly modelované jako analogový počítač. Vzhledem k tomu, že sám Walter toto považoval více za demonstraci možností mozkových buněk než za pokrok v oblasti robotiky, v dnešní době se jen velmi obtížně hledají detaily jeho výzkumu. Přesto se jedná o v mnoha směrem revoluční roboty.

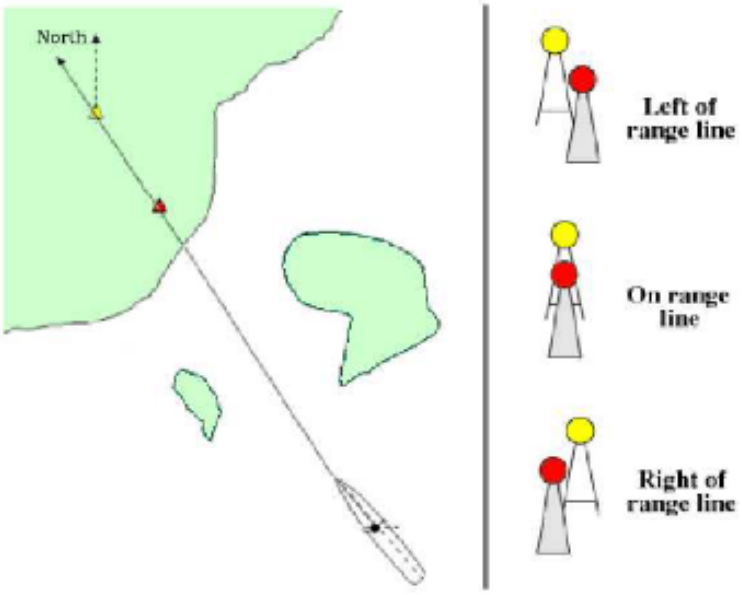
Již poněkud sofistikovanější a v praxi použitelnější metodu představili Silverman a kol. v roce 2002 [28]. Jejich robot používal programovou architekturu, kdy chování pro každý z robotových úkolů bylo zapouzdřeno do modulu (podobně, jako se nyní děje s balíčky ve frameworku ROS 7.1). Samotné navádění bylo řešeno jak pasivně pomocí velké desky oranžové barvy přímo nad dokovací stanicí, tak aktivně pomocí IR majáku. Úspěšné zadokování robot poznal podle skokové změny napětí na baterii a právě podle IR majáku a vzdálenosti od něj. Celý systém přitom počítal s relativně specifickým způsobem dokování s poměrně velkým konektorem jak na straně robota, tak na straně stanice.

Cassinis a kol.ve své práci představil navádění robota na principu dvou za sebou umístěných světel, určujících přímou linii ke stanici [9]. Stejných principů se využívalo během středověku v mořeplavectví, kdy se takto označovaly bezpečné koridory pro přiblížení do přístavu (Obrázek 4.3). Stejného principu zarovnání dvou objektů se využívá i v mechanických mířidlech u zbraní.

Myšlenka celého navádění je taková, že nad dokovací stanicí svítí v přímce za sebou dvě rozlišitelná světla[1]. Tato světla jsou přitom v různé výšce, jak zachycuje Obrázek 4.2. Robot podle jejich vzájemné polohy poté dokáže určit, zda se od stanice nachází vlevo, vpravo, či je na ideální přibližovací přímce (tedy přímo před ní). Princip je dobře ilustrován na Obrázku 4.4.



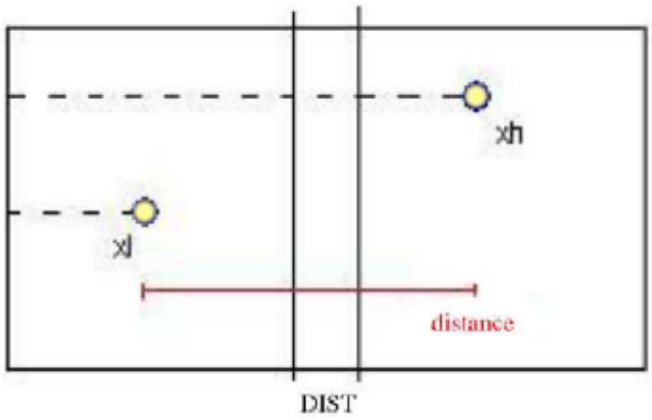
Obrázek 4.2: Pozice světel při jejich použití k navádění. Obrázek přejat z [9].



Obrázek 4.3: Princip pozičních světel na ilustraci navádění lodě mezi dvěma ostrovy. Obrázek přejat z [9].

Samotný algoritmus lze neformálně popsat takto:

[1]V práci se píše o dvou světlech, ale pokud je robot vybaven dostatečně kvalitní kamerou, lze použít jakékoli dva objekty.



Obrázek 4.4: *distance* odpovídá horizontální vzdálenosti dvou bodů `xl` a `xh`. Pokud je jejich horizontální vzdálenost menší než DIST, předpokládá se, že robot jede v ideálním kurzu. Obrázek přejat z [9].

1. Robot najde v obraze z kamery aktivní značky (světla). Zpracování obrazu je v tomto případě nesmírně jednoduché, značky/světla jsou velmi jasné halogenové žárovky, stačí tedy nastavit určitý limit pro jasnost pixelu a hledat shluky pixelů, kde je daný limit překročen. Tento limit je přitom dynamicky měněn, jelikož ve větší vzdálenosti je jas značek nižší a naopak. V ideálním případě jsou nalezeny dva shluky pixelů, které odpovídají dvěma značkám.

2. Vypočítá se těžiště obou shluků a označí se jako `xl` pro dolní značku a `xh` pro horní značku.

3. Následně se získá horizontální vzdálenost těžišť (vizte Obrázek 4.4). Pokud je hodnota *distance* kladná (bod `xl` je vlevo), pak se robot přibližuje zprava a musí zatočit doleva a opačně.

4. Po celou dobu pohybu nesmí robot ztratit ani jedno ze světel ze zorného pole.

Dalším principem, který je naznačen v [33], je použití magnetické navigační pásky, po které se robot dostane až ke stanici. Tento princip je jistě využitelný ve výrobnách, skladech a jiných průmyslových objektech, kde se již nyní používají k navádění magnetické, případně různě barevné pruhy.

Pokud se ovšem robot pohybuje po libovolné trajektorii, je využití tohoto systému složitější, ale přesto možné. Magnetická páska bude vyvedena jen do určité vzdálenosti před stanici a robot, který bude libovolně jezdit, bude moci při přejetí magnetické pásky začít sledovat její směr a tím se nakonec dostat ke stanici. Pokud by zvolil špatný směr, bylo by pouze zapotřebí se na konci magnetické pásky otočit a nyní již sledovat správný směr. Toto využití magnetické pásky je již možné i tam, kde se roboti po magnetických páskách pohybovat nemusejí, je ovšem třeba zajistit navádění k samotné pásce a tedy doplnit ho ještě jinou metodou.

Další možností aktivního navádění je princip, který se používá i v dokovacích stanicích robotických vysavačů[2] a byl popsán v [18].

Princip je dobře ilustrován na Obrázku 4.5. Dokovací stanice vysílá signál pomocí několika IR LED se směrovým vysíláním – je tedy jasné, v jakém úhlu bude možné signál přijmout. V původní práci má každá dioda vysílací směr zhruba 40 °. Jednotlivé vysílací výseče se přitom neúplně překrývají, čímž při k diodách vznikne $2k-1$ oblastí, které od sebe lze rozlišit přijímanými signály. Každá dioda přitom vysílá nějak specifický signál. Dosah diod je také omezen velmi přesným řízením vstupního proudu, aby bylo dosaženo optimálního dosahu v řádu nižších jednotek metrů (v [18] se hovoří specificky o dvou metrech).

Na přední straně robota jsou poté nainstalovány IR přijímače, které mohou zachytit signály z jednotlivých IR diod. Jedna z použitelných konfigurací vyžaduje celkem šest IR přijímačů, kde dva budou umístěny těsně (odchylka 10 stupňů) vedle sebe v přední části robota a zbylé čtyři po 45 stupních po jeho obvodu (vizte Obrázek 4.5).

Ve chvíli, kdy se robot snaží nabít, začne hledat signály vysílané stanicí a ve chvíli, kdy nějaký zaznamená, je schopen rozeznat, jaká dioda tento signál vyslala a z toho určit, v jaké výseči se daný senzor nachází. Robot se následně snaží dostat do centrální výseče. Přesný algoritmus není v práci uveden.

Další možností je použít např. LIDAR nebo jinou metodu měření vzdálenosti, poskytující dostatečné rozlišení, a hledat přímo geometrický popis dokovací stanice (zmíněno jako možné řešení v [26]). Tato metoda poskytuje poměrně dobrou přesnost, vyžaduje ovšem dostatečně přesnou metodu měření vzdálenosti a zároveň může vykazovat vysokou míru falešně pozitivních rozpoznání za předpokladu, že dokovací stanice má generickou geometrii.

Posledním způsobem je vizuální rozpoznávání a navigování pomocí tagů umístěných na dokovací stanici (podrobnější informace o tazích v 4.1). Tato metoda je použita v [15], [26] a [23]. Ke své realizaci vyžaduje tato metoda kameru s přiměřeným rozlišením vzhledem k velikosti tagu a požadované vzdálenosti detekce. Samotný algoritmus poté funguje tak, že se robot snaží v obraze z kamery identifikovat specifický tag dokovací stanice. Konkrétní algoritmus rozpoznání tagu přitom není důležitý, pokud ho robot je schopen v rozumném čase provést vlastními výpočetními prostředky, nebo má k dispozici jiný způsob, jakým může v obraze rozpoznat požadované informace (odesílání informací na server apod.).

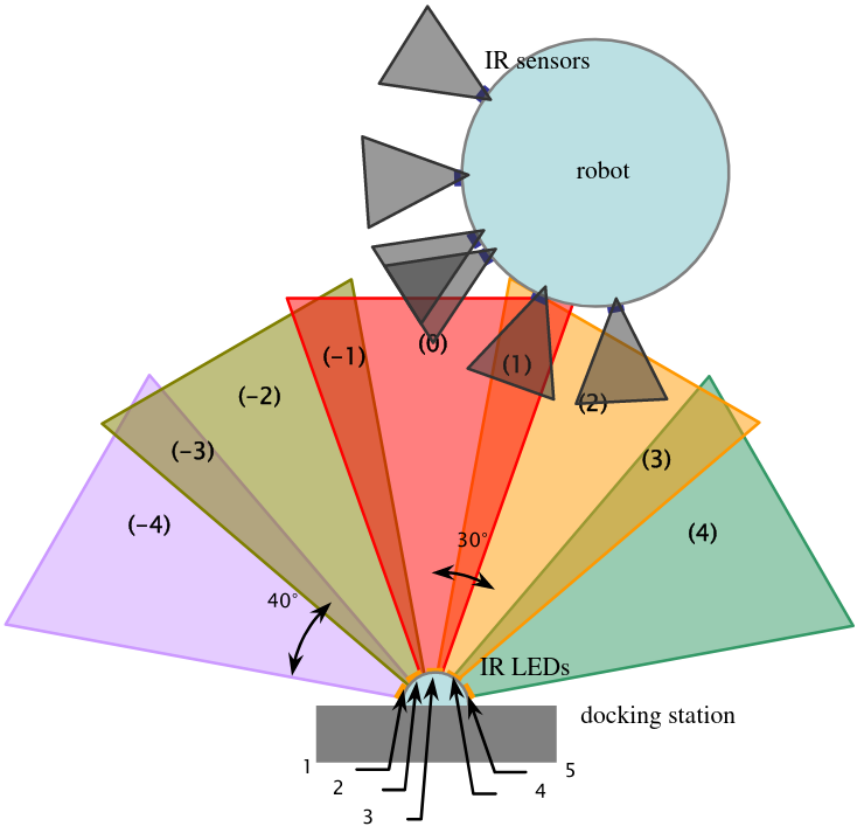
Ve chvíli, kdy daný tag identifikuje, může z pokřivení jeho obrazu v důsledku pozorování pod úhlem zjistit i svou relativní polohu vzhledem k tagu a tedy i ke stanici. Poté stačí jen zkonstruovat trasu, která robota nejprve dostane do pozice přímo před stanicí a následně dojet až k ní. Některé algoritmy jsou schopné správnou orientaci robota a jeho přibližování řešit naráz, jsou ale zpravidla náročnější na výpočet.

Tagy

Jak bylo zmíněno výše, některé práce používaly pro navigaci k dokovacím stanicím *tagy* neboli značky – přesně specifikované obrazy, které umožňovaly dokovací stanici nejen identifikovat dokovací stanici, ale často také podle deformace tagu vypočítat relativní svou polohu vůči této stanici.

Po zevrubném průzkumu problematiky se ukázalo, že tato problematika je daleko komplexnější, než se může na první pohled zdát, a proto byla vytvořena samostatná podsekce,

[2]Technologie dokovacích stanic pro vysavače jako Roomba jsou neveřejné a tato informace je tedy neoficiální, zjištěná nadšenci, kteří některé funkce podobných stanic zjistili pomocí reverzního inženýrství.



Obrázek 4.5: Ukázka principu činnosti doku s naváděním pomocí IR LED. Popis v textu. Obrázek přejat z [18]

kde budou v úvodu popsány společné znaky všech hlavních prakticky používaných tagů[3] a následně probrány hlavní koncepty spolu s typickými implementacemi.

Obecnými cíli a kritérii, která jsou důležitá při tvorbě, používání a vyhodnocování kvality tagů [14], jsou:

1. Poměr falešně pozitivních shod. Toto zahrnuje případy, kdy byl v prostředí rozpoznán tag i přesto, že se tam žádný nevyskytoval.
2. Záměna dvou existujících tagů.
3. Poměr falešně negativních shod, tedy situace, kdy nebyl rozpoznán validní tag.
4. Velikost tagu.
5. Odolnost vůči zhoršeným podmínkám (šum, osvětlení...).
6. Roztřesenost rozpoznávání, neboli jak hodně „osciluje“ rozpoznaný tag v rámci reálné scény. V případě navigace rorbotů toto není tolik podstatné, důležitějším se toto kritérium ovšem stává v aplikacích augmentované reality.

[3] V angličtině se používá výraz *fiducial marker* neboli referenční značka, v této práci ale bude nejčastěji používáno slovo *tag*.

Dále nás může v konkrétních aplikacích zajímat například rychlost a výpočetní náročnost rozpoznávání, zvláště v případě slabšího hardwaru robota. Pokud se má robot podle tagu navigovat, je také důležité, zda jde z tagu odvodit i informace o vzájemné poloze stanice (nebo jakéhokoli jiného cíle) a robota.

Obecně se dá říci, že prakticky všechny tagy používají pouze dvě barvy (prakticky vždy černou a bílou). Důvodem je, že takové barevné schéma neklade takové nároky na kamery (která může zaznamenávat scénu jen v odstínech šedé) a zároveň poskytuje největší kontrast mezi použitými barvami.

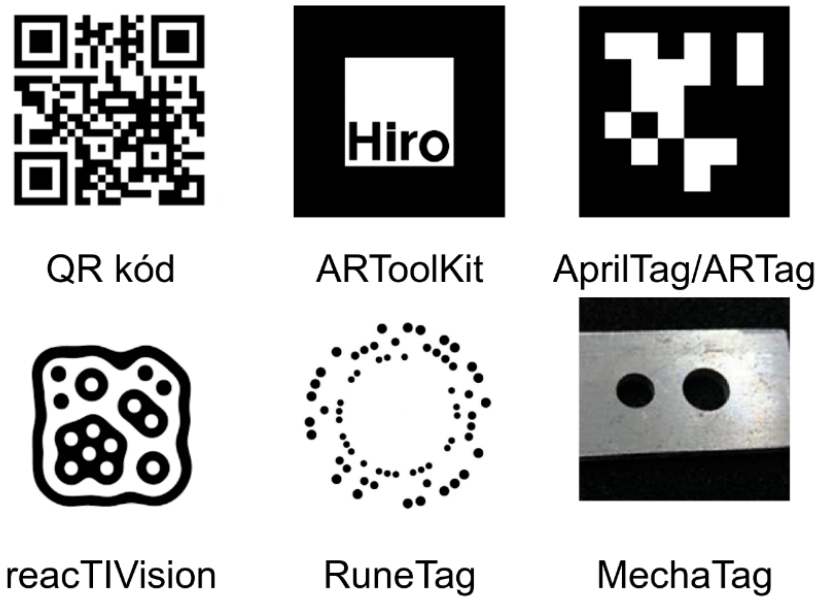
Co se týče samotného kódování informace, většina tagů používá vizuálně podobný systém jako QR kód, tedy čtvercový tag, vyplněný menšími čtverci bílé a černé barvy, kde jeden čtverec odpovídá jednomu bitu informace. Jedná se například o ARtag [14], AprilTag [22] [32] [19] a CyberCode [24]. Existují ovšem i tagy založené na kódování informací do kružnic (RuneTag [5]) nebo úplně jiné systémy, založené například na topologii (reacTIVision [2]). ARToolKit [21] poté dovoluje použít jako tag libovolný obraz v černém rámečku. MechaTag [13] je poté minimalistickou značkou v podobě dvou nad sebou umístěných kružnic různého průměru. Níže budou některé systémy stručně představeny.

Dle [14] je nejpoužívanějším tagem na poli augmentované reality pravděpodobně ARToolKit [21]. Používá čtvercové černobílé tagy s černým rámečkem a bílým středem, na kterém může být zobrazen libovolný černobílý motiv (text, pixel-art...). Při rozpoznávání se tag porovnává s databází známých vzorů a vrací se ten s nejmenší Hammingovou vzdáleností. Na jednu stranu umožňuje ARToolKit velkou variabilitu motivů díky prakticky libovolné velikosti a rozlišení, na druhou stranu vyžaduje kameru s poměrně vysokým rozlišením a velký výpočetní výkon pro rozpoznání tagu, jelikož je třeba vždy projít celou databázi známých vzorů. V robotice se přitom tolik nepoužívá.

Tyto problémy přitom odstraňuje ARTag [14], který z ARToolKitu přímo vychází. Na rozdíl od ARToolKitu se jedná o pevně danou matici 10x10 jednotek. Rámeček je široký 2 jednotky, což dává matici 6x6 pro samotná data. Existuje přitom celkem 2002 tagů s unikátní informací, kde 1001 z nich má černý rámeček, zbytek poté bílý. 36bitová informace z tagu přitom obsahuje zakódované 10bitové slovo. Pro zvýšení robustnosti celého systému je použito metody kontrolních součtů (CRC) a dopřednou korekci chyb (forward error correction, FEC).

Podobný systém jako ARTag využívá i AprilTag [22] [32] [19]. Vychází přitom právě z ARTagu, navíc ovšem garantuje minimální Hammingovu vzdálenost mezi dvěma tagy při jakémkoli otočení. Zároveň se zvýšila i přesnost lokalizace tagu v prostoru. Další důležitou výhodou byla otevřená licence (jak ARToolKit, tak ARTag nebyly publikovány pod otevřenou licencí [32]). Samotný tvar a popis AprilTagů se liší podle konkrétního typu a množství požadované informace (4-12 bitů), tyto typy jsou označované jako rodiny tagů. Nejpopulárnější je verze 36h11. Jedná se o velmi populární tagy, které jsou stále ve vývoji (poslední verze, AprilTag3, byla představena v roce 2019 [19]). V robotice jsou přitom tyto tagy jedněmi z nejvíce používaných.

Všechny výše popsané typy přitom používaly podobný vizuální záznam jako QR kódy, tedy matici černobílých bodů. Některé typy tagů ovšem používají i jiné principy. Například RuneTag používá k záznamu dat body rozprostřené po několika soustředných kružnicích. Celý princip záznamu se podobá záznamu dat na jednu plotnu pevného disku. Jsou zde jednotlivé kružnice/úrovně, sektory a bloky/sloty. V každém slotu se pak může nacházet tečka, jejíž velikost se dynamicky mění, čím blíže středu se nachází, tím je menší. Situaci znázorňuje Obrázek 4.7.



Obrázek 4.6: Porovnání několika vybraných technologií tagů. AprilTag a ARTag přitom mohou využívat stejnou grafickou reprezentaci (AprilTag označuje tuto rodinu tagů jako 36h11), informace jsou ovšem kódovány jinak. AprilTag navíc používá i jiné rodiny.

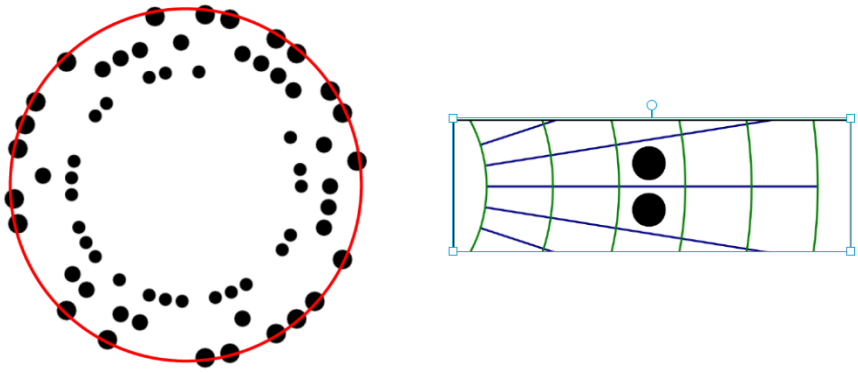
Ve chvíli, kdy algoritmus správně určí kružnici (v obraze se bude jevit jako elipsa), je možné z jejích parametrů určit vzájemnou polohu kamery (a potažmo i robota) a tagu.

Dalším zmíněným typem tagu je reacTIVision [2]. Tento tag byl původně vytvořen pro reacTable, experimentální hudební nástroj. Tento tag je přitom založen na topologickém rozpoznávání – každý tag je složen ze zanořených bílých a černých oblastí, které dohromady dávají unikátní topologii, nezávislou na natočení nebo deformaci (vizte Obrázek 4.8). Samotné tagy jsou přitom generované genetickým algoritmem, jak je popsáno v [3]. Kromě vizuální zajímavosti (což byl i jeden z cílů návrhu) a originálního přístupu má reacTIVision i tu výhodu, že je rozpoznatelný v jakékoli poloze. Na druhou stranu je tento tag nevhodný ke zjišťování vzájemné polohy, jelikož v základní podobě neobsahuje žádné pevně dané hrany.

Posledním zmíněným tagem je MechaTag. Tento tag je složen pouze ze dvou děr o průměrech 8 a 10 mm se středy vzdálenými 15 mm. Primární výhodou tohoto tagu je použitelnost v prostředí, kde není nutné přenášet žádnou informaci, ale pouze poskytnout referenční bod. Zároveň je tento typ daleko odolnější než všechny ostatní představené varianty, které jsou často pouze vytištěné na papíře a tedy náchylné ke zničení. Praktická použitelnost pro účely identifikace dokovací stanice se ovšem jeví jako nízká.

4.2 Principy dokování a připojení

V minulé sekci byly představeny základní způsoby, jak může být robot navigován k dokovací stanici. V této sekci budou ukázány postupy a způsoby, jakými může robot zadokovat u stanice ve chvíli, když je v těsné vzdálenosti. Jinými slovy, budou zde ukázány různé způsoby, kterými se může robot propojit s dokovací stanicí (nejen) za účelem nabíjení.



Obrázek 4.7: Ukázka RuneTagu. Vlevo červeně vyznačena jedna úroveň, vpravo ukázka dvou zaplněných a mnoha nezaplněných slotů. Obrázky přejaty z [5].

V současné době existuje jen velmi málo konektorů, které by byly použitelné pro autonomní roboty, kde stále velkou roli hraje nepřesnost. Z tohoto důvodu všechny odborné články zmíněné v této práci používají vlastní konektory a způsoby propojení, často nepoužitelné nikde jinde. Níže bude ukázáno několik z těchto řešení spolu s krátkým popisem.

Ve svých pracích [28], [27] vytvořili Silverman a kol. komplexní a poměrně bezpečný dokovací systém, sestávající se z konektoru dokovací stanice a konektoru robota.

Na dokovací stanici je umístěn poměrně velký konektor s trychtýřovitým naváděcím límcem o velikosti zhruba 15x18 cm (šířka x výška), Tento konektor má 2 pasivní stupně volnosti. může se pohybovat podél osy z (pohyb nahoru-dolů) a dále se může kolem osy z otáčet, čímž poskytuje zhruba 60° vstupní dokovací okno pro robota. Nákres je na Obrázku 4.9 vlevo. Na straně robota má konektor 1 stupeň volnosti, umožňuje otáčení podél osy z, nákres se nachází Obrázku 4.9 vpravo.

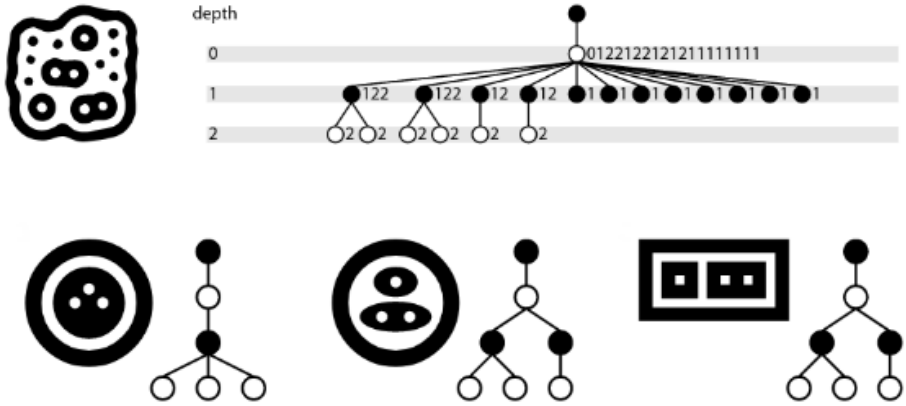
Tento konektor poskytuje velkou toleranci k nepřesnostem, které mohou vzniknout při přibližování, na druhou stranu konektor zabírá na robotovi mnoho prostoru a je poměrně složitý.

Druhý způsob je své podstatě nejjednodušší možnou konstrukcí dokovacího konektoru, kterou používají například [15], [23] nebo [9]. Konektor je přitom tvořen dvěma (případně více) kovovými pásy na jedné straně a menšími styčnými plochami na straně druhé.

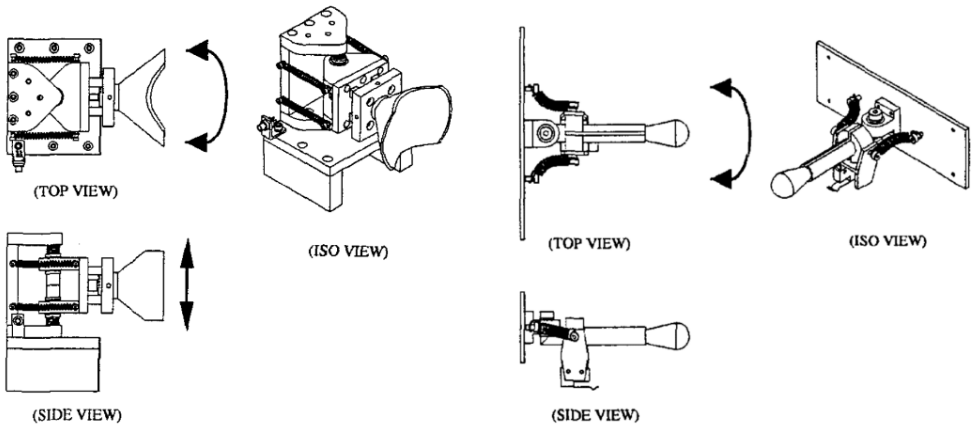
V [15] je použita asi nejjednodušší verze. Dokovací stanice je tvořena několika měděnými pásy, které jsou umístěné vodorovně nad sebou. Konektor na straně robota je poté tvořen třemi nad sebou umístěnými válci z vodivého kovu, které jsou zároveň odpruženy pro lepší dosednutí. Velmi podobného principu je využito i v [23].

Práce Cassinise a spol. popisuje poněkud sofistikovanější přístup [9]. Na straně robota se o připojení starají 4 tlusté kovové desky položené na plocho (vizte Obrázek 4.10 vpravo, horní fotografie). Na straně stanice již nejsou pouze válcovité kusy kovu, ale je v nich vysoustružena drážka pro přesnější zapadnutí desek a zlepšení kontaktu.

Jak je vidět z 4.10, všechny konektory vystavují poměrně velkou plochu konektorů pod napětím a minimálně během nabíjení jsou tedy poměrně nebezpečné. Na druhou stranu, je jednoduché je vyrobit i s použitím poměrně malého počtu nástrojů poměrně rychle a snadno.



Obrázek 4.8: Princip topologického kódování reacTIVisionu. Vpravo dole d-touch, předchůdce reacTIVisionu lišící se hranatým tvarem. Obrázky přejaty z [2] a [3].



Obrázek 4.9: Ukázka konektorů z [28]. Obrázky přejaty od tamtéž.

Obrázek 4.10: Vlevo nahoře řešení z [15], vlevo dole z [23], vpravo z [9]. Obrázky přejaty od tamtéž.

Kapitola 5

Robot Trilobot a jeho renovace

Jelikož tato práce předpokládá i vytvoření funkční dokovací stanice, bylo třeba zvolit vhodného robota, pro kterého budou obě součásti vytvořeny. Takový robot ovšem nebyl k dispozici, bylo proto rozhodnuto, že v rámci této práce se naváže na předchozí práci autora, která se zabývala mj. upgradem robota Trilobot [4]. Bylo také rozhodnuto, že nebude vytvořen pouze jeden robot, ale budou upgradováni všichni dostupní Triloboti, celkem 7 kusů, kteří budou moci být využiti i mimo tuto diplomovou práci, například při výuce.

V následující sekci (5.1) bude popsán stav, v jakém se Triloboti nacházeli před upgradem. Sekce 5.2 poté popíše realizovaný upgrade.

5.1 Historie robota Trilobot

Trilobot byl původně vyroben kolem roku 1998 firmou Arrick Robotics jakožto edukativní a výzkumný robot. Vzhledem ke svému stáří již nějakou dobu neodpovídá současným požadavkům na výuku a výzkum, ovšem vzhledem k jeho kvalitnímu šasi, rozumné velikosti a stále hojně používanému diferenciálnímu řízení se během let objevilo několik pokusů o jeho upgrade a případné opětovné využití. Žádný z těchto projektů ovšem nikdy nebyl realizován na více než jednom robotovi.

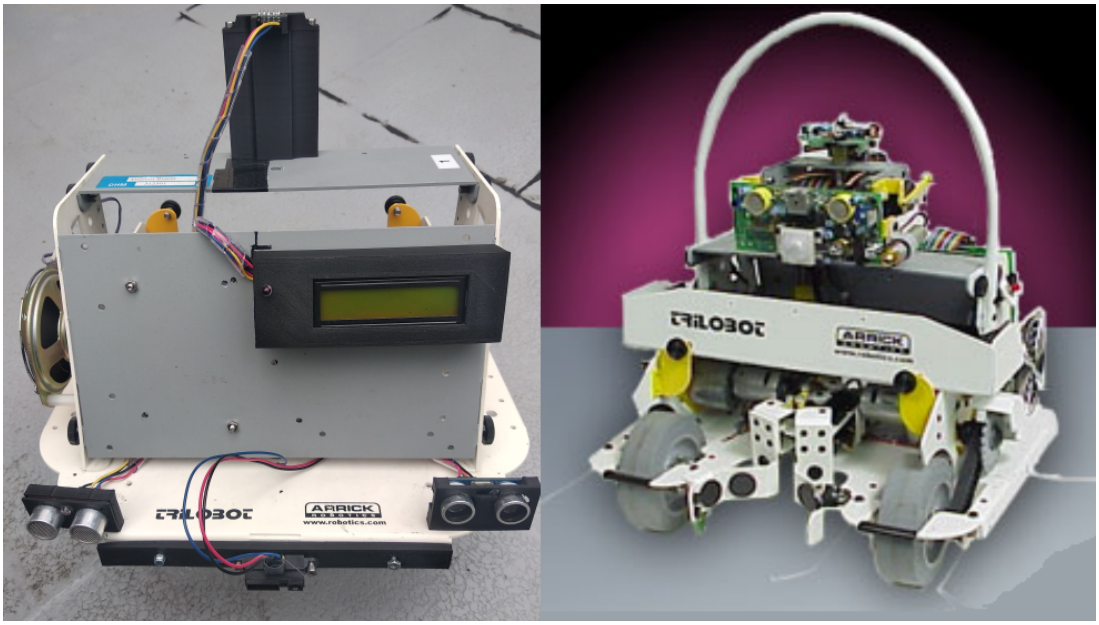
Vzhledem k autorovým zkušenostem s Triloboty, kdy jednoho z nich vylepšil s cílem demonstrovat na něm výstupy své bakalářské práce [4], bylo rozhodnuto, že mimo samotné zadání této práce bude navíc vytvořeno 7 renovovaných Trilobotů, na kterých bude možné mj. demonstrovat i výsledky této práce (nabíjecí systém, dokovací stanice). Upgrade byl přitom koncipován tak, aby výslední roboti byly použitelní i ve střednědobém časovém horizontu.

5.2 Upgrade hardwaru

V této sekci bude popsán upgrade hardwaru robota Trilobot. V úvodu sekce bude vysvětlena celková architektura nového Trilobota. Poté budou detailněji popsány jednotlivé komponenty.

Původní hardware Trilobotů byl již zastaralý a pro použití v dnešní době naprosto nevhodný. Tento hardware byl tedy kompletně nahrazen a z původních Trilobotů bylo použito pouze kovové šasi a hnaná kola.

Hlavní řídící jednotkou je Raspberry Pi 4 ve 4GB verzi spolu s dotykovým displejem připojeným pomocí rozhraní DSI, nahrazující původní kovový kryt na zadní straně Trilobota



Obrázek 5.1: Ukázka Trilobotů. Vpravo původní Trilobot z roku 1998, vlevo vylepšená verze, která byla vytvořena pro účely [4].

(vizte Obrázek 5.1 vlevo, kde je tento kryt natočen směrem na kameru). Pro nízkoúrovňové úkoly, například komunikaci po sběrnici I^2C nebo přes sériovou linku, bylo Raspberry Pi doplněno deskou Arduino Mega 2560 a řadičem Sabertooth 2x5 pro ovládání motorů. Stejně jako v autorově dřívějším upgradu [4], Arduino obstarává řízení motorů a přeposílání jimi produkované zpětné vazby, sběr dat ze senzorů vzdálenosti SRF-08 a údajů z IMU. Raspberry Pi je poté určeno na vysokoúrovňové řízení, plánování cesty atp.

K Raspberry Pi je poté kromě displeje připojena i multifunkční stereoskopická kamera Intel Realsense D435i. Tato kamera umí snímat okolí ve viditelném i IR spektru a dále obsahuje i vlastní IMU. Kromě toho disponuje i možností snímat hloubkovou mapu okolí, tzv. „pointcloud“.

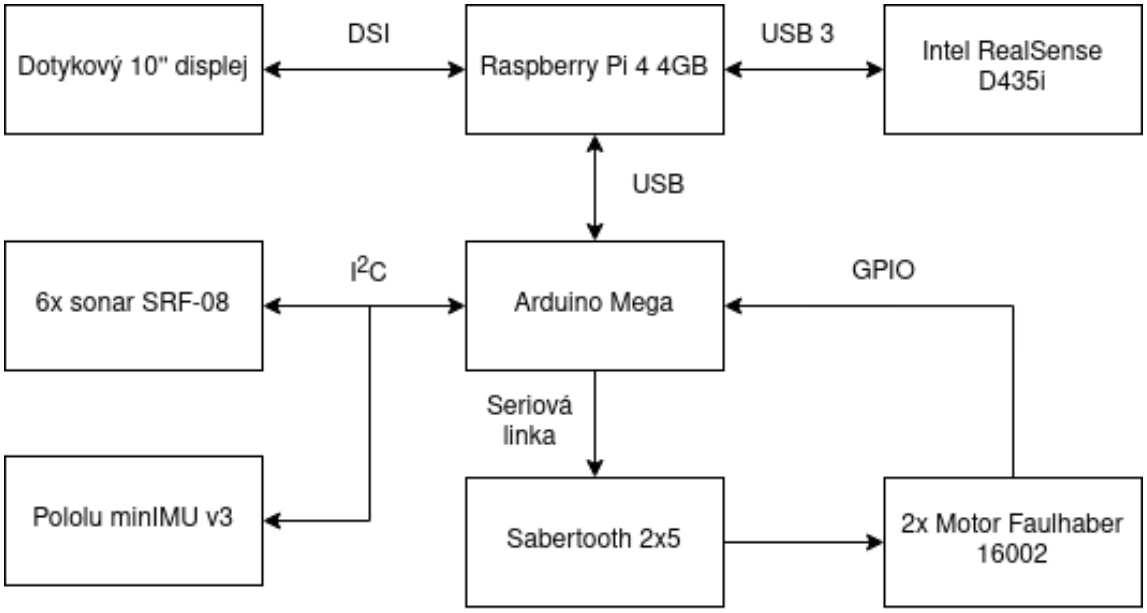
Tento dvojdeskový přístup byl zvolen kvůli snazší implementaci nízkoúrovňové komunikace na platformě Arduino, možnosti znovupoužít kód z předchozího upgradu a zjednodušení činnosti hlavní desky.

Napájení robota je možné realizovat několika způsoby:

- Pomocí USB-C konektoru na desce Raspberry Pi, která poté napájí i Arduino a ostatní senzory. V tomto módu přitom nefungují motory.
- Pomocí 12 V síťového adaptéru.
- Pomocí vestavěné 14.8V čtyřčlánkové Li-Pol baterie.

Původní motory byly vyměněny za motory Faulhaber 16002 s rotačními dvoukanálovými enkodéry, které poskytují zpětnou vazbu o pohybu. Kvůli omezenému počtu pinů pro HW přerušení na desce Arduino Mega byl použit pouze jeden kanál každého enkodéru. Druhý je sice fyzicky zapojen, ale neumožňuje generovat přerušení a proto není v rámci této práce využit. Při řízení motorů se vycházelo z [4].

O softwarové řízení se stará hlavní deska Raspberry Pi, na které běží standardní linuxová distribuce Ubuntu Mate spolu s frameworkem ROS – Robot Operating System 7.1. Důvody



Obrázek 5.2: Základní propojení komponent upgradovaného robota Trilobot včetně rozhraní.

pro použití frameworku ROS jsou poté zjevné, jedná se asi o nejrozšířenější platformu pro tvorbu robotických systémů na světě.

Komunikace mezi deskami Raspberry Pi a Arduino Mega probíhá pomocí USB, na SW úrovni poté pomocí frameworku ROS a jeho balíčku `rosserial`. Raspberry Pi dále komunikuje s dotykovým displejem, připojeným speciálním plochým kabelem a kamerou Intel RealSense, která je připojena pomocí USB-3 kvůli vysokému datovému toku.

Deska Arduino Mega poté komunikuje se šesti sonary SRF-08 a inerciální jednotkou minIMUv3 přes sběrnici I^2C a pomocí jednosměrné sériové linky i s deskou Sabertooth 2x5. Tato deska přitom již přímo řídí motory.

Pro získávání informací o okolí je robot vybaven šesti sonary SRF-08, uspořádanými do nepravidelného šestiúhelníku po obvodu robota v úhlech zhruba 60 stupňů. Robot je také vybaven RGB-D kamerou Intel RealSense D435i. Schematicky je zapojení komponent znázorněno na Obrázku 5.2 včetně použitého rozhraní na jejich propojení.

Níže budou detailněji popsány jednotlivé komponenty.

Popis použitých komponent

V této sekci budou postupně detailněji popsané jednotlivé komponenty, které byly použity při upgradu.

Raspberry Pi 4 model B je čtvrtou generací celosvětově známého jednodeskového počítače. Obsahuje 4 jádra ARM Cortex-A72 na frekvenci 1.5 GHz. Použitá verze disponuje 4 GB RAM. Jako displej a zároveň vstupní periferie pro ovládání je zvolen dotykový displej přímo určený pro Raspberry Pi o úhlopříčce 10 palců. Jako operační systém bylo zvoleno Ubuntu Maté. Důvodem je přímá podpora frameworku ROS na systému Ubuntu. Důvodem pro zvolení grafického prostředí Maté jsou jeho menší požadavky na hardware v porovnání se standardním grafickým prostředím GNOME.

Arduino Mega 2560 je mikrokontrolerová prototypovací deska založená na MCU ATmega2560. Disponuje 54 digitálními I/O piny, z nichž 15 umožňuje PWM výstup. Dále je k dispozici 16 analogových vstupních pinů, 4 UART rozhraní, I^2C rozhraní aj. K počítači (v tomto případě k Raspberry Pi) se připojuje pomocí USB kabelu. Napájí se buď přímo přes USB, nebo pomocí souosého konektoru. Napájení je možno realizovat i pomocí dvojice pinů `Vin` a `GND` (vstupní napětí v rozsahu 7-12 V) nebo pinů `5V` a `GND` (regulovaných 5V).

Sabertooth 2x5 je řídící deskou pro diferenciálně řízené roboty. Deska umožňuje zpracovat až 3000 rychlostních příkazů za sekundu a každému z motorů je schopna nárazově dodat až 10 A. Doporučeným dlouhodobým maximálním odběrem je poté 5 A na motor. Díky přepínačům lze dosáhnout mnoha různých módů komunikace, v této práce je využit mód nazvaný „simplified serial“, což ve zkratce znamená, že nadřazená deska (v tomto případě Arduino Mega) odesílá po seriové lince jeden byte s informací o chodu obou připojených motorů. Napájení je řešeno přímo z baterie/síťového adaptéru.

SRF-08 je ultrazvukovým senzorem komunikujícím přes rozhraní I^2C. Maximální teoretická vzdálenost, ve které by měl být schopen odhalit překážku, je zhruba 11 metrů[1].

Modul minIMU-v3 na Trilobotovi zůstal z předchozího upgradu [4]. Kombinuje v sobě gyroskop, akcelerometr a magnetometr a komunikuje prostřednictvím sběrnice I^2C.

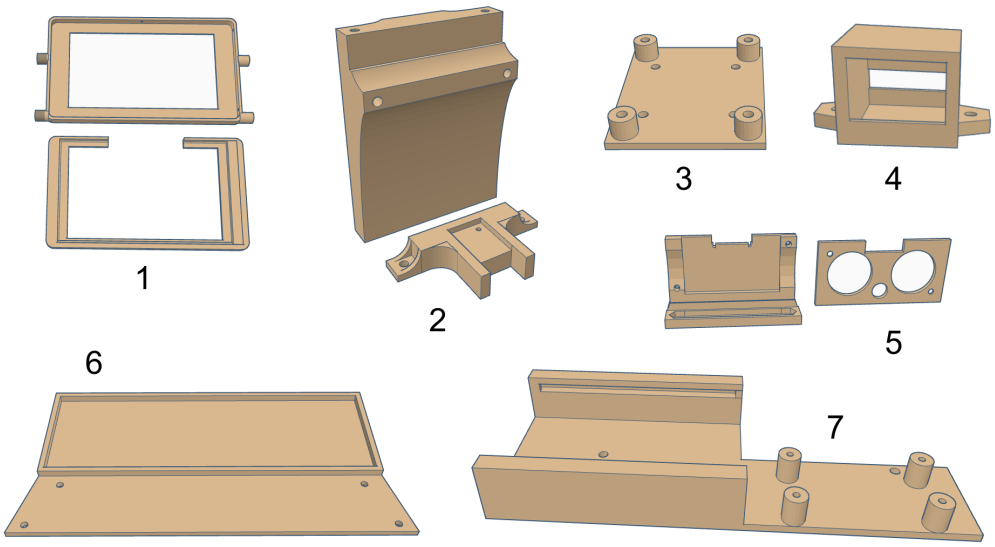
Obvod 4S 40A slouží jako ochrana proti podbití baterie a bude také sloužit jako nabíjecí obvod v navrhované nabíječce 6. Výstupní napětí (cca 12-16 V podle nabití baterie) je poté měněno step-down měničem na stabilních 5V pro napájení Arduina, Raspberry a senzorů. Motory a deska Sabertooth jsou přitom napájeny přímo neregulovaným napětím z baterie.

Kromě samotných komponent bylo navrženo a vytištěno na 3D tiskárně také velké množství adaptérů a dílů. Jejich seznam s popisem je uveden níže. Miniatury komponent jsou uvedeny v Obrázku 5.3 (poměrové velikosti nesouhlasí), obrázky modelů v plné velikosti poté v příloze A.

- Adaptér pro dotykový displej, který nahradil původní kovový zadní kryt, který byl přesunut doprostřed robota jako platforma pro další součástky.
- Adaptér desky Sabertooth 2x5.
- Vyvýšený adaptér pro kameru a minIMU.
- Adaptér pro nabíjecí obvod 4S 40A a step-down konvertor napětí.
- Kryt zapínacího tlačítka.
- Adaptér pro ultrazvukový senzor SRF-08.
- Rozšiřující platforma pro baterii. Na místě, kde je na Trilobotovi nyní umístěna baterie, se původně nacházel drapák, který ovšem byl z většiny kusů dávno odstraněn. Vzniklý prázdný prostor byl proto překlenut a využit jako platforma pro baterii.
- Adaptér pro nepájivé pole. Postranní sloty jsou určeny pro nestandardní dvouřadá nepájivá pole (vytvořena na míru, využití pro napájení a I^2C sběrnici), prostřední slot je pro standardní malé nepájivé pole o rozměrech 10x17 pinů.

Vzhledem k enormnímu množství práce a času, které by byly vyžadovány pro kompletaci všech 7 robotů, byla část práce delegována jako trojčlenný projekt v rámci předmětu ROBa. Cílem tohoto nově vzniklého projektu bylo seznámit se s konceptem nového Trilobota na již

[1] Dáno maximálním časem měření, což je 60 ms.



Obrázek 5.3: Miniatury jednotlivých modelů komponent, které byly následně vytištěny na 3D tiskárně. 1: adaptér pro dotykový displej (přední a zadní část), 2: adaptér pro Intel RealSense a minIMU, 3: adaptér pro Sabertooth 2x5, 4: kryt zapínacího tlačítka, 5: adaptér pro sonary SRF-08, 6: Platforma pro baterii, 7: platforma pro napájecí obvody.

existujících exemplářích a poté pod vedením autora této práce dokončit zbývající jednotky. Toto dokončení se sestávalo z upevňování 3D tištěných dílů a senzorů na jejich místa, zapojování kabeláže atp. V rámci tohoto projektu bylo vytvořeno 5 robotů, z nichž 4 byli kompletní. Komponenty na poslední kus nebyly k dispozici z důvodu zapůjčení na realizací jiných projektů. První dva prototypy autor sestavoval sám.

Realizace upgradu prvních kusů probíhala od června do prosince 2021. V první fázi byly upgradováni 2 Triloboti. V této fázi docházelo k častým změnám uspořádání a polohy jednotlivých komponent. Výslednou podobu Trilobota můžete vidět na fotografiích 5.4, 5.5.

Součástí upgradu bylo vytvořit i kód pro desku Arduino Mega, který by vytvářel rozumné rozhraní pro získávání dat ze senzorů a řízení motorů na straně jedné a zasílání dat do Raspberry Pi na straně druhé. Kód vychází z autorovy předchozí práce [4]. Detailní popis kódu lze nalézt v samostatné sekci 7.3.

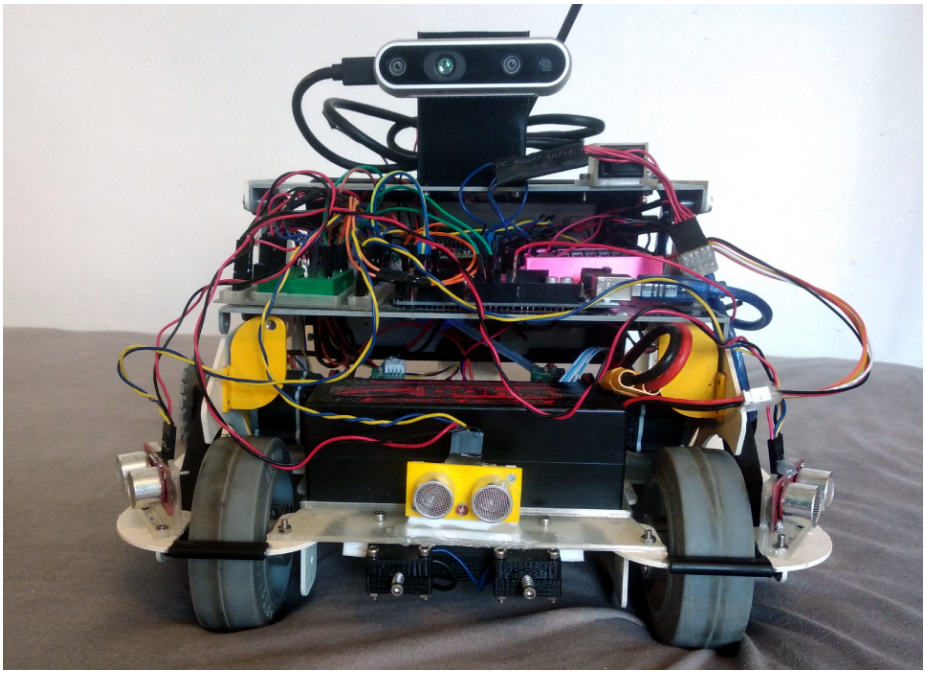
Níže je možné vidět fotografie Trilobota po upgradu (Obrázky 5.4 5.5).

5.3 Upgrade softwaru

Trilobot bude programován převážně na úrovni Raspberry Pi, kód pro Arduino Mega by nemělo být nutné při běžných robotických úlohách měnit, pokud se k robotovi nebudou připojovat další senzory.

Jako programovací framework pro Trilobota byl zvolen ROS – Robotic Operating System[2]. Tento framework byl zvolen, jelikož se jedná o celosvětově nejpoužívanjěší framework

[2]I přes slovní spojení *Operating System* – operační systém – se o operační systém nejedná a potřebuje skutečný operační systém, na kterém může fungovat, jedná se tedy o jakýsi middleware či framework. V případě Trilobota se jedná o Ubuntu Maté.



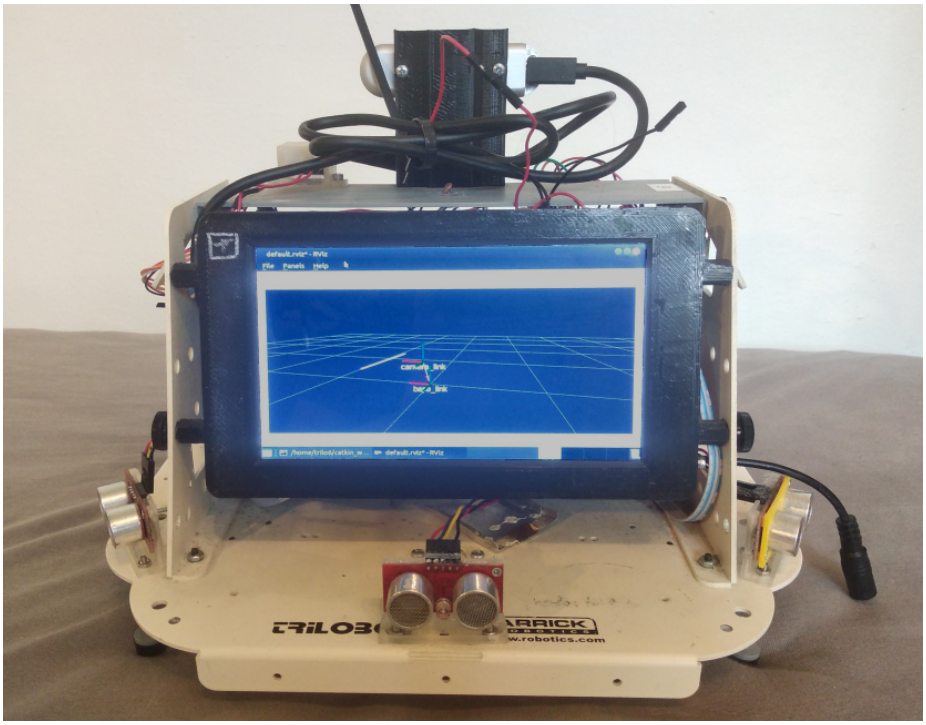
Obrázek 5.4: Pohled na upgradovaného Trilobota zepředu. Deska, na které se nachází Arduino Mega a nepájivé pole, původně sloužila jako zadní kryt.

pro vývoj robotů. Další nespornou výhodou je existence mnoha již hotových balíčků (vizte 7.1) pro mnoho různých senzorů a aplikací, vývoj tedy může být mnohem rychlejší.

Architektura ROSu je založena na rozdělení kódu do malých navzájem komunikujících programů podobně, jako velí filozofie vytváření terminálových utilit v Linuxu, kdy každá z nich je úzce specializovaná na konkrétní činnost a jejich kombinací a vzájemným provzáním lze dosáhnout složitějších úkonů. V ekosystému ROS se těmto malým programům říká `nodes`, uzly. Jednotlivé uzly spolu komunikují prostřednictvím zpráv, které mají danou strukturu, a tzv. *topics*/témat.

Více uzlů, společně s požadovanými zprávami a dalšími náležitostmi, může být sloučeno do balíčku, který se stará o určitou funkcionalitu. Může například ovládat kameru, kde jeden uzel se bude starat o poskytování telemetrie a metadat, další o samotný obraz atp.

ROS přitom pomocí `rosserial` může komunikovat i s deskou Arduino Mega, která je použita na Trilobotovi. Není tedy nutné vytvářet vlastní komunikační protokol mezi Arduinem a Raspberry/ROSem.



Obrázek 5.5: Pohled na Trilobota zezadu. Zde se původně nacházel kovový zadní plát, nyní se zde nachází dotykový displej.

Kapitola 6

Dokovací stanice a nabíjecí systém

V této kapitole bude popsán návrh dokovací stanice a nabíjecího systému, který bude následně instalován na Trilobota. V první sekci bude popsán návrh nabíječky a představen její 3D model určený pro tisk na 3D tiskárně. V další sekci se následně proberou detaily nabíjení a kontroly stavu baterie s pomocí obvodu 4S 40A a zpracování dat o baterii pomocí desky Arduino.

6.1 Dokovací stanice

Po prostudování dostupných zdrojů bylo rozhodnuto, že návrh dokovací stanice nebude založen na žádném z existujících řešení prezentovaných výše, ale bude navržen a vytvořen od začátku. Na úvod byly vytyčeny body, které by výsledný návrh dokovací stanice měl splňovat. Tyto body ve stručnosti popisují základní požadavky na výrobu, funkcionalitu a způsob navádění robota ke stanici.

Následně byla provedena analýza požadavků na návrh stanice, ze které vyplynuly tyto body a omezení, kterými by se měl návrh řídit, spolu s jejich odůvodněním:

- Stanice by měla být tisknutelná (byť třeba po částech) na 3D tiskárně, z důvodu snadné tvorby prototypů.
- Celý systém (tedy dokovací stanice a konektor na straně robota) musí dohromady akceptovat určité nepřesnosti ve vzájemné poloze a i přes to umožnit nabíjení.
- Stanice by neměla mít samovolně pohyblivé části kvůli zjednodušení návrhu i údržby. Jinými slovy, neměla by obsahovat mechanické zajištění konektoru.
- Stanice by měla být naprosto pasivní, neměla by tedy obsahovat žádnou naváděcí elektroniku, opět kvůli jednoduchosti návrhu a provozu. Jelikož se počítá s tím, že nabíjecí obvod bude součástí robota (vizte 6.2), není důvod mít na straně stanice žádnou podpůrnou elektroniku. Absence aktivního navádění také umožní její využití jiným, fyzicky kompatibilním robotem bez nutnosti složité změny naváděcího softwaru.
- Stanice nebude aktivně poskytovat robotovi údaje o své poloze. Bude k ní ovšem připevněn tag, který umožní robotovi navigovat se ke stanici pomocí obrazu z kamery. Volba tagu bude popsána dále.

- Konektory budou mít přiměřenou velikost na straně robota i stanice. Nebude se tedy jednat o např. vodivé pásy jako v případě [15].
- Stanice bude obsahovat nepohyblivé mechanické prvky, které usnadní robotovi navádění ke stanici v poslední fázi přiblížení. Příkladem mohou být naváděcí rampy/ližiny apod.

Nejdůležitějšími aspekty z výše uvedených jsou přitom poslední dva. Mechanické navádění robota je důležité mj. z toho důvodu, že kamera je na robotovi umístěna poměrně vysoko a tedy není možné vizuálně kontrolovat vzájemnou polohu konektorů. Zároveň mechanické prvky usnadní samotné navádění, které nebude muset být tolik přesné. Reálná implementace bude vycházet z dokovacích stanic pro robotické vysavače, které navádějí tyto vysavače pomocí drážek pro jejich kola.

Druhým aspektem je vytvoření samotných konektorů, které nebyly přímo specifikovány v kritériích pro návrh. Jak již bylo řečeno, žádné dostupné konektory nenabízejí dostatečnou toleranci v zapojování, jelikož jsou většinou přizpůsobeny pro lidského uživatele. Náhodou byl ovšem objeven způsob nabíjení, který se po prvotním průzkumu ukázal jako slibný. Jedná o magnetické konektory, kdy se na libovolný kus magnetického kovu připevní dostatečně silný neodymový magnet. Pokud se dodrží opačná polarita, vznikne dvojice magnetů, které přes kov dobře vedou elektrický proud i poměrně vysokých hodnot (nabíjení by mohlo využívat až jednotky ampér) a zároveň jde snadno spojit díky magnetům.

Prototyp a první testování

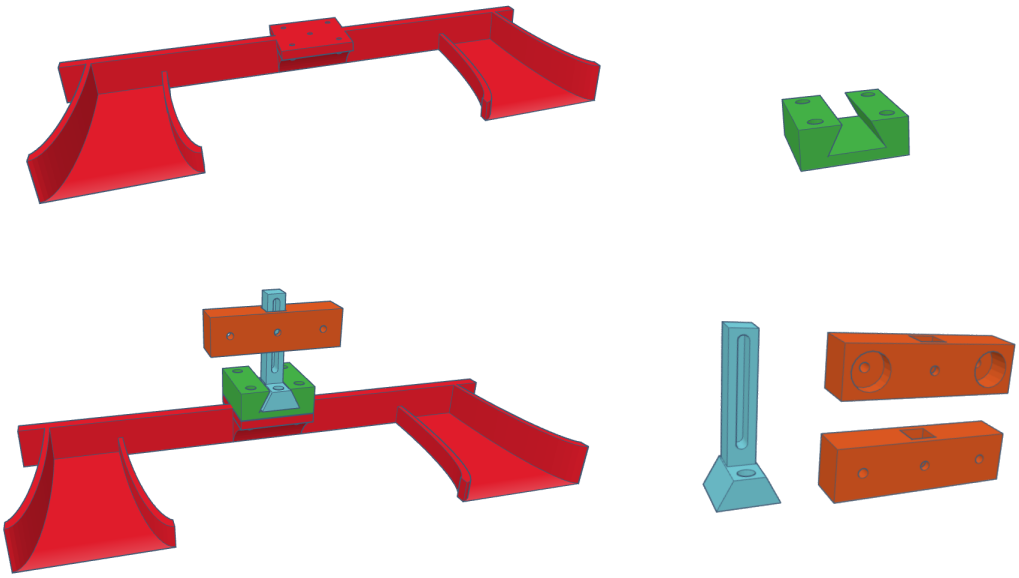
V začátku práce byl vytvořen prvotní prototyp na Obrázku 6.1. Mechanické navádění bylo řešeno rozšířenými nájezdy pro kola Trilobota, které se směrem k nabíječce zužovaly až na šířku kola s drobnou rezervou v řádu milimetrů. Pohyb v osách x a z byl řešen v dokovací stanici, kdy je možné nabíjecí hlavu posouvat dopředu a dozadu, případně nahoru a dolů.

Uprostřed základny stanice se nacházela plocha pro přišroubování adaptéru nabíjecí aparatury (na obrázku zeleně) pomocí čtyř šroubů M3. Předpokládalo se, že design samotné nabíjecí části se bude průběžně měnit, proto byla navržena modulárně. Do adaptéru se nasouvala část, která umožňovala pohyb po osách x i z (na obrázku modře), v dané poloze se fixovala utažením šroubu. Poslední částí (na obrázku oranžově) je část obsahující samotné konektory.

Stanice neměla žádný prostor pro umístění tagu, neměla zapojený zdroj energie a sloužila tedy pouze pro ověření konceptu. Májezdy se při pilotních testech ukázaly jako funkční a praktické řešení pro finální navádění. Nabíjecí hlava ovšem vykazovala nedostatečné zajištění v určené poloze, kdy docházelo k jejímu kývání. Tato hlava tedy byla v další iteraci přepracována.

Následně došlo k vytvoření ještě několika verzí této stanice, konečný model je poté možné vidět na Obrázku 6.2. Oproti původnímu prototypu došlo k řadě vylepšení. Původní nájezdy byly zhruba o 50 % prodlouženy a rozšířeny, což umožňuje ještě větší kompenzaci chyb navigace robota. Zároveň ovšem větší nájezdy zajišťují zvýšenou stabilitu stanice. Nájezdy byly také odděleny od samotného jádra stanice, což umožňuje jejich snadnou výměnu (například v situaci, kdy by stanici měl využívat robot s jiným rozvorem kol).

Samotné jádro dokovací stanice bylo zesíleno, centrální platformu pro přišroubování nabíjecí hlavy nahradily dva montážní otvory, do kterých se zasunují vodicí dráhy. Na tyto dráhy je následně připevňována samotná nabíjecí hlava. Na rozdíl od původního prototypu



Obrázek 6.1: Model návrhu dokovací stanice. Červeně základna stanice, zeleně adaptér s kolejnicí, modře adaptér pro pohyby v osách x a z, oranžově 3D tištěná část konektoru ze zadní (nahoře) i přední strany (dole). Vlevo dole poté model sestavené stanice bez šroubů a samotných magnetických konektorů.

umožňuje nyní pouze pohyb po vertikální ose. Nad ní je poté umístěn rámeček s tagem, který také umožňuje změnu výšky.

Za tag použitý pro rozpoznávání stanice byl zvolen AprilTag. Tento tag je v robotice hojně používaný a tedy otestovaný, plní účel rozpoznání stanice a její polohy vůči robotovi a existuje pro něj podpůrný balík ve frameworku ROS, není tedy nutné ručně implementovat rozpoznávací algoritmus, případně ho portovat do frameworku.

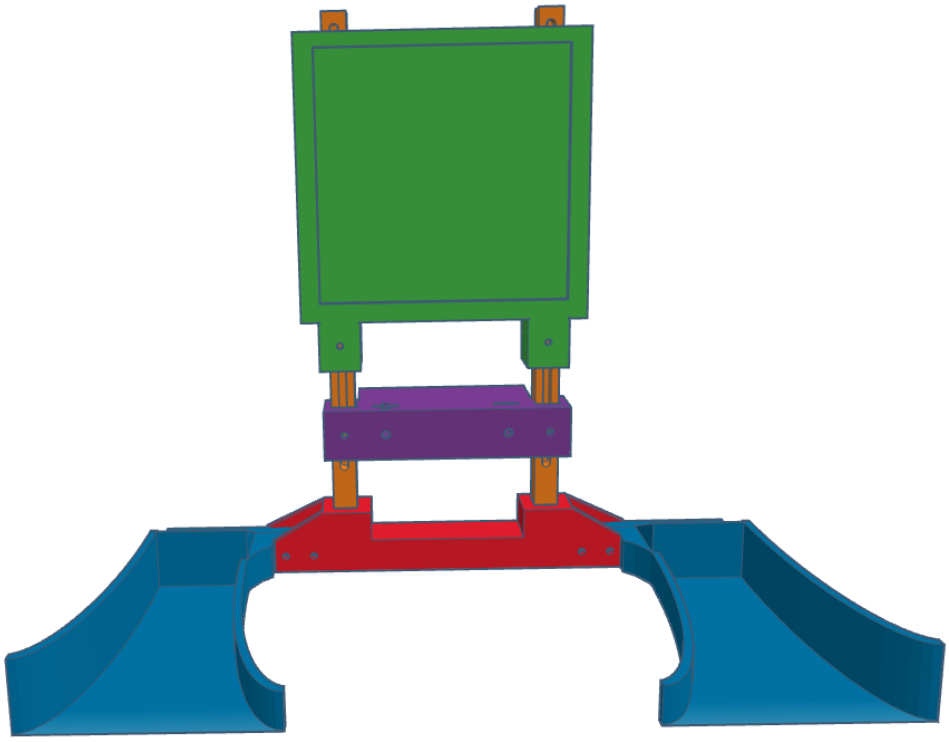
Celkově došlo k rozdělení stanice na menší části, které lze podle potřeb nezávisle měnit, čímž vznikla modulární nabíjecí platforma, která je snadno modifikovatelná i pro případné jiné roboty. Celkový design také obsahuje méně pohyblivých částí a v praktických zkouškách působí robustněji. Nejslabším místem se staly vertikální vodicí dráhy, díky modulárnímu designu ovšem není problémem dráhy v případě poškození vyměnit nebo je nahradit např. za kovové tyče odpovídajícího průměru. Otvory v nabíjecí hlavě i rámečku jsou s touto alternativou kompatibilní.

6.2 Nabíjení na straně robota

Tato sekce se bude zabývat návrhem konektorového modulu, v druhé části poté nabíjecím systémem a systémem rozvodu energie.

Nabíjecí konektory

V případě nabíjecího konektoru na straně robota byl opět vytvořen prvotní návrh. V něm se jako velmi slibný jevil princip magnetického spojování konektorů, kdy měly být ke styčným plochám připevněny magnety. Tento způsob měl zajišťovat, že po přiblížení budou mít



Obrázek 6.2: Model konečné podoby dokovací stanice. Červeně základna stanice, ze které byly odděleny samotné nájezdy (modře). Tyto byly rozšířeny a prodlouženy. Místo původní jedné vodicí dráhy byly použity dvě (oranžově), na kterých je kromě samotné nabíjecí hlavy (fialově) uchycen i rámeček s AprilTagem (zeleně).

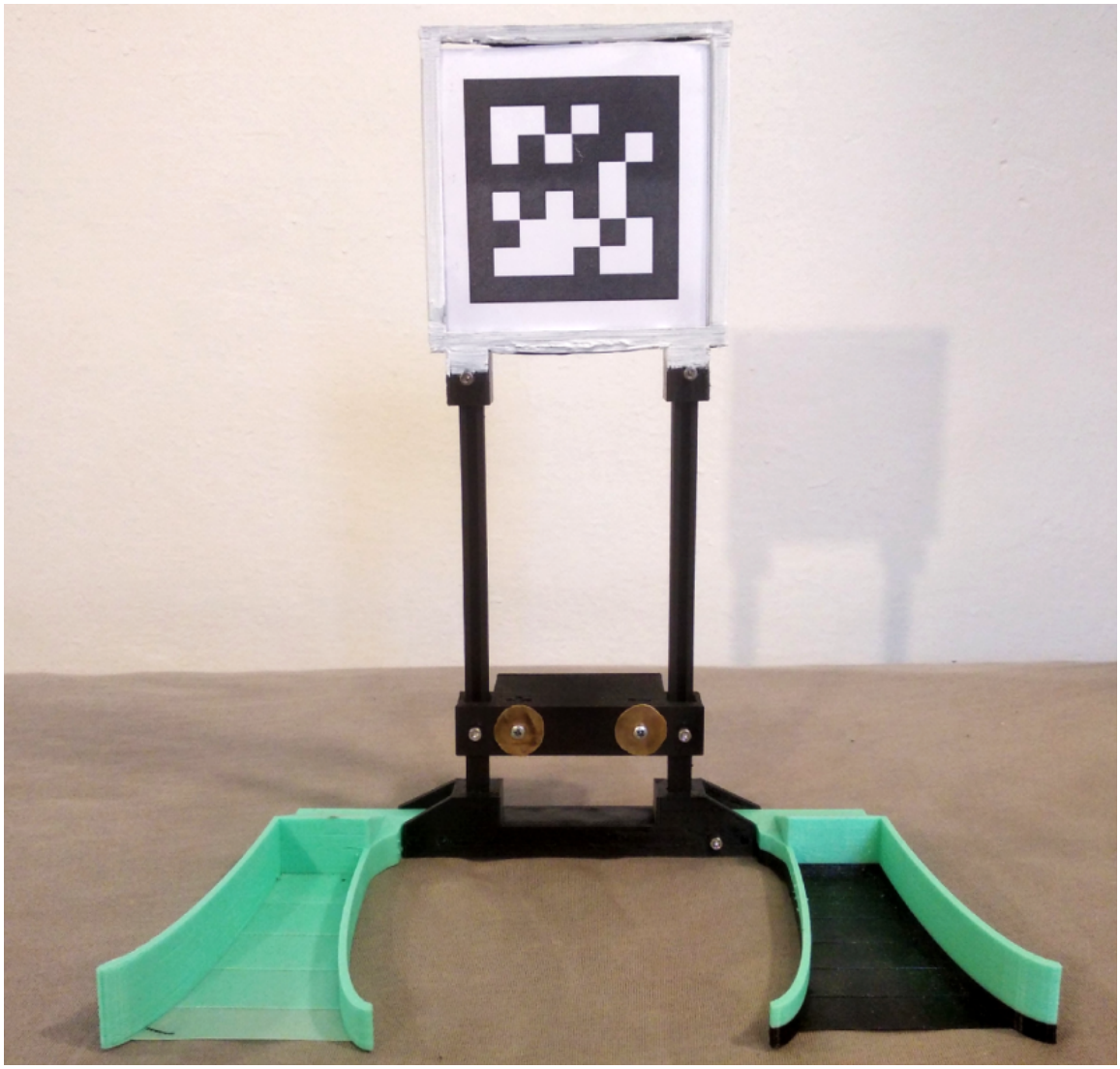
konektory samy snahu se spojit. Díky použité kombinaci magnetů mělo také být obtížné obrátit polaritu díky odpuzování souhlasných pólů.

Na Obrázku 6.4 je vidět prvotní prototyp uchycení konektorů na straně robota. Pohyb celého uchycení je možný v ose y (do stran) pro případnou jednorázovou úpravu vzájemné polohy.

Během pokračujících prací se použití magnetů jako prostředku k pevnějšímu spojení a zábrany proti přepólování ukázalo jako zbytečné. Samotné magnety nebyly schopné zajistit dostatečně kvalitní spojení a tento nápad se tedy ukázal jako nepraktický. V nové verzi byly magnety zachovány, ale slouží primárně jako hlava pístu v šachtě (vizte dále).

Vzhledem k úpravám dokovací stanice bylo třeba upravit i konektorový modul na straně robota. Upravený návrh je vidět na Obrázku 6.5. Oproti původnímu návrhu došlo k přesunu samotných konektorů ze spodní do přední části celého modulu. Samotné vodivé plochy (šrouby) jsou uchyceny ve válcové šachtě s průměrem odpovídajícím průměru magnetu, který slouží jako hlava pístu a vede šroub v šachtě. Kabely od těchto ploch jsou poté vyvedeny ven z šachty malým výřezem.

Šrouby v reálném modulu přesahují samotný modul zhruba o 5-10 mm, kde je v této poloze zajišťuje pružina. Šrouby se při tlaku stlačují a zajíždějí do modulu, čímž kompenzují



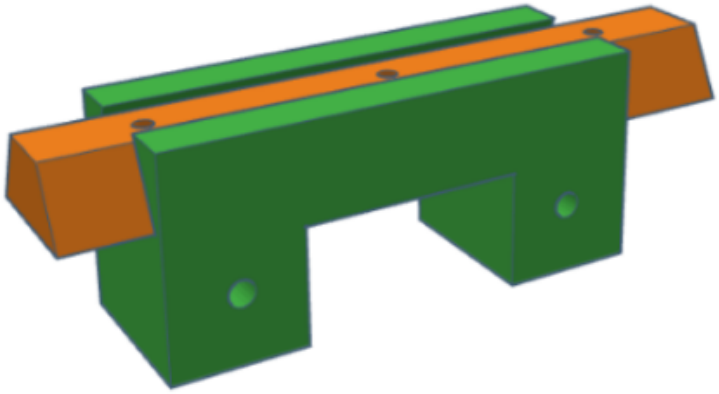
Obrázek 6.3: Konečná podoba dokovací stanice.

polohové i úhlové nepřesnosti ve vzájemné poloze robota a stanice. Reálný stav je vidět na fotografii 6.6. Prvotní testy ukázaly, že tento návrh je prakticky použitelný.

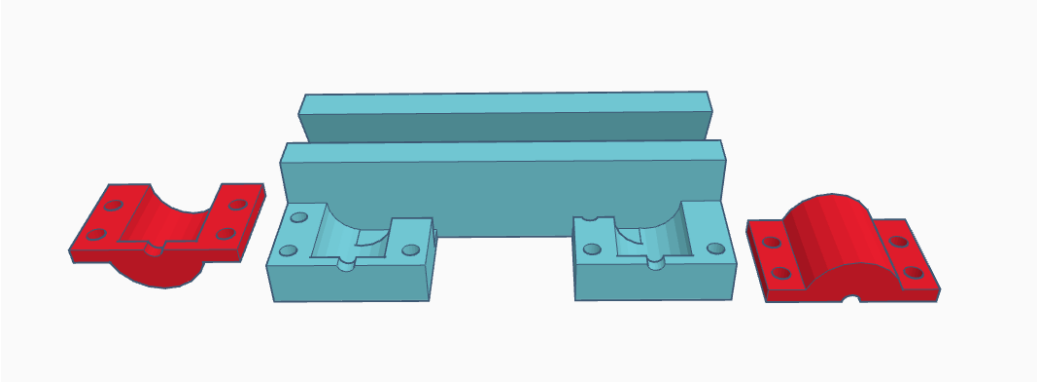
Systém rozvodu energie

Nabíjecí obvod bude pouze rozšířením současného obvodu, který se stará o kontrolu napětí baterie a rozvod energie (jeho blokové schéma je možné vidět na Obrázku 6.7), který byl vytvořen během upgradu Trilobota.

K napájení je využita čtyřčlánková li-pol byterie o nominálním napětí 14.8 V. Ta je napojena na obvod BMS (angl. *battery management system*) 4S 40A, který se stará o nabíjení i vybíjení baterie a balancování jejích jednotlivých článků. Z výstupu BMS obvodu je přímo napájena deska Sabertooth 2x5 a motory. Ostatní komponenty jsou napájené přes převodník napětí, který napětí z baterie snižuje a stabilizuje na 5 V. Nabíjení probíhá buď pomocí souosého konektoru nebo výše popsaných konektorů pro dokovací stanici. Vstupní napětí přitom musí být 12 V. Toto napětí je nejdříve dalším převodníkem zvýšeno na 16-18



Obrázek 6.4: Původní návrh konektoru na robotovi.

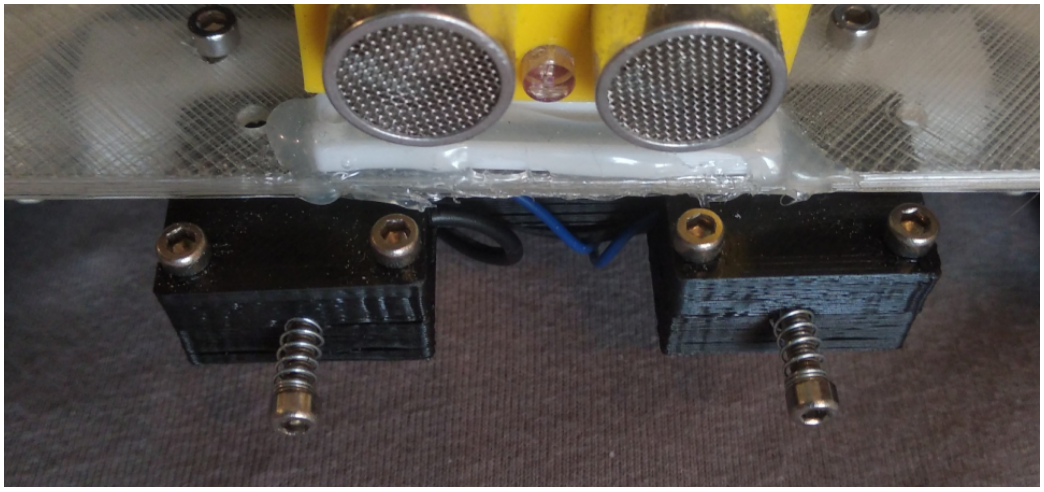


Obrázek 6.5: Konečný návrh konektorového modulu. V hlavní části (modře) došlo k přesunu samotných konektorů do přední části. Uchycení vodivých ploch je poté řešení přišroubováním vrchních částí konektoru (červeně).

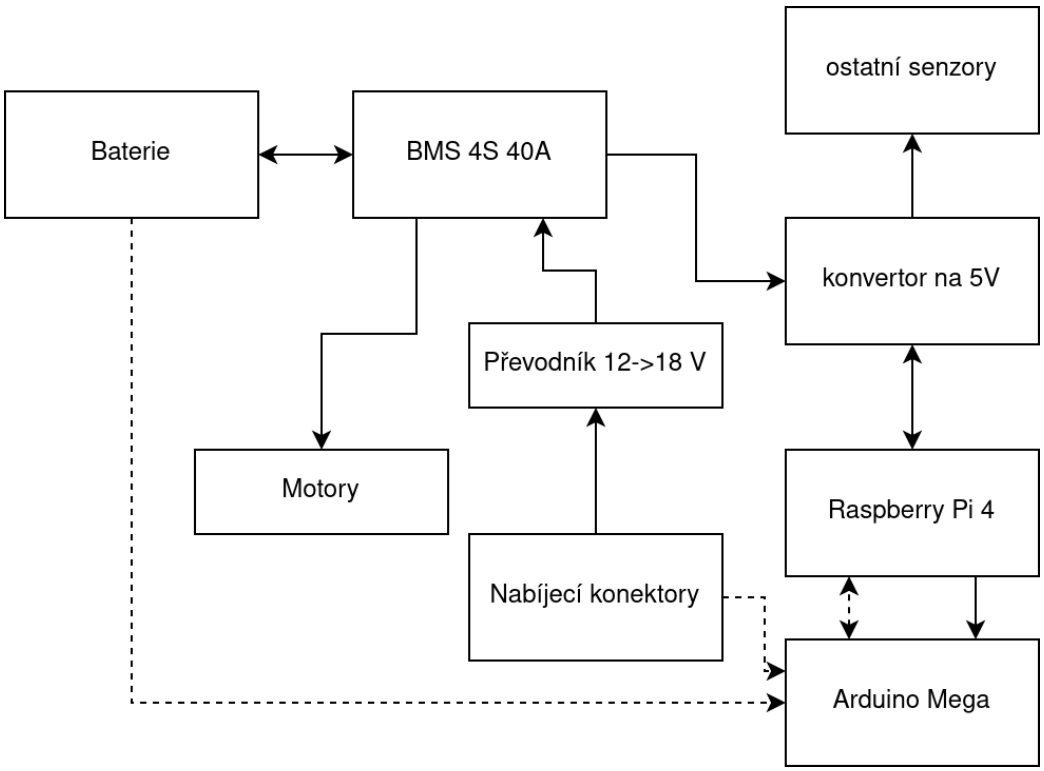
V a přivedeno do BMS obvodu, který již toto napětí využívá k nabíjení baterie. Pokud je baterie odpojena, energie přes BMS proudí přímo do převodníku na 5 V.

Napětí 5 V je poté rozvedeno do všech ostatních komponent, primárně do Raspberry Pi, které následně napájí i Arduino. Ostatní senzory jsou napájené přímo z převodníku.

Obvod BMS bohužel nemá dedikovaný výstup, který by signalizoval potřebu nabití a/nebo stav napětí na jednotlivých článcích baterie. Stav nabíjení tedy zda se robot právě nabíjí, je tedy řešen měřením napětí mezi svorkami napájecích konektorů. Pokud mezi nimi není žádné napětí, robot se nenabíjí, pokud je zde napětí naměřeno, dochází k nabíjení. jelikož je vstupní napětí 12 V, pro 5V logiku desky Arduino je využito děliče napětí, který omezuje vstupní napětí (a díky odporu v řádu stovek tisíc Ω i proud), které je přivedeno na piny Arduina. Tento údaj je již softwarově přeposílán do Raspberry Pi.



Obrázek 6.6: Ukázka konektorů přímo na robotovi. Všimněte si odpružení konektorů, kompenzující drobné odchylky ve vzájemné poloze robota a stanice.



Obrázek 6.7: Schéma toku energie a řízení. Šipky určují směr řízení nebo toku energie. Plná čára znázorňuje tok energie, tečkovaná poté řízení/komunikaci, která má spojitost s napájením.Arduino tedy zaznamenává stav baterie a to, zda se robot v danou chvíli nabíjí. Tyto informace poté posílá do Raspberry Pi přes USB za využití frameworku ROS, resp. jeho součásti `rosserial`.

Kapitola 7

Software

Ač má tato práce jako hlavní cíl vytvořit pouze nabíjecí stanici a k ní příslušný program, bude v této sekci popsána architektura robota Trilobot jako celku. Kapitola je členěna následovně. V první sekci 7.1 je popsána obecná architektura systému na Raspberry Pi, tedy použitý operační systémy, použité frameworky a podobně. V sekci 7.2 je popsána architektura hlavního ovládacího programu ve frameworku ROS včetně s ním komunikujících uzlů. Ve třetí sekci 7.3 je popsána architektura firmware pro desku Arduino Mega.

7.1 Základní architektura na Raspberry Pi

Jak již bylo zmíněno, Trilobot je ovládán jednodeskovým počítačem Raspberry Pi. Tento počítač ke své činnosti vyžaduje standardní operační systém, který podporuje procesory ARM. Ekosystém okolo této desky nabízí vlastní linuxovou distribuci s názvem Raspberry Pi OS (dříve Raspbian). Tento systém založený na Debianu je oficiálním a doporučeným operačním systémem pro Raspberry Pi všech verzí. Díky použitému desktopovému prostředí LXDE je systém poměrně nenáročný. Tento systém ovšem nebyl použit kvůli několika důvodům. Prvním z nich je, že Raspberry Pi OS oficiálně nepodporuje framework ROS, což by potenciálně mohlo vést k problémům s instalací a používáním tohoto frameworku. Druhým důvodem byla absence 64bitové verze systému v době zahájení činnosti na tomto projektu[1].

Jak již bylo zmíněno, stěžejním kritériem pro výběr systému byla nativní podpora frameworku ROS, což výběr omezilo prakticky pouze na operační systém Ubuntu a jeho deriváty. Bylo proto použito Ubuntu MATE, využívající desktopové prostředí Maté. Tento systém je podobně jako Raspberry Pi OS poměrně nenáročný, ale existuje i v 64bitové verzi a nativně podporuje ROS. Po jistých konfiguracích, týkajících se především velikosti ovládacích prvků a textu, je také relativně snadno použitelný na dotykové obrazovce, kterou Trilobot disponuje.

Nad operačním systémem je potom nainstalován framework ROS, o kterém bude následující subsekce.

ROS – Robot Operating System

Robot Operating system, známější pod svou zkratkou ROS, je dle [12] „open-source meta-operační systém určený pro programování robotů. Umožňuje mj. hardwarovou abstrakci, nízkoúrovňové řízení periferií, ale také implementaci vysokoúrovňových algoritmů, napří-

[1] V současné době (květen 2022) již ovšem 64bitová verze Raspberry Pi OS existuje.

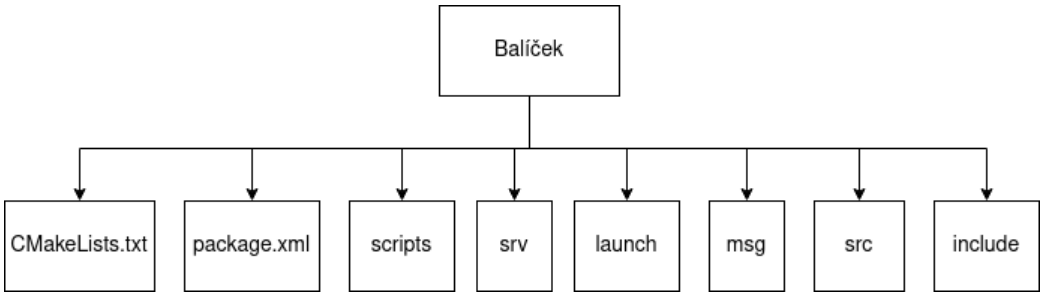
klad pro počítačové vidění nebo umělou inteligenci“ (volně přeloženo). Hlavním cílem frameworku ROS je usnadnit vývoj robotů tím, že se vytvoří pokud možno co největší množství standardizovaných řešení mnoha problémů v robotice: lokalizace, mapování, navigace, SLAM, řízení standardních typů podvozků aj.

Hlavním konceptem frameworku na souborové úrovni jsou balíčky. Tyto balíčky obsahují všechnu funkcionalitu spjatou s konkrétním cílem, může se tedy jednat o řízení konkrétního typu podvozku, implementaci jednoho nebo více algoritmů pro plánování cesty a podobně. Jednotlivé balíčky jsou reprezentovány jako složky s předepsanou strukturou, schematicky zachycenou na 7.1.

Soubor CMakelist.txt je standardním CMake souborem, ve kterém jsou obsaženy detaily k sestavení balíčku (závislosti na dalších balíčcích atp.). Soubor package.xml je tzv. *package manifest*, obsahuje informace o balíčku důležité pro lidské uživatele. Jedná se tedy o název, popis, kontakt na vývojáře, licenci a podobně.

Složka `scripts` již obsahuje zdrojové kódy, v tomto případě skripty v jazyce Python, případně shellové skripty s příponou `.sh`. Ve složce `srv` jsou definovány služby (angl. *services*), které budou podrobněji popsány dále. Složka `launch` obsahuje *launch* soubory, tedy XML soubory, které umožňují spustit více ROS uzlů jedním příkazem, což se hodí zejména pro rozsáhlejší aplikace. Složka `msg` obsahuje definice zpráv, které definuje sám balíček (nejedná se tedy o standardní zprávy, zprávy frameworku ROS budou probrány podrobněji dále). Složky `src` a `include` poté slouží pro kód napsaný v jazyce C++. Zdrojové soubory jsou ve složce `src`, hlavičkové soubory poté ve složce `include`. Pokud balíček některou složku nevyužívá, nemusí ji mít vytvořenou.

Občas se v balíčcích vyskytují i další složky, nepatřící do standardu, například `param` nebo `config`, sloužící pro uložení konfiguračních souborů.



Obrázek 7.1: Adresářová struktura ROS balíčku.

Ze softwarového pohledu jsou aplikace napsané v ROS rozděleny do mnoha relativně jednoduchých skriptů nebo spustitelných souborů, které běží souběžně a vyměňují si informace právě pomocí ROS infrastruktury (systém pro zasílání zpráv, případně služby a akce, bude popsáno dále). Tomuto celku, který je tvořený vzájemně komunikujícími programy, se v terminologii ROS říká „výpočetní graf“. Jednotlivé aplikace (v ROS terminologii odpovídají uzlům) jsou spouštěny buď pomocí launch souborů nebo přímo prostřednictvím příkazu `rosrun`. Tyto programy – uzly – tedy distribuovaně pracují vždy na své části problému, přičemž si data vyměňují několika způsoby: zasíláním zpráv na daná témata (angl. `topics`), využitím služby (angl. *service*) nebo pomocí akce (angl. *action*). Tuto funkcionalitu přitom poskytuje právě framework ROS. Všechny výše uvedené způsoby komunikace způsoby budou popsány dále.

Celý výpočetní graf částečně centralizovaný, i když komunikace jako taková probíhá na principu *peer to peer* spojení. Centrálním uzlem je tzv. *ROS Master*, hlavní uzel, který

funguje jako adresář, kde jednotlivé uzly zjišťují identitu ostatních, jimi nabízená témata, služby a akce. Jakmile uzly zjistí identitu protějšku, již mezi sebou komunikují přímo.

Nejčastějším způsobem komunikace mezi dvěma uzly je zasílání zpráv. Tyto zprávy jsou určitého formátu, který je definován příslušným souborem s příponou `.msg`. Tyto zprávy jsou buď poskytované nějakým jiným balíčkem (včetně standardních, definovaných v balíčku *std_msgs*), nebo vytvořeny nově. Je přitom silně doporučeno používat již existující a známé formáty zpráv, pokud pro daný účel existují, a nevytvářet zbytečně vlastní. Použití již existujících zpráv zlepšuje začlenění vytvářeného balíku do zbytku výpočetního grafu. Zprávy jsou zasílány na konkrétní téma (*topic*). Platí přitom, že jedno téma obsahuje zprávy pouze jednoho formátu. Jednotlivé uzly spolu komunikují přímo, jak bylo popsáno výše. Publikující uzly přitom nabízejí publikovaná témata přes hlavní uzel (ROS Master), uzly-odběratelé stejným způsobem zjišťují identitu publikujícího uzlu. Hlavní uzel tedy funguje jako adresář a katalog témat.

Implementace komunikace pomocí zpráv probíhá tak, že na straně produkujícího uzlu se vytvoří *ROS Publisher*, který bude na dané téma produkovat zprávy daného formátu. Na straně konzumujícího uzlu bude vytvořen *ROS Subscriber*, který bude na každé přijetí zprávy na daném tématu reagovat voláním *callback* funkce.

Druhým způsobem komunikace je služba. Tento způsob komunikace funguje na principu žádost-odpověď(angl. *request-response*). Jeden uzel, klient, žádá druhý uzel, server, o nějakou službu, například vzdálený výpočet, dodání dat a podobně. Z principu se jedná o spojení 1 ku 1. Tento model se hodí například v situaci, kdy uzel potřebuje od serveru konkrétní informace, jejichž vysílání zprávou by bylo nepraktické nebo nežádoucí.

Posledním typem komunikace je akce (angl. *action*). Z praktického hlediska se jedná o podobný model komunikace jako služba, počítá se ovšem s tím, že akce trvá delší dobu. Klient je proto informován o stavu požadavku. Typickým využitím je zasílání navigačních cílů do uzlu `move_base`, který se stará o plánování cesty.

ROS celkově klade důraz na znovupoužitelnost, modularitu a univerzálnost. Tyto cíle jsou z velké části naplněny. Ovšem vzhledem k tomu, že každý balíček je udržovaný jiným vývojářem nebo jejich skupinou, liší se styl i úroveň dokumentace, která je často neúplná, zastaralá, či přímo chybějící. Efektivita frameworku také klesá při velkém počtu současně běžících uzlů, kdy se začíná projevovat režie způsobená systémem zasílání zpráv.

Transformace ve frameworku ROS

Jedním z důležitých úkolů robota je být schopný lokalizovat se, jinými slovy, určit svoji polohu v mapě. Ve frameworku ROS je toto řešeno tzv. „stromem transformací“ (angl. *transformation tree*). Jednoduše řečeno, v každém ROS systému existuje několik souřadnicových systémů, které se vzájemně liší polohou počátku a natočením os. Mezi jednotlivými systémy se body v prostoru převádějí pomocí transformací. Tyto transformace poté tvoří strom transformací.

V této subsekci budou nejprve vysvětleny základní souřadnicové systémy společné většině robotů, poté budou představeny i systémy specifické pro robota Trilobot.

Hlavní souřadnicový systém se nazývá `map`[2]. Jeho počátek se nachází zpravidla v místě první inicializace robota, pokud si tento robot mapu vytváří, případně se nachází v jed-

[2]Teoreticky existuje ještě souřadnicový systém `world`, který odpovídá poloze na planetě Zemi. Jeho počátek leží na průsečíku nultého poledníku s rovníkem na úrovni hladiny moře. Jeho použití je ale velice zřídkavé. V některých aplikacích je definován také jaké počátek nějakého omezeného prostoru, který je mapován více mapami, například plocha výrobního závodu, kde každá z budov má vlastní mapu.

nom z rohů mapy, pokud je tato mapa dostupná předem. Na tento systém (v terminologii ROS se souřadnicové systémy nazývají *frames*, rámce) poté navazuje systém `odom`. Tento systém má počátek soustavy souřadnic v bodě, kde je robot inicializován v rámci mapy, při první inicializaci jsou tedy rámce totožné, ovšem okamžité po inicializaci se s největší pravděpodobností začnou lišit (vizte dále).

Transformace mezi rámci `map` a `odom` poté vyjadřuje chybu odometrie vůči poloze robota. K výrazné změně transformace mezi těmito dvěma rámci dochází například při prokluzu kol. Transformace dále vyjadřuje prvotní posun a natočení při inicializaci robota. V čase se tedy transformace mění, konstantní (v terminologii ROS „statická“) by zůstávala jen v případě, kdy by byla odometrie dokonalá.

Následují rámce `base_footprint` a `base_link`, které určují polohu a orientaci robota. Rámce se dle definice mírně liší, `base_footprint` určuje pozici robota na podložce (ve výšce 0), `base_link` by poté měl být ve středu centrálního prvku robota, i když je často rámec `base_footprint` vynecháván a místo něj používán jen `base_link`. Tento postup se uplatnil i v případě robota Trilobot, jelikož tento robot nemá žádné pohyblivé části a mít dva různé rámce vyjadřující prakticky totéž by bylo zbytečné. Transformace mezi rámci `base_link` (případně `base_footprint` a `odom` vyjadřuje cestu, kterou robot urazil.

Ostatní rámce jsou již specifické pro robota Trilobot. Prvním z nich je několik rámců, které se týkají kamery Intel RealSense, z nichž nejdůležitější je rámec `camera_link`, vyjadřující střed kamery. V tomto a od něj odvozených rámcích jsou popsána data z kamer. Kromě základního rámce existují i rámce zvlášť pro akcelerometr, gyroskop, barevnou a hloubkovou kameru, které reflektují drobné rozdíly ve fyzickém umístění senzorů v rámci pouzdra kamery.

Posledním rámcem je rámec dokovací stanice, `dock`. Tento rámec přitom odpovídá spíše než rámci samotné dokovací stanice rámci použitého tagu[3]. Zvláštností tohoto rámce je osa z, směřující nikoli kolmo vzhůru, ale do prostoru rovnoběžně s podložkou. Díky stromu transofrmací a automaticky získané transformaci mezi `camera_link` a `dock` je tato zvláštnost nepodstatná a správci rámců nečiní žádné problémy.

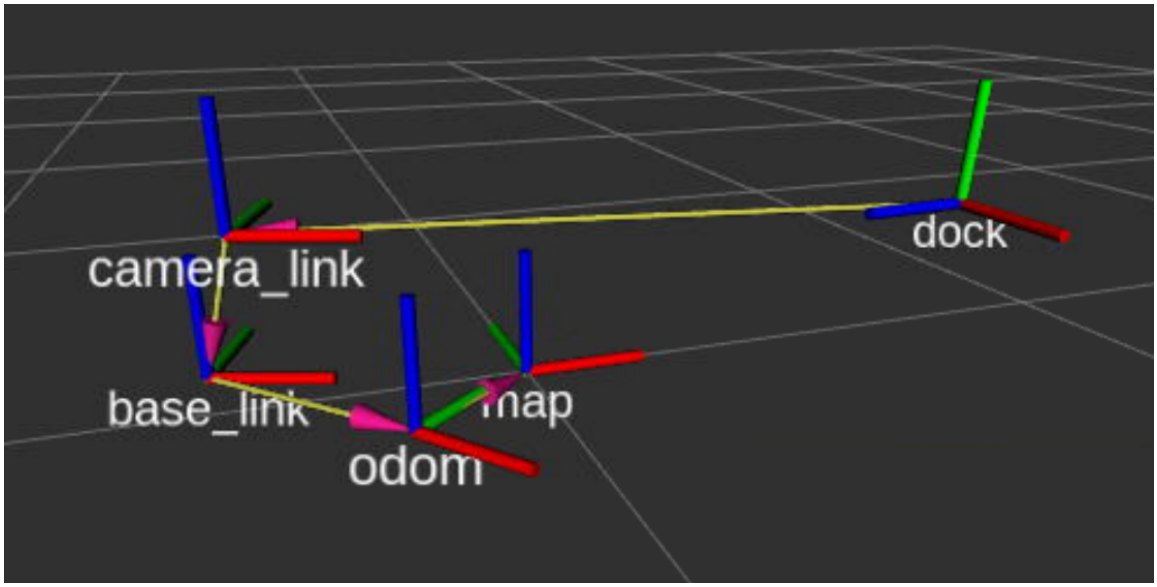
Na obrázku 7.2 je poté vidět vizualizace jednotlivých rámců, přičemž referenční rámec je `map`, jeho počátek souřadnic byl tedy určen jako hlavní a všechny ostatní se dopočítávaly pomocí transformací. Robot byl inicializován nově, původně tedy byly rámce `map` a `odom` totožné. Jenže kvůli zaseknutí robota (robot byl chvíli při pohybu držen na místě, aby bylo dosaženo prokluzu kol a tím i chyby v odometrii) se muselo přistoupit ke kompenzaci, která se projevila jako rozdílná poloha rámců `map` a `odom`. Rámec `dock` má poté rozdílnou orientaci os, osa Z (modře) nemíří nahoru, ale je rovnoběžná s podložkou.

7.2 Architektura hlavního programu

Při tvorbě software bylo jedním z cílů maximálně využít možností, které nabízí framework ROS, specificky již hotové balíčky pro konkrétní problémy, jakými jsou například navigace nebo SLAM. Toto umožnilo nejen snadnější implementaci, ale také snadnější budoucí údržbu, jelikož již existující balíky jsou zpravidla dokumentovány a používány napříč různými roboty, jejich API je tedy mezi ROS vývojáři obecně dobře známo.

Podrobné schéma jednotlivých uzlů, spolu s typy zpráv a použitými tématy, je k dispozici na 7.3. Ve zbytku subsekce budou dále popsány jednotlivé standardní balíky, které byly použity. Vlastní kód specifický pro robota Trilobot bude probrán v samostatné subsekci

[3]Jak je zmíněno v sekci 6.1, jedná se o Apriltag, konkrétně o rodinu 36h11.



Obrázek 7.2: Ukázka rámců, použitých v této práci. Červeně osa X (standardně osa pohybu dopředu-dozadu), zeleně osa Y (pohyb do stran) a modře osa Z (standardně osa pro pohyb nahoru-dolů).

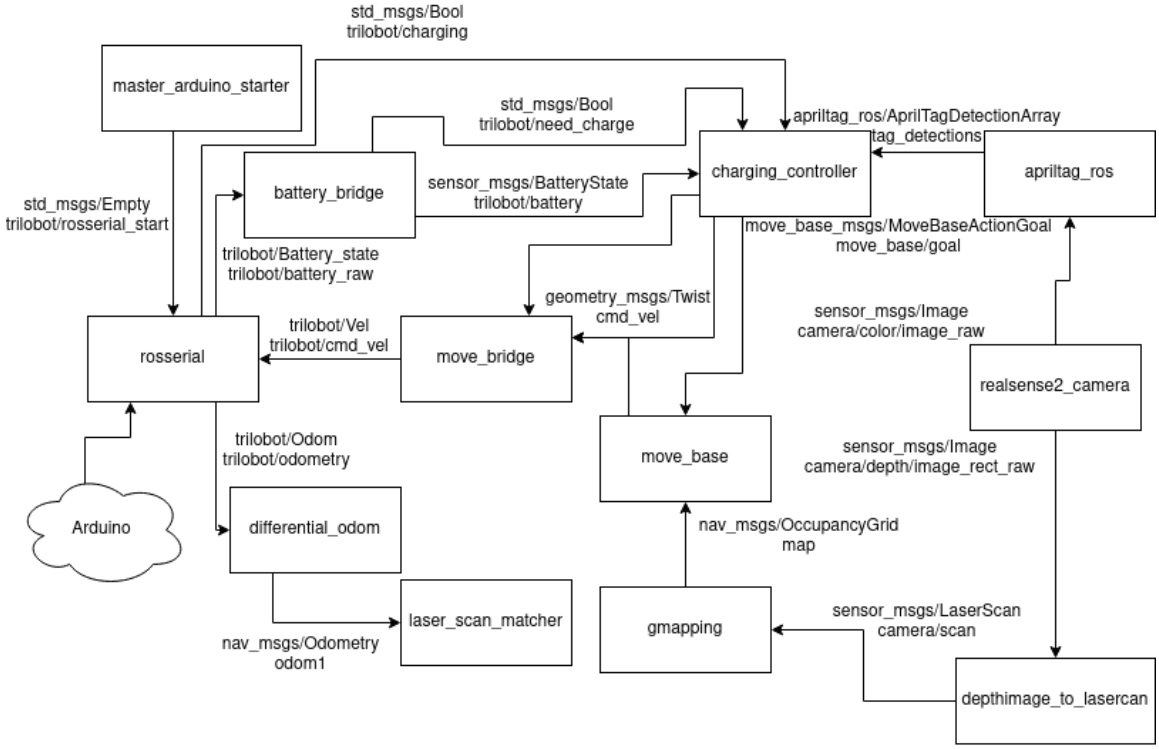
7.2.1. Tento kód (sdružený do ROS balíčku `trilobot`) přitom zajišťuje komunikaci s deskou Arduino přes `rosserial`, připravené launch soubory včetně konfigurací použitých balíčků, vlastní ovládací kód, který umožňuje využít dokovací stanici aj.

Jedním z hlavních cílů práce bylo vytvořit program, který robotovi umožní využívat dokovací stanici, jinými slovy, program umožní robotovi stanici najít a rozpoznat, zjistit vzájemnou polohu robota a stanice, naplánovat cestu a tuto cestu absolvovat. U stanice se dále musí být schopný připojit tak, aby probíhalo nabíjení, a po jeho ukončení se opět odpojit.

Rozhodl jsem se pro vytvoření univerzálního ovládacího programu, který robotovi umožní realizovat simultánní lokalizaci a mapování (SLAM), navigaci obecně k jakémukoli cíli na mapě a plné využití všech relevantních senzorů. Framework ROS přitom poskytuje prakticky všechny potřebné algoritmy v již hotových balících, jakými jsou `move_base` pro plánování cesty a různé mapovací/SLAM balíky, například `gmapping` nebo `hector_mapping`. Je přitom silně nedoporučováno psát vlastní algoritmy ve vlastních balících pokud již takové existují, zejména kvůli kompatibilitě se zbytkem frameworku ROS.

Ač se může zdát, že vytváření programu v ROS je jen triviálním „spouštěním balíků“, realita je poněkud odlišná, jelikož kvůli snaze o co největší univerzálnost obsahují balíky často i stovky konfiguračních proměnných, jejichž vzájemné vyladění zabírá největší část celé implementace. Při návrhu a popisu schématu z 7.3 proto začnu nejprve s minimalistickým příkladem, který budu postupně rozšiřovat.

Pokud má být mobilní robot schopen mapovat své okolí a pohybovat se v něm, je třeba použít dva hlavní balíky, jak je vidět na schématu 7.4: `move_base` a jeden z balíků, který umožňuje mapování a/nebo SLAM, jakými jsou například rtabmap, hector_slam nebo gmapping. Balík `move_base` slouží k plánování cesty. Kvůli jeho univerzálnosti je třeba poměrně složitého nastavování v několika konfiguračních souborech, ale vzhledem k tomu, že zprovoznění `move_base` nebylo hlavním cílem práce, nebude tu konfigurace

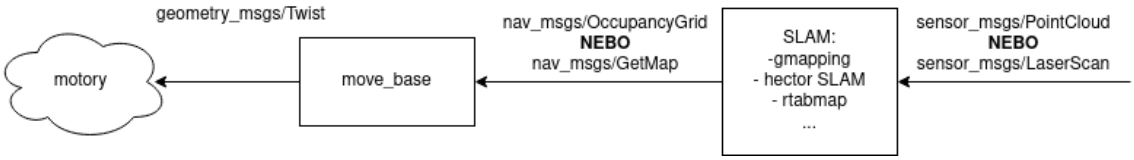


Obrázek 7.3: Blokové schéma programu ve frameworku ROS. Podrobnější vysvětlení v textu.

detailněji vysvětlována. V konfiguračních souborech (složka `param` v kořenovém adresáři balíku `trilobot`) jsou ovšem všechny konfigurační soubory i s komentářem.

Balík `move_base` pracuje na dvou úrovních, které odpovídají lokálnímu a globálnímu plánování. Mapy pro oba plánovače mají formu cenové mapy (angl. *cost map*) a v této mapách hledají nejvhodnější cestu do cíle. Důvod pro použití dvou map je ten, že globální mapa poskytuje představu o celé cestě, ale její přepočítávání je poměrně náročné. Zároveň je také zbytečné přepočítávat celou mapu, jelikož robot v daném čase využívá především údaje v jeho bezprostředním okolí. Pro tyto účely slouží lokální mapa, která zachycuje relativně malé okolí robota a může být tedy přepočítávána častěji.

Zatímco globální plánovač byl ponechám standardní, lokální plánovač byl změněn na `teb_local_planner` poté, co standardní plánovač `base_local_planner` nebyl schopen trasu adekvátně naplánovat a absolvovat. Plánovač `teb_local_planner` je přitom mimořádně vhodný pro roboty s diferenciálním řízením podvozku. Využívá přitom *timed elastic band* algoritmu, který byl popsán v [25], ke kterému přidává některá rozšíření, která předcházejí uváznutí algoritmu v lokálních minimech např. kvůli překážkám. Detaily v této práci nebudou rozebírány.



Obrázek 7.4: Minimalistické schéma jádra ovládacího programu pro mobilního robota v ROS.

Cíl se uzlu `move_base` (stejné jméno jako balík) lze zadat buď zasláním zprávy typu `geometry_msgs/PoseStamped` na téma `move_base_simple/goal`, nebo lze využít akce. První způsob je jednodušší, ale uzel, který pohyb inicioval, nemá informace o jeho průběhu. Druhý způsob je poněkud složitější na implementaci, ale umožňuje průběžně sledovat, zda již bylo cíle dosaženo. Výše zmíněný typ zprávy obsahuje kromě časového razítka požadovanou polohu (bod v 3D prostoru, pro 2D použití se ponechává výšková souřadnice nulová) a orientaci (formou kvaternionu), zpravidla v rámci `odom` nebo `map`.

Co se týče SLAM, ROS nabízí několik balíků, podporujících buď pouze mapování[4], či případně kompletní SLAM funkcionalitu.

V tomto projektu je použit balík `gmapping`, který je zástupcem druhé zmíněné skupiny, a tedy poskytuje jak lokalizaci, tak mapování. Jedná se asi o nejvíce používaný balík pro tyto účely, zároveň není příliš výpočetně náročný.

Gmapping k tvorbě mapy využívá data z laserového dálkoměru a není schopen využít data ve formě oblaku bodů (angl. *pointcloud*) nebo hloubkového obrazu, které produkuje kamera Intel Realsense. Toto je ovšem poměrně častý problém, proto existuje jednoduchý balík `depthimage_to_laserscan`, který z hloubkového vezme jen několik řádků, které následně zkombinuje, čímž vznikne ekvivalent laserového skenu, který je již pro gmapping použitelný. Gmapping dále využívá odometrii. V původním návrhu byla odometrie poskytována uzlem *differential_odom* (součást balíku `trilobot`, bude popsáno dále). Jednalo se ovšem o odometrii spočtenou na základě tzv. *dead-reckoning* pouze z dat enkodérů. Tato metoda bohužel nereflektovala drobné prokluzy např. při otáčení a způsobovala proto velmi často způsobila ztrátu polohy. Do systému byl proto přidán uzel `laser_scan_matcher`, který k ke zpřesnění odometrie využívá i data ze simulovaného laserového skeneru, čímž došlo k výraznému zlepšení.

Na ilustraci 7.4 je zachyceno, že zprávy typu `geometry_msgs/Twist`[5] putují k motorům. Toto je velmi abstraktní, ač správná formulace, v případě konkrétního robota je ovšem nutné řešit, jakým způsobem budou zprávy typu `Twist` transformovány na příkazy, které uvedou do provozu skutečné motory, a to na požadované rychlosti. Tuto transformaci zajišťuje uzel `move_bridge`, který je součástí balíku `trilobot` (vizte 7.2.1).

Jak již bylo několikrát zmíněno, dokovací stanice je identifikována pomocí Apriltagu. Tento tag a jeho poloha jsou zjišťovány z RGB obrazu kamery Intel Realsense. Tato kamera má vlastní ROS balík s názvem `realsense2_camera`, který zajišťuje komunikaci s hardwarem kamery a vytváření ROS-kompatibilních zpráv s daty. Obraz z RGB kamery zpracovává uzel `apriltag_ros`, který v obraze hledá příslušné tagy (v tomto případě pouze jediný) a pokud je najde, vysílá do zbytku ROSu jejich rámec, v případě Trilobota nazývaný `dock`.

Posledním balíkem třetí strany, který dosud nebyl probrán, je `rosserial`. Tento uzel umožňuje desce Arudino, aby mohla komunikovat se zbytkem frameworku ROS na desce Raspberry. Uzel překládá zpráávy mezi těmito dvěma deskami a přenáší je po USB sběrnici s pomocí vlastního protokolu.

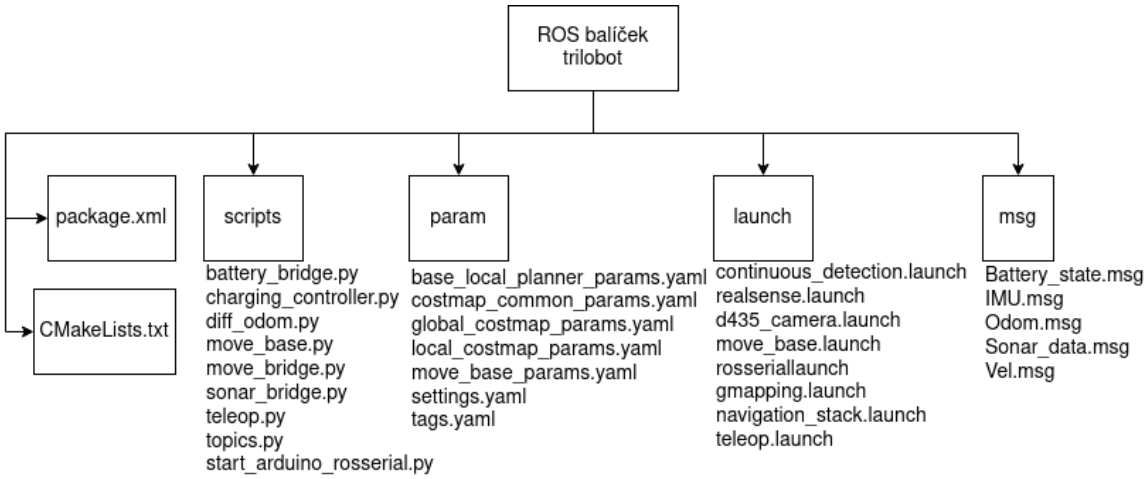
7.2.1 ROS balíček `trilobot`

V předchozích subsekcích se probírala hlavně struktura aplikace a jednotlivé uzly a balíky třetích stran, které byly v rámci projektu použity. Tato subsekce je zaměřena na vlastní

[4] lokalizační funkcionalitu je možné doplnit jiným vhodným balíkem, například na principu Kalmanova filtru nebo adaptivní Monte Carlo lokalizace, oba postupy jsou implementovány v ROS balících.

[5] Zpráva typu `Twist` obsahuje údaje o požadované lineární a úhlové rychlosti ve všech třech osách.

autorovu práci, tedy zdrojové kódy ve frameworku ROS[6]. Tyto kódy přitom zajišťují primárně komunikaci s deskou Arduino a překlad komprimovaných zpráv do standardních a dále jeden z hlavních cílů této práce, tedy navigaci robota k dokovací stanici a nabíjení. Balíček bude popisován postupně dle své adresářové struktury, kterou můžete vidět na 7.5. Výjimku tvoří soubor `charging_controller.py`, který implementuje hlavní funkcionalitu, tedy navádění robota k dokovací stanici, který bude popsán v samostatné subsekci 7.2.2.



Obrázek 7.5: Přehledová struktura balíčku `trilobot`.

Soubory `CMakeLists.txt` a `package.xml` obsahují informace pro překlad pomocí nástroje `catkin` (speciální překladač pro ROS balíky) a základní údaje o balíku.

Ve složce `scripts` jsou umístěny skripty v jazyce Python. Soubory, `battery_bridge.py`, `move_bridge.py`, `move_base.py` `sonar_bridge.py` a `diff_odom.py` se starají především o konverzi ze zpráv, které jsou používané Arduinem, do standardních typů zpráv. Ve většině případů se jedná skutečně pouze o přepis údajů ze strohých zpráv z Arduina do poněkud výřečnějších ROS zpráv. Pro příklad zde uvádím strukturu zprávy `Battery_state`, která je specifická pro balík `ros-trilobot` a zprávy `BatteryState`, což je standardní ROS zpráva pro zasílání informací o baterii 7.6. Uzel `move_bridge`, který je spouštěn ze souboru `move_base.py`, slouží pro překlad příkazů o pohybu robota ze standardních zpráv `geometry_msgs/Twist` do redukovaných zpráv pro Arduino. Zároveň také implementuje (ve třídě `MoveBridge` ze souboru `move_bridge.py`) dva různé toky příkazů, standardní a prioritní. Druhý zmíněný typ se uplatňuje při pohybu k dokovací stanici, jelikož se předpokládá, že potřeba nabít se má přednost před jinými úkoly robota.

V současné době je pouze vypsáno varování při prioritním pohybu, v budoucnu by bylo dobré naprosto oddělit témata pro zasílání zpráv pro „běžný“ a „prioritní“ pohyb. Jelikož ale uzel `move_base`, který se stará o plánování cesty i zasílání samotných příkazů pro pohyb nejen v době navádění k dokovací stanici, není schopen dynamicky měnit témata, na jaká zasílá tyto příkazy, není kompletní oddělení příkazů implementováno.

Soubor `start_arduino_rosserial.py` zajišťuje, že se bude vykonávat hlavní smyčka kódu Arduina. Pokud by se na tématu `trilobot/rosserial_start` po určitou dobu neobjevila žádná zpráva, smyčka v Arduinu se přestane vykonávat.

[6]Výjimku tvoří soubor `diff_odom.py`, který autor převzal. Detaily v textu u popisu jednotlivých souborů.

Soubor `topics.py` obsahuje centralizovaný přehled používaných témat s krátkým komentářem. Důvod pro vyčlenění témat do samostatného souboru je jejich jednodušší údržba a zároveň eliminace chyb v důsledku překlepů.

Soubor `teleop.py` umožňuje Trilobota řídit přes terminálové rozhraní s využitím klávesnice. Nepoužívá uzel `move_base` pro zadávání cílů, ale přímo zasílá zprávy na téma `cmd_vel`.

Battery_state	**BatteryState**
float32 cell1	std_msgs/Header header
float32 cell2	float32 voltage
float32 cell3	float32 temperature
float32 cell4	float32 current
bool charging	float32 charge
	float32 capacity
	float32 design_capacity
	float32 percentage
	uint8 power_supply_status
	uint8 power_supply_health
	uint8 power_supply_technology
	bool present
	float32[] cell_voltage
	float32[] cell_temperature
	string location
	string serial_number

Obrázek 7.6: Porovnání zprávy balíčku `trilobot/Battery_state` a standardní zprávy `BatteryState`

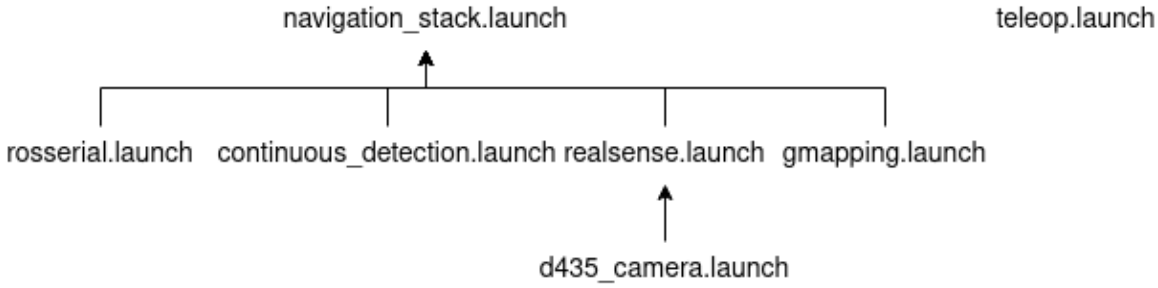
Soubor `diff_odom.py` není autorovým dílem, ale byl zkopírován a upraven z ROS balíku `diff-drive` od Jona Stephana [30]. Tento skript odebírá zprávu typu `trilobot/Odom` s počtem pulzů každého z enkodérů od začátku pohybu a směrem, kterým se každé z kol otáčí. Skript v původní podobě počítal pouze s tím, že se kola diferenciálního podvozku otáčí pouze dopředu. Jelikož ale Trilobot umožňuje mj. otáčení na místě, kdy se jedno kolo otáčí v opačném směru než druhé, byl skript upraven tak, aby akceptoval i obrácený směr otáčení a korektně publikoval data pro odometrii.

Ve složce `param` jsou uloženy konfigurační soubory pro balík *move_base* a `apriltag_ros`. Jak již bylo řečeno, detailní popis všech parametrů, kterých je několik desítek, by zabral zbytečně mnoho místa a není pro tuto práci klíčový. Ve zdrojovém kódu je ovšem vše detailně komentováno.

Složka `launch` obsahuje tzv. *launch files*. Tyto soubory umožňují spouštění a konfiguraci více balíků zároveň. V rámci tohoto balíku byla použita strategie relativně malých a jednoduchých launch souborů a jejich hierarchické spojování, jak je vidět na obrázku 7.7.

Hlavním souborem je `nav_stack.launch`. Tento soubor spouští další launch soubory, uvedené v přehledu níže:

- Soubor `realsense.launch` (momentálně spouštěný externě, bude vysvětleno), který spouští uzel *realsense2_camera* pro komunikaci s kamerou Intel RealSense.
- Soubor rosserial.launch, který zajišťuje komunikaci s deskou Arduino Mega.
- Uzel `differential_odom`, spouštěný souborem `diff_odom.py`.



Obrázek 7.7: Struktura launch souborů v balíku `trilobot`. Šipka znamená, že zdrojový balík je spouštěn v rámci cílového.

- Soubor `move_base.launch`, který spouští stejnojmenný uzel pro plánování cesty a její absolvování.
- Soubor `gmapping.launch`, který se stará o SLAM.
- Soubor `continuous_detection.launch`, který v obraze z kamery detekuje dokovací stanici podle jejího tagu AprilTag.

Soubor `realsense.launch` slouží ke spouštění ROS *wrapperu* pro kameru Intel Realsense. Tento wrapper, zejména na slabším hardwaru, ovšem ne vždy korektně provede inicializaci kamery a je nutné ho restartovat, proto, i když by měl být spouštěný z hlavního launch souboru, je tento spouštěn samostatně, aby bylo možné ho snadno restartovat. Kromě launch souboru samotné kamery `d435_camera.launch`, který obsahuje konfiguraci a spouštění samotných uzlů, je ještě v rámci tohoto souboru spouštěna statická transformace mezi rámcem `base_link` a `camera_link`, která vyjadřuje polohu kamery vůči robotovi. Soubor má také možnost spustit filtr dat z IMU modulu (`imu_filter_madgwick`), který je ale v současné chvíli vypnutý, jelikož se data z IMU nepoužívají.

Soubor *rosserial.launch* se stará o spouštění uzlu `rosserial` pro komunikaci s Arduinem. Mimo to ovšem spouští i všechny „můstky“, které překládají zprávy mezi komprimovaným a standardním formátem.

Soubor `move_base.launch` slouží ke spuštění uzlu `move_base` stejnojmenného balíku, spolu s načtením parametrů ze souborů ve složce `param`. Funkcionalita tohoto uzlu je detailněji popsána v 7.2.

Soubor `gmapping.launch` spouští a konfiguruje stejnojmenný uzel, jehož cílem je SLAM.

Posledním je soubor `continuous_detection.launch`. Tento soubor spouští uzel `apriltag_ros_continuous_node` z balíčku `apriltag_ros`. Cílem tohoto uzlu je odebírat obraz z kamery Intel RealSense, hledat v něm tag příslušící dokovací stanici, a podat informace o transformaci tohoto tagu relativně k rámci *camera_color_optical_frame*, což je jeden z rámců kamery.

Poněkud stranou celé hierarchie je soubor `teleop.launch`. Tento soubor spouští uzel `trilobot_teleop` ze souboru `teleop.py` a uzel `key_teleop` ze stejnojmenného balíčku. Pro ovládání robota klávesnicí je nutné ještě spustit soubor *rosserial.launch*. Toto dělené ovládání je způsobeno tím, že během vývoje bylo často třeba převzít ovládání např v situaci, kdy selhala lokalizace/mapování a hrozil náraz robota.

Dosud nezmíněné senzory a jejich podpora v balíku `trilobot`

V textu byly několikrát zmíněny ultrazvukové sonary SRF-08 a modul minIMU-v3, ovšem jejich podpora v ROSu nebyla dále rozvedena, jelikož nejsou nutné pro hlavní cíl celé práce, navigaci k dokovací stanici.

Ultrazvukové sonary SRF-08 jsou pro účely proaktivní navigace a mapování poměrně nevhodné, jelikož nejsou schopné provádět měření na vysokých frekvencích. Teoretická maximální frekvence měření při 60 ms, což je minimální měřicí čas daný dokumentací, je 16.7 Hz, v současné konfiguraci pouze 10 Hz kvůli velmi konzervativní periodě měření 100 ms. Další stinnou stránkou sonarů je jejich velmi široký rozsah, který dále znepřesňuje výsledky.

Výhodou je ovšem jejich umístění po celém obvodu robota, což zajišťuje informaci o překážkách i mimo zorný úhel kamery. Toto by šlo využít pro reaktivní vrstvu přímo na úrovni desky Arduino, která by mohla velmi rychle reagovat na neočekávané překážky a nouzově zastavit robota rychleji, než kdyby se tak dělo přes standardní cestu na desce Raspberry Pi. Jelikož ale tato funkcionalita nebyla vyžadovaná a ani by nebyla prospěšná současnému určení, nebyla proto implementována a jedná se o jedenu ze zajímavých možností rozšíření robota.

Modul minIMU-v3 sice poskytuje poměrně důležitá data o poloze robota, která by šla dobře využít v odometrii, zvlášť v kombinaci s IMU z kamery RealsenSense, jelikož ale deska Raspberry Pi trpí nedostatkem výkonu pro současné úlohy, nepoužívá současný ovládací program data ani z jedné z dostupných IMU jednotek. Softwarová podpora je tedy na úrovni kódu desky Arduino, data se ovšem nikam nezasílají a ve většině testů není modul minIMU ani fyzicky osazen na robotovi.

7.2.2 Navádění Trilobota k dokovací stanici, soubor `charging_controller.py`

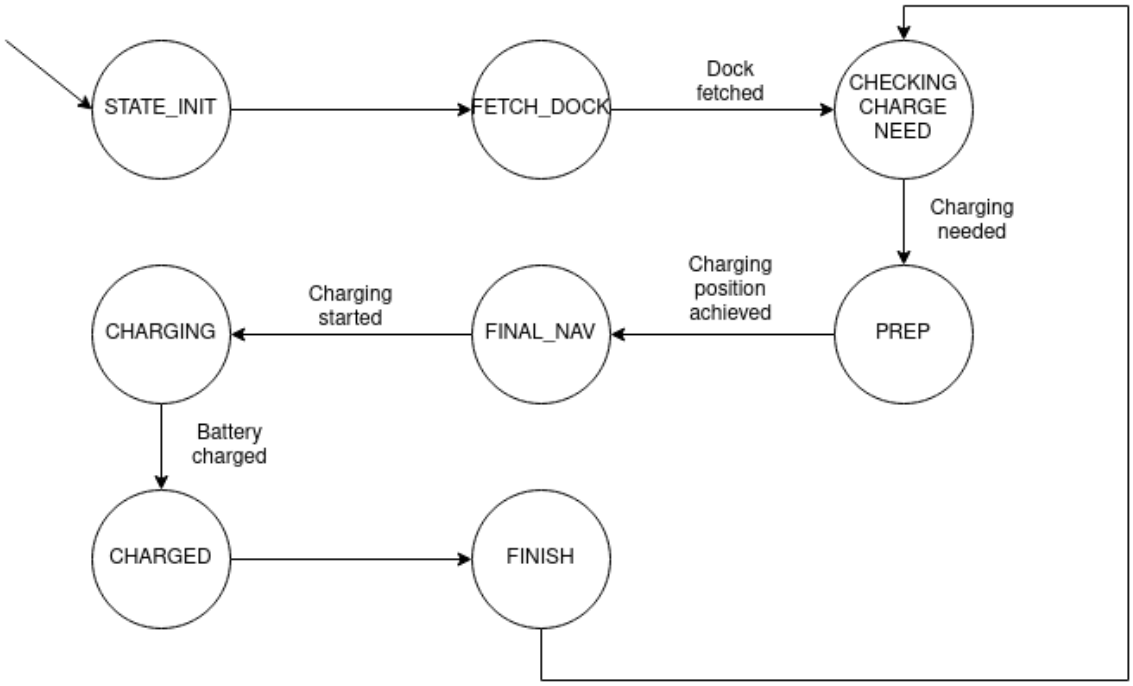
V této subsekci bude detailně popsán princip, jakým se robot navádí k dokovací stanici. Algoritmus se nachází v souboru `charging_controller.py`. Nejprve bude poskytnuto shrnutí hlavní myšlenky algoritmu, poté bude probrán detailněji.

Po inicializaci program nejprve čeká, než bude identifikována dokovací stanice. Tento stav je indikován tím, že je dostupná transformace do rámce `dock` a zároveň je přijata zpráva od uzlu `apriltag_ros`, informující o nalezení tagu v obraze. Ve chvíli, kdy je dokovací stanice nalezena, je její pozice uložena, pro budoucí navigaci. Poté program kontroluje stav napětí na článcích baterie a zároveň kontroluje, zda na tématu `trilobot/charge_needed` není obdržena zpráva. Pokud ano, je zahájena nabíjecí fáze. Nejprve se robot dostane do nabíjecí polohy přímo před dokovací stanicí ve vzdálenosti cca 30 cm. Tato první fáze cesty je řešena pomocí uzlu `move_base` a jeho plánování cesty. Poté již robot jede dopředu a případně lehce upravuje svou polohu vzhledem ke stanici. Jakmile je zaznamenáno, že se Trilobot nabíjí, přestane se pohybovat dopředu a čeká, až se baterie nabije. Ve chvíli, kdy uplyne čas potřebný pro nabití, robot se odpojí a odjede od nabíjecí stanice.

Na 7.8 je znázorněn algoritmus v podobě konečného automatu. Níže bude algoritmus popsán detailně.

Stav `INIT` je startovním stavem, neprobíhá v rámci něj žádná akce. Je ovšem ponechám jako potenciální místo inicializace případných rozšíření, která by to vyžadovala. Prozatím se ovšem automaticky přechází do dalšího stavu, `FETCHING_DOCK`.

Ve stavu `FETCHING_DOCK` je získávána nabíjecí pozice před dokovací stanicí. Tato pozice se získá pomocí zpráv od uzlu `apriltag_ros_continuous_node`. Přičemž příslušná *callback* funkce nezískává údaje o poloze stanice ze zprávy samotné, ale její přijetí pouze indikuje dostupnou polohu ve stromu transformací, odkud je také získána. Nabíjecí pozice se nachází



Obrázek 7.8: Algortimus navádění k dokovací stanici.

určitý počet centimetrů přímo před nabíjecí stanicí, tento počet centimetrů udává konstanta `CP_OFFSET`. Experimentálně byla konstanta nastavena na hodnotu 0.5 m. Ve chvíli, kdy je k dispozici tato nabíjecí pozice, přechází algoritmus do stavu `CHECKING_CHARGE_NEED`.

Ve stavu `CHECKING_CHARGE_NEED` je pouze kontrolovaný stav vnitřní proměnné, která je aktualizovaná v *callback* funkci příslušného *ROS Subscriberu*. Existují přitom dvě možnosti, jak vyvolat potřebu nabíjení, buď automaticky pomocí monitoringu napětí na bateriích (který probíhá již v Arduinu) a zasláním příslušné zprávy, nebo zasláním zprávy na téma `trilobot/charge_anyway`. Druhý případ vyvolá dokovací proceduru nehledě na stav baterie. Primárně sloužil pro potřeby testování, ale robot v principu nemusí čekat, než budou jeho články vybité, aby je dobil, proto byla funkcionalita zachována. Ve chvíli, kdy je třeba robota nabít, přechází se do stavu `PREP`.

Ve stavu `PREP` se odehrává poměrně velká část celého procesu. Nejprve je uzlu `move_bridge`, který překládá povely pro pohyb Trilobota, oznámeno, že začal probíhat prioritní pohyb. Detaily stran prioritního pohybu jsou popsány u příslušného uzlu 7.2.1. Následně je nabíjecí pozice (získaná ve stavu `FETCHING_DOCK` nastavena jako současný cíl pro pohyb a odeslána jako akce do uzlu `move_base`. Ve chvíli, kdy je pohyb do nabíjecí pozice úspěšně dokončen, přechází algoritmus do stavu `FINAL_NAV`.

Ve stavu `FINAL_NAV` dochází k finálnímu přibližování ke stanici. Toto přibližování přitom není řešeno pomocí plánování cesty a uzlu `move_base`, ale pomocí vzájemné polohy robota a stanice a geometrické dopočítávání lineární a úhlové rychlosti. Pohyb přitom končí ve chvíli, kdy se robot začne nabíjet (svými kontakty se tedy napojí na dokovací stanici). Ve chvíli, kdy se robot začne nabíjet, ještě mírně popojede dopředu, aby došlo ke stlačení kontaktů a tím vytvoření bezpečnějšího spoje. Pokud robot zaznamená, že již nedostává aktualizace o poloze dokovací stanice, vyhodnotí tento stav tak, že se již nachází natolik blízko, že kamera nezabírá celý tag. Pokud se poté do určitého krátkého času (řádově jednotky sekund)

nezačne nabéíjet, vyhodnotí robot situaci jako neúspěšný pokus o zadokování a popojede zhruba o 50 cm zpět s cílem pokusit se znovu zadokovat. Ve chvíli, kdy je robot úspěšně zadokovaný a nabíjí se, přechází do stavu `CHARGING`.

Ve stavu `CHARGING` se robot nabíjí. Robot v současném stavu nemá možnsot, jak průběžně monitorovat stav nabití baterie, jelikož BMS obvod neposkytuje žádnou zpětnou vazbu o průběhu nabíjení. Proto před samotným připojením robota ke stanici je hrubě odhadnut nabíjecí čas. Z dat o současném napětí článků baterie pomocí lineární aproximace robot nejprve odhadne zbývající kapacitu v článcích. Následně odhadne i čas nabíjení. počítá přitom s tím, že se po celou dobu bude nabíjet konstantním proudem 2 A. Takto získaný čas je opravdu jen velmi hrubým odhadem celkového času potřebného k nabití, ale v praxi se ukázal jako dostatečný. Nabíjení přitom může končit buď po uplynutí daného času, nebo při zaslání zprávy na téma `trilobot/disconnect_anyway`. V takovém případě se přechází do stavu `CHARGED`.

Ve stavu `CHARGED` robot vyjede z dokovací stanice, bude se přitom pohybovat nejmenší možnou rychlostí pozadu po dobu 1 s. Poté přechází do stavu `FINISHED`.

Ve stavu `FINISHED` již robot pouze resetuje vnitřní proměnné a zašle zprávu o ukončení prioritního pohybu. Uzel poté přechází zpět do stavu `CHECKING_CHARGE_NEED` a celá smyčka se opakuje.

7.3 Architektura firmware pro Arduino Mega

Firmware pro Arduino Mega vycházel z autorovy dřívější práce [4], část kódu je tedy přejatá právě z této práce. Jedná se především o základ pro ovládání motorů, práci se SRF-08 a minIMU-v3. Kód byl ovšem mohutně rozšířen a vylepšen, především byla přidaná podpora rozhraní `rosserial`, která umožňuje firmware komunikovat s frameworkem ROS běžícím na Raspberry Pi.

Nejdříve bude popsaná obecná struktura kódu a poté budou detailněji popsány jednotlivé třídy.

V hlavním souboru `rosserial_arduino.ino` se nachází hlavní řídící funkce kódu, `setup()` a `loop()`. V první jmenované funkci dochází k inicializaci rozhraní `rosserial` a vytvoření ovládacích tříd pro motory, sonary a baterii. Ve funkci `loop()` jsou poté periodicky updatovány jednotlivé komponenty pomocí funkce`update()`. Perioda vykonávání hlavní smyčky přitom může být nastavena pomocí konfigurační konstanty `CYCLE_DURATION`. Kód přitom zaručuje, že smyčka se bude vykonávat nejméně takovou dobu, která je nakonfigurována v této konstantě, není ovšem zaručeno, že se smyčka (například kvůli přílišné zátěži) nebude vykonávat déle. Při testech ovšem bylo časování vždy dodrženo a je tedy pravděpodobné, že Arduino je výpočetně dostatečně výkonné. Perioda hlavní smyčky je 20 ms, což se ukázalo jako ideální kompromis mezi výpočetní náročností a rychlostí publikování dat. Příkazy pro motory mají periodu nastavenu na 100 ms, jelikož nebyl důvod měnit příkazy do motorů rychleji, vzhledem k relativně nízké robotově rychlosti.

Třída `Motor_driver`, která se nachází v souboru `motors.h`, slouží pro ovládání motorů podle údajů o požadované rychlosti z Raspberry Pi a zasílání informací o ujeté vzdálenosti pro účely odometrie. Ujetá vzdálenost se získává pomocí rotačních enkodérů, kde jeden kanál z každého enkodéru je připojen na pin Arduina, který podporuje přerušení. Bohužel, Arduino nemá k dispozici dostatek pinů podporujících přerušení pro připojení obou kanálů obou enkodérů, čímž se snížila přesnost a ztratila informace o směru otáčení kol. Přesnost ovšem zůstala stále dostatečná (zhruba 2 stupně otočení kola na jeden enkodérový pulz), směr otáčení lze zjistit z jiných zdrojů (například současný stav proměnných

`power_l` a `power_r`, které udávají aktuální výkon motorů). Každé volání funkce `update()` poté odpovídá pseudokódu na 1.

Algorithm 1 Pseudokód funkce `update()` pro aktualizaci stavu motorů.

```
Get current ticks from encoders and send them via rosserial
if No cmd_vel update for CMD_VEL_TIMEOUT then
    Set speed for both motors to 0
else
    Compute new power for motors        ▷ Power = driving value for Sabertooth 2X5
    Set the new power
end if
```

Třída `Motor_driver` přitom pouze jen nepřevádí požadovanou rychlost ze zpráv z `rosserial` do vnitřní reprezentace a neovládá řadič motorů Sabertooth, implementuje také ochranu před nečekaným ukončením proudu příkazů z hlavní desky Raspberry Pi (pomocí konfigurační konstanty `CMD_VEL_TIMEOUT`) a také zajišťuje tzv. „soft-start“: výkon pro danou rychlost není nastaven nárazově, ale mění se pozvolna, čímž jsou zlepšeny jízdní vlastnosti a pohonná soustava robota se tolik nenamáhá.

Třída `Sonar_driver` ze souboru `srf08.h` ovládá 6 sonarů SRF-08. Sonary přitom s deskou Arduino komunikují pomocí I^2C sběrnice. Pseudokód funkce `update()` je popsán na 2. Sonary přitom neměří vzdálenost okamžitě, ale potřebují zhruba 60 ms (v kódu kvůli rezervě nastaveno 100 ms) času, než je k dispozici výsledek měření.

Algorithm 2 Pseudokód funkce `update()` pro ovládání sonarů SRF-08.

```
if measuring in progress then
    if current_time ≥ SRF08_MEASURE_TIME then
        Gather results from all sensors and send them via rosserial
    end if
    if current_time ≥ SRF08_UPDATE_INTERVAL then
        Set command to sensors to start measuring
    end if
end if
```

Třída `Battery_driver` (která se nachází v souboru `battery.h`) poté slouží ke sběru dat o baterii a nabíjení. Jedná se tedy především o čtení napětí jednotlivých článků baterie a případnou detekci, zda se Trilobot právě nabíjí. Pseudokód aktualizační funkce `udpate()` je k dispozici na 3.

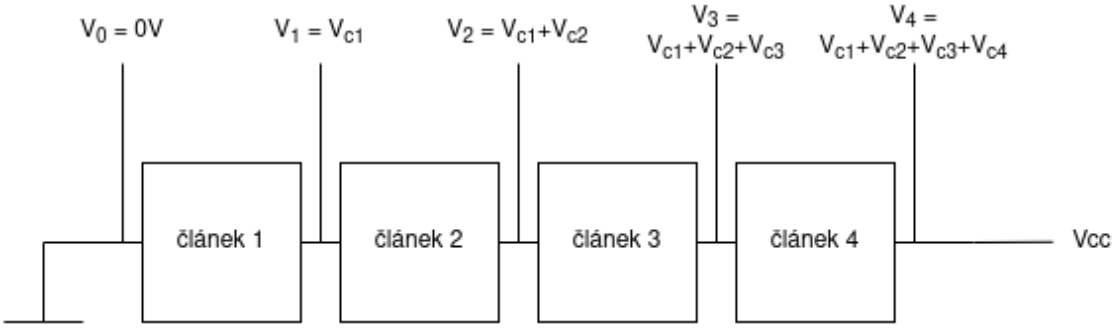
Algorithm 3 Pseudokód funkce `update()` pro sběr dat o baterii.

```
if current_time ≥ BATT_UPDATE_TIME then
    Get current voltages of all cells and send them via rosserial
    if pin PIN_CHARGE_CHECK is HIGH then
        Send true to „trilobot/battery_charging“ topic
    end if
end if
```

Pro získání informace o tom, zda se Trilobot nabíjí, slouží pin specifikovaný konstantou PIN_CHARGE_CHECK. Tento pin přitom měří napětí na děliči napětí mezi nabíjecími elektrodami robota. Dělič napětí omezuje maximální napětí na pinu na 4.5 V při vstupním

nabíjecím napětí 12 V. Zároveň velikost rezistorů (několik set tisíc Ω) zajišťuje minimální proud tekoucí napěťovým děličem a zanedbatelné ztráty energie.

Bylo třeba také vyřešit způsob, jakým měřit napětí jednotlivých článků baterie. Baterie má celkem 4 li-pol články spojené seriově, což odpovídá 14.8 V nominálního napětí. Bohužel nelze měřit napětí každého z článků vůči stejnému nulovému potenciálu. Lze měřit pouze napětí všech článků mezi tím současným a zemí, jak zachycuje ilustrace 7.9. Jelikož napětí jednoho li-pol článku je zhruba 3.7 V, již dva články by překročily maximální povolené napětí na pinu 5 V desky Arduino. Byl proto vytvořen čtyřcestný napěťový dělič, který zajišťuje, že jednotlivá napětí (na ilustraci značena V_1 až V_4) nepřekročí maximální povolené napětí. Z jednotlivých napětí V_1 až V_4 šly již napětí jednotlivých článků spočítat snadno.



Obrázek 7.9: Ilustrace problému měření napětí jednotlivých článků baterie. Napětí V_{cn} je napětí článku n, napětí V_n je napětí měřené v daném místě.

Konverze zpráv z `rosserial` do standardního formátu

Framework ROS obsahuje velké množství standardních zpráv, se kterými jsou jednotlivé balíky schopné pracovat, čímž se dále zlepšuje celková modularita a možnost snadno přidat další balíky. Bohužel, kvůli požadované obecnosti jsou zprávy často poměrně rozsáhlé. Deska Ardino přitom nedisponuje dostatečnými paměťovými, výpočetními ani přenosovými kapacitami pro využívání těchto zpráv. Byly proto vytvořeny vlastní zprávy, které slouží pro výměnu dat mezi deskami Arduino a Raspberry Pi, přičemž na straně Raspberry Pi byly vytvořeny speciální překladové uzly, nazývané *bridges*. Tyto „mosty“ konvertují zprávy ze speciálním zpráv balíku Trilobot do standardních ROS zpráv.

Kapitola 8

Testování

V této kapitole budou popsány výsledky testování algoritmu pro navádění k dokovací stanici. Celý systém přitom explicitně testován nebyl, jelikož je ale jeho chod vyžadován pro správnou funkci nabíjecího algoritmu, lze tvrdit, že základní ověřeno funkčnosti proběhlo. Nejdůležitějšími částmi algoritmu byly navádění k nabíjecí pozici pomocí mapy plánování cesty a dále finální přiblížení. Každá z těchto částí byla testována samostatně a následně byl algoritmus testován jako celek. Každý z těchto tří testů bude popsán v samostatné sekci.

Během první vlny testování přitom ve výpočetním grafu 7.3 nebyl přítomen uzel `laser_scan_matcher` pro zpřesnění odometrie a byl používaný standardní lokální plánovač `base_local_planner`. Nepřesná lokalizace spolu s nevhodným plánovačem způsobovaly nespolehlivé dokování a navádění, kdy robot byl schopen v dobrých podmínkách zadokovat zhruba v 70 procentech případů, úspěšnost ale velmi závisela na konkrétním prostředí. Byla proto zpřesněna odometrie využitím dat ze simulovaného laserového skeneru a použit plánovač `teb_local_planner`, který se více hodí pro diferenciální podvozky. Níže získané výsledky již byly naměřeny s těmito úpravami.

8.1 Test navádění podle mapy

Navádění podle mapy testuje více než samotný kód správné nastavení uzlů `move_base` a `gmapping`. V obou případech se jedná o komplexní uzly s desítkami konfiguračních parametrů, jejichž nastavení může zabrat až desítky hodin, proto je test konečného nastavení velmi důležitý. Test probíhal tak, že robot byl před každým pokusem inicializován před dokovací stanicí a poté přesunut (pomocí dálkového ovládání) do náhodné pozice a orientace v prázdném pokoji 3x3 m. Následně byl vyslán požadavek na okamžité nabití. Prvotní inicializace před dokovací stanicí zajistila, že robot již na začátku získal informaci o poloze dokovací stanice, kteoru si následně udržel i po ztrátě stanice „z dohledu“. Cílem bylo autonomně dosáhnout nabíjecí pozice zhruba 0.5 metru před dokovací stanicí, přičemž robot by měl směřovat k ní. Odchylka v poloze byla stanovena na 0.25 m, úhlová odchylka na 0.1 rad.

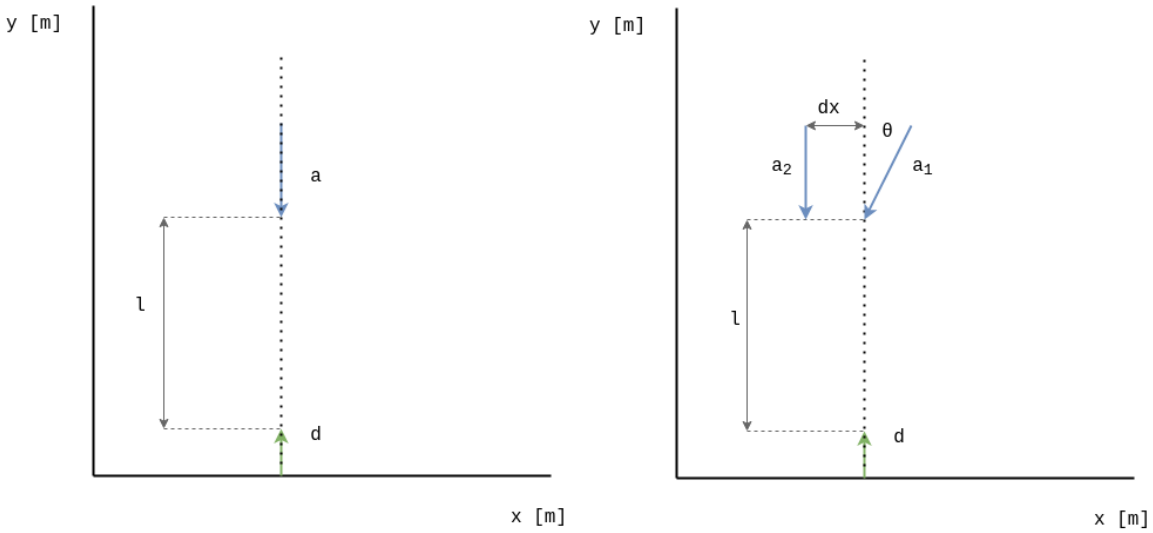
Robot úspěšně dosáhl cíle v 8 případech z 10, ve 2 selhal SLAM algoritmus, kdy nejprve došlo k chybné lokalizaci, což vedlo k chybnému mapování. Tím vznikla zpětnovazební smyčka, kdy robot ztratil informaci o svém okolí a musel být restartován SLAM algoritmus (`gmapping`). V ostatních případech robot úspěšně dosáhl předepsané pozice před dokovací stanicí. Přesnost dosažení cíle byla zpravidla lepší než požadovaná, ve dvou případech se ovšem reálná cílová pozice od ideální lišila téměř přesně hraniční hodnotu uvedenou výše.

8.2 Test finálního přiblížení

Tento test kontroloval, zda je správně implementován algoritmus pro manuální přiblížení, kdy již není možné využít navigační uzel `move_base`. Proběhly celkem dva testy:

1. Test z ideální pozice: vzdálenost $l \in < 0.4, 0.6 > m$ před dokovací stanicí, minimální stranová i úhlová odchylka.

2. Test hraniční, ovšem nikoli ideální pozice: $l \in < 0.3, 0.8 > m$ před dokovací stanicí, úhel $\theta \in < -0.3, 0.3 > rad$ (úhel, který svírá robot s ideální přibližovací trajektorií k dokovací stanici), stranové odchylky cca 0.2 m. Robot přitom dokovací stanici viděl.



Obrázek 8.1: Ilustrace testů finálního přibližování. Vlevo první test, vpravo druhý. Písmeno l znázorňuje vzdálenost robota od stanice, vektor a znázorňuje polohu robota, vektor d polohu stanice. Stranová odchylka je značena jako dy, úhlová odchylka jako θ.

V prvním testu robot zadokoval v 10 případech z 10 bez obtíží. Ve druhém testu robot zadokoval na první pokus v 8 případech z 10. Za neúspěch v obou případech mohla chybná navigace a nepřesnost v navádění, kdy sice robot eliminoval úhlovou odchylku, stranová odchylka ovšem nebyla kompletně opravena, robot zajel mimo naváděcí platformy a nebyl schopen tuto odchylku opravit ani při opakovaných pokusech. Výchozí pozice ovšem byly horší, než jakých bylo zpravidla dosahováno první fází navigace.

8.3 Test celého algoritmu

Celý algoritmus byl testován stejně, jako část s naváděním pomocí mapy. Robot byl tedy inicializován před dokovací stanicí, poté přesunut na náhodnou pozici s náhodnou orientací v místnosti 3x3 m a následně byl vyslán příkaz k nabití. Robot úspěšně zadokoval v 8 případech z 10. V jednom případě došlo opět k chybě ve SLAM algoritmu a ve druhém případě plánovač zvolil nevhodnou trajektorii, která vyústila ve srážku s překážkou. Ve druhém jmenovaném případě sice robot nakonec zadokoval, ale pokus nebyl hodnocen jako úspěšný, jelikož náraz do překážky se nedá považovat za korektní naplánování cesty.

8.4 Zhodnocení výsledků

Výsledky testů ukazují, že algoritmus je principiálně funkční, v některých případech ovšem dochází k nežádoucímu chování. Celkově ovšem lze algoritmus hodnotit jako dostatečný.

Kapitola 9

Závěr a budoucí práce

V průběhu práce byly prostudovány algoritmy pro plánování cesty od těch nejzákladnějších (bug algoritmy) přes ty relativně pokročilé (PRM) až po specializoané přístupy (využití ACO). Zároveň byla provedena rešerše na poli dokovacích stanic a principů navigace včetně tzv. tagů. Dále byla navržena dokovací stanice a nabíjecí systém. Během prvotního testování byly sesbírány údaje pro další vylepšení návrhu této nabíječky, které byly v další iteraci zapracovány. Na straně robota byl navržen systém nabíjení a monitoringu baterie, který byl reálně vytvořen na robotovi Trilobot a testován během vývoje softwaru.

Proběhl také upgrade robotů Trilobot do technického a výkonového stavu, který umožňuje jejich opětovné využití nejen v rámci této práce, ale i během výuky, projektů a podobně. V rámci upgradu byl vylepšen hardware, roboti byli osazeni novými motory, deskami Raspberry Pi a Arduino Mega, sonary SRF-08 a kamerami Intel RealSense. Bylo také navrženo a 3D vytištěno velké množství adaptérů a úchytů pro tyto roboty.

Po stránce softwarové byl robot upgradován na operační systém Ubuntu MATE, ovládací kód je napsán ve frameworku ROS. Kód umožňuje robotovi provádět SLAM, vyhledávání dokovací stanice a zadokování za účelem nabíjení.

Byla vytvořena modulární dokovací stanice, která umožňuje nabíjení robota Trilobot. Důraz byl kladen na jednoduchost designu, snadnou tisknutelnost na 3D tiskárně a možnost výměny jednotlivých komponent z důvodu poškození a/nebo změny designu.

Testování ukázalo, že implementovaný algoritmus plní zadaný cíl (využívání dokovací stanice), byť ne vždy je cíle dosaženo. Celková úspěšnost je ovšem poměrně vysoká.

Co se týče možného využití, robot je funkčním celkem, který by mohl být využit například v projektech, či jako demonstrační robot.

Robot dále nabízí i mnoho námětů k budoucí práci. Zajímavou výzvou by bylo využít sonary SRF-08 jako dodatečný zdroj informací o okolí, případně k enkodérové odometrii přidat údaje z IMU jednotek. Po hardwarové stránce by bylo vhodné doplnit Raspberry Pi a případně i kameru Intel RealSense aktivním chlazením. Jelikož existuje sedm identických robotů (někteří ještě v posledním stadiu dokončování), je možné zapracovat i na komunikaci těchto robotů a například se pokusit o multirobotové mapování rozsáhlejších oblastí.

Seznam obrázků

Literatura

[1] Bateman, T. *France approves fully autonomous bus for driving on public roads in a European first* [online]. euronews.com [cit. 2022-01-01]. Dostupné z: https://www.euronews.com/next/2021/12/01/france-approves-fully-autonomous-bus-for-driving-on-public-roads-in-a-european-first.

[2] Bencina, R., Kaltenbrunner, M. a Jordà, S. Improved Topological Fiducial Tracking in the reacTIVision System. In:. červenec 2005, s. 99–99. DOI: 10.1109/CVPR.2005.475. ISBN 0-7695-2372-2.

[3] Bencina, R. a Kaltenbrunner, M. The Design and Evolution of Fiducials for the reacTIVision System. In: *Proc. 3rd International Conference on Generative Systems in the Electronic Arts*. 2005.

[4] Beran, J. *Výukový tutorial pro použití Arduina v robotice*. Brno, CZ, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/22824/.

[5] Bergamasco, F., Albarelli, A., Rodola, E. a Torsello, A. RUNE-Tag: A high accuracy fiducial marker with strong occlusion resilience. In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. June 2011, s. 113–120. DOI: 10.1109/CVPR.2011.5995544. ISSN 1063-6919.

[6] Brand, M., Masuda, M., Wehner, N. a Yu, X.-H. Ant Colony Optimization algorithm for robot path planning. In: *2010 International Conference On Computer Design and Applications*. 2010, sv. 3, s. V3–436–V3–440. DOI: 10.1109/ICCDA.2010.5541300.

[7] BringAuto. *Last Mile Delivery Robot* [online]. BringAuto, 2022 [cit. 2022-05-12]. Dostupné z: https://bringauto.com/en/.

[8] Buchmann, I. a Inc, C. E. *Batteries in a Portable World: A Handbook on Rechargeable Batteries for Non-engineers*. Cadex Electronics, 2001. ISBN 9780968211823. Dostupné z: https://books.google.cz/books?id=YIBhAAAACAAJ.

[9] Cassinis, R., Tampalini, F., Bartolini, P. a Fedrigotti, R. Docking and charging system for autonomous mobile robots. Leden 2005.

[10] Choset, H. Coverage of Known Spaces: The Boustrophedon Cellular Decomposition. *Autonomous Robots*. Prosinec 2000, sv. 9, s. 247–253. DOI: https://doi.org/10.1023/A:1008958800904.

[11] Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W. et al. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, květen 2005. ISBN 0262033275.

[12] Dattalo, A. *What is ROS?* [online]. ROS [cit. 2022-01-01]. Dostupné z: http://wiki.ros.org/ROS/Introduction.

[13] Digiacomo, F., Bologna, F., Inglese, F., Stefanini, C. a Milazzo, M. MechaTag: A Mechanical Fiducial Marker and the Detection Algorithm. *Journal of Intelligent Robotic Systems*. Listopad 2021, sv. 103. DOI: 10.1007/s10846-021-01507-x.

[14] Fiala, M. ARTag, a fiducial marker system using digital techniques. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005, sv. 2, s. 590–596 vol. 2. DOI: 10.1109/CVPR.2005.74.

[15] Guangrui, F. a Geng, W. Vision-based autonomous docking and re-charging system for mobile robot in warehouse environment. In: *2017 2nd International Conference on Robotics and Automation Engineering (ICRAE)*. 2017, s. 79–83. DOI: 10.1109/ICRAE.2017.8291357.

[16] International, S. *Electrolux Trilobite* [online]. Wikipedia, 2021 [cit. 2021-12-25]. Dostupné z: https://en.wikipedia.org/wiki/Electrolux_Trilobite.

[17] International, S. *SAE Levels of Driving Automation* [online]. SAE International, 2021 [cit. 2021-12-25]. Dostupné z: https://www.sae.org/blog/sae-j3016-update.

[18] Kim, K. H., Choi, H., Yoon, S., Lee, K., Ryu, H. et al. Development of docking system for mobile robots using cheap infrared sensors. Prosinec 2021.

[19] Krogius, M., Haggenmiller, A. a Olson, E. Flexible Layouts for Fiducial Tags. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. October 2019.

[20] Logiwa. *Warehouse Robotics* [online]. Logiwa, 2021 [cit. 2021-12-25]. Dostupné z: https://www.logiwa.com/blog/warehouse-robotics.

[21] Looser, J. *ARToolKit Documentation* [online]. ARToolKit [cit. 2021-12-30]. Dostupné z: http://www.hitl.washington.edu/artoolkit/documentation/index.html.

[22] Olson, E. AprilTag: A robust and flexible visual fiducial system. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, s. 3400–3407. DOI: 10.1109/ICRA.2011.5979561.

[23] Quilez, R., Zeeman, A., Mitton, N. a Vandaele, J. Docking autonomous robots in passive docks with Infrared sensors and QR codes. Červen 2015. DOI: 10.4108/icst.tridentcom.2015.259673.

[24] Rekimoto, J. a Ayatsuka, Y. CyberCode: Designing Augmented Reality Environments with Visual Tags. In:. New York, NY, USA: Association for Computing Machinery, 2000, s. 1–10. DARE '00. DOI: 10.1145/354666.354667. ISBN 9781450373265. Dostupné z: https://www2.sonycsl.co.jp/person/rekimoto/papers/dare2000.pdf.

[25] Roesmann, C., Feiten, W., Woesch, T., Hoffmann, F. a Bertram, T. Trajectory modification considering dynamic constraints of autonomous robots. In: *ROBOTIK 2012; 7th German Conference on Robotics*. 2012, s. 1–6.

[26] Romanov, A. M. a Tararin, A. A. An Automatic Docking System for Wheeled Mobile Robots. In: *2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*. 2021, s. 1040–1045. DOI: 10.1109/ElConRus51938.2021.9396509.

[27] Silverman, M., Jung, B., Nies, D. a Sukhatme, G. Staying alive longer: Autonomous robot recharging put to the test. Prosinec 2021.

[28] Silverman, M., Nies, D., Jung, B. a Sukhatme, G. Staying Alive: A Docking Station for Autonomous Robot Recharging. In:. Květen 2002, sv. 1, s. 1050–1055. DOI: 10.1109/ROBOT.2002.1013494.

[29] skitterbot. *Grey Walter's tortoises* [online]. YouTube, 2008 [cit. 2021-12-29]. Dostupné z: `https://www.youtube.com/watch?v=lLULRlmXkKo`.

[30] Stephan, J. *Differential-drive* [online]. Jon Stephan, 2015 [cit. 2022-05-14]. Dostupné z: `https://github.com/jfstepha/differential-drive`.

[31] Walter, G. *The Living Brain*. W. W. Norton, 1953.

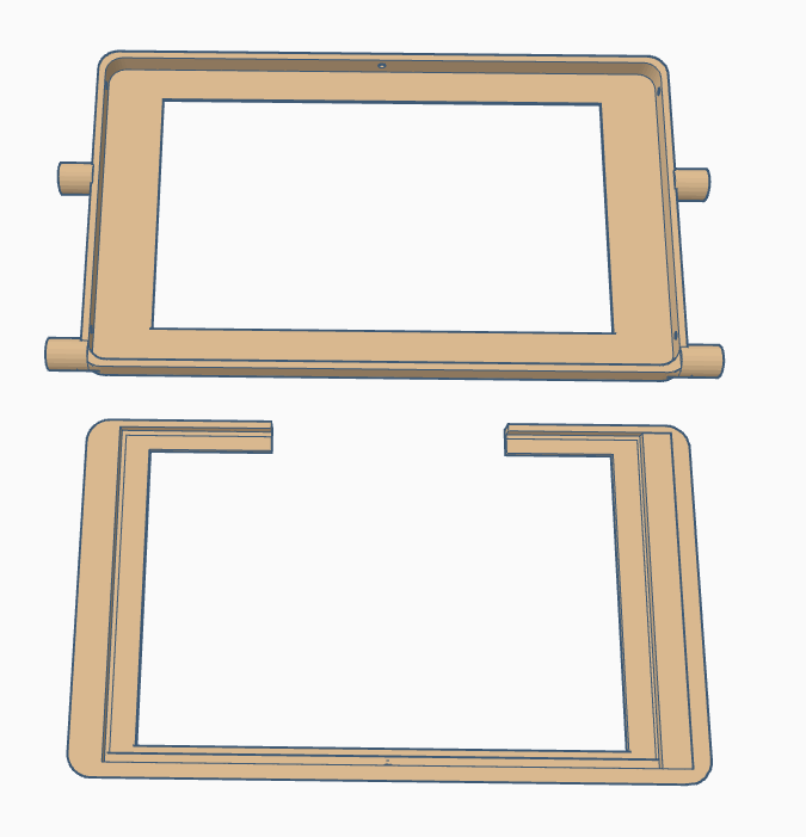
[32] Wang, J. a Olson, E. AprilTag 2: Efficient and robust fiducial detection. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, s. 4193–4198. DOI: 10.1109/IROS.2016.7759617.

[33] Wu, Y.-C., Teng, M.-C. a Tsai, Y.-J. Robot docking station for automatic battery exchanging and charging. In: *2008 IEEE International Conference on Robotics and Biomimetics*. 2009, s. 1043–1046. DOI: 10.1109/ROBIO.2009.4913144.

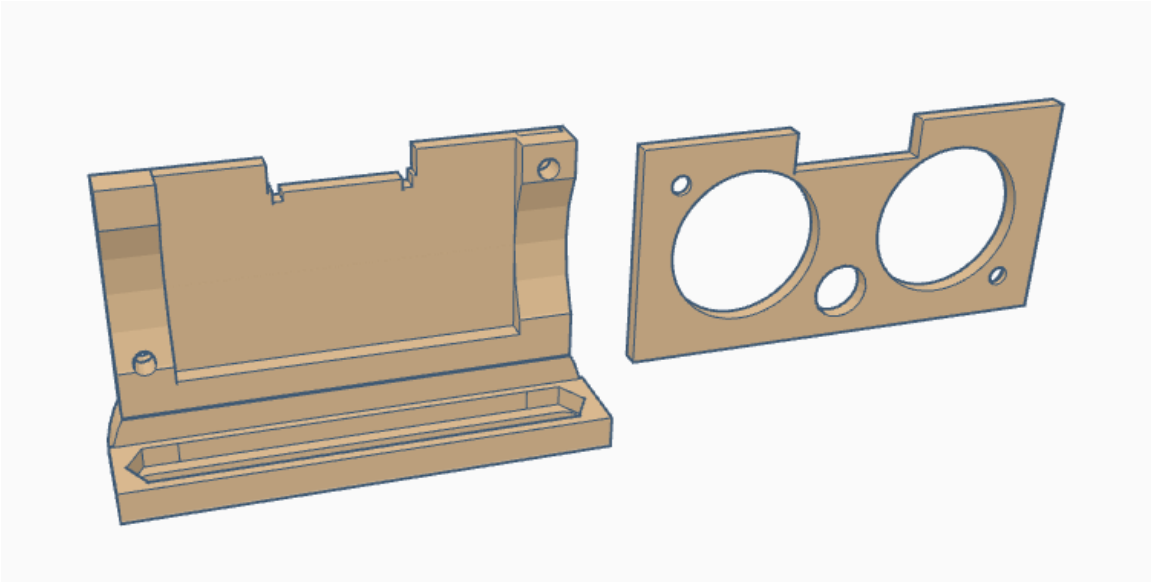
Příloha A

Modely 3D tištěných komponent

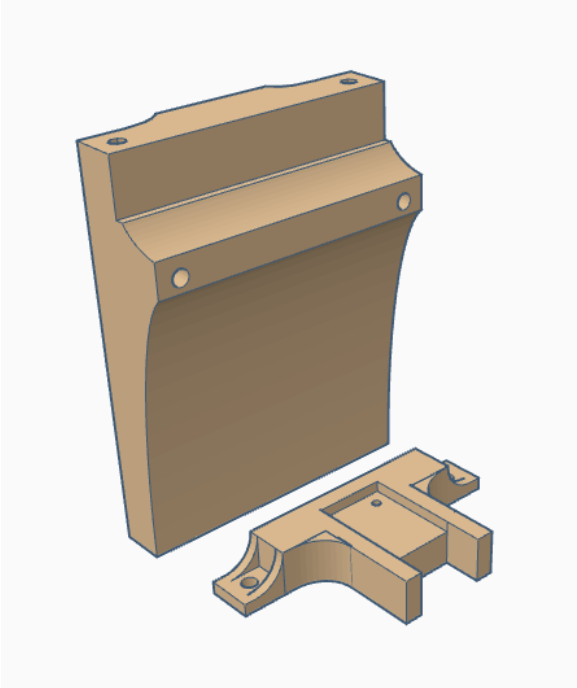
V této příloze budou ukázány 3D tištěné díly na robota v dostatečné velikosti. Velikosti přitom neodpovídají realitě, stejně tak nesouhlasí poměry mezi jednotlivými Obrázky.



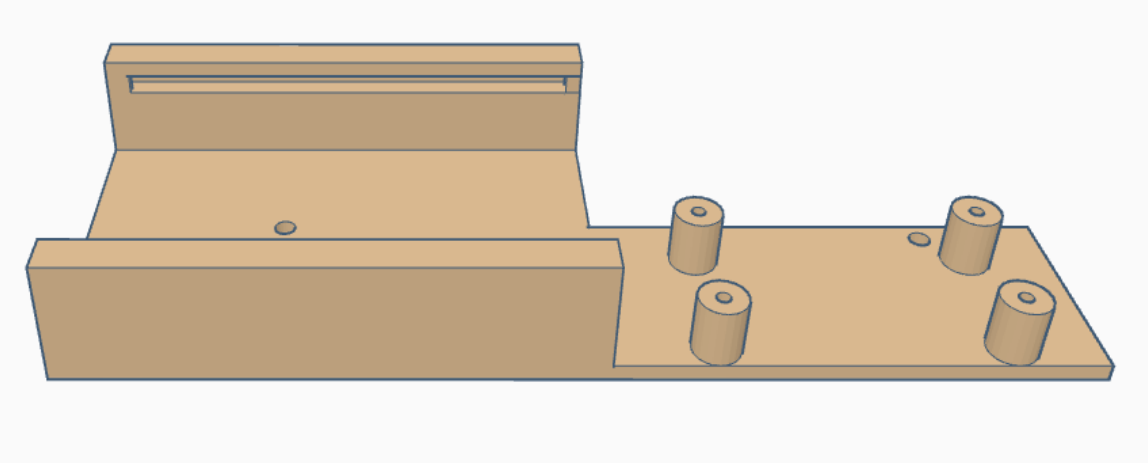
Obrázek A.1: Adaptér na Raspberry Pi a dotykový displej. Oba díly do sebe zapadají, pro pevnější spojení je možné je sešroubovat. Velikost byla navržena tak, aby šel použít místo zadního krytu původního Trilobota.



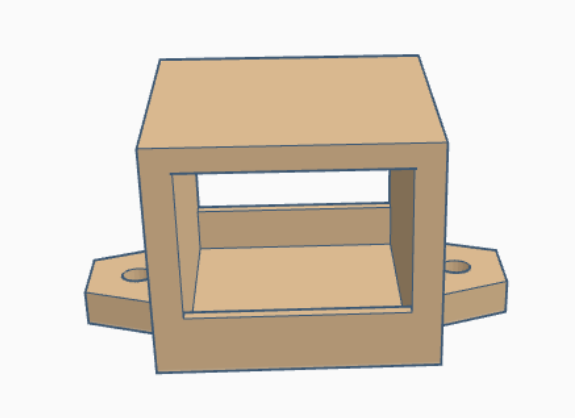
Obrázek A.2: Adaptér na senzory vzdálenosti SRF-08. Lze přitom použít bez části vpravo, která má chránit samotný tištěný spoj. Jelikož byly otvory pro uchycení adaptérů vrtány ručně s poměrně velkou tolerancí vůči nepřesnostem, umožňuje sonar variabilní rozestup montážních otvorů (místo dvou otvorů pro šrouby je v dolní části podlouhlý otvor pro libovolné umístění šroubů).



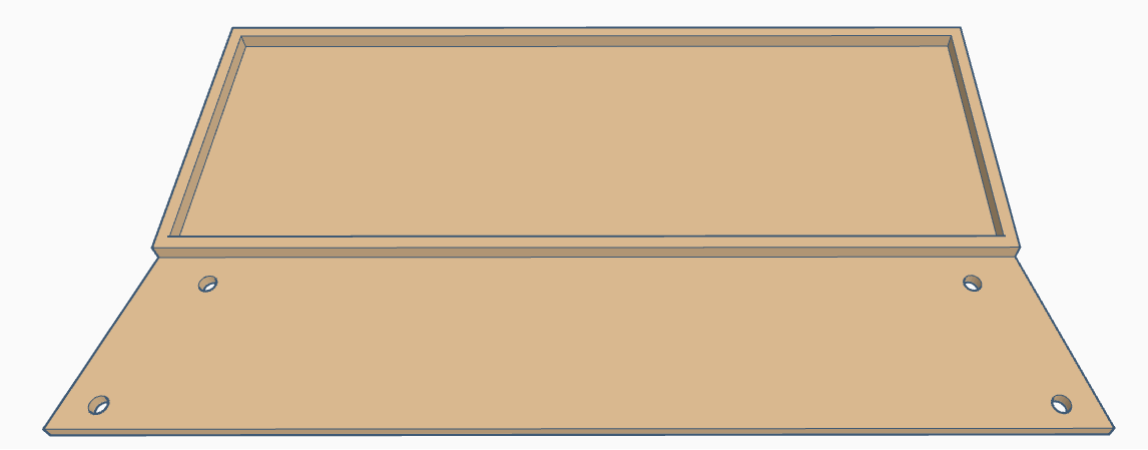
Obrázek A.3: Montáž pro kameru Intel RealSense a minIMU.



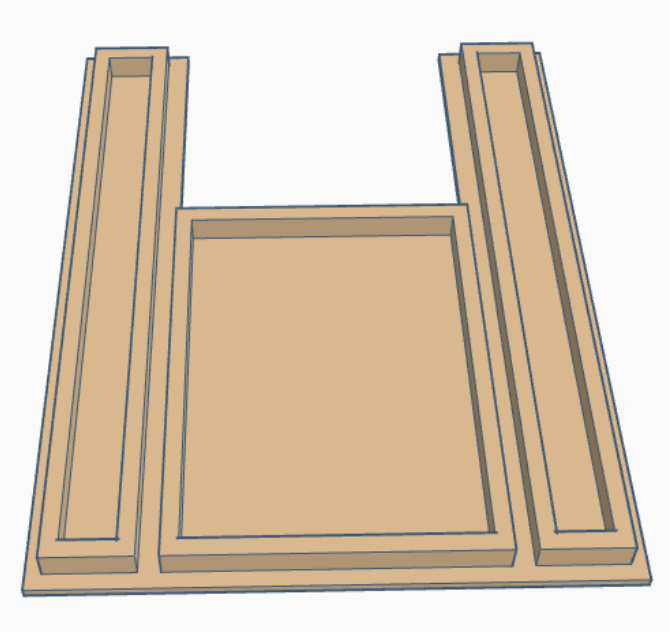
Obrázek A.4: Adaptér na BMS obvod 4S 40A a 5V stabilizátor/konvertor.



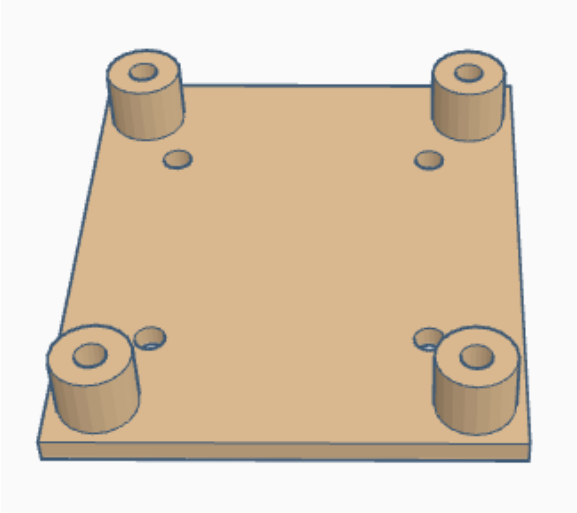
Obrázek A.5: Krytka spínače pro zapnutí robota.



Obrázek A.6: Platforma pro baterii a zakrytí předního otvoru Trilobota.



Obrázek A.7: Adaptér na nepájivá pole.



Obrázek A.8: Adaptér na desku Sabertooth 2x5.

Příloha B

Struktura přiložené SD karty

Struktura přiložené SD karty je následující:

- `Text`: V této složce se nachází technická zpráva jak ve formátu pdf, tak jako zdrojové soubory.
- `Code`: V této složce se nachází kód pro robota Trilobot.
  - `rosserial-trilobot`: V této složce se nachází firmware pro desku Arduino.
  - `trilobot`: V této složce se nachází ROS balík.
- `stl`: V této složce se nachází stl modely všech použitých komponent, které byly následně tištěny na 3D tiskárně.