VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

SAMOŘÍZENÍ MODELU AUTA V NEZNÁMÉM PROSTŘEDÍ POMOCÍ SLAM

SELF DRIVING OF CAR MODEL IN UNKNOWN ENVIRONMENT USING SLAM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **FILIP JAHN**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JOSEF STRNADEL, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

142949

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Jahn Filip**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Samořízení modelu auta v neznámém prostředí pomocí SLAM**
Kategorie:	Vestavěné systémy
Akademický rok:	2022/23

Zadání:

1. Proveďte rešerši technik a prostředků SLAM (Souběžná lokalizace a mapování, angl. Simultaneous Localization And Mapping) vč. jejich případných vylepšení, např. aktivní SLAM (ASLAM).
2. Seznamte se s konstrukcí a výbavou modelu auta na podvozku DFRobot ROB0170 (NXP Cup Race Car). Proveďte rešerši prostředků (řídicí deska, čidla, akční členy, operační systém aj.) a metod řízení modelů založených na tomto podvozku, obzvláště těch ze soutěže NXP CUP.
3. Vybavte model z bodu 2 základními prostředky řízení pohybu. Naimplementujte programovou vrstvu umožňující ovládat pohyb (vpřed, vzad, vlevo, vpravo) modelu.
4. Navrhněte rozšíření modelu z bodu 3 o prostředky umožňující implementaci technik SLAM. Rozšíření realizujte a implementujte nad ním zvolenou množinu technik SLAM.
5. Zvolte sadu min. tří úloh samořízení realizovaného modelu s hledáním řešení na bázi výstupů zvolených SLAM technik. Proces hledání řešení úloh algoritmizujte a implementujte. Schopnost modelu samočinně řešit úlohy experimentálně ověřte.
6. Zhodnoťte vlastnosti realizovaných přístupů k samočinnému řešení řídicích úloh pomocí SLAM, diskutujte související chyby/nejistoty a vlivy, nastiňte možné směry pokračování v projektu.

Literatura:

- Lluvia I., Lazkano E., Ansuategi A.: Active Mapping and Robot Exploration: A Survey. *Sensors*. 2021, Vol. 21, No. 7, s. 2445:1-26. DOI 10.3390/s21072445.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání,
- splnění první části bodu 4 zadání, představení možných přístupů k implementaci technik SLAM a k samořízení modelu pomocí výstupů SLAM.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Strnadel Josef, Ing., Ph.D.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	31.10.2022

Abstrakt

Tato práce si klade za cíl sestavit model vozidla, které bude schopno autonomně se pohybovat v prostředí a zároveň bude mapovat své okolí. Dalším cílem práce bylo podrobněji porozumět vestavným systémům a jejich vývoji, a proto bylo programování záměrně realizováno na hardwarové úrovni (bare metal) bez využití operačního systému, či jiných, již existujících řešení. Z technik SLAM byla vybrána grid-based metoda, která využívá mřížku (grid) jako základní prostorovou reprezentaci prostředí. V rámci této metody jsou senzory využívány k měření vzdálenosti a určování polohy robota v daném prostoru. Tyto informace jsou následně zpracovány a použity k vytvoření mapy prostředí, kterou robot využívá ke své orientaci a pohybu v daném prostoru. Po projetí předem neznámé dráhy robot sestaví mapu prostoru a uloží ji jako excelový soubor na SD kartu, aby byla mapa jednoduše čitelná. Přínosem této práce je podrobné popsání jednotlivých použitých součástek. Práce byla od začátku psána tak, aby byly jednotlivé moduly samostatně funkční. Tím vznikly knihovny, které když se vloží do projektu, budou plně funkční.

Abstract

This thesis aims to build a model of a vehicle that will be able to autonomously navigate in the environment while mapping its surroundings. Another goal of the work was to understand embedded systems and their development in more detail, and therefore the programming was deliberately implemented at the hardware level (bare metal) without the use of an operating system or other existing solutions. From the SLAM techniques, a grid-based method was chosen, which uses a grid as the basic spatial representation of the environment. In this method, sensors are used to measure the distance and determine the position of the robot in a given space. This information is then processed and used to create a map of the environment, which the robot uses to orient and move through the space. After traversing a previously unknown path, the robot builds a map of the space and saves it as an excel file on an SD card to make the map easy to read. The contribution of this thesis is the detailed description of each component used. The work was written from the beginning so that the individual modules are independently functional. This created libraries that when inserted into the project will be fully functional.

Klíčová slova

SLAM, model autonomního vozidla, grid-based SLAM, autonomní řízení, mapování okolí robota

Keywords

SLAM, autonomous vehicle model, grid-based SLAM, autonomous driving, robot environment mapping

Citace

JAHN, Filip. *Samořízení modelu auta v neznámém prostředí pomocí SLAM*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Josef Strnadel, Ph.D.

Samořízení modelu auta v neznámém prostředí pomocí SLAM

Prohlášení

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

.....
Filip Jahn
8. května 2023

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Josefu Strnadelovi, Ph. D. za zodpovězení mých nesčetných otázek, které mi pomohly pochopit principy použité v této práci a za ochotu pomoci s mnoha problémy, se kterými jsem se setkal.

Obsah

1 Úvod **3**

2 Rešerše problematiky **4**
2.1 SLAM . . . 5
2.1.1 Životní cyklus robota . . . 5
2.1.2 Metody SLAM . . . 6
2.2 Související oblasti . . . 7
2.2.1 Rozdělení robotů . . . 7
2.2.2 Kinematika . . . 8
2.2.3 Odometrie . . . 11
2.2.4 Vestavný systém . . . 13
2.2.5 Měření . . . 14

3 Rešerše zvolených komponent **17**
3.1 Konstrukce . . . 17
3.2 Elektřina, rozvody . . . 18
3.3 Řídicí jednotky . . . 19
3.3.1 Motory . . . 19
3.3.2 Servo . . . 22
3.4 Rešerše senzorů . . . 23
3.4.1 Senzory pro detekci jízdního pruhu . . . 23
3.4.2 Senzory pro mapování . . . 25
3.5 SD slot . . . 30
3.6 Arduino . . . 32
3.7 Výsledná podoba robota . . . 33

4 Popis realizace **34**
4.1 Inicializace . . . 34
4.2 Hlavní programová smyčka . . . 36
4.3 Pohyb robota . . . 38
4.3.1 Výpočet ujeté vzdálenosti . . . 38
4.3.2 Výpočet polohy robota . . . 38
4.4 Mapování . . . 40
4.4.1 Reprezentace mapy . . . 40
4.4.2 Algoritmy mapování pohybu . . . 41
4.4.3 Pohyb v bloku . . . 42
4.4.4 Přemístění do jiného bloku . . . 43
4.4.5 Mapování okolního prostředí . . . 44

4.4.6 Ukládání mapy 46
4.4.7 Rovnice výpočtu nastavitelných parametrů mapy vzhledem k místu v paměti 48
4.4.8 Návrhy na zlepšení algoritmů 49

5 Experimentální část **50**
5.1 1. úkol 50
5.2 2. úkol 53
5.3 3. úkol 54

6 Závěr **56**

Literatura **57**

A Manual **60**

Kapitola 1

Úvod

Tato bakalářská práce se zabývá jedním z nejvíce vzrušujících témat současnosti - autonomními vozidly. Tato revoluční technologie představuje klíč k bezpečnějšímu a efektivnějšímu provozu na silnicích a nabízí nám nekonečné možnosti, jak zlepšit naši každodenní mobilitu.

Autonomní vozidla představují revoluční technologii, která má potenciál výrazně zlepšit bezpečnost silničního provozu. Díky pokročilým senzorům a algoritmům jsou tato vozidla schopna rychle reagovat na změny v okolním prostředí a přizpůsobit se jim. Tím eliminují mnoho lidských chyb, které jsou běžnou příčinou dopravních nehod. Výsledkem je snížení počtu dopravních nehod a zranění, či úmrtí na silnicích. Autonomní vozidla nabízejí rovněž větší efektivitu a pohodlí pro cestující, kteří mohou během cesty využívat čas například k práci nebo odpočinku. Navíc se autonomní vozidla mohou stát základem pro vytvoření nových mobility služeb a transformovat způsob, jakým se lidé pohybují v městských oblastech. Díky tomu se autonomní vozidla stávají nejen převratnou technologií, ale i klíčem k lepšímu a bezpečnějšímu budoucímu silničnímu provozu.

Jedním z klíčových aspektů autonomního řízení je schopnost vozidla „vidět“ a „rozumět“ okolnímu prostředí. To je zajištěno využitím metod jako je SLAM (simultánní lokalizace a mapování), dynamickému mapování a senzorům. Metody umožňují vozidlům „rozpoznat“ okolí, vytvořit co nejpřesnější mapu a navigovat se bezpečně v reálném čase.

V práci se zaměříme na hloubkové senzory jako zdroj dat pro SLAM a dynamické mapování, a na jejich využití k vytvoření přesného modelu okolí vozidla. Dále se budeme věnovat problematikám jako je měření, kinematika či odometrie, které nám umožní pochopit principy, které jsou nezbytné pro autonomní vozidlo.

Cílem této práce je nejen poskytnout přehled o metodách SLAM, přístupů a technologií. Ale sestavit reálný model vozidla, vybavit ho senzory a implementovat autonomní řídicí algoritmus, který vozidlo (robota) bude provádět předem neznámým prostředím. V implementaci je kladen důraz na dynamické mapování, které je šetrné k místu v paměti, což by v budoucnu mohlo vést k levnějším komponentám ve vozidlech a tím snížit výslednou cenu automobilu.

Kapitola 2

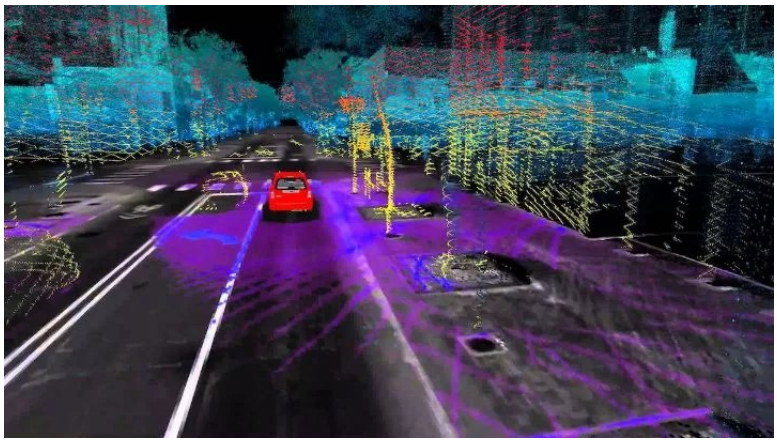
Rešerše problematiky

Simultánní lokalizace a mapování (*SLAM*) je technologie pro autonomní roboty, která jim umožňuje poznávat a mapovat okolní prostředí v reálném čase. Tato technologie se často používá v mobilních robotech, dronech a bezpilotních vozidlech, aby se mohly samostatně navigovat v prostředí, ve kterém se pohybují. [15]

SLAM sestává ze dvou hlavních komponent:

- **Mapování:** Robot vytváří mapu okolního prostředí. Mapa může obsahovat informace o geometrii prostoru, překážkách a dalších objektech.
- **Lokalizace:** Robot umisťuje sebe sama do mapy. To mu pak umožňuje pohyb a navigování.

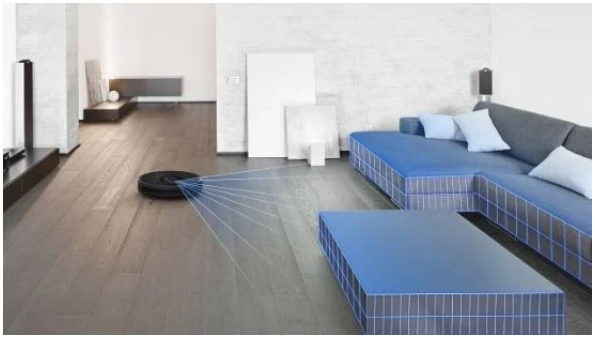
SLAM 2.1 je složitý proces, který vyžaduje použití různých senzorů a algoritmů pro zpracování dat, výpočtů polohy a korekci mapy. Jeho účelem je umožnit autonomním robotům pohybovat se po prostředí bez předem známé cesty, nebo bez nutnosti externího navádění.



Obrázek 2.1: Ukázka mapování užitím SLAM [15]

Jedním z největších problémů je schopnost spolehlivě se lokalizovat v prostředí a přesně mapovat okolní objekty. To může být komplikováno řadou faktorů, jako je například omezená kvalita dat ze senzorů, různé pohyblivé překážky nebo rušení, které mohou zkreslovat výsledky. Proces může být časově náročný a může vyžadovat vysoký výpočetní výkon, což může být úskalí pro některé typy robotů s omezenými zdroji. Kupříkladu takový domácí vysavač 2.2 nebude disponovat takovým výpočetním výkonem jako autonomní vozidlo.

Kromě toho je třeba vzít v úvahu i bezpečnost a spolehlivost takovýchto robotů. Některé chyby nebo nedostatky mohou vést k nebezpečným situacím nebo dokonce k poškození robotů či okolního prostředí. Je důležité, aby byly použity spolehlivé senzory a algoritmy pro zajištění přesnosti a spolehlivosti výsledků.



Obrázek 2.2: Domácí chytrý vysavač mapující obývací pokoj [33]

2.1 SLAM

Následující kapitola více představí různé metody SLAM - jak robot dokáže samostatně plnit zadané úkoly. Jak se zorientuje v prostředí, ve kterém nikdy předtím nebyl a jaké techniky se pro to využívají. Následující sekce popíší postupný obecný proces životního cyklu robota - od jeho inicializace, přes mapování, až po splnění úkolu. Závěr této kapitoly pojednává o možnostech, čím lze robota vybavit a co všechno je k tomu zapotřebí.

2.1.1 Životní cyklus robota

Počáteční inicializace

Autonomní robot začíná tím, že se zavádí do prostředí, ve kterém bude pracovat. Tento proces se nazývá *začlenění* nebo-li *inicializace* a zahrnuje několik kroků. Prvním je nastavení robotického systému. Před zahájením práce je třeba nakonfigurovat hardware a software robotického systému tak, aby byl připraven k práci. To zahrnuje například připojení senzorů a dalšího vybavení, nastavení komunikačních kanálů a nastavení kontrolních algoritmů. V některých případech může být vhodné získat předběžnou mapu prostředí, ve kterém bude robot pracovat. Tato mapa může být získána pomocí skenování okolí, nebo předem namapovaných dat. Následuje inicializace procesu SLAM. Ten se spouští, aby robot mohl začít poznávat a mapovat okolní prostředí pro počáteční inicializaci v prostoru. Dále je zapotřebí nastavit robotovi úkol, který bude plnit, nebo nějaký cíl – to může zahrnovat například navigaci k určitému místu, shromažďování dat nebo vykonávání určitých úkonů. Po nastavení cíle nebo úkolu se robot přepne do autonomního režimu a začne samostatně plnit stanovený úkol. Tento proces zahrnuje použití různých algoritmů pro plánování trasy, rozpoznávání překážek a dalších činností.

Autonomní režim

Ve smyčce se pak provádí typické operace, které mají za cíl splnění zadaných podmínek.

Přijímání a zpracovávání dat ze senzorů. Robot na začátku cyklu musí vždy pracovat s aktuálnímy daty a proto přijímá data ze senzorů (čímž mohou být kamery,

LIDAR, radary nebo např. ultrazvukové senzory), která používá k poznávání a mapování. Tato data jsou zpracovávána pomocí algoritmů, aby se získaly informace o okolním prostředí a aktualizovala se mapa.

Plánování trasy. Po aktualizaci mapy je zapotřebí zanést sebe sama do prostoru a aktualizovat plán trasy. Tento plán zahrnuje překonání překážek, vyhýbání se nebezpečným místům a navigaci k cíli.

Ovládání pohybu. Robot používá kontrolní algoritmy k řízení pohybu podle plánu trasy. To může zahrnovat ovládání pohonných jednotek, řízení směru a rychlosti nebo ovládání dalších částí robota, jako jsou ramena nebo koncové manipulátory. V našem případě se jedná o kontrolu servomotoru pro zatáčení přední nápravy a řízení pohonných jednotek. O tom ale více v této kapitole 2.2.2.

Kontrola a řízení procesu. Různé algoritmy pro kontrolu a řízení průběhu úkolu. Například využití algoritmů pro vyhodnocení stavu a stanovení dalšího postupu.

Výstup a komunikace. Robot může také používat různé výstupní zařízení, jako jsou zvukové a světelné signály nebo tiskové zprávy, pro komunikaci se svým okolím nebo pro oznámení stavu nebo výsledků úkolu.

Existuje několik různých metod pro SLAM, které robot využívá ke zpracování své polohy a okolního prostoru. V práci jsou popsány pouze ty nejběžnější a nejpoužívanější.

2.1.2 Metody SLAM

Monte Carlo

Jedna z nejběžnějších metod založena na simulaci náhodných událostí k výpočtu pravděpodobnostního rozložení možných stavů robota a mapy [13]. Generují se náhodné odhady polohy a mapy, a pak se pro každý odhad vyhodnocuje, jak dobře se shoduje s měřeními, která byla získána. Tyto odhady se pak používají k vytvoření pravděpodobnostního rozložení možných stavů robota a mapy. V průběhu času se tyto odhady aktualizují na základě nových měření, čímž se zlepšuje přesnost odhadu stavu robota a mapy, nebo-li pozice robota v mapě.

Metoda Monte Carlo se často používá v robotice, protože umožňuje robotovi pracovat s neustále se měnícími a neznámými chybami v měřeních a také umožňuje kombinovat různé typy měření, jako jsou například odometry nebo snímání laserových snímačů. Metoda je výkonná a robustní, ale má několik nedostatků, jako je vysoká náročnost na výpočetní výkon a potřeba velkého množství dat pro korektní a deterministické určení polohy.

Metodou se ale řeší například globální lokalizace, či problém únosu[1] [17].

Kalmanův filtr

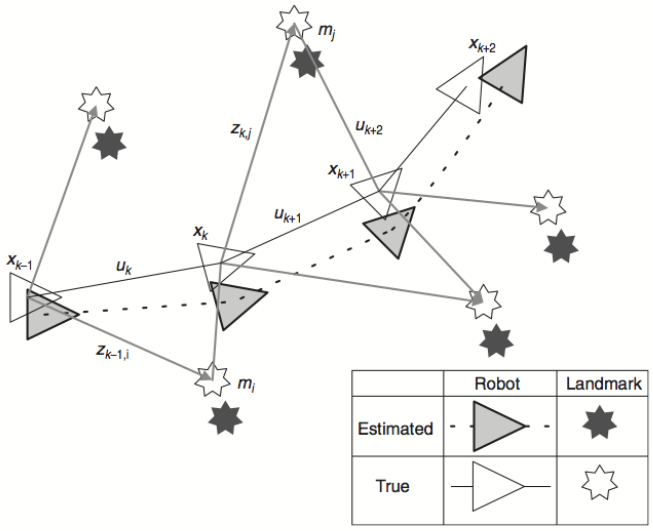
Princip fungování Kalmanova filtru je založen na Bayesově principu[2].

Kalmanův filtr se skládá ze dvou kroků: predikce stavu a aktualizace stavu. Názorná ukázka na obrázku 2.3. V prvním kroku používá filtr model chyby k určení, jak se stav systému od posledního měření změnil. Tuto informaci pak používá k určení, jak by se stav systému měl vyvíjet do budoucna. Ve druhém kroku pak filtr používá nové měření k

[1] Problém únosu je metoda ověřování schopnosti robotu se znovu lokalizovat po závažnější chybě. Pro ověření přemístíme robota z lokalizovaného místa do nového prostoru, který ještě nemá zmapovaný a sledujeme, zda-li se dokáže z této změny vzpamatovat.

[2] Bayesův princip se používá k určení pravděpodobnosti, že se určitá událost stane na základě pravděpodobností jiných událostí, které již nastaly.

aktualizaci svého odhadu stavu. Měření se kombinuje s předpovědí stavu z predikce stavu pomocí Bayesova principu, aby se určil nejlepší odhad stavu systému. Kalmanův filtr tedy pracuje s neustálým kombinováním modelu chyby a měření, které pomáhají při vytváření nejlepšího odhadu stavu systému v daném okamžiku. Je velmi flexibilní a může být upraven pro různé typy systémů a různé typy chyb. Výhoda je, že v každé iteraci pracuje pouze s posledním odhadem a nejnovějším řešení, díky čemuž má nízké nároky na paměťový prostor.



Obrázek 2.3: Predikce stavu je označena šedým trojúhelníkem, reálná poloha je znázorněna bílým trojúhelníkem. Odchylka se v čase bez použití korelace čím dál tím zvětšuje. [31]

Rozšířený Kalmanův filtr (*EKF*)

Jedná se metodu pro řešení problémů soustav, které jsou řízeny nelineárními rovnicemi stavu a měřeními, která jsou taktéž nelineární. EKF je variantou Kalmanova filtru, který pracuje s lineárními modely a měřeními. Tato metoda, rovněž jako její předchůdkyně, pracuje ve dvou krocích: predikce stavu a aktualizace stavu. Při predikci se používá nelineární model stavu k určení, jak se stav systému od posledního měření změnil. Tuto předpověď pak používá k aktualizaci odhadu stavu pomocí nového měření pomocí metody EKF.

2.2 Související oblasti

Následující sekce pojednává o praktičtějších věcech. Od přehledu robotů - pro definici jaké jsou možnosti a co se ve světě využívá - po výpočet pohybu v prostoru na základě vybraného robota. To navazuje na předešlou kapitolu, protože u robotů s různými pohyby se vypočítává pohyb jiným způsobem. A posledně popis měření a systém robota, který ho řídí.

2.2.1 Rozdělení robotů

Robotů je dnes celá řada a tak je můžeme řadit do různých kategorií. Na obrázku 2.4 je ukázka široké škály využití robotů v dnešní době. Aby bylo možné správně propočítat pohyb robota, bude třeba si nejprve definovat, s jakým robotem se pracuje. Technologie SLAM je určena pro autonomní roboty, jak již bylo zmíněno dříve 2. Mobilní koloví roboti

mohou být klasifikováni do několika kategorií podle své konstrukce, schopností nebo účelu. Dělení může být například dle těchto kritérií:

- Pohyb: Roboti mohou být buďto stacionární, nebo mobilní. Stacionární bývají pouze na jednom místě bez možnosti pohybu. Naopak mobilní roboti jsou využíváni pro pohyb v nějakém prostoru.
- Způsob pohybu: Kola, pásy, nohy a jiné.
- Použití: Roboti navrženi k různým účelům. Průzkumy, průmysl, domácnost, vojenské účely a mnoho dalšího.
- Typy ovládání: Ovládáni mohou být vzdáleně, například užitím nějakého joysticku nebo konzoly, nebo mohou být autonomní, kdy je robot schopen samostatně vykonávat určitý druh práce.

Tato práce se zaobírá kategorií mobilních robotů a konkrétně podkategorií autonomní roboti. Ta zahrnuje autonomní nebo-li bezpilotní roboty, jenž jsou schopni mimo pohybu po silnicích nebo v terénu bez řidiče navíc samostatně plnit úkoly bez přímého lidského dozoru. Mohou být využíváni pro přepravu osob nebo nákladů a často jsou vybaveni různými senzory a algoritmy pro navigaci a rozpoznávání překážek. Mohou být používáni v různých odvětvích, jako je průmysl, zemědělství nebo zdravotnictví. Pokud jsou vybaveni umělou inteligencí, pak mají schopnost učit se a adaptovat se na nové situace, což jim umožňuje zlepšovat svou efektivitu a přesnost.

Obrázek 2.4: Různé druhy robotů [34], [32], [23], [10], [26], [29]

2.2.2 Kinematika

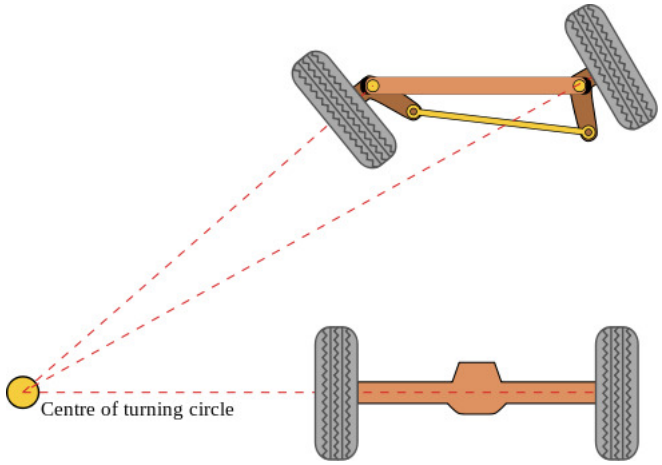
Kinematika je oblast fyziky, blíže pak mechaniky, která se zabývá studiem pohybu těles a popisem jeho geometrických a kinetických veličin. Obecně se zkoumá, jak se mění poloha, rychlost a zrychlení těles v čase, nikoliv však příčiny těchto pohybů - jako jsou síly, nebo momenty. Na to se zaměřuje oblast zvaná dynamika. Kinematika se rovněž zabývá popisováním trajektorií při pohybu těles a určováním času potřebného pro uražení určité vzdálenosti, nebo pokud má těleso urazit určitou vzdálenost předem danou rychlostí. [37]

V oblasti robotiky a autonomního řízení je kinematika stěžejní pro plánování pohybu robota a následné řízení jeho trajektorie, nebo pro propočty ujeté vzdálenosti a orientace v prostoru. V procesech SLAM pak umožňuje plánovat optimální trajektorii pohybu na

základě informací o poloze robota a mapy okolního prostředí. Pokud tedy robot 'vidí' před sebe, nebo alespoň zná trasu, kterou má projet, obor kinematika napomáhá svými rovnicemi najití a projetí té nejoptimálnější trasy.

V automobilovém průmyslu a v robotice je důležitá kinematika řízení kol, zejména v souvislosti s takzvaným Ackermannovým řízením. Tento typ řízení umožňuje vozidlu projíždět oblouky, aniž by sklouzával pneumatiky do stran při sledování cesty v zatáčce. Řízení kol je nezávislé, což znamená, že se vnitřní a vnější kolo otáčí různě – tedy pod jinými úhly. V případě, že vozidlo projíždí zatáčku, vnitřní kolo má menší poloměr dráhy než kolo vnější.

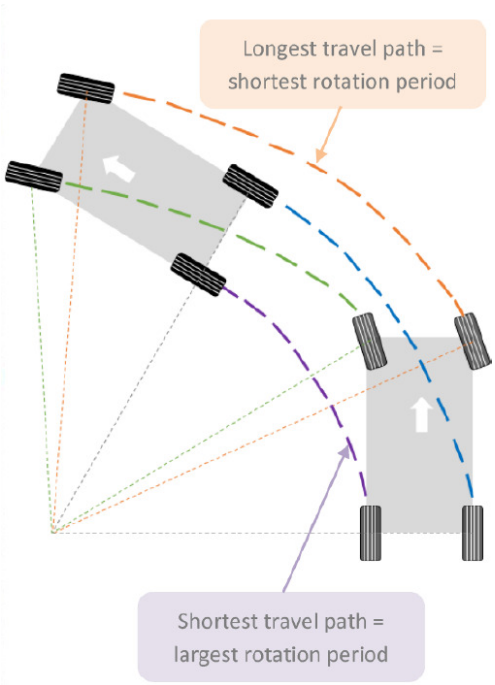
Geometrickým řešením je, že všechna kola mají uspořádány poloměry kružnic se společným středem, jak je patrno z obrázku 2.5. Protože jsou zadní kola pevná (nemohou se otáčet), musí být tento střed na přímce prodloužené od zadní nápravy. Aby se všechny osy potkaly v jednom bodě, musí se vnitřní kolo (protože je blíže od daného bodu) pootočit o větší úhel, než kolo vnější. [28] Vizte následující obrázek, který znázorňuje problematiku.



Obrázek 2.5: Ackermannovo řízení, kde přední kola směřují do stejného bodu, čímž musí mít vnitřní kolo větší úhel natočení, než kolo vnější. [12]

Díky tomu tedy všechna kola projíždí stejnou křivku, což téměř zajistí hladký průjezd zatáčkou. Na následujícím obrázku je vidno, že vnější strana kol musí ujet výrazně delší trasu, než kola na vnitřní straně[3] . Aby se kola pohybovala po určité křivce stejně, a tedy robot následoval požadovanou kružnici, je zapotřebí, aby se kola na vnější straně točila rychleji, než kola na straně vnitřní. Ujeté vzdálenosti jednotlivých kol znázorňuje následující obrázek 2.6.

[3]Můžete si všimnout, že obrázek není úplně nejpřesnější. Přední kola (resp. všechna kola, jenom ta přední nemají přesné grafické znázornění) by měla svírat s přímkou vedoucí do středu kružnice úhel o devadesáti stupních. Pro ilustrační ukázku ujeté délky jednotlivých kol je ale obrázek naprosto dostačující.



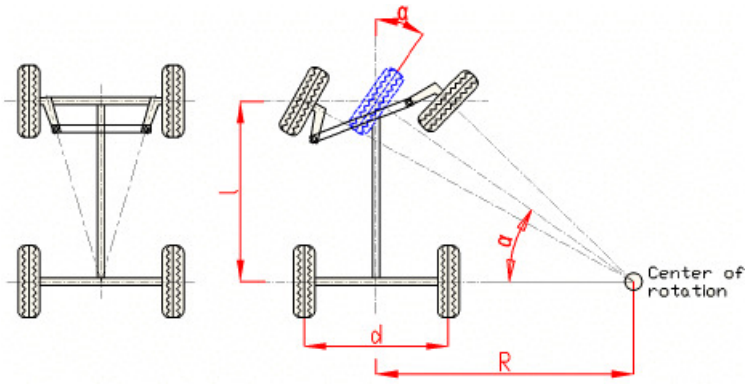
Obrázek 2.6: Ujetá vzdálenost jednotlivých kol při Ackermannovu řízení [21]

Aby nedocházelo k poskakování předních kol při zatáčení, je třeba buďto zpomalit otáčení kol na vnitřní straně nebo zrychlit otáčení kol na straně vnější nebo obojí zároveň. Tento proces se nazývá diferenciální řízení, jenž je založen na různých rychlostech jednotlivých kol. Výsledkem je pak plynulé zatáčení [19].

Pro výpočet různorodých rychlostí pro levou a pravou stranu robota je třeba znát úhel zatáčení a rychlost vozidla. Protože má při zatáčení levé kolo jiný úhel než kolo pravé, zavádí se tzv. prostřední virtuální kolo, které je uprostřed robota, viz obrázek 2.7 [14]. Pro toto kolo potřebujeme znát úhlovou rychlost, což bude reprezentovat rychlost robota a vypočítá se následovně:

$$\omega = \frac{v}{r} \tag{2.1}$$

v je obvodová rychlost kola a r je poloměr kola.



Obrázek 2.7: Střední virtuální kolo, které se zavádí z důvodu jednoduššího výpočtu pohybu robota [19]

Úhel směřování středního kola (R) je vypočítán goniometrickou funkcí, přičemž (l) je rozvor kol - což je vzdálenost předních kol od zadních - a (d) je rozchod kol - vzdálenost středovými liniemi kol, které jsou v jedné nápravě.

$$R = \frac{l}{\tan \alpha} \tag{2.2}$$

Pak lze dopočítat rychlosti pro levé a pravé kolo následovně. Úhlová rychlost kol na levé a pravé straně vozidla je závislá na rozchodu kol a úhlové rychlosti středového kola. Výsledkem je úhlová rychlost pro kola na levé straně (ω_L) a kola na pravé straně (ω_R).

$$\omega_L = \omega * (1 + \frac{d}{2 * R}) \tag{2.3}$$

$$\omega_R = \omega * (1 - \frac{d}{2 * R}) \tag{2.4}$$

2.2.3 Odometrie

Odometrie je metoda měření pohybu, který se využívá pro určení polohy a orientaci robota. Primárně se využívá informací z jeho pohybových senzorů, především z enkodérů na kolech. Enkodéry měří počet otáček kol, což umožňuje vypočítat ujetou vzdálenost. Úhel natočení robota jsme schopni vypočítat na základě vědomostí o natočení kol přední nápravy a ujeté vzdálenosti. Tyto informace pak lze využít k aktualizaci odhadu polohy robota.[4]

Roboti využívají odometrii při plánování a navigaci v prostoru, protože jim umožňuje určit jejich polohu v reálném čase a sledovat svůj pohyb v prostoru.

Nicméně, odometrie není zcela přesná, protože bývá ovlivněna chybovými faktory, jako jsou například klouzání kola na nerovném povrchu nebo přítomnost šumu v senzorech. Kvůli těmto faktorům může být odometrie nepřesná a může vést k nesprávnému určení polohy robota.

Aby se minimalizovaly tyto chyby, často se používají i další senzory, jako jsou například snímače vzdálenosti, kamery a další, které mohou poskytnout další informace o prostoru. S využitím metod SLAM se dokáže zajistit správná lokalizace robota v prostoru, i přes veškeré nepřesnosti.

Při odometrii, jak již bylo zmíněno, se využívá enkodérů, které slouží pro měření otoček kola. To zajišťuje schopnost určit ujetou vzdálenost, nebo třeba aktuální rychlost pohybu. Zpravidla se neměří pouze celé otáčky kola, to vedlo k velice hrubým výsledkům a robot by tak měl velice zkreslené informace o tom, kde se nachází. Měří se zlomky otáček, což znamená, že za jednu otočku se provede x měření, díky čemuž se zvyšuje přesnost a robot tak má aktuální informace, na základě kterých je schopen vykonávat přesnější řízení a manévrování.

Výpočet ujeté vzdálenosti na základě počtů otoček kola

Aby bylo možné určit, jak se robot v prostoru pohybuje, je třeba vypočítat ujetou vzdálenost, což vypočítáme z ujeté vzdálenosti kol. Díky tomu můžeme následně vypočíst jeho souřadnice, což pro nás bude stěžejní prvek pro orientaci v prostoru.

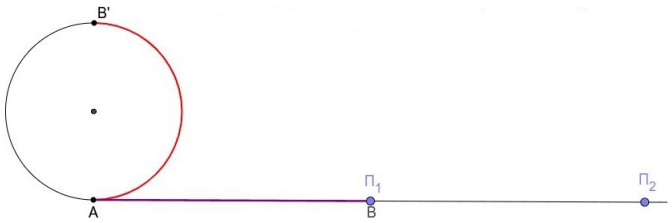
[4] Říkáme odhadu, protože i přes veškeré správné matematické rovnice může docházet k různým jevům, které ovlivňují pohyb robota a tak je realita jiná, než si myslíme. Například taková jízda po koberci nebo po ledu. Jízdní vlastnosti jsou radikálně odlišné než když jedeme po ideálním povrchu, kterým mohou být třeba parkety.

Pro výpočet ujeté vzdálenosti kola musíme znát poloměr, resp. průměr kola. Z matematiky ze základní školy víme, že obvod kruhu se dá vypočítat následující rovnicí, kde (D) je průměr kola, potom (O) je obvod kola, což je patrné i z obrázku 2.8.

$$O = \pi * D \tag{2.5}$$

Potom vynásobením počtů otoček (N) dostáváme ujetou vzdálenost kola (d):

$$d = \pi * D * N \tag{2.6}$$



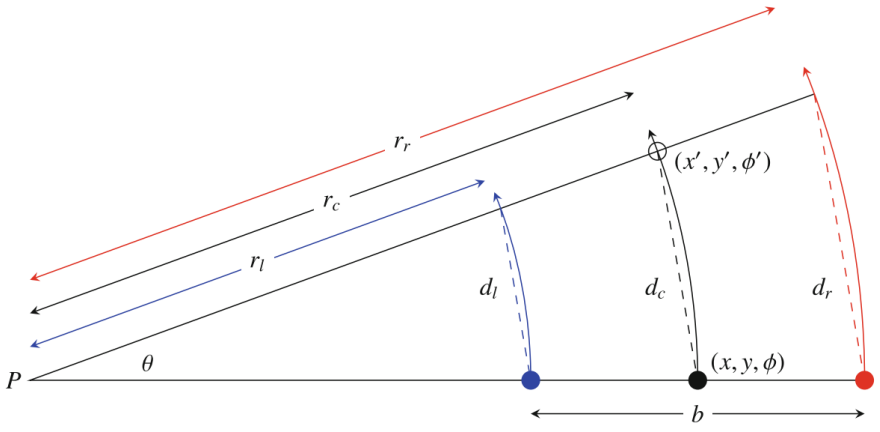
Obrázek 2.8: Obvod kola [30]

Výpočet souřadnic robota v prostoru

Aby bylo dopočítáno polohy robota, využije se předešlého výpočtu ujeté vzdálenosti kola, pro které byly výpočty provedeny (d_l, r_l). Aby bylo možné získat ujetou vzdálenost robota, bude zapotřebí dalších rovnic, které budou ukázány na následujícím příkladu.

Robot směřující nahoru [7]. Modrá tečka je levé kolo, červená tečka je pravé kolo a černá střed robota. Nyní je potřeba spočítat novou pozici a aktuální směr - tedy úhel, pod kterým robot směřuje. Počáteční souřadnice jsou (x, y) a úhel směru je ϕ. Dohromady zapsáno jako (x, y, ϕ). Počáteční směr je nastaven na $(\phi = \pi/2)$, aby to sedělo s obrázkem 2.9. Po ujetí určité vzdálenosti po křivce se robot pootočí o $(\phi' = \phi + \theta)$, kde θ se určí tímto vztahem [25]:

$$\theta = d_l/r_l = d_r/r_r \tag{2.7}$$



Obrázek 2.9: Výpočet nové polohy robota [7]

Následně se dopočítá průměr ujetých vzdáleností kol, díky čemuž je získána ujetá vzdálenost prostředního kola, což je střed robota (d_c).

$$d_c = \frac{d_l + d_r}{2} \tag{2.8}$$

A konečně jsou dopočítány nové souřadnice robota.

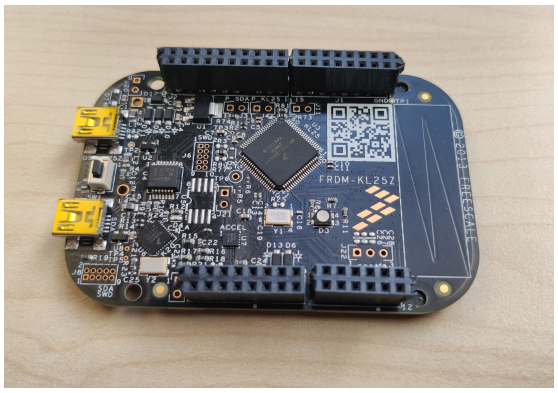
$$x' = x + d_c * \cos\theta \tag{2.9}$$

$$y' = y + d_c * \sin\theta \tag{2.10}$$

2.2.4 Vestavný systém

Vestavěný systém (tzv. *embedded system*) je jednoúčelový počítač, který se zaměřuje na právě jednu aktivitu. V tomto počítači je zabudován řídicí systém, který má za úkol plnit předem danou úlohu. Na rozdíl od univerzálních počítačů, jako jsou osobní počítače, jsou tyto systémy typicky návrhem určeny pro specifické účely a jsou optimalizovány pro konkrétní úkoly. Vestavěné systémy lze nalézt v mnoha zařízeních, jako jsou mobilní telefony, bankomaty, automobily, domácí spotřebiče, průmyslová zařízení a mnoho dalšího. Jsou všude kolem nás a kolikrát si ani neuvědomujeme, že pracujeme s počítačem.

Vestavěné systémy sestávají z mikroprocesorů, pamětí, periferních zařízení, oscilátoru a vstupně/výstupních pinů. Tyto prvky jsou integrovány na jediné desce plošných spojů (PCB), která zajišťuje propojení jednotlivých částí systému a jeho řízení. Vestavěnému počítači se říká mikrokontrolér (MCU). Ten se mimo svou velikost vyznačuje tím, že má malou spotřebu energie. Jak takový počítač vypadá je k nahlédnutí na následujícím obrázku 2.10.



Obrázek 2.10: Mikrokontrolér KL25Z

Další zařízení, která se často připojují k MCU na jeho IO piny, jsou například senzory, displeje, tlačítka a podobně. Periferie uvnitř MCU mohou zahrnovat ADC převodníky, DAC převodníky, PWM regulátory, komunikační rozhraní (UART, SPI, I2C, CAN), časovače a další.

Pro SLAM je nevyhnutelné použít vestavěných systémů. Díky tomu dokáže robot vykonávat algoritmy a de facto přemýšlet (např. při použití umělé inteligence). Pro mapování a lokalizaci robota v prostoru lze využít vestavěných systémů se senzory. Takovými senzory jsou kamery, LiDAR, ultrazvukové senzory a další, které mohou snímat okolí a vytvářet 2D, nebo 3D mapy.

Vestavěné systémy jsou nutné pro SLAM, protože musí být dostatečně malé a výkonné, aby mohly být použity v autonomních zařízeních. Tyto systémy musí být schopny zpracovat obrovské množství dat z různých senzorů v reálném čase, což vyžaduje vysokou výpočetní sílu a efektivní algoritmy. Proto jsou vestavěné systémy klíčovými komponenty pro úspěšnou implementaci SLAM v autonomních zařízeních.

Celkově lze říci, že vestavěné systémy jsou klíčovými komponentami pro mnoho moderních technologií, včetně autonomních vozidel, robotů a virtuální reality. V kombinaci se správnými senzory a algoritmy mohou vestavěné systémy umožnit zařízením mapovat okolí a lokalizovat se v něm, což umožňuje řízení a navigaci v reálném čase bez nutnosti manuálního řízení. Díky tomu mohou být autonomní zařízení využívána v mnoha oblastech, jako jsou průmysl, doprava, medicína a další, a přinášet významné výhody v efektivitě, bezpečnosti a komfortu pro uživatele.

2.2.5 Měření

Nauka a věda o měření se nazývá metrologie. Primárním cílem každého měření je minimalizace chyb, které jsou neoddiskutovatelné, a přiblížení se co nejvíce reálným a správným výsledkům. Každé měření je pokaždé zatíženo určitou chybou, což je odchylka mezi naměřenou a skutečnou hodnotou. Některé chyby mohou naměřenou hodnotu zvětšit, jiné zmenšit [18]. Úkolem pro kalibrační laboratoře je definovat množství chyb, které mohou nastat v určitých podmínkách, nazýváno *toleranční pole*. Chyby mohou záviset na různých faktorech, mnohokrát jejich kombinacemi.

Teorie chyb

Předpokládejme, že pro určitou veličinu je pouze jedna správná hodnota nazývána *pravá hodnota*. Ta může být změřena pouze *ideálním měřením*, což je měření, které obsahuje nulovou odchylku – nemá žádnou chybu. Cílem měření je tedy přiblížit se co nejvíce této pravé hodnotě a minimalizovat nahodilé chyby.

Existují dva typy chyb – **nepoznatelné** a **poznatelné**. Poznatelné chyby se ale stanovují z omezeného počtu pozorování, takže z širšího hlediska mohou být rovněž nepoznatelné, neb nejsme schopni provést nekonečno pokusů [36]. Z daného počtu pozorování se ale dostane nějaká znatelná chyba, která když je systematická – tedy dokáže se zjistit její příčina vzniku – může se provést korelace, které chybu odstraní.

Chyby se dělí následovně (dle [22], slide 13-15):

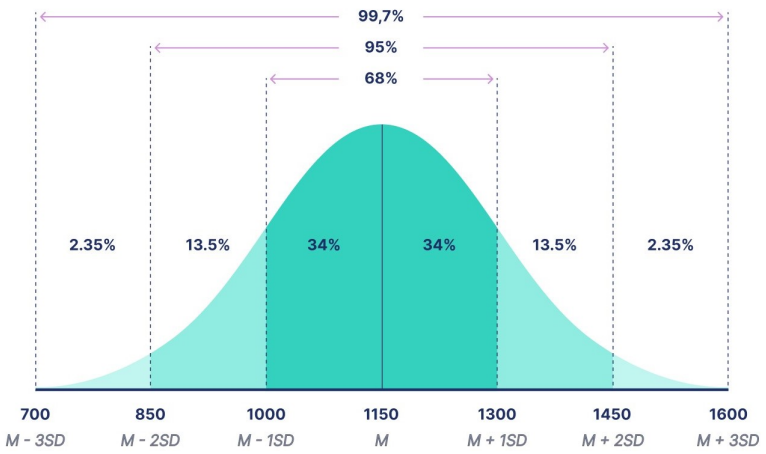
- **Náhodné chyby**: tvořeny z neznámých, či nepoznaných důvodů. Jsou zjistitelné pouze opakovaným měřením v konzistentních podmínkách. Takovéto chyby jdou poznat právě opakovaným měřením, kdy výsledky měření nejsou vždy stejné. Pokud nedokážeme přijít na příčinu vzniku chyby, pak chyba spadá do této kategorie.

- **Systematické chyby**: často se jedná o vnější vlivy na měřící zařízení. Může se jednat o vnější rušení, šum, teplotní či mechanické zatížení. Pokud se podaří nalézt vlivy ovlivňující měření, jejich eliminací se přiblížíme pravé hodnotě.

- **Hrubé chyby**: způsobeny lidským faktorem. Bývají odstraňovány přísnou kontrolou prací a školením personálu. Mezi tuto skupinu chyb patří chybný výklad výsledků, nesprávné zvolení přístroje na určité měření, výpočetní chyby aj.

Teorie nejistot

Pojem nejistota je přidružen k výsledku měření a vyjadřuje omezené vlastnosti tohoto procesu. Nejistotu jsme schopni zmenšovat, nikoli však absolutně eliminovat. Teorie nejistot se tedy zaměřuje na kvantifikaci nejistot a popis, jak jsou chyby v měření rozděleny a jak tyto nejistoty ovlivňují jejich přesnost. Cílem tohoto přístupu je určit interval hodnot, ve kterém je s určitou pravděpodobností pravá hodnota [18]. Takovým intervalem může být např. Gaussovo rozdělení, které je užitečné pro popis náhodných veličin, které mají centrální limitní vlastnost, tj. jejich průměr se blíží k normálnímu rozdělení s rostoucím počtem měření 2.11.



Obrázek 2.11: Gaussovo normální rozdělení [8]

To znamená, že pravá hodnota se nachází s určitou pravděpodobností v okolí naměřené hodnoty, přičemž je okolí určeno nejistotou [6]. Ke stanovení velikosti nejistoty existují dvě metody, které se dají kombinovat pro lepší přesnost: **typ A** a **typ B**.

Nejistota typu A

Tato metoda vychází ze statistického zhodnocení opakovaného měření. Počet měření při užití této metody by měl být více jak 10, tedy $n > 10$. Jinak by se měla využít metoda B, protože výsledná hodnota určená pomocí vztahu (2.13) by byla málo spolehlivá.

Při použití metody typu A se postupuje následovně. Nasbírá se soubor měření, přičemž se měří stejná veličina za konstantních podmínek. Odhad výsledné hodnoty pro počet měření n, kde $(n > 1)$, je reprezentován hodnotou aritmetického průměru podle vztahu:

$$\hat{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.11}$$

Nejistota odhadu (u_{Ay}) hodnoty X se určí jako výběrová směrodatná odchylka střední hodnoty. Prvně ale potřebujeme určit výběrovou směrodatnou odchylku, která udává šířku intervalu, do kterého s určitou pravděpodobností padne naměřená hodnota.

$$s(x) = \sqrt{\frac{1}{(n-1)} \sum_{i=1}^{n} (x_i - \hat{x})^2} \tag{2.12}$$

Výběrovou směrodatnou odchylku střední hodnoty pak možno získat vztahem:

$$s(\hat{x}) = \frac{s(x)}{\sqrt{n}} \tag{2.13}$$

Nejistota typu B

Vyhodnocení standardních nejistot vstupní veličiny typu B je založeno na ostatních přístupech. Metodou B jsme schopni např. odhadnout vliv náhodných chyb. Standardní nejistota se odhaduje pomocí racionálního úsudku na základě možných informací. Kupř. způsob určení nejistoty měřicího přístroje udaný výrobcem přístroje, nebo na základě již získaných zkušeností.

Pokud ovlivňující vlivy nabývají hodnot v rozsahu definovaném výrobcem (= měří se za výrobcem stanovených pracovních podmínek), určí se interval, ve kterém hodnota x měřené veličiny X s velkou pravděpodobností leží, přičemž se implicitně předpokládá rovnoměrné rozložení pravděpodobnosti $k = 3$, tj., že pravděpodobnost výskytu libovolné hodnoty z tohoto intervalu je stejná, pak:

$$u_B(x) = s(x) = \frac{\hat{x}}{\sqrt{3}} \tag{2.14}$$

Je-li výsledek měření získán z hodnot několika veličin, je výsledná nejistota typu B rovna odmocnině součtu variancí (disperzí) a kovariancí těchto veličin, typicky doplněných o váhové koeficienty.

Další nejistoty

Někdy nemusí vyčíslení nejistoty podle typu A ($u_A(x)$), nebo podle typu B ($u_B(x)$) stačit a proto existují další nejistoty. Právě takovou může být kombinovaná nejistota, která se určuje jako odmocnina ze součtu čtverců obou nejistot:

$$u_{Cy} = u_C(x) = \sqrt{u_{Ay}^2 + u_{By}^2} \tag{2.15}$$

Aby bylo dosaženo co nejlepšího intervalu pokrytí, rozšiřuje se standardní nejistota koeficientem rozšíření k_r, z čehož vzniká rozšířená nejistota a ta je pak vyjádřena vztahem:

$$U = k_r * u \tag{2.16}$$

Kapitola 3

Rešerše zvolených komponent

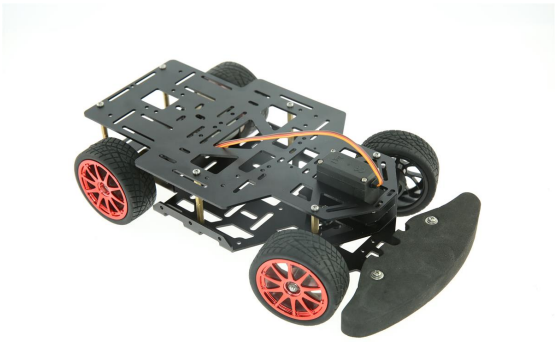
Robot sestavá z pevné konstrukce. Je řízen mikrokontrolérem FRDM-KL25Z, jenž je velice levná vývojová platforma z rodiny Kinetis L série KL2x, založena na ARM Cortex-M0+ procesoru. Kontrolér je mozkem celého robota, jehož procesor pracuje na frekvenci 48MHz. Bohužel má poměrně malou paměť a to přesně 128kB flash a 48kB SRAM. Nás primárně zajímá paměť SRAM, jelikož na ni se bude ukládat dynamicky se tvořící mapa.

3.1 Konstrukce

Konstrukce robota je ze sady DFRobot ROB0170, a vypadá následovně 3.1. Konstrukce byla zakoupena díky přihlášení se na soutěž NXP CUP. Na oficiálních stránkách mají odkaz na zakoupení modelů. NXP vydává nové modely pro každoroční soutěž, ale mnoho týmů, které jezdí na soutěž už delší dobu, využívají své starší verze modelů. Konstrukce dojde rozdělána v krabici do posledního šroubku.

Jednotlivé komponenty jsou přehledně odděleny do sáčků. Návod, který byl přiložen bohužel neodpovídal komponentám, které byly uvnitř balení, a tak bylo zapotřebí zapnout selský rozum a konstrukci sestavit vlastními silami. Bylo zde spousta součástek a mnoho z nich nebylo využito.

NXP tím dává prostor pro kreativitu a každý model na dráze je tím jedinečný, protože každého napadne jiné řešení, jak model sestavit. Ve výsledku vypadají modely podobně. Pokud se ale zanoříme do maličkostí, jak se k určitým problémům přistupovalo, nastartuje se promenáda skvostných nápadů.



Obrázek 3.1: Konstrukce vozidla

Problémy

Při konstrukci modelu, která byla prvním krokem, se objevilo pár problémů.

Jedním z nich bylo přidělání ozubeného kolečka na poháněcí tyč vedoucí od BLDC motorů. To slouží pro převedení kinematické síly rotace motoru na kola, čímž dojde k pohybu vozidla. Díky tolika součástkám navíc, v balení přímo u motorů, jsem nerozuměl, jak má kolečko na hladké kovové tyči držet. Kolečko jsem přilepil speciálním lepidlem na příslušnou tyč, ale bylo zapotřebí mít vše správně rozmyšlené, protože po přilepení se s kolečkem už nepohne. Pokud bych netrefil korektní pozici umístění ozubeného kolečka, mohlo by se stát, že by se neprotklo s druhým ozubeným kolečkem, které je přišroubováno přímo na kolo. Tím pádem by se rotace nedostala ke kolu a to by se tak nerozjelo. Díky tomu, že na jedné straně bude přidělán senzor na otočení kola, bylo třeba vše řádně změřit a nechat si případný prostor. [1]

Dalším závažným problémem byla táhla vedoucí od servomotoru ke kolům. Vypadají následovně 3.2. Ty se mi dostala překvapivě krátká, což způsobovalo, že byla kola natočena k sobě a směřovala směrem k robotu. Odchylka byla cca 5° směrem k robotu, což by zapříčiňovalo chybovost pohybu a zvětšovalo by to nepřesnost dalších výpočtů. Zkoušel jsem proto sehnat delší šroubky, abych dokázal prodloužit táhlo, čímž by se kola srovnala do rovnoběžek. Aby bylo zajištěno komfortního nastavování táhla, tak je jedna strana šroubku vždy levotočivá. Tím pádem, v ideálním případě, přiděláme jednu stranu táhla ke kolu, druhou stranu táhla k servu a postupně povolujeme šroubek, což způsobí, že se obě strany povolují současně a my si tak můžeme navolit ideální vytočení šroubku.

Obrázek 3.2: Táhlo řízení

Jelikož je ale jedna strana levotočivá, sehnat takový šroubek je nadlidský úkol. Takové šroubky nejsou v normálních obchodech prodejné. Jsou nesehnatelné, protože se normálně nikde nepoužívají.

Nakonec mne napadlo zajít do obchodu s RC modely, kde se dala patřičná táhla koupit. Následně jsem je musel zkrátit a zabrousit, neboť mnou požadovanou délku jsem nebyl stavu sehnat.

3.2 Elektřina, rozvody

Celému systému dodává energii baterie Kavan Li-Po 2700 mAh, která poskytuje napětí 11.1V, což potřebují BLDC motory pro své korektní fungování.

Elektřina se distribuuje do všech periferií skrze kabelové svorky (wago), které mají pět portů. Tedy, z baterie proudí elektřina do waga (plus a minus - každý má své), a odsud se rozštěpí do:

- Motorů (2x - levý a pravý)

[1] To, že budeme používat nějaký senzor na snímání otočení kol jsem v době konstrukce robota ještě nevěděl. Tušil jsem už ale, že něco podobného může nastat a proto jsem si vyměřil takovou vzdálenost, aby pro případný senzor zbylo dostatek místa.

- Step-down měniče pro periferie
- Stabilizátor napětí pro servo

Step-down měnič pro periferie

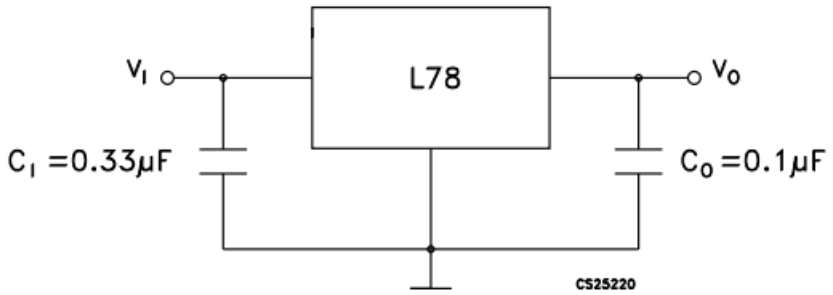
Hlavní měnič, pro zajištění elektřiny pro mikrokontrolér, všechny senzory a pro napájení osvětlení.

Napětí se zde mění z 11.1V na 5.5V. Z výstupu vedou kabely do rozvodové desky, na kterou jsou přidělány veškeré napájecí kabely k již zmíněným modulům.

Stabilizátor pro servo

Servo vyžaduje 4.8 - 6V, přičemž potřebuje alespoň 0.8A. První měnič není schopen poskytnout takový výkon pro všechny periferie a servo, proto je použit lineární regulátor napětí - 5V typ 7805, který je zapojen následovně.

Na vstupu má napájení ze step-down měniče. Mezi plus a minus na rozvodné desce, které vedou ze step-down měniče, je kondenzátor 330nF. Odtud vedou kablíky k napájecí nožce stabilizátoru. Zem je přidělána na minus na rozvodné desce. U výstupu je další kondenzátor o velikosti 100nF. Tímto máme zajištěné napájení pro servo. Schéma zapojení je ukázáno na obrázku 3.3.



Obrázek 3.3: Zapojení stabilizátoru napětí [3]

3.3 Řídicí jednotky

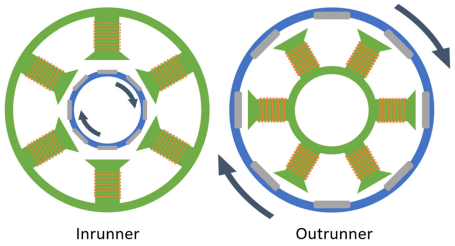
Robot má k dispozici dva BLDC motory pro zajištění pohybu vpřed a vzad a servo motor, který koriguje natočení předních kol, díky čemuž může zatáčet. Obě jednotky vyžadují napájení a řídicí signály z mikrokontroléru. Motory jsou umístěny u zadních kol, kdežto servo motor je u přední nápravy.

3.3.1 Motory

Vozidlo je poháněno dvěma BLDC motory o výkonu 930kV, které vypadají takto 3.6. Ty jsou připojeny na elektronické regulátory otáček (ESC), které slouží k regulaci otáčení motorů.

BLDC motor (tzv. Brushless DC) je typ elektrického motoru, který se skládá z rotujícího rotoru s permanentními magnetky a statoru s cívkami, které jsou napájeny střídavým proudem z ESC. BLDC motory se dělají ve dvou provedeních, tzv. *inrunner* a *outrunner*.

Inrunner má rotor rotující uvnitř statického těla, kdežto outrunner je statický uvnitř a rotuje po vnějším obvodu, jak je ukázáno na obrázku 3.4.

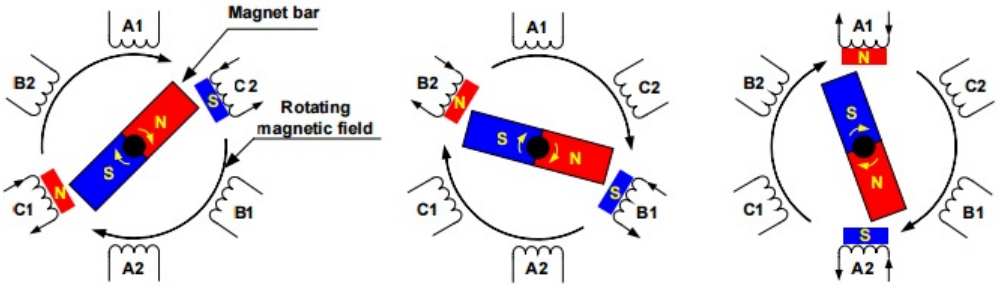


Obrázek 3.4: Typy BLDC motorů [24]

Pro vysvětlení principu fungování budou uvedeny motory třífázové, které jsou zapojeny do hvězdy. Bývají i jiná zapojení, ale třífázové je nejpoužívanější. [9].

Cívky jsou umístěny po obvodu statoru, což je pevná část motoru. Každá cívka má na opačné straně svůj protipól a když na cívku, a její protipól, je přiveden proud, vytvoří se elektromagnetické pole. To interaguje s magnetickém polem rotoru a díky tomu se rotor pootočí. Rotor má permanentní magnetické pole díky magnetkám připevněných staticky k rotoru. Toto pole se tedy otáčí v závislosti na vytvořeném elektromagnetickém poli cívkami.

Řídicí jednotka sleduje polohu rotoru a zasílá střídavě signály do příslušných cívek, aby docházelo k měnění elektromagnetického pole ve směru chtěného pohybu motoru. To je znázorněno zde 3.5.



Obrázek 3.5: Rotace BLDC motorů [4]

ESC je klíčový komponent pro řízení rychlosti a směru otáčení motoru. Přijímá signál z řídicí jednotky (MCU) a na základě tohoto signálu upravuje napětí a frekvenci, které posílá na motor. To umožňuje precizní kontrolu nad rychlostí a směrem otáčení motoru.

Na obrázku 3.7 je možnost vidět na levé straně kabely plus a minus pro zapojení k baterii. Pak jsou zde kabely bílý a černý, které se připojují na řídicí jednotku. Bílým kabelem se vede PWM signál, černý slouží pro uzemnění pro stabilizaci signálu. Na pravé straně jsou tři stejné kabely, které se zapojují na motor.

Zapojení

Doporučuje se použít kabelové svorky, které se připevní od motoru k ESC. Je to z toho důvodu, že pokud by byly kabely prohozeny, prohodila by se polarita otáčení motoru. To bude mít za následek, že se motor bude točit opačným směrem, než bylo zamýšleno.

Obrázek 3.6: BLDC motor 930kV

Obrázek 3.7: BL HELI ESC 30A

Já kabelové svorky neměl a tak jsem kabely k ESC napájel naslepo. Zapojení jednoho motoru se mi podařilo napájet správně a motor se točil na správnou stranu. U druhého motoru jsem takové štěstí neměl a tak se točí na opačnou stranu, než bych potřeboval.

Řešením byla softwarová úprava, kdy jsem tomuto motoru otočil signál. To znamená, když jede robot dopředu, jednomu motoru se zasílá klasický signál a druhému motoru musíme dát signál stejné rychlosti, avšak pro otáčení se na druhou stranu.

TBD	**Využití**
`Piny MCU`	PTA12 (levý motor), PTA13 (pravý motor)
`Pin Mux Control`	FTM1_CH(0/1) (Alternative 3)
`Konfigurace pinu`	Output
`Frekvence signálu`	50Hz

Tabulka 3.1: BLDC motory

Řízení

Ovládání BLDC motoru z řídicí desky je tedy skrze ESC. ESC přijímá signál z řídicí jednotky, jenž obsahuje informaci o požadované rychlosti otáčení motoru. Tento signál je *PWM*[2] (*pulse-width modulation*), kde jednotlivé rychlosti jsou popsány v tabulce 3.2. Obecně platí, že čím delší je šířka pulzu, tím rychleji se motor otáčí a naopak. Pozor však, že se pohybujeme pouze v rozmezí cca 35% až 45% pulsní šířky. Co je míň, či víc, nebere ESC v potaz.

Komunikace mezi ESC a BLDC motorem funguje rovněž na principu PWM, ale tato problematika byla popsána již dříve (3.3.1). Pro naše potřeby musíme hlavně vědět, jak komunikovat s ESC a jak mít vše správně zapojeno.

Robot má umožněn pohyb vzad pouze s jedním řádem rychlosti, protože v konečném důsledku není couvání žádoucí. Couvá se jedině tehdy, když robot vyjede mimo jízdní pruhy a podaří se mu to detekovat 4.2.

Střída pro zastavení motorů byla odchycena logickým analyzátorem (např. [5]) a prvotním užitím Arduina, kde jsou připravené knihovny a implementace je tak značně ulehčena. Další hodnoty byly vybrány experimentálně.

[2]PWM je metoda řízení napětí, kdy se napětí na výstupu periodicky mění mezi maximální a minimální hodnotou

Rychlost	Střída
Jízda vzad	6.7
Zastavení motorů, robot stojí na místě	7.365
Pomalejší jízda vpřed	7.72375
Rychlejší jízda vpřed	8.0825
Rychlejší jízda vpřed	8.44125
Nejrychlejší jízda vpřed	8.8

Tabulka 3.2: BLDC motory PWM

3.3.2 Servo

Servo motor se používá k natáčení nápravy předních kol, což umožňuje zatáčení. Na robotovi je použit servo motor *MG996R*, který je vyfocen na obrázku 3.8

Obrázek 3.8: Servo motor MG996R

Je zapotřebí zajistit napájení 5V, jinak servo nemá dostatečný výkon pro svou práci. To bylo zajištěno stabilizátorem napětí, jak jsme si popsali (3.2). Servo je připojeno k MCU na pin *PTD5*, který umožňuje funkcionalitu PWM skrze Timer / PWM Module (TMP).

	Využití
Piny MCU	PTD5
Pin Mux Control	FTM0_CH5 (Alternative 4)
Konfigurace pinu	Output
Frekvence signálu	50Hz

Tabulka 3.3: Servo motor

Rovněž funguje na principu PWM, kdy 2% znamená natočení úplně doleva a 12% pak doprava. 7% pak znamená jízdu vpřed v ideálním případě. Avšak z nějakého důvodu použitý servo motor má 90° při 7,5% (proto se nenechte zmást rozdílem obrázků, které lze nalézt na internetu, a mým kódem). Následně řídicí knihovna uvažuje tři možnosti natočení předních kol na každou stranu. Vzhledem k tomu, že 1° je 0,083%, lze snadno dopočítat procenta signálu, která se budou zasílat. Viz následující tabulka (3.4).

Předem stanovená natočení kol dále pomohou pro rychlejší výpočty pohybu robota, neb potřebné výpočty, které souvisí s natočením serva, budou uloženy při kompilaci programu do tabulky. Tím pádem se ušetří nějaký čas, kdy by se musely počítat sin, cos funkce.

Tyto výpočty souvisí s lokalizací a pohybem robota. Jestli si vzpomenete na ackermannovo řízení (2.2.3), při zatáčení musíme nejprve vypočítat střed, kolem kterého se točíme. Kvůli tomu, že jsme vždy nějak natočeni a jsme někde v prostoru, nedokážeme bez výpočtů určit, kde přesně se střed kružnice, kterou následujeme, nachází.

Stupně	Procenta
45	3.75
60	5
75	6.25
90	7.5
105	8.75
120	10
135	11.25

Tabulka 3.4: Servo motor PWM

Víme-li, jaké natočení serva máme, můžeme před-počítat poloměr této kružnice. Vypočítáno následujícím vztahem, kde r je poloměr kružnice, L je vzdálenost přední nápravy od zadní a α je úhel natočení přední nápravy.

$$r = L/\tan(\alpha) \tag{3.1}$$

Stupně	Poloměr otáčení v cm
45	17.5
60	30.3
75	65.3
90	7.5
105	65.3
120	30.3
135	17.5

Tabulka 3.5: Poloměry kružnic při zatáčení

3.4 Rešerše senzorů

Senzory jsou využity k získávání informací o okolním prostředí, či překážkách nebo k zaznamenávání určitých jevů, jako je detekce otočení kola a dalších. Algoritmus Grid-based SLAM pracuje tak, že se do mapy zaznamenávají data ze senzorů a následně se využívá kombinace těchto dat k aktualizaci mapy a lepší určení pozice robota. Senzory jsou rozděleny podle pracovní náplně.

3.4.1 Senzory pro detekci jízdního pruhu

Barevné senzory

jsou strategicky umístěny úplně vepředu a detekují barvu pod senzorem. Pakliže by došlo k rozeznání jízdního pruhu (totiž bílé barvy), máme tak dost času na natočení přední nápravy, čímž dojde k zatočení vozidla. Senzory jsou umístěny jen pár centimetrů nad povrchem země, aby byla data co nejpřesnější.

Byl vybrán detektor barvy *TCS3200* 3.9, který disponuje čtyřmi LED světly, které využívá pro přisvícení povrchu. Díky tomu nedochází k mylným jevům, kdy - pokud je

špatné světlo - si robot zastíní, pročež jiné senzory, které byly testovány, hlásily, že nejsme senzorem nad jízdním pruhem, což byla mylná úvaha.

Obrázek 3.9: Detektor barvy TCS3200

Chybovost je dle dokumentace [2] 0.2% při 50kHz. Pokud ale senzor přizvedneme od povrchu dále, chybovost kvadraticky roste. Experimentálně bylo zjištěno, že při cca 4cm od povrchu senzor hlásí rozdíl mezi bílou a černou barvou minimální jednotky. Díky tomu by mohlo často docházet k nesprávné interpretaci dat ze senzorů a robot by mohl začít zatáčet.

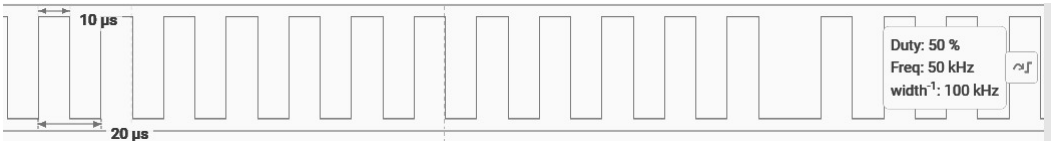
Z toho důvodu jsou senzory umístěny do jednoho centimetru nad povrchem. Díky tomu je chybovost zanedbatelná a senzor tak zasílá správná data.

Popis senzoru

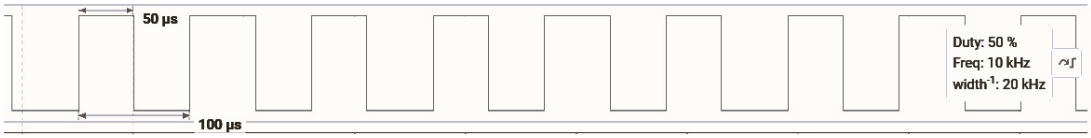
Senzor zasílá čtvercový signál, který sestává z opakujících se čtvercových vln. Tyto vlny se střídají mezi dvěma úrovněmi napětí - vysoké a nízké (typicky 5,5V a 2,5V). Měřením jejich frekvence získáváme hodnotu detekovaného světla. [20]

Senzory jsou připojeny k pinům mikrokontroléru 3.6, jejichž funkcionalita je nastavena na GPIO[3]. Je rovněž zapnuto přerušení při padající hraně signálu.

Pro výpočet frekvence signálu je použita input capture funkcionalita [16] mikrokontroléru, která se spouští vždy při přerušení. Funkce pak zaznamenává časy spadené hrany, kde jsou tyto časy následně využity pro výpočet trvání předešlé periody. Jak lze vidět na obrázku - pokud je doba periody kratší, jedná se o bílou barvu 3.11, pokud delší, jedná se o černou barvu 3.10. Více barev se neuvažuje.



Obrázek 3.10: Čtvercový signál černé barvy



Obrázek 3.11: Čtvercový signál bílé barvy

[3]Nastavení pinu na mikrokontroléru zahrnuje konfiguraci jeho vlastností, jako je směr (vstup nebo výstup), počáteční stav a nebo přerušení.

	Využití
`Pin Mux Control`	GPIO (Alternative 1)
`Piny MCU`	PTD0 - levý, PDT4 - pravý, PDT3 - střední senzor
`Konfigurace pinu`	GPIO, pull up
`Přerušení`	Ano, falling edge[4]
`Metoda zpracování signálu`	Input capture

Tabulka 3.6: Barevné senzory

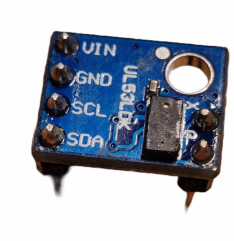
3.4.2 Senzory pro mapování

Laserové senzory

slouží k mapování prostoru. Jsou umístěny na stranách robota. Vysíláním laserových paprsků skenujeme okolí. Detekujeme-li překážku, zanášíme do mapy její výskyt jako pevnou překážku.

Popis senzoru

Pro měření vzdálenosti je použit laserový senzor *VL53L0X* 3.12. Ten je vybaven laserovým zdrojem světla, který odesílá laserový paprsek směrem k cíli. Paprsek se odrazí od cíle a senzor měří dobu, kterou trvalo světlu, než se vrátilo zpět. Tuto dobu lze použít k výpočtu vzdálenosti robota k cíli.



Obrázek 3.12: Laserový senzor VL53L0X

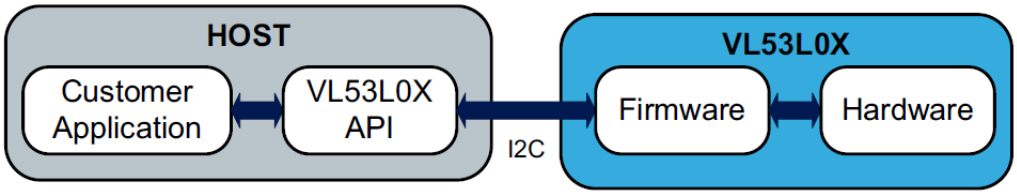
VL53L0X má rozsah měření od 2cm do 2m a nabízí vysokou rychlost měření. Je také kompaktní a nízkonapěťový, což umožňuje snadnou integraci do menších robotů, kteří nedisponují velkými bateriemi. Senzor má velikost 29.5mm x 22mm a tak je možné ho umístit do malých prostor, kam by se např. ultrazvukový senzor nevešel.

Přesnost senzoru je uvedena +/-3%. Při testování senzoru na různé povrchy mne velice překvapilo, že sklo senzoru nedělá prakticky žádný problém. Odchylku nejistoty jsem vypočítal metodou typu A na +/-5% (2.2.5). Na druhou stranu senzoru dělá problém zrcadlo. Tam snímá vzdálenost odrazu objektu v zrcadle. Experimentálně bylo zjištěno, že pokud by jel robot rovnoběžně se zrcadlem, snímá svou vzdálenost od zrcadla, předpokládaně. Je-li zrcadlo pod jiným úhlem a senzor tak nestojí proti sobě, měří vzdálenost odrazu objektu v zrcadle.

Komunikace se senzorem

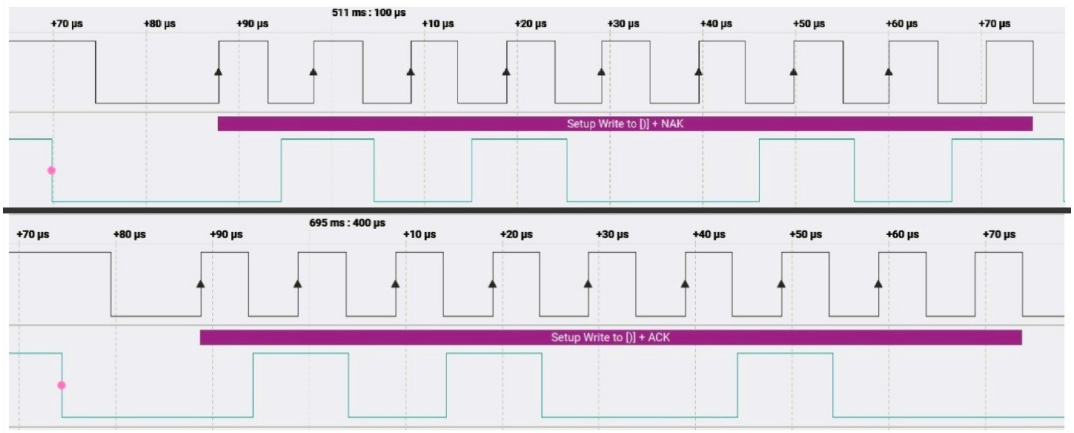
Komunikace se senzory je implementována s použitím I^2C protokolu. Adresa senzoru je standardně nastavena na 0x29. Úroveň napětí I^2C je mezi 3-5V. Stejně tak i napájecí napětí. Výhodou je rozsah do 5V, tím odpadá případná nutnost řešení level shifteru [5].

V dokumentaci se píše [35], aby se použilo vytvořené *API* (Application Programming Interface) pro komunikaci se senzorem. Znázorněný diagram ukazuje, jakým způsobem probíhá komunikace mezi MCU a senzorem 3.13. Snažil jsem se senzor tímto způsobem rozjet, avšak jsem se zasekl na dlouhou dobu a do zdárného konce jsem nedošel. Okolo API je potřeba postavit aplikaci, která si volá funkce z API a následně napojit komunikaci na námi vytvořenou knihovnu, která bude obsluhovat sériovou komunikaci.



Obrázek 3.13: VL53L0X popis funkce systému [35]

Jak bylo již zmíněno, nakonec se senzor nepodařilo korektně zprovoznit s deskou KL25Z a proto bylo třeba najít jinou cestu. Tou bylo využití Arduina, jehož úloha bude popsána v sekci 3.6. Na následujícím obrázku 3.14 je ale výstřižek odchycené komunikace osciloskopem. Nahoře je komunikace s deskou KL25Z, dole je deska Arduino. Ač se signály zdají být prakticky stejné, se zápisem stejných dat, senzor na tento paket odeslal desce KL25Z NAK a desce Arduino ACK. Nedokázal jsem přijít proč se tak stalo, proto jsem přidal Arduino desku, se kterou komunikace funguje. Můžete nahlédnout, že signály jsou opravdu velice podobné, časově sedí, daty sedí, ale na jednu desku senzor odpoví, na druhou ne...



Obrázek 3.14: Výstřižek komunikace mezi (KL25Z/Arduinem) a laserovým senzorem. Vrchní odchycená komunikace je s deskou KL25Z - nefunkční, spodní komunikace je s deskou Arduino - funkční.

[5] Level shifter je zařízení, které slouží pro převod signálové úrovně z jednoho napěťového rozsahu na jiný. Tak umožňuje propojit různá zařízení s různými napěťovými úrovněmi signálu.

Ultrazvukové senzory

jsou využity pro detekci překážek jak fixních, tak pohyblivých, v těsné blízkosti auta. Senzor je připevněn nad servo motorem vepředu. Pokud senzor detekuje nějakou překážku, auto okamžitě zastaví, aby nedošlo ke střetu. Vzhledem k tomu, že uvažujeme primárně jízdu vpřed, stačí tak detekce pouze před vozidlem.

Popis senzoru

V této skupině senzorů byl vybrán *Ultrasonic senzor M5* 3.15 využívající ultrazvukové vlny k určení vzdálenosti k předmětu. Senzor emituje ultrazvukovou vlnu a poté měří dobu trvání, než se vrátí odraz zpět. Tuto dobu lze použít k výpočtu vzdálenosti k překážce. Senzor má efektivní rozsah měření od 30 - 150cm a proto je využit primárně pro účely detekce překážky před robotem.

Obrázek 3.15: Ultrazvukový senzor M5

Komunikace se senzorem

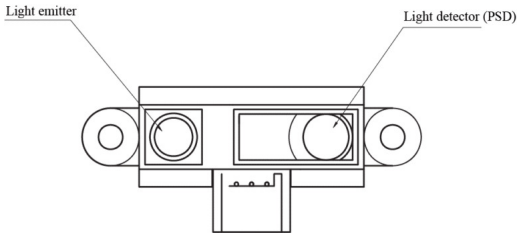
Komunikace se senzorem je rovněž skrz I^2C sběrnici. Adresa senzoru je standardně nastavena na 0x57 [27]. Senzor nakonec nebyl použit pro svou nepřesnost. Navíc jsem ani nebyl schopen senzor zprovoznit s deskou KL25Z, protože k senzoru neexistuje žádná dokumentace popisující implementaci, resp. komunikaci (nebo se mi ji nepodařilo najít). Senzor byl zprovozněn pouze skrze Arduino, přičemž odchylka měření byla příliš velká, aby bylo senzor možné použít a věřit mu.

Infračervené senzory

byl využit místo ultrazvukového senzoru. Slouží tedy rovněž pro detekci překážek před vozidlem. Je to jednoduchý infračervený senzor použitelný pro měřící vzdálenost 20-150cm.

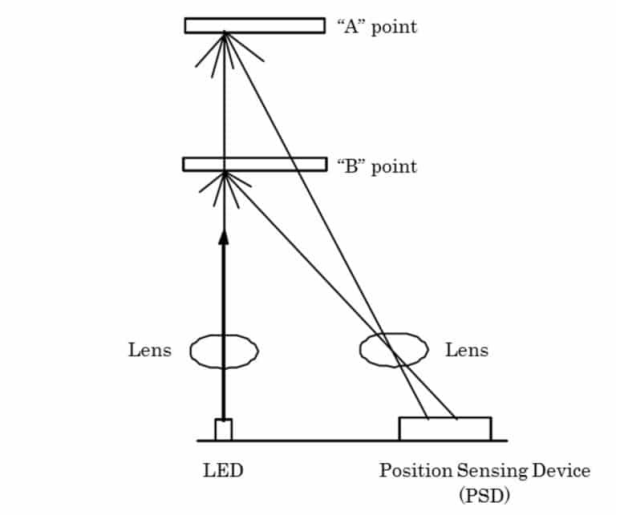
Popis senzoru

Byl vybrán senzor *SHARP GP2Y0A02YK0F* 3.16. IR senzor používá paprsek infračerveného světla k odrazu od objektu k měření jeho vzdálenosti. Vzdálenost se vypočítá pomocí triangulace paprsku světla. Senzor se skládá z IR LED a světelného detektoru nebo PSD (Position Sensing Device). Když se paprsek světla odrazí od objektu, odražený paprsek dosáhne detektoru světla a na PSD se vytvoří „optická skvrna“.



Obrázek 3.16: IR senzor SHARP

Když se změní poloha objektu, změní se také úhel odraženého paprsku a poloha bodu na PSD. Viz bod A a bod B na obrázku níže 3.17.



Obrázek 3.17: Změna pozice před senzorem

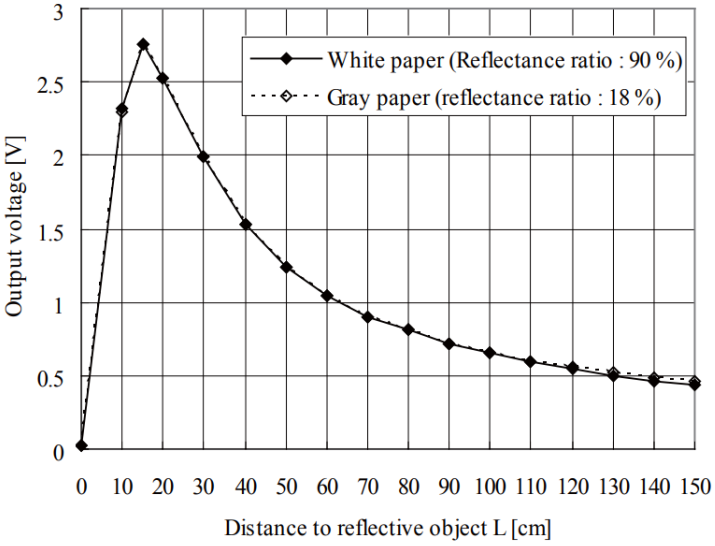
Snímač má vestavěný obvod pro zpracování signálu. Tento obvod zpracovává polohu optického bodu na PSD pro určení polohy (vzdálenosti) odrazného předmětu. Vydává analogový signál, který závisí na poloze objektu před senzorem.

	Využití
`Piny MCU`	PTE20
`Pin Mux Control`	ADC0_DP0 (Alternative 0)
`Konfigurace pinu`	Input

Tabulka 3.7: Infračervený senzor

Komunikace se senzorem

IR senzory vzdálenosti vydávají analogový signál, který se mění v závislosti na vzdálenosti mezi senzorem a objektem. Z dokumentace můžete vidět [1], že výstupní napětí senzoru se pohybuje od 2,8V, když je objekt vzdálený 15cm, do 0,4V, když je objekt vzdálený 150cm. Graf 3.18 také ukazuje, proč použitelný rozsah detekce začíná na 10 cm. Lze si všimnout, že výstupní napětí předmětu vzdáleného 5cm je stejné, jako výstupní napětí předmětu vzdáleného 62cm. Použitelný rozsah detekce proto začíná po vrcholu - zhruba na 15cm, nebo-li 2,8V.



Obrázek 3.18: Změna pozice před senzorem

Vzdálenost objektu od senzoru je možné spočítat následujícími rovnicemi, které byly získány z knihovny k senzoru[6].

$$distance = value * 5/1023; \tag{3.2}$$

$$distance = 60.375 * pow(distance, -1.16); \tag{3.3}$$

Magnetické senzory

Tyto senzory slouží k detekci otočení kola využívající Hallova jevu, díky čemuž je sledována ujetá vzdálenost.

Popis senzoru

Používá se senzor s označením 55100 3.19. Senzor využívá vzniku elektrického napětí v materiálu pod vlivem magnetického pole. Když je senzor umístěn v blízkosti magnetu, magnetické pole magnetu ovlivní pohyb nábojů v senzoru a vytvoří se elektrické napětí. Tohle napětí je potom zpracováváno senzorem a může být použito k určení vlastností magnetu nebo k jeho detekci.

[6]Knihovna k senzoru na GitHub: https://github.com/guillaume-rico/SharpIR

Obrázek 3.19: Hallův senzor 55100

Komunikace se senzorem

Na robotovi jsou dva magnetky, které jsou přilepeny na kole. Jsou umístěny tak, aby bylo zaznamenáno pootočení kola o polovinu otočky. Jakmile je zaznamenána detekce magnetu senzorem, na výstupním (modrém) vodiči je staženo napětí na 0V, a tudíž se objeví logická nula. Je implementováno přerušení na spadnutí hrany, při kterém se inkrementuje ujetá vzdálenost.

	Využití
`Pin Mux Control`	GPIO (Alternative 1)
`Piny MCU`	PTD2
`Konfigurace pinu`	GPIO, pull up
`Přerušení`	Ano, falling edge

Tabulka 3.8: Hallův senzor

3.5 SD slot

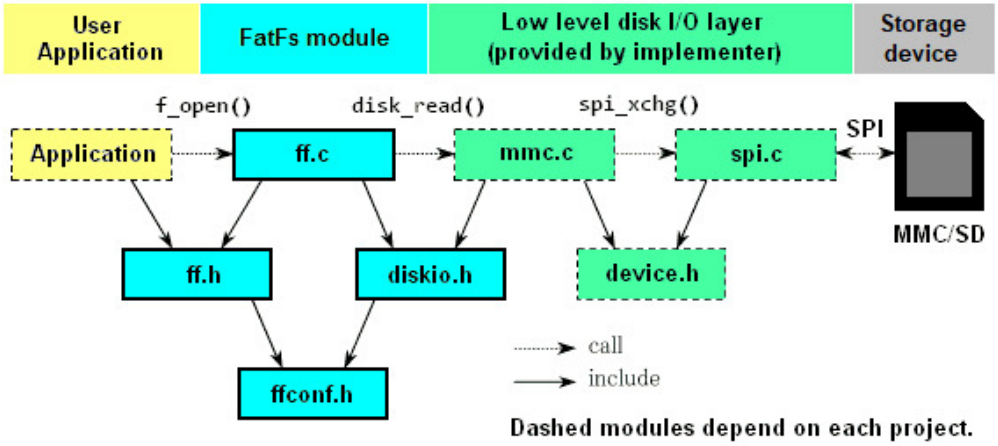
Pro co nejjednodušší zobrazení výsledné mapy bylo rozhodnuto nahrát mapu na SD kartu, kde bude jeden soubor s názvem *MAP.csv*. Díky tomuto formátu je mapa čitelná bez nutnosti zásahů od uživatele, který s mapou pracuje. Pro uložení na SD kartu je třeba SD slotu, který je připojen k MCU a díky tomu může robot mapu uložit.

Jedná se o modul čtečky karet microSD s označením SparkFun BOB-00544, což je malé zařízení určené pro snadné a rychlé čtení a zápis dat z mikroSD karet. Tento modul je navržen tak, aby byl snadno použitelný. Na následujícím obrázku je fotografie tohoto modulu.

Obrázek 3.20: Modul čtečky karet microSD - SparkFun

Modul čtečky karet microSD BOB-00544 je vybaven SPI rozhraním, což zajišťuje vysokou rychlost přenosu dat a umožňuje snadné připojení k řídícím jednotkám. Modul také obsahuje logiku pro detekci karty, kde při absenci karty je pin *CD* stažen na logickou nulu, zatímco když je karta zasunuta ve slotu, stav tohoto pinu je logická jednička.

Pro vytváření souborů, zápisu do souboru a obecně práci se souborovým systémem na kartě je třeba využít knihovnu FastFS[7]. Ta poskytuje základní operace při práci se složkami a soubory. Na následujícím obrázku je k nahlédnutí schéma, co knihovna obsahuje a co si uživatel této knihovny musí dopsat.



Obrázek 3.21: Schéma knihovny FastFS, která zajišťuje korektní práci se souborovým systémem na SD kartě. [11]

Uživatel musí mít naimplementovanou aplikaci, která si volá knihovní funkce. Zároveň je ale třeba dopsat komunikační protokol SPI. Následující tabulka pak ukazuje, na jaké piny je slot zapojen a do jakých módů se musí piny nastavit.

	Využití
`Pin Mux Control`	SPI0 (Alternative 1)
`Piny MCU`	PTC4(SS), PTC5(SCK), PTC6(MISO), PTC7(MOSI),

Tabulka 3.9: Zapojení SD slotu

Můžete si povšimnout, že pin *CD* nevyužíváme. Komunikační protokol je uzpůsoben tak, že data při neodpovídání slotu zasílá nepřetržitě, přičemž svítí LED na mikrokontroléru značící odesílání dat. Takže pokud by se nám stalo, že zapomeneme SD kartu vložit, můžeme ji vložit i tehdy, kdy MCU data ukládá.

Poslední zmínka se týká napájení, kde je třeba 3.3 voltů, podobně jako laserové senzory. Takové napájení není možno z baterie - z té máme proud snížený pouze na 5V. K tomuto využíváme mj. Arduino desku, ze které je stabilních 3V3 zajištěno. Více o této desce si povíme v následující kapitole.

[7]Odkaz na knihovnu: http://elm-chan.org/fsw/ff/00index_e.html

3.6 Arduino

Arduino je vývojová platforma založená na mikrokontroléru. Deska obsahuje vše, co je potřeba pro vývoj elektronických projektů. Mezi jeho výhody patří jednoduchost použití, nízká cena a obrovské množství dostupného materiálu a knihoven. Desky Arduino se liší v závislosti na velikosti, funkcionalitě a ceně a jsou vhodné pro různé úrovně zkušeností od začátečníků po profesionální inženýry. Arduino desky lze programovat pomocí Arduino IDE, což je multiplatformní vývojové prostředí, které obsahuje editor kódu, kompilátor a ladící nástroje.

Kvůli delšímu záseku s laserovými senzory jsem se rozhodl využít jeho výpomoc. Bylo potřeba dodělat okolní mapování, které navazovalo na data z tohoto měření. Využitím arduina ušetříme rovněž procesorový čas, kdy bychom pouze čekali na přijetí dat ze senzorů. K arduinu jsou tak připojeny dva laserové senzory, které jsou umístěny na levé a pravé straně robota.

	Využití
`Piny MCU`	PTC1 (CLK), PTC2 (SDA)
`Pin Mux Control`	kPORT_MuxAlt2 (Alternative 2)
`Konfigurace pinu`	Input/Output

Tabulka 3.10: Arduino zapojení s KL25Z

Životní cyklus

Arduino nejprve inicializuje oba senzory a přidělí jim IIC adresy. Levý má adresu 0x30 a pravý 0x31. Senzory se dají vypnout a restartovat, pokud na pin s označením X (resp. XSHUT) připojíme kabel a nastavíme hodnotu do logické nuly. Tak je senzor ve stavu restartování a my tak můžeme druhému nastavit adresu. Pak uděláme to samé pro senzor druhý.

Následně s využitím knihovny *Adafruit_VL53L0X.h*, konkrétně funkce rangingTest(), provedeme měření pro jeden i druhý senzor. Data pak odesíláme na naší primární desku KL25Z, která má IIC adresu 0x7E a tato měření se opakují po celou dobu, dokud arduinu neodebereme proud.

Propojení s KL25Z

Desky, jak již bylo řečeno, jsou propojeny skrze protokol IIC. Arduino má funkci „mastera“[8] a odesílá data na desku KL25Z, která je naopak nastavena jako „slave“[9]. Deska KL25Z má implementováno DMA (*Direct Memory Access*), což je technologie, která umožňuje přímý přístup k paměti bez potřeby procesoru. Je řízena speciálními obvody v počítači, které

[8]Řídící zařízení (master) v I2C sběrnici je zařízení, které řídí celou komunikaci na sběrnici a vysílá řídící signály. Master zařízení zahajuje přenos dat tím, že odešle startovací signál, vybere příjemce a odesílá nebo přijímá data. Master také řídí takzvaný tok dat, což znamená, že určuje, kdy mohou začít být data odesílána nebo přijímána na sběrnici a kdy se mají zastavit.

[9]Podřízené zařízení (slave) je na druhé straně zařízení, které přijímá data od masteru a odpovídá na ně. Slave neovládá komunikaci na sběrnici, ale pouze reaguje na signály a data zaslaná masterem. Slave zařízení může být připojeno ke sběrnici I2C až 127, přičemž každé zařízení má svou jednoznačnou adresu

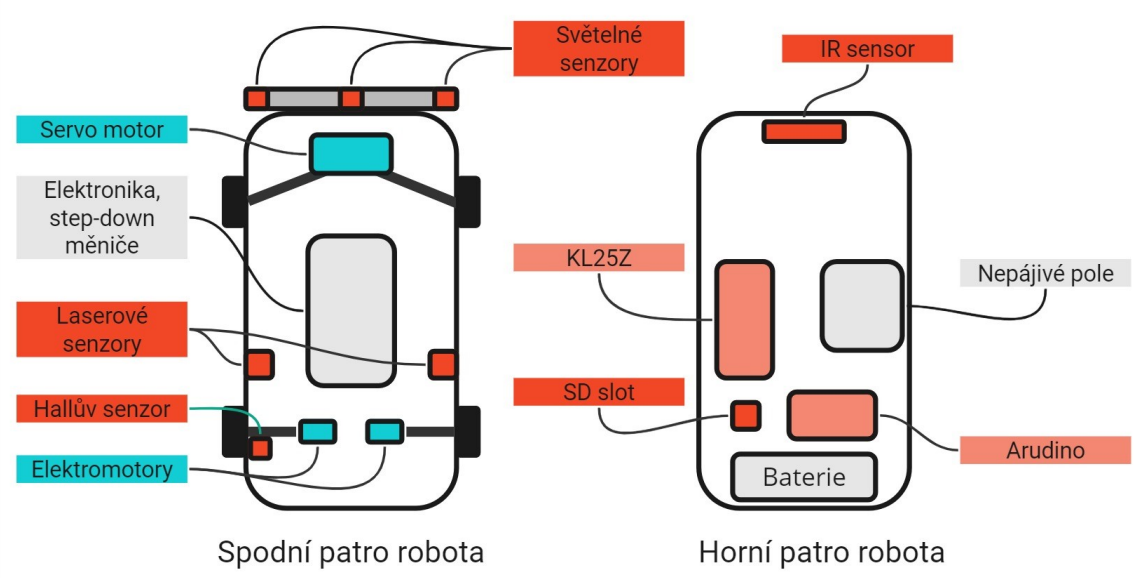
umožňují přenos dat mezi periferiemi a pamětí bez zásahu procesoru. Tento proces je velmi rychlý a účinný, protože procesor není zatížen přenosem dat.

DMA ukládá přijatá data do přiděleného pole, které je nastaveno na velikost 4 bajty. V poli se vždy udržuje pouze poslední, nejnovější, záznam a to ve formátu [idSenzoru, naměřenáHodnota, idSenzoru, naměřenáHodnota]. Pokud procesor potřebuje data, zpracuje prvně toto pole a pak s těmito hodnotami dále nakládá.

3.7 Výsledná podoba robota

Na následujícím obrázku 3.22 lze vidět rozložení komponent na robotovi. Ten sestává ze dvou pater, přičemž spodní patro je využito primárně pro rozvody elektřiny a mechanické části, jako jsou např. motory. Vrchní patro je pak určeno pro spojení veškerých senzorů s deskou, plus napájení senzorů, které je distribuováno skrze nepájivé pole.

Snažil jsem se dostat i baterii do spodního patra, protože je nejtěžším článkem, mimo konstrukci jako takovou. Pokud by se mi ji podařilo dostat dolů doprostřed, snížil bych těžiště robota a tak by mohlo být rozjíždění energeticky úspornější a zatáčení přesnější. Neřešíme zde ale závodění a tak rychlost a větší náklon do zatáček nepůsobí žádné problémy.



Obrázek 3.22: Diagram robota

Na obrázku, pro větší přehlednost, nejsou ukázány rozvody elektřiny a propojení jednotlivých komponent s deskami. To by celý obrázek kompletně znejasnilo. Samozřejmě veškeré komponenty potřebují napájení a propojení s deskami, buďto s jednou, nebo druhou. Tím, že nebylo využito žádné nástavby a je vše zapojeno napřímo drátky, je výsledný robot lehce matoucí svou složitostí. Pouze využitím kamery bychom odstranili velký počet drátků vedoucích od nepájivého pole, kde máme rovněž napájení pro veškeré periferie, k senzorům.

Elektřina tedy proudí z baterie do pole s elektronikou, která skýtá waga a step-down měniče. Odtud proudí do nepájivého pole, kde přes stabilizátor napětí proudí do všech periferií. Napájení pro desku KL25Z bereme přímo ze step-down měniče, neb má i výstup USB. Arduino má pak napájení skrze Vin.

Kapitola 4

Popis realizace

Pro lepší orientaci ve zdrojových textech je přiloženo rozepsané schéma adresářové stromové struktury bakalářské práce, které je k nalezení na konci této práce, tedy zde A.

Robot se probouzí k životu zapojením baterie a přivedením elektřiny do celého systému. Tímto se nastartuje mikrokontrolér, který provede úvodní inicializaci. Ta skýtá zapnutí hodin, inicializaci veškerých pinů, nastartování ladicích výpisů (které fungují pouze v případě zapojení s PC) a rozsvícení LED na MCU bílou barvou, značící čekání na pokyny. Poslední věcí je inicializace dotykového senzoru. Pak už se čeká na další akce. Pro spuštění dalších úloh musíme přidržet prst v okolí středu dotykového senzoru. Zapojením baterie se rovněž rozsvítí světla vepředu na robotovi a rozjedou se některé senzory. Jelikož je ale MCU ve fázi čekání, nepůsobí to žádné ruchy a nechtěné akce.

K nalezení: složka *board*

4.1 Inicializace

Složky: `startup, map`.

Jakmile se poprvé dotkneme senzoru, nastartujeme inicializaci desky (*startupBoard*()) a periferií (*startupPeripherals*()). Postupně probíhá inicializace:

- PWM pro všechny motory, včetně servo motoru.
- Přerušení pro Hallovy senzory.
- I2C rozhraní v módu slave, sloužící pro komunikaci s další deskou.
- ADC převodník pro IR senzor.

Dále se nastavují připojená periferní zařízení. Těmi jsou světelné senzory, motory a servo motor.

Motory (3.3.1)

`startup_peripherals.c/initMotors`()

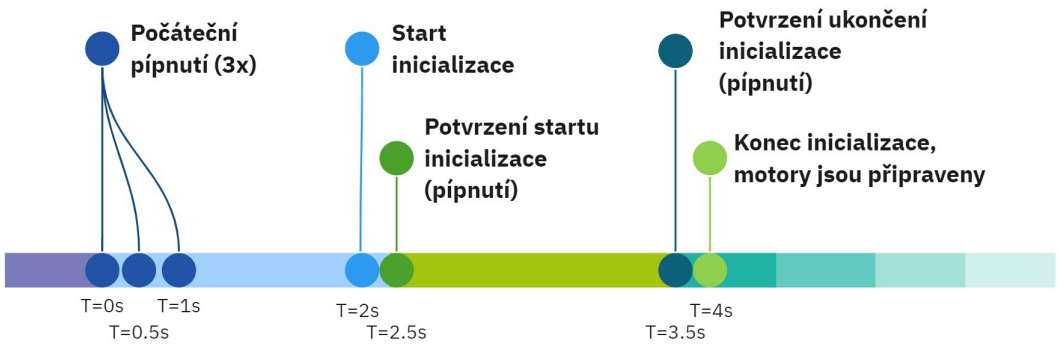
Než začne zasílání patřičných signálů na ESC se záměrem řízení motoru, je třeba korektní inicializace. Mimo správného zapojení, nastavení příslušných pinů a jejich módů a dalších záležitostí je třeba inicializovat ESC, čímž se nastartuje.

Při zapojení baterie vydá ESC inicializační znělku–tři krátké, postupně zvyšující se tóny. Tím dává najevo, že očekává inicializaci, která probíhá následovně. Je třeba v krátké době

(po další pípnutí) projet rozmezí (pulsní šířky) signálů, od nejnižší hodnoty po nejvyšší. Jako první se zasílá pulsní šířka 2.88%, přičemž se provádí 100 iterací, při kterých se k šířce připočte šířka 0.06% a skončí se na cca 11%. V každé iteraci se setrvá po dobu 2 ms, aby došlo ke korektnímu rozeznání ESC modulem. Jakmile ESC zaznamená nejnižší hodnotu, pípne poprvé, jakmile se dokončí inicializace, pípne podruhé a tím jsou motory připraveny k pohybu.

Může se stát, že ESC identifikuje na jednom motoru vrchol inicializace dříve a jeden motor se tak roztočí. Setkal jsem se i s tím, že to udělal u obou a robot se mi rozjel velice rychle po stole, narazil a urval světelné senzory vepředu. Proto doporučuji, aby robot nestál na povrchu při inicializaci. Osvědčilo se mi položit robota na zem až tehdy, kdy je kompletně inicializován. Dalším řešením by mohlo být zkrácení cyklu, čímž by došlo ke zmenšení maximální rychlosti a motory by nemohly jet tak rychle. Je využito ale pouhých cca 40% z této škály, takže by to ve výsledku nebyl problém.

Na následujícím obrázku 4.1 je nástin časové osy inicializace motorů. Časové hodnoty jsou spíše ilustrativní, jelikož start inicializace závisí na aplikaci. Na startu závisí potvrzení startu inicializace. Každopádně mezi startem inicializace a koncem inicializace by měla být alespoň vteřina, kdy se postupně zvětšuje pulsní šířka, aby došlo ke korektní inicializaci.



Obrázek 4.1: Časová osa inicializace motorů

Servo motor (3.3.2)

`startup_peripherals.c/initServo`()

Inicializace servo motoru není potřeba. Pokud má servo dostatečné napětí pro sepnutí spojky a pootočení, není třeba ničeho dalšího a servo je připraveno.

I přes to bylo implementováno otočení zcela doprava a následně doleva, abychom došlo k ověření už při inicializaci, že se servo chová korektně a dle očekávání. Navíc, kdyby kolům něco překáželo, robot stojí na místě a nedojde tak k havárii.

Světelné senzory (3.4.1)

`startup_peripherals.c/startupSensorCapture`()

Světelným senzorům se inicializují piny, zapnou se hodiny a nastaví se zachycení vstupu na vzestupnou hranu. Zapnutím přerušení jsou senzory již plně aktivní a signály jsou průběžně čteny. Senzory zasílají signály už při zapojení baterie, ale až provedením této inicializační funkce se začnou data číst a zpracovávat.

Infračervený senzor (3.4.2)

`startup_board.c/init_adc()`

SHARP senzoru se inicializují hodiny a konfigurace je nastavena na ADC. Inicializace senzoru není třeba, neb nefunguje na protokolech, ale přímo na spojitém signálu. Vyčte se ze senzoru napětí a přepočítá se na vzdálenost.

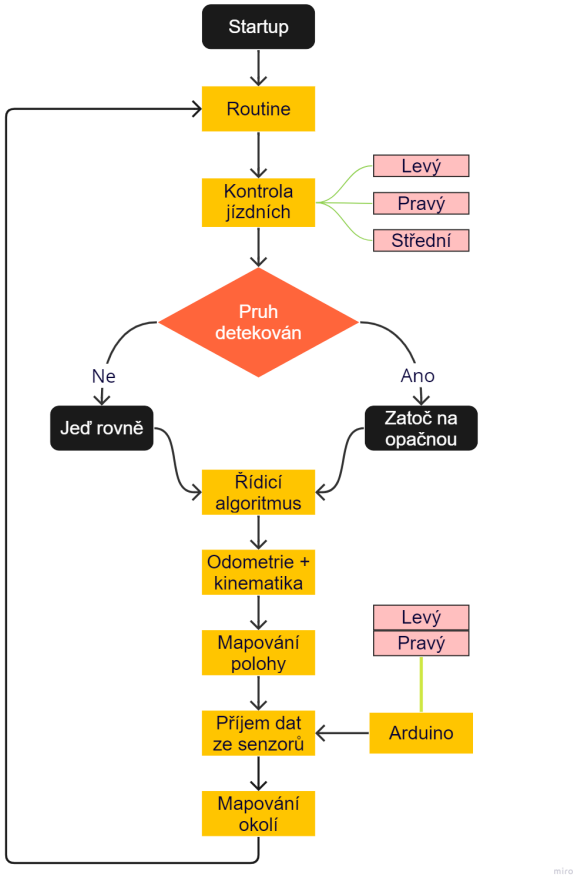
4.2 Hlavní programová smyčka

`routine.c`

Hlavní smyčka zastírá největší část kódu, která se vykonává neustále dokola, dokud nenastane určitý jev, který přesune program do závěrečné fáze. Tím, že je projekt psán bare metal, neuvažujeme žádné procesy, které by vznikaly a zanikaly a tak potřebujeme jednu hlavní nekonečnou smyčku, ve které se v návaznosti budou provádět dané operace, které si teď přiblížíme.

Smyčka sestává z následujících sekcí, přičemž ty „velké“ a složité jsou odděleny v sekcích samostatně, aby bylo dost prostoru je dostatečně probrat. Ostatní, ne tolik markantní a složité, budou popsány zde.

Diagram hlavní smyčky je znázorněn na následujícím obrázku 4.2. Jakmile hlavní smyčka skončí, program se přesune do závěrečné fáze, která skýtá uložení vytvořené mapy na SD kartu.



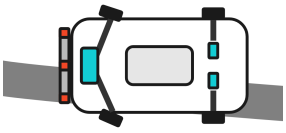
Obrázek 4.2: Diagram hlavní smyčky

Kontrola jízdních pruhů

`routine.c/checkLines`()

Postupně se prochází data z levého a pravého senzoru. Pokud je pod nějakým zaznamenána barva, tj. hodnota je větší jak 100 (zjištěno experimentálně), je tato informace zaznamenána pro řídící algoritmus.

Pokud je ale zaznamenána barva zároveň i pod středovým senzorem, zaznamená se tento údaj pro následné ostřejší zatáčení.



Obrázek 4.3: Detekce jízdního pruhu středovým senzorem

Řídicí algoritmus

`control_unit.c/controlUnit`()

V závislosti na straně detekce jízdního pruhu se odvrací přední kola na opačnou stranu. Pokud robot jede rychleji a má začít zatáčet, algoritmus zpomalí jeho rychlost a natočí kola. Pokud není po delší dobu zaznamenán jízdní pruh, robot se rozjede rychleji, přičemž směřuje rovně. Pokud robot zaznamená jízdní pruh pouze středovým senzorem, zastaví, vrátí se o kousek zpět a pak dále pokračuje.

Tím, že robot nevidí před sebe, protože se nepodařilo stihnout zprovoznit kameru, je rychlost velice omezena. Pokud je detekce jízdních pruhů pouze pod senzory, musí tomu být uzpůsobena rychlost robota, aby bylo možné pruhy detekovat a zároveň filtrovat (průměrovat) měřená data. To proto, aby nedocházelo k mylným úsudkům z měření a následným špatným rozhodnutím.

Příjem dat ze senzorů

`routine.c/processSensorData`() | `routine.c/checkStop`()

Data z Arduina jsou zapisována do pole za pomoci DMA 3.6. Jediná nevýhoda tohoto přístupu je, že procesor neví, jak jsou data stará.

Arduino zasílá data několikrát za vteřinu, takže by byla data neaktuální pouze v případě, kdy by se Arduino nějak zaseklo a data přestala chodit. To je ale ošetřeno tím, že při přečtení dat z pole jsou hodnoty v poli vynulovány. Pokud by tedy nastala nějaká podobná chyba, alespoň nezůstanou stará data jevící se jako aktuální. Data z levého a pravého senzoru jsou tedy **uložena do globálních proměnných**, odkud jsou dále zpracována.

Následně se čte hodnota infračerveného senzoru. Pak za pomoci rovnic pro výpočet vzdálenosti 3.2 je vypočteno, kolik cm před robotem je nějaká překážka. Pokud je detekována překážka do 15cm, vozidlo zastaví, aby nedošlo ke střetu. Pakliže po nějakém čase překážka pomine, vozidlo se opět rozjede a pokračuje v práci.

4.3 Pohyb robota

Změnu směru jízdy dokáže robot zajistit natočením přední nápravy a následným pohybem vpřed či vzad. V závislosti na natočení nápravy se robot pohybuje buď po přímce (to když není náprava nikterak natočena a jedeme tak rovně), což je znázorněno na obrázku 4.4, nebo po kružnici, znázorněno zde 4.5

Obrázek 4.4: Jízda rovně. Přímky kol přední nápravy se nikterak neprotínají s přímkou zadních kol.

Pro zjištění změny pozice vůči předchozímu výpočtu (jinak by nebylo třeba provádět další výpočty) je třeba dvou údajů – natočení nápravy (známo) a ujetá vzdálenost robota (potřeba výpočtu).

4.3.1 Výpočet ujeté vzdálenosti

`mapping.c/calcNewDistance`()

Pro výpočet ujeté vzdálenosti robota je znám vstupní parametr, čímž jsou otočky na levém zadním kole, které snímá Hallův senzor. Ujetou vzdálenost levého kola pak možno vypočíst následující rovnicí 2.6, přičemž D je přibližně rovno 6.7cm. Jedna otočka kola je tedy posun levého kola o cca 21cm.

Tato vzdálenost se rovná ujeté vzdálenosti robota pouze tehdy, pokud jede rovně. Protože, jak bylo ukázáno na tomto obrázku 2.6, pokud zatáčí, kolo ujede buďto více, nebo méně, než kolo střední, které je referencí pohybu robota.

Pak se použitím následujícího algoritmu získá ujetá vzdálenost středního kola, resp. robota při zatáčení.

4.3.2 Výpočet polohy robota

`mapping.c/calcNewPosition`()

Pakliže robot zatáčí, vypočítá se nejprve střed kružnice, kolem které se točí. Musíme si uvědomit, že je robot někde v prostoru a je nějak natočen, proto rovnice počítá se všemi proměnnými. To na základě předcházející polohy, kdy začal zatáčet, poloměru kružnice - který je znám - a úhlu směřování v době, kdy začal zatáčet - ten je rovněž znám.

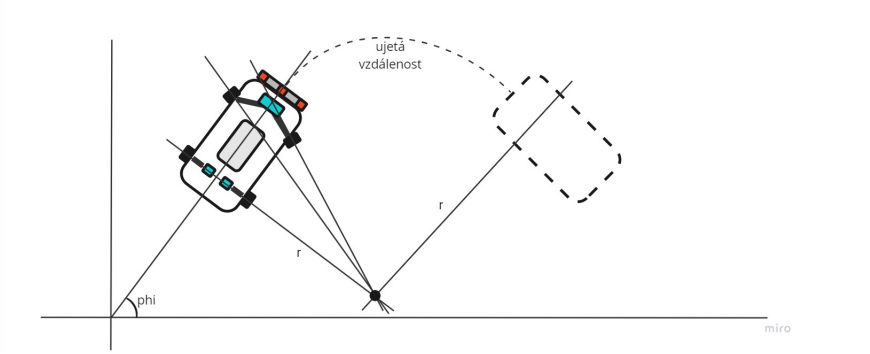
```
centerOfCircle[X] = prevPos.x - radius * sin(angle);
centerOfCircle[Y] = prevPos.y - radius * cos(angle);
```

Obrázek je pouze ilustrativní, aby přiblížil problematiku výpočtu.

Čárkovaný obrys auta je aktuální robota pozice, detailnější model je předcházející pozice.

Algorithm 1 Přepočet ujeté vzdálenosti při zatáčení

```
1: function calcNewDistance(turningSide, newDistanceLW)
2:     phi ← newDistanceLW/radius
3:     distanceLeftCenterWheel ← 4.5
4:     if turningSide == TURNING_LEFT then
5:         currentRadius ← distanceLeftCenterWheel + currentRadius
6:         angleHeading ← phi + angleHeading
7:     else
8:         currentRadius ← distanceLeftCenterWheel − currentRadius
9:         angleHeading ← phi − angleHeading
10:    end if
11:    newDistance ← phi ∗ currentRadius
12: end function
```



Obrázek 4.5: Zatáčení v prostoru

Následně je dopočítán posun robota na této kružnici, což vrátí souřadnice [x, y]. Následuje výpočet úhlu, který robot svíral na předchozí pozici se středem kružnice. Pak je vypočteno úhlové posunutí na základě ujeté vzdálenosti a poloměru kružnice. Sečtením těchto dvou úhlů se získá finální úhel, se kterým se bude počítat v následujícím algoritmu *finalTheta*. Novou pozici lze pak získat následujícím algoritmem (algoritmus nastiňuje i výpočet středu kružnice).

Algorithm 2 Výpočet pozice robota

```
1: function calcNewPosition(distance, newDistanceLW)
2:     if steer = GO_DIRECT then
3:         theta ← atan2(prevPos.y − centerOfCircle.y, prevPos.x − centerOfCircle.x)
4:         deltaTheta ← distance/radius
5:         finalTheta ← theta + deltaTheta
6:         newPos.x ← centerOfCircle.x + radius ∗ cos(finalTheta)
7:         newPos.y ← centerOfCircle.y + radius ∗ sin(finalTheta)
8:     else
9:         newPos.x ← prevPos.x + distance ∗ cos(angleHeading)
10:        newPos.y ← prevPos.y + distance ∗ sin(angleHeading)
11:    end if
12:    return newPos.x, newPos.y
13: end function
```

Tím se získala aktuální pozice robota, která musí být zanesena do mapy, o čemž pojednává následující kapitola.

4.4 Mapování

`map_operations.c`

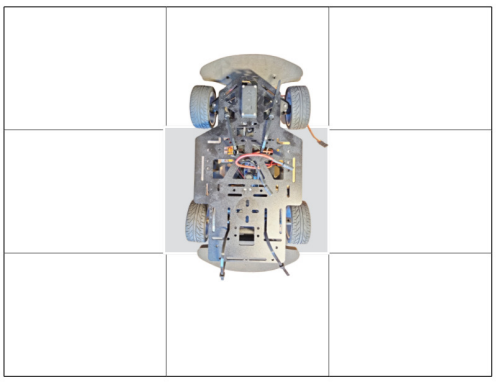
Pro naše účely nejlépe sedí metoda Grid-based (*GB*). Ta se zaměřuje na rozdělení mapy do mřížky buněk a použití dat z laserového senzoru k aktualizaci obsazenosti každé buňky. Pozice robota se pak stanoví identifikací buňky s nejvyšší pravděpodobností obsazenosti. Jedním z hlavních výhod GB je jeho jednoduchost. Mřížka obsazenosti se snadno interpretuje a snadno se s ní pracuje. Navíc, tato metoda má tendenci být méně náročná na výpočetní zdroje než jiné metody.

Na druhou stranu má několik nedostatků. Například, při malé velikosti buněk mřížky mohou být detaily mapy ztraceny. Kromě toho, pokud robot jezdí rychle nebo se pohybuje v prostředí s velkým množstvím překážek, může být obtížné udržet aktuálnost mapy. Proto je třeba korektně zvolit velikost políčka v závislosti na prostoru a množství překážek, či korigovat rychlost, aby nedocházelo ke ztrátám dat.

4.4.1 Reprezentace mapy

Mapa je tvořena bloky, které sestávají z polí. Každý blok má odkaz na 2D pole, které reprezentuje mapu a souřadnice bloku [x, y]. Počet polí v bloku je inicializován při překladu aplikace. Rovněž tak velikost jednoho pole. Jednotlivé bloky jsou dynamicky alokovány, což se děje na základě dvou faktorů. Jednak trasy, kudy projíždíme a jednak záznamů ze senzorů. Bud´ přejedeme do nového bloku, nebo senzor zaznamená překážku někde v dálce. Mapa na samém počátku je k nahlédnutí na obrázku 4.6.

Následující obrázek je ilustrací pro lepší představu mapy. Počáteční orientaci si můžeme zvolit a při překladu aplikace robot bude směřovat daným směrem v mapě. Zde je pro ukázku nastaveno počáteční směřování na 90°. Směr se nastavuje makrem *STARTING_HEADING_ANGLE* v souboru *global_macros.h*



Obrázek 4.6: Počáteční orientace robota v mapě

Struktura mapy

Zde je výtažek struktury bloku. Z objektů se nakonec nepoužívá *map_Road* a *map_Line*. K tomu by robot potřeboval vidět před sebe, aby mohl s jistotou říct, kde se jízdní pruhy

nachází. Je to ale ponecháno, kdyby to v budoucnu chtěl někdo využít. Zároveň není problém přidat jakýkoliv jiný záznam, který by se využíval.

```
//* Map possible objects
typedef enum _map_object {
  map_Empty = 0U,
  map_CurrentPosition,
  map_Track,  // I was there in prev. steps
  map_Road,   // Road (between lines)
  map_Line,  // Line detected on current field of the map
  map_Wall  // Wall detected on field
}map_object_t;

//* One block in a map
typedef struct map_blk
{
    map_object_t **block; // 2D field of map objects
    int corX;
    int corY;
}map_block;
```

4.4.2 Algoritmy mapování pohybu

Starý algoritmus mapování

Zprvu byly bloky propojeny jako obousměrně vázaný seznam. Každý blok měl ukazatele nahoru, doleva, doprava a dolů. Byl zde ale problém s nalezením a propojením nově vytvořeného bloku se sousedícími již alokovanými bloky.

Příklad 4.4.1. *Posuneme se (v blocích) nahoru, doprava a pak dolů. Takže máme aktivní blok, který by měl souřadnice [1, 0], protože jsme vedle úvodního. Jak se mohu dozvědět o stávajících sousedních blocích? Zda blok [0, 0] existuje?*

Napadlo mě řešení, kde by si robot ukládal projeté bloky do lineárního seznamu, čímž bych mohl v maximálně lineárním čase najít existující blok a propojit aktuální s nalezeným mezi sebou. Na to navázal problém s bloky, které byly inicializovány díky zaznamenání překážky senzory.

Pro záznam fixní překážky někde v dálce bylo zapotřebí alokovat několik bloků, než jsme se dostali k bloku, do kterého měla být informace uložena. Jenže co kdyby robot udělal otočku a najel do bloku, který byl inicializován tímto způsobem? Bylo by velmi obtížné zjistit, zda daný blok existuje, a přesun do něj by vyžadoval velmi složitý algoritmus. Navíc pro propojení bloků je třeba mít ukazatele na dané bloky. V řešení se však udržuje informace pouze o aktuálním bloku, a k dalším se lze dostat pouze přes ukazatele. Takže k nalezení bloku, do kterého se má robot přesunout, by musel jít algoritmus po již vytvořených blocích, aby získal jeho ukazatel, pročež by aktivní (kde se robot nachází) s tímto (do kterého se chce přesunout) propojil navzájem. Všechny bloky by musely být někde uloženy. Musel by však být vymyšlen algoritmus, který by byl schopen najít daný blok, a ten by byl zbytečně komplikovaný.

Nynější algoritmus

`common/hash_table.c` Proto jsem zvolil jinou strategii a všechny bloky jsou uloženy v hashovací tabulce. Ta sestává z klíče a hodnoty, kde klíč je jednoznačný identifikátor (ID) a hodnotou je ukazatel na blok v paměti. Jednotlivé bloky jsou tedy uloženy v tabulce a dostaneme se k nim na základě jejich souřadnic.

```
typedef struct Node {
    int key;
    map_block *value;
    struct Node *next;
} Node;

typedef struct HashTable {
    int size;
    int count;
    Node **table;
} HashTable;
```

ID je spočítáno rovnicí:

$$id = ((y * MAP\_ROWS * MAP\_COLUMNS) + x + 1) * 31 \tag{4.1}$$

Srovnání algoritmů

Díky nynější technice se ušetří navíc 4B místa na jednom bloku. Nyní jeden blok v hashovací tabulce zabere 12B. Ukazatel na blok = 4B, ukazatel na následující blok = 4B a ID (*int*) má opět 4B. Blok jako takový má ukazatel na 2D pole - to měl i ve starém řešení - a souřadnice bloku byly taktéž využívány, takže z tohoto pohledu jsou bloky stejné. Ve starém řešení měl ale jeden blok čtyři ukazatele (na každou stranu) a to bylo dohromady 16B.

Ve starším řešení bylo lehčí výslednou mapu uložit. Jednoduše bychom procházeli bloky od levého horního bloku a díky propojení bychom ji postupně celou prošli. Pokud by nastal jev, že by bloky netvořily čtverec či obdélník, při průchodu bychom chybějící bloky doplnili, aby se ve výsledném souboru dalo vyznat. Mimo ukládání zde bylo mnoho problémů, které by se musely lepit složitými algoritmy.

V novějším případě mapa nezabírá skoro žádné přebytečné místo a máme tak pouze záznamy o potřebných informacích. Pro uložení fixní překážky tak nemusíme alokovat mnoho bloků, abychom se dostali k požadovanému, ale jenom si alokujeme ten jeden a uložíme si ho do tabulky.

Složitější bude výslednou mapu ukládat, protože nebude stačit projít alokované bloky a ty postupně uložit. Budeme muset propočítávat, kolik bloků mezi dvěma existujícími zbývá doplnit prázdným místem. O tom ale více v kapitole 4.10.

4.4.3 Pohyb v bloku

`mapping.c/mapping()` | `map_operations.c/moveInMap()`

Jestliže se má robot přesunout do jiného pole, ať už v rámci aktivního bloku, či jiného, záleží na nastavení velikosti pole a výpočtech odometrie a kinematiky 2.2.2. Funkce `calcNewPosition()` 4.3.2 vrátila nové souřadnice. Na jejich základě je možnost určit, zda se má posunout do jiného pole, nebo nikoliv. To určuje následující algoritmus, který prvně určí souřadnice v bloku, kde by se měl robot nacházet. Následovně přepočte y složku, protože

2D pole v jazyce C je indexováno vzestupně směrem dolů (tedy čím větší hodnota, tím je níž). Zde je ale použit souřadný systém, kde y složka roste směrem vzhůru. Proto je potřeba ji přepočítat a otočit ji podle prostřední řádky. Pokud se přesunul do jiného pole, oproti předchozí pozici, je třeba určit, jakým směrem se má posunout o jedno pole v mapě.

Algorithm 3 Pohyb mezi poli v rámci jednoho bloku

```
1: function MAPPING
2:     moveToField_x ← (newPos.x/MAP_BLOCK_SIZE)%MAP_COLUMNS
3:     moveToField_y ← (newPos.y/MAP_BLOCK_SIZE)%MAP_ROWS
4:     moveToField_y ← MAP_ROWS/2 − (moveToField_y − MAP_ROWS/2)
5:     if moveToField_x ≠ currPosInBlk.Col or moveToField_y ≠ currPosInBlk.Row then
6:         direction ← getDirection(moveToField_x, moveToField_y)
7:         moveInMap(direction)
8:     end if
9: end function
```

4.4.4 Přemístění do jiného bloku

`map_operations.c/moveBetweenBlocks`()

Pokud se přesouvá do jiného bloku, vypočteme si jeho ID na základě souřadnic bloku. Následně je třeba ověřit, zda-li blok již existuje. Pokud ne, dojde k alokování a inicializaci, a následně je uložen do hashovací tabulky. Dále, už v obou případech, se nastavuje blok jako aktuální v globální proměnné *currentBlockInMap* a dojde k zanešení polohy robota do již aktivního bloku. Zde je lehký nástin jak probíhá pohyb mezi bloky:

Algorithm 4 Pohyb mezi jednotlivými bloky

```
1: function moveBetweenBlocks(direction)
2:     X ← 0
3:     Y ← 0
4:     getCoordinates(direction, &X, &Y)
5:     ID ← getUniqueID(X, Y)
6:     pBlockToMove ← searchItemInHT(ID, X, Y)
7:     if pBlockToMove == NULL then
8:         pBlockToMove ← createNewBlock(ID, X, Y)
9:     end if
10:    currentBlockInMap ← pBlockToMove
11: end function
```

Velikost jednoho bloku se dá vypočítat následující rovnicí:

$$Velikost\_bloku = pocet\_radku * pocet\_sloupcu * velikost\_pole^2, \tag{4.2}$$

Pokud algoritmus řekne, že se má robot přesunout do jiného bloku, musí nejprve zjistit, zda daný blok není již inicializován.

Pokud je, nastaví tento blok jako aktivní a nastaví si svou pozici v bloku - *currPosInBlk[X]*, *currPosInBlk[Y]*. Pokud blok ještě neexistuje, musí se alokovat, inicializovat a uložit do tabulky. Následně se nastaví jako aktivní a přenastaví své souřadnice v bloku.

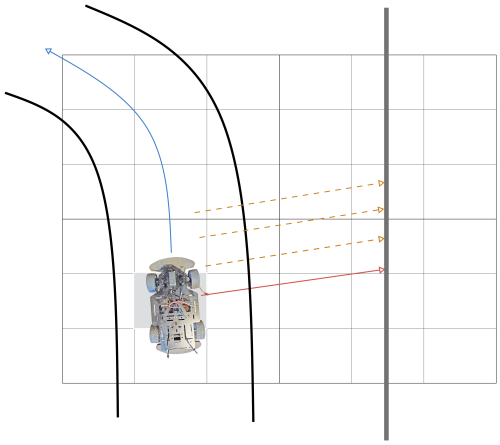
Je zde užit dvojí souřadnicový systém. Jeden má za úkol udržovat informace o pozici robota v aktivním bloku (*currPosInBlk[X]*, *currPosInBlk[Y]*), a druhý má za úkol udržovat informace k danému aktivnímu bloku a to, jaké jsou jeho souřadnice v mapě (*currentBlockInMap.corX*, *currentBlockInMap.corY*).

4.4.5 Mapování okolního prostředí

`mapping.c/saveSensorData()` | `map_operations.c/saveBarrierToMap()`

Bez mapování okolí by bylo mapování trasy robota zcela zbytečné, protože právě z okolního prostředí se dá určit, zda-li jsou výpočty pohybu robota korektní, či nikoliv. V metodách SLAM se běžně užívají algoritmy, které sledují reálnou a vypočtenou polohu a na základě toho jsou schopny korelovat chybovost odometrie, nebo kinematických výpočtů.

V řešení takové algoritmy nejsou využity, ale mapování okolí se provádí. Implementace těchto algoritmů by mohlo být rozšíření této práce, protože je to obsáhlé a složité téma, které by bylo na samostatnou bakalářskou práci. Proces mapování je znázorněn na následujícím obrázku 4.7.

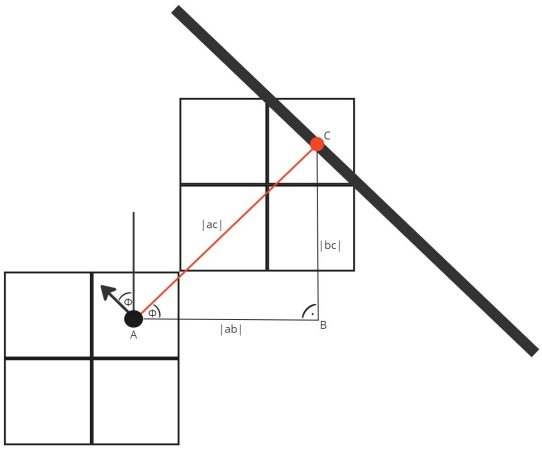


Obrázek 4.7: Proces mapování

Pokud je zaznamenána překážka v jedné z proměnných *leftLaserValue*, nebo *rightLaserValue*, voláme funkci *saveBarrierToMap()*. V té se musí nejprve spočítat, kde se záznam vyskytuje s vědomostí úhlu, kterým robot směřuje a zaznamenanou vzdáleností. Dále je třeba vypočítat, do jakého bloku a pole záznam spadá a konečně dojde k jeho umístění.

Výpočet, o kolik polí se zaznamenaná překážka vyskytuje

Na začátku je znám úhel směřování a vzdálenost překážky. V závislosti na tom, který ze senzorů (levý, nebo pravý) překážku zaznamenal, je upraven úhel přičtením, resp. odečtením 90°. Vypočítáme přilehlou stranu /*ab*/ a protilehlou stranu/*bc*/, na jejichž základě je možné dopočítat, o kolik řádků a sloupců je překážka vzdálena vůči poloze robota. Pro snadnější představu vizte následující obrázek 4.9.



Obrázek 4.8: Mapování okolního prostředí. Výpočet souřadnice detekované překážky za použití trigonometrických rovnic.

Přilehlá strana je vypočtena rovnicí:

$$|ab| = |ac| * \cos\phi \tag{4.3}$$

Protilehlá pak obdobně:

$$|bc| = |ac| * \sin\phi \tag{4.4}$$

Výsledky jsou zatím posuny v centimetrech, ty je třeba převést na posuny v polích za pomoci jednoduchého vydělení centimetrů s velikostí bloku. Následným přičtením k aktuální pozici jsou získány souřadnice překážky a pak jsou finálním propočtem získána pole v daném nalezeném bloku. Může se to zdát lehce matoucí, proto je zde uveden algoritmus 5, který bude snad stravitelnější.

Algorithm 5 Zanesení detekované překážky do mapy

```
1:  function saveBarrierToMap(laserSide, value, angle)
2:      if laserSide == left then
3:          angle ← angle − 90
4:      else
5:          angle ← angle + 90
6:      end if
7:      ab ← value * cos(angle)
8:      bc ← value * sin(angle)
9:      shiftInX ← bc/MAP_BLOCK_SIZE
10:     shiftInY ← ab/MAP_BLOCK_SIZE
11:     fieldX ← currPosInBlk.Col + shiftInX
12:     fieldY ← currPosInBlk.Row + shiftInY
13:     shiftInXBetweenBlks ← fieldX/MAP_COLUMNS
14:     shiftInYBetweenBlks ← fieldY/MAP_COLUMNS
15:     blockX ← currentBlockInMap− > corX + shiftInXBetweenBlks
16:     blockY ← currentBlockInMap− > corY + shiftInYBetweenBlks
17:     block ← getBlock(blockX, blockY)
18:     block[fieldY][fieldX] ← mapWall
19: end function
```

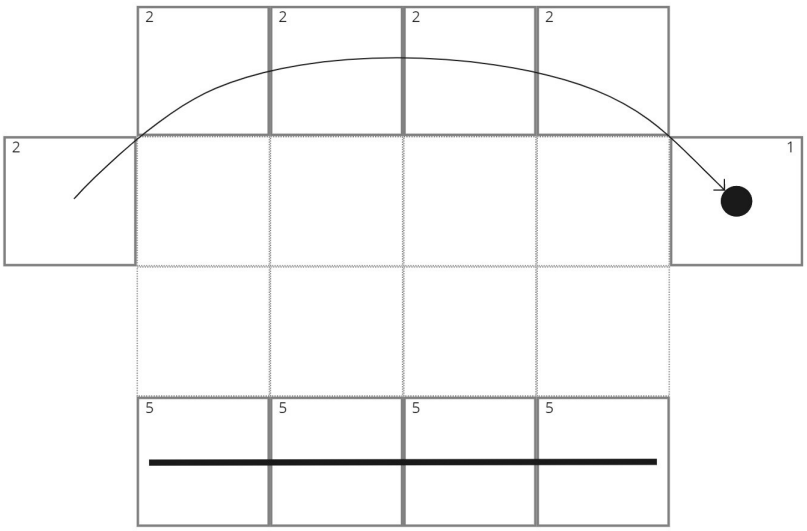
4.4.6 Ukládání mapy

`save_map.c/saveMap`()

Mapa je ukládána na SD kartu 3.5 ve formátu .csv, aby bylo možné mapu ihned číst a lehce zpracovávat. Zápis do souboru je možný využitím funkce *f_write(string, size)*, kde *string* je řetězec, který má být zapsán do souboru a *size* je délka tohoto řetězce. Zde nebude rozebráno do podrobna, jaké veškeré funkce jsou potřeba pro korektní zápis, spíše je zaměření na ukládání bloků. Tato funkce je ale uvedena z důvodu, že je potřeba řetězce. To je důležité dvakrát podtrhnout, protože díky tomu je algoritmus velice komplikovaný. Problematika je představena na následujícím příkladu.

Příklad 4.4.2. *Robot vyjel z levého bloku a pokračoval do pravého, přičemž zaznamenal fixní překážku napravo od něj, jak je vidno z obrázku 4.9. V rozích bloků jsou uvedena čísla, která jsou pouze pro uvědomění, že blok musel být alokován. Jinak jsou daná čísla v jednotlivých polích bloku, kudy robot projížděl, či kde překážku zaznamenal.*

Bloky uprostřed jsou vyznačeny tečkovaně, což značí, že se v daném místě nic nenachází a proto tyto bloky nebyly alokovány a tedy neexistují. Jsou zde pouze pro lepší představu a aby nebylo matoucí, že někde dole jsou další bloky.

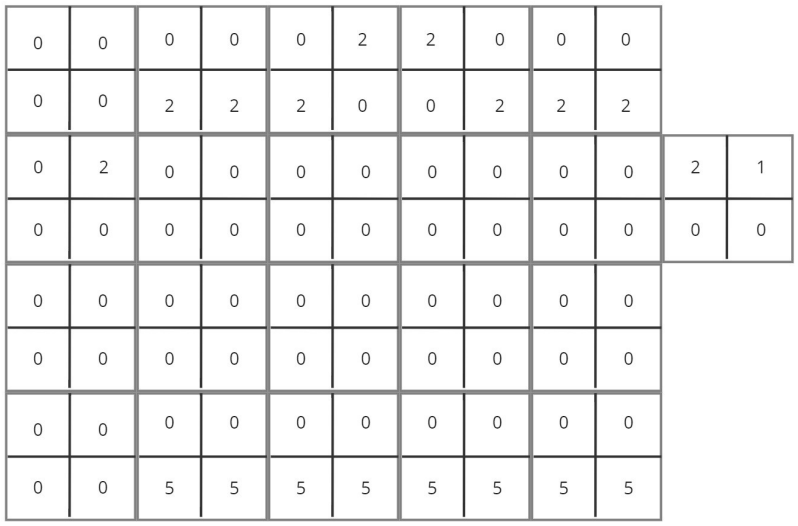


Obrázek 4.9: Příklad mapy, kterou máme uložit na SD kartu.

Jednotlivé bloky jsou roztříštěny po paměti MCU, konkrétněji jsou někde na haldě. Ukazatele na tyto bloky jsou uloženy v hashovací tabulce. Aby bylo možné bloky korektně uložit do souboru, bude nejprve potřeba veškeré alokované bloky vložit do pole, kde si dojde k jejich seřazení podle y souřadnice a následně dle x souřadnice.

Aby bylo možné bloky zapsat do souboru, nejprve je třeba vytvořit již zmíněný řetězec, který by vypadal pro první dva řádky z obrázku 4.10 následovně:

string="0,0,0,0,0,2,2,0,0,0,\n,0,0,2,2,2,0,0,2,2,2,\n"

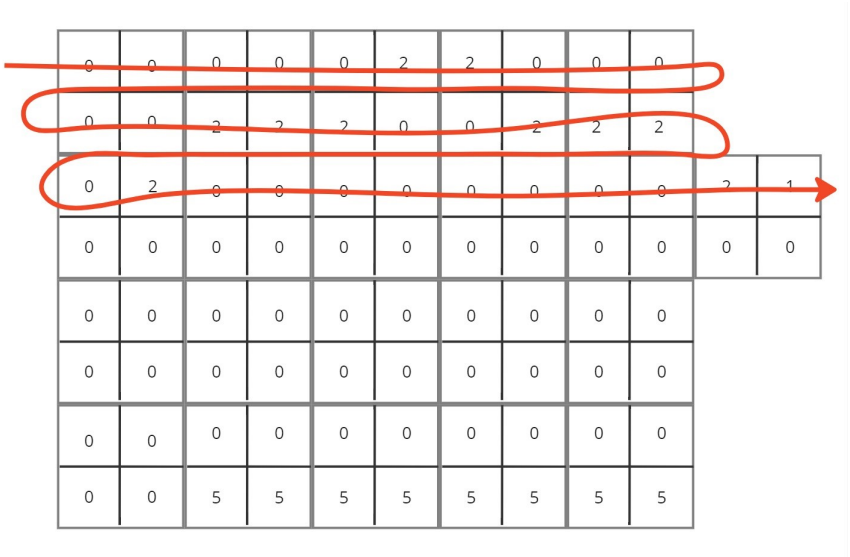


Obrázek 4.10: Mapa doplněná o chybějící části, která má být uložena.

Díky tomuto příkladu je doufám jasnější, co je úskalím této problematiky. Nejprve je třeba vytvořit prázdné bloky, které jsou buďto mezi dvěma bloky, které jsou vzdáleny o více jak jeden blok (to jest pro příklad druhý řádek bloků). Dále vytvořit prázdné bloky, pokud je mezi bloky někde mezera (třetí řádek bloků a indexuji od 1). A na závěr, pokud je v nějakém řádku nejlevější blok více vlevo, než první blok v řádku (tedy x souřadnice prvního bloku v řádku je větší, jak x souřadnice bloku nejvíce vlevo), je třeba doplnit na začátku patřičný počet bloků, aby došlo ke korektnímu zarovnání.

Následně se prochází postupně veškeré bloky, vždy po jednotlivých řádcích bloků a po jednotlivých řádcích polí, a ukládáme hodnoty z polí do řetězce. Začíná se prvním řádek bloků, prvním řádky polí. Pak, pokud má blok vícero řádků polí, se pokračuje stejným řádkem bloků, ale dochází k posunutí na další řádek polí. A tak se postupně uloží celá mapa.

Pokud se koukáte do kódu, tak se tam děje všechno zároveň. Tedy - prochází se celé pole bloků. Postupně se zpracovávají řádky bloků, přičemž pokud je blok první a některý je více vlevo, zanáší se řádek polí prázdných znaků (a to vždy pro každý řádek toho bloku). Pak se zanáší znaky z aktuálně zpracovávaného řádku polí. Pokud se přejde do pole, kdežto předcházející blok je vzdálen, tedy není sousedící, zanáší se opět prostorová výplň. A takto se zpracovávají postupně veškeré řádky bloků. Následující obrázek 4.11 snad pomůže k lepšímu porozumění.



Obrázek 4.11: Ukázka průchodu při ukládání mapy.

4.4.7 Rovnice výpočtu nastavitelných parametrů mapy vzhledem k místu v paměti

Bloky mají souřadnice a jsou ukládány do hashovací tabulky, která je staticky alokována na 50 záznamů. Tato velikost je naprosto dostatečná pro velké místnosti. Naopak je velikost tabulky možná nadimenzovaná. Pro větší haly by se dalo využít přealokování tabulky pro více záznamů, aby bylo vyhledávání rychlejší. Prázdná inicializovaná hašovací tabulka zabere 256B místa na haldě (heapu) v paměti z celkových 14 kB. Máme tak k použití dalších 13,77kB na jednotlivé bloky. Jak už jsme si vypočetli, záznam v hashovací tabulce zabírá 16B. Jeden blok zabírá 12B (ukazatel na 2D pole, souřadnice = 2x integer). Velikost 2D pole v paměti závisí na počtu řádků a sloupců. Výslednou velikost můžeme vypočítat následujícím vztahem:

$$velikostBloku = (pocetRadku * 4) * (pocetSloupcu * 1) \tag{4.5}$$

Dle prostoru, který má robot projet, můžeme spočítat, jestli nastavení parametrů bloků dokáže pokrýt celý prostor, či robotu dojde paměť.

Velikost pole bude stěžejní pro výsledný záznam v poli. Určení, má-li být informace do mapy zaznamenána, je na základě počtu výskytu informace v poli.

Příklad 4.4.3. *Než přejedeme do dalšího pole, zaznamenáme překážku v devíti z dvaceti měření. To nesplňuje podmínku 75% výskytů z měření a proto informaci neukládáme*[1].

Velikost pole bloku by měla být minimálně 1 otočky kola, takže uvažujme minimální velikost pole řádově 10cm. Po zvolení velikosti pole a počtu řádků a sloupců si můžeme

[1]Pozn.: V mém řešení jsem tuto logiku nakonec neimplementoval. Jakmile senzor zaznamená překážku, zanášíme ji do mapy. Bylo by to ale lepší, přesnější řešení. Fungovalo by takto: z každého měření by se vzala hodnota, i ta maximální (tzn. že senzor nic nezaznamenal). Pro každé pole bychom ale museli počítat, kolikrát jsme pro dané pole měření provedli, abychom při záznamu mohli vypočítat, jaký je průměr daných měření. Ve výsledku bychom kupř. zaznamenali v určitém poli 4x překážku z 5 měření v daném bloku a tak bychom zanesli informaci o fixní překážce do daného pole.

vypočítat, kolik bloků bude využito.

$$pocetBloku = \frac{prostor(cm^2)}{pocetRadku * pocetSloupcu * velikostPole} \tag{4.6}$$

Pak následující rovnicí dopočítáme, kolik místa v paměti zhruba namapování zabere.

$$pocetBajtu = pocetBloku * velikostBloku * 16B \tag{4.7}$$

Pokud je tato výsledná velikost větší, jak 13.77kB, musíme zvětšit jeden z parametrů - počet řádků, počet sloupců nebo velikost pole.

4.4.8 Návrhy na zlepšení algoritmů

Algoritmy na propočítávání pohybu robota a mapování samotné využívají algoritmus podobný algoritmům *LineDDA*[2] a *CircleByPoints*[3], které se využívají v grafice pro svou jednoduchost. Jednoduchost z pohledu intuice a pochopení, avšak na výpočet jsou velice náročné, protože počítají jednak s desetinnými čísly *float* či *double*, užitím *sin* a *cos* funkcí se výpočty enormně zpomalují a konečně využíváním funkcí jako jsou kupř. *round()*, které *float* převádí na *int*. Tyto funkce jsou využity v poměrně velkém zastoupení v tomto řešení, a proto by šlo algoritmy zrychlit. Byla snaha psát práci co nejoptimálněji, avšak některé propočty stále užívají zmíněné funkce.

Z těchto důvodů, které jsou pro MCU náročné, by bylo lepší používat například *floating point*[4] aritmetiku, nebo sofistikovanější algoritmy, které jsou šetrnější k náročnosti výpočtů. Například u počtů při zatáčení - kružnice - by se dal najít rozhodně lepší způsob.

Pohyb se počítá vždy v daném místě v prostoru. Je dán úhel, pod kterým se robot nachází vůči inicializační poloze, ujetá vzdálenost - je někde a je nějak natočen - a s tím počítají rovnice, čímž vyžadují použití právě goniometrických funkcí. Zrychlení výpočtů by se dalo zrychlit třeba tak, že by se pozice robota transformovala do středu souřadného systému, tam by se provedly výpočty spojené s rasterizací jednotlivých polí a na závěr by se vše vrátilo do původní polohy. To byla vůbec první myšlenka. Pohyb robota se ale počítá při ujetí nějaké vzdálenosti, což znamená, že dojde k pootočení kola, což se detekuje pouze jednou za polovinu otočení kola. Je tak dost času na dané výpočty a proto byla zvolena metoda, která je implementována.

Dalším zlepšením by mohlo být použití akcelerátoru a magnetometru, aby vypočítané hodnoty korelovaly více s realitou a nebyly to pouze „ideální“ hodnoty. Pokud by se robot nacházel na ledové ploše a roztočil motory, kola by se sice točila, takže by docházelo k mapování, ale robot by prakticky stál na místě. Mapování by tak nesedělo s realitou, čemuž by se mělo předcházet užitím správných technologií.

[2]Popis LineDDA algoritmu například zde: https://shorturl.at/bqCQU
[3]CircleByPoints algoritmus: https://shorturl.at/NOPU2
[4]Vysvětlení floating point aritmetiky zde: https://shorturl.at/vCG25

Kapitola 5

Experimentální část

5.1 1. úkol

Prvním úkolem pro robota bude nejzákladnější mapování - pouze pohybu. První úkol sestává ze tří částí, kdy robota pustíme pouze rovně, pak ho necháme udělat kolečko doleva a následně doprava.

Při prvním úkolu uvažujme nastavení parametrů mapy následovně:

- **Počet polí v bloku** = 10x10 polí
- **Velikost jednoho pole** = 10x10 cm

Jízda rovně

Pustíme robota rovně na vzdálenost 41 cm. Tím by se měl posunout o 4 pole doprava, jelikož musíme brát v potaz, že na začátku jsme uprostřed počátečního pole. Tudíž pro vyjetí z pole o rozměrech 10x10 cm je třeba ujet vzdálenost alespoň 5cm. Nebo-li užitím rovnice $posun = vzdalenost/velikostPole$ si můžeme vypočítat, že se máme posunout o 4 pole.

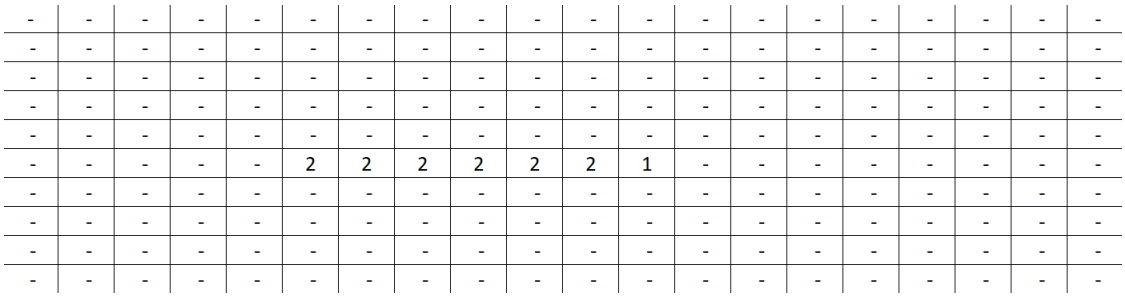
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	2	2	2	2	1
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-

Obrázek 5.1: 1. úkol - jízda rovně, pole = 10x10 cm

Mapování jízdy rovně probíhá bez problémů. Vlezli jsme se do jednoho bloku, takže nebylo třeba alokovat další blok napravo. Robot se zastavil po 48cm, protože otočky kol Hallovým senzorem se počítají pouze 2x za otočení kola. Tím se nám dostává poměrně velká chybovost.

Ta by se dala řešit enkodérem, který je schopen zaznamenávat otočky kola po x stupních (jsou různé typy enkodérů, některé jsou schopny zaznamenat otočení např. o 1 stupeň). Tak

bychom měli mnohem přesnější přehled o tom, v jaké pozici se kolo nachází a jakou vzdálenost jsme ujeli. Pokud bychom nastavili pole na 5x5 cm, výsledek by měl být následující: $posun = 41/5 = \lceil 9 \rceil$. Měli bychom se posunout o 9 polí.



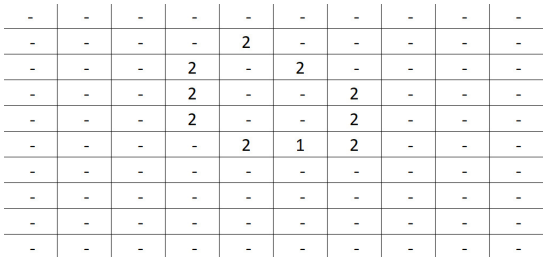
Obrázek 5.2: 1. úkol - jízda rovně, pole = 5x5 cm

Jak je patrno z obrázku, posunuli jsme se ale pouze o 6 polí. To je způsobeno chybou v algoritmu, který uvažuje posunutí se vždy maximálně o jeden blok. Měli bychom tak mít nastavenou velikost bloku minimálně jednoho kroku, což je polootočení kola (protože máme dva Hallovy senzory).

Tento algoritmický problém by se dal řešit výpočtem nové pozice v bloku. Zanesením pozice do mapy a propojením předcházející polohy s aktuální značkami trasy (tedy číslem 2). Byla by zde přípustná odchylka při zatáčení, ale trasa robota by nebyla s prázdnými místy mezi zanesenými polohami [1]

Jízda doleva

Pustíme robota a nastavíme mu zatáčení plně doleva, tj. za násilného užití funkce (*turnLeftCustom*(*MAX_STEER_LEFT*);). Při běžném běhu si tyto knihovní funkce pohybu robot volá sám na základě vnějších vlivů. Pro testovací účely ale vypneme modul *control_unit*, který se stará o řízení robota a nastavíme mu zatáčení na levou stranu. Mj. pole je opět nastaveno na 10x10 cm. Robotu nastavíme rovněž podmínku, aby zastavil po 110 cm, což je otočka o 360°. Vypočteno rovnicí $vzdalenost = 2 * \pi * 17.5 = \lceil 110cm \rceil$.



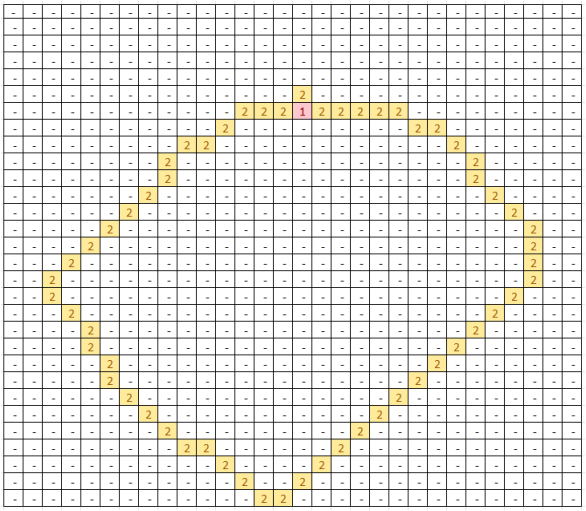
Obrázek 5.3: 1. úkol - jízda doleva

Je patrno, že se robot zastavil v počátečním bodě. Může se zdát, že by kolečko mělo být lepšího tvaru, to je ale opět způsobeno výpočtem polohy robota, která je navázána na ujetou vzdálenost, která působí tuto nepřesnost. Objetá kružnice tedy není nejhezčí, ale je správná. Robot ji totiž projel na pravé straně úplně vpravo a rovněž tak na levé straně. Dostal se ale o pár cm za hranici, proto se zdá být kružnice „bramborovitého“ tvaru.

[1] Pozn.: Pokud bychom měli opravdu malé měřítko mapy, při zatáčení by se stalo, že by se nám nevykreslila křivka, ale přímka z bodu minulé polohy do bodu aktuální polohy.

Jízda doprava

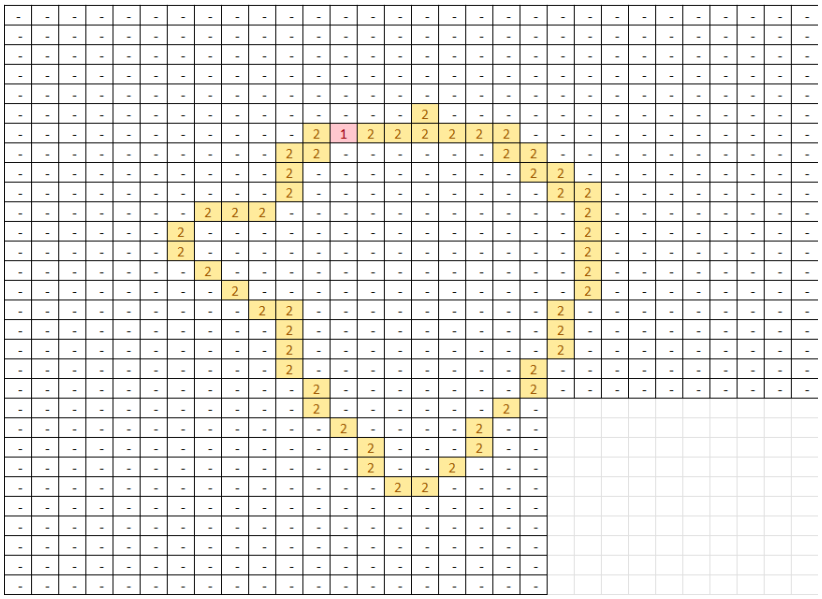
Abychom nedostali stejný výsledek, akorát na opačnou stranu, změňme zatáčení z maximálního na minimální. Tedy robot bude objíždět kružnici, která má poloměr - ne 17.5 cm - ale 65.3 cm. Ujetá vzdálenost robota bude 411 cm po zaokrouhlení nahoru.



Obrázek 5.4: 1. úkol - jízda doprava

Na obrázku můžeme vidět poněkud zajímavější a větší mapu. Mapování se zde zdá být docela přesné, s tímto výsledkem jsem spokojen a nemám moc co vytknout. Žlutě je zvýrazněna projetá trasa, červeně pak místo, kde se robot zastavil.

Až po napsání tohoto textu jsem si všiml, že jsem ponechal rozměry pole 5x5 cm. Namapovaná kružnice se mi ale poměrně líbí a tak ji tu nechám. Přiložím však mapu při zvolení 10x10 cm. Ta je zajímavá díky zřetelnému dynamickému mapování, kde se mapují pouze podstatné části a nezabírá se paměť zbytečnými daty.



Obrázek 5.5: 1. úkol - jízda doprava, pole = 10x10 cm

Kromě dynamického mapování, které funguje dle očekávání, si můžeme všimnout dalších dvou věcí, které dle očekávání naopak vůbec nejsou. Jednak se podívejme, kde robot po udělání kolečka skončil. To mělo být správně v nejvyšším bodě kružnice, ale robot popojel asi o 20 cm dál. Při testování skončil +- tam, kde začal, reálně. Mapování se ale počítá pouze z odometrie/kinematiky a nebere v potaz žádné okolní, reálné jevy. To znamená, že pokud někde v algoritmu nastane určitá chyba, propaguje se dál a velice pravděpodobně se v čase zvyšuje.

Druhou anomálií je výběžek vlevo.

5.2 2. úkol

Druhým úkolem bude rozšířené mapování skýtající využití laserových senzorů k mapování svého okolí.

Jízda rovně

Nejprve pro ověření správného mapování pusťme robota rovně podél zdi, která je vzdálena cca 25cm na levé straně a podívejme se na výsledek.

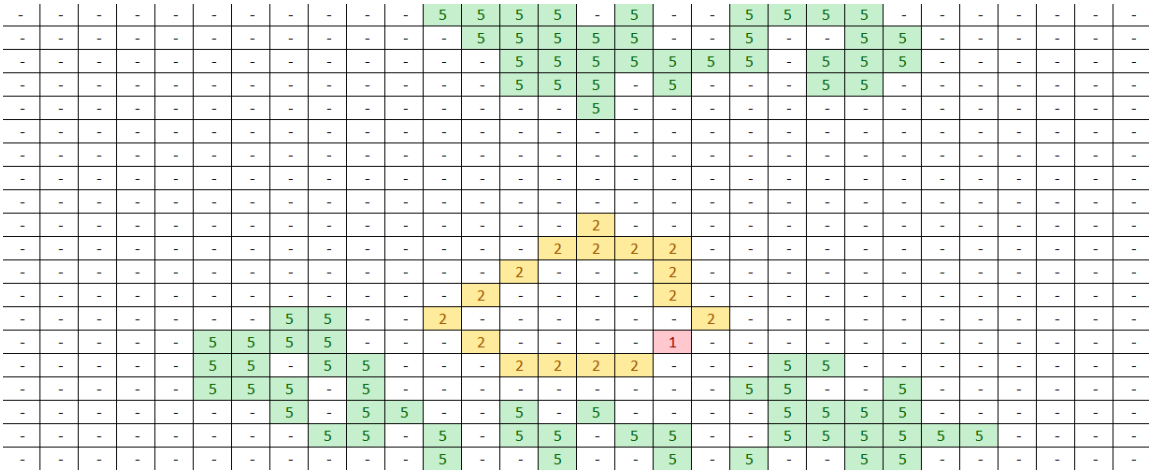
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	5	5	5	5	5	5	5	5	5	5	5	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	2	2	2	2	2	2	2	2	2	2	1	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Obrázek 5.6: 2. úkol - jízda rovně, pole = 10x10 cm

Výsledná mapa se zdá být bez chyby. Žádné odchylky v zachycení dat senzory, propočet zanesení překážky do mapy se také zdá být v pořádku. Toto byl ale nejlehčí úkol, bez propočtů směřujících uhlů a podobně 4.4.5. Nyní robota pustíme na chodbě, která má protilehlé zdi vzdáleny 3 metry a robotu zadáme, aby udělal kolečko.

Zatáčení

Robotovi nastavíme otočení přední nápravy na poslední možnost = střední, kde se poloměr zatáčení rovná 30.3 cm. Zatáčet bude doleva. Spusťme robota a podívejme se na výsledek.



Obrázek 5.7: 2. úkol - zatáčení, pole = 10x10 cm

Zde už je mapa o něčem jiném. Robota jsem pustil blíže jedné straně - pravé - a výsledek tomu také nasvědčuje. Mapování polohy robota už jsme probírali, zde pouze poukáži na fakt, že na mapě je konečná poloha za objetím kružnice, ale reálně se zastavil zhruba tam, kde začal.

Zajímavostí je zde ale mapování zdí. Na levé straně (z počáteční pozice robota - tedy nahoře) proběhlo poměrně uspokojivě. Překvapuje mne, že na pravé straně je více záznamů, než na straně levé. To může být způsobeno možným odrazem paprsku, nebo robot v daný moment propočítával něco jiného, čímž nenasbíral tolik dat, ze kterých by určil, že se tam vyskytuje překážka.

Na druhé straně, tedy pravé - spodní - je situace opačného charakteru. Záznamů překážky máme dost. Jak se robot točí, stává se často (jak je vidno), že zaznamenaní překážky spadá do různých bloků. Řekl bych, že se jedná o zaokrouhlovací chybu, díky které robot zanáší záznamy buď do jednoho bloku, či druhého. Pokud bychom ve finále zprůměrovali zanesená data, dostali bychom poměrně slušnou představu, kde se zeď reálně vyskytuje.

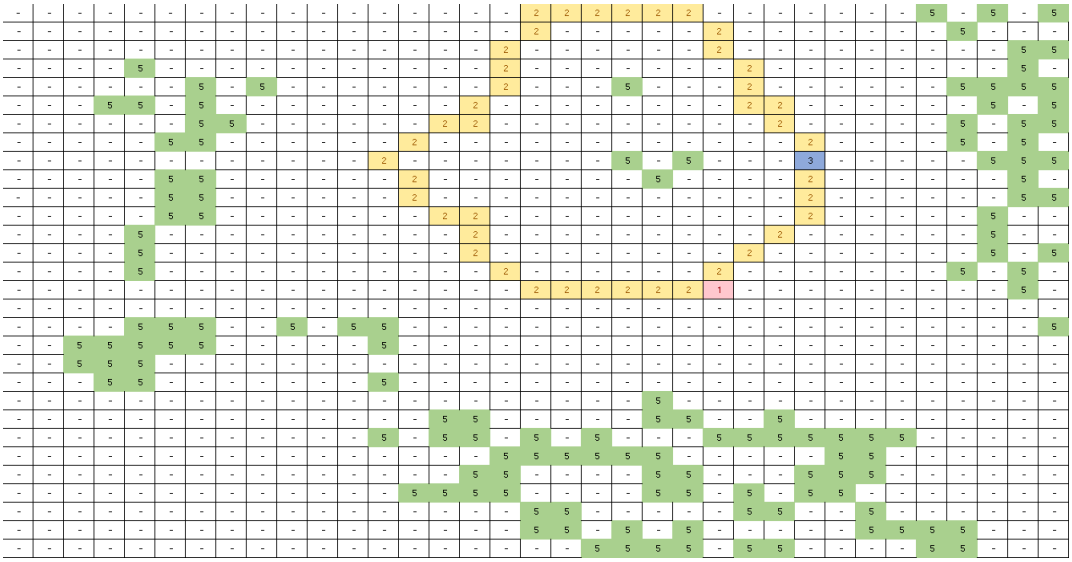
5.3 3. úkol

Nyní robotovi zadejme nejtěžší úkol, který skýtá veškeré podmínky. Těmi jsou:

- Jízda po okruhu = detekce jízdních pruhů
- Mapování pohybu robota
- Mapování překážek
- Detekce pohybu před robotem = využití IR senzoru

Pokud robot detekuje překážku před sebou, zastaví a zanese do mapy číslo 3 (které jsem přidal pouze pro toto testování, proto v kódu není k nalezení). Testuji v prostoru, kde jsou ze tří stran pevné překážky. Dráha má charakter oválné závodní tratě typu NASCAR[2], bez klopení. Délka rovinky je 80 cm a pak vždy následuje zatáčka.

[2]Závody NASCAR se konají na oválných tratích, většinou v USA, ale také v Kanadě a Mexiku. NASCAR závody jsou charakteristické vysokými rychlostmi a blízkými souboji mezi vozidly, což často vede k dramatickým a napínavým závodům.



Obrázek 5.8: 3. úkol - okruh, pole = 10x10 cm

Rozměr mapy je už poměrně velký, ale díky vyznačeným hodnotám v mapě je zřetelné, kudy robot projížděl (žlutá barva), kde skončil (červená buňka), kde zastavil, protože jsem mu vlezl do mapy (modrá buňka) a fixní překážky (zelené buňky).

Zde je mnoho zajímavého. Jednak jsem nesprávně odhadl délku okruhu, takže robot popojel o kus dál za cílovou rovinku, což je patrno i z mapy a nyní to poměrně sedí s ujetou vzdáleností.

Dalším zajímavým jevem je mapování překážek, když robot zastavil, aby předešel nehodě. V cestě napravo je vidět modrá buňka, kde jsem robotu vešel do cesty. Na té straně rovněž proběhlo vícero výpočtů překážek, než na straně druhé - tak si vysvětluji, že na pravé straně je vícero zanesených dat, než na straně levé.

Mezi levou a střední-spodní částí by mělo být *map_Empty*, neb jsou tam dveře, které byly otevřeny a tak bych očekával, že tam žádná překážka nebude. Je ale patrno, že kvantitativně je to zanedbatelné vůči ostatním shlukům.

Mezi pravou stranou a střední-spodní částí překážek je mezera - předpokládám že z důvodu rohu, který byl už moc vzdálený.

Nad okruhem (myšleno v mapě) není žádný záznam, což odpovídá realitě. Na té straně není žádná zeď, ani překážka, takže v tomto ohledu je to správně.

Posledním zajímavým jevem je detekce čehosi uprostřed mapy (a okruhu). To budu já, čekající, až robot dokoná svůj úkol, abych si vyzvedl data.

Kapitola 6

Závěr

V rámci této bakalářské práce byl navržen a sestaven autonomní model vozidla, který využívá metody SLAM. Byla provedena kompletní instalace všech potřebných technologií, včetně MCU a senzorů, a následně byly všechny komponenty propojeny a zprovozněny. V průběhu vývoje byl vytvořen řídicí algoritmus, který umožňuje vozidlu pohybovat se a sbírat data. Hlavním zaměřením bylo dynamické mapování pomocí SLAM metody grid-based, kde jsou udržována pouze podstatná data, což je vhodné pro roboty s omezenou pamětí.

Výsledky dosažené touto metodou jsou uspokojivé, leč by mohla být práce dále rozšířena např. o průměrování shluku dat a tím získání preciznější polohy fixní překážky. O kameru, díky čemu by měl robot představu, co se nachází před ním, což by umožnilo rychlejší průjezd trasou.

Literatura

[1] *GP2Y0A02YK0F*. E4-A00101EN. SHARP, leden 2006 [cit. 2023-04-19].

[2] *TCS3200, TCS3210 PROGRAMMABLE COLOR LIGHT-TO-FREQUENCY CONVERTER*. TCS3200. TAOS, červenec 2009 [cit. 2023-04-16].

[3] *Datasheet - Positive voltage regulator ICs*. DS0422. ST, říjen 2018 [cit. 2023-04-21].

[4] Admin. *Brushless DC (BLDC) Motor* [online]. electricalbaba, 2016 [cit. 2023-04-16]. Dostupné z: https://electricalbaba.com/brushless-dc-bldc-motor/#google_vignette.

[5] Admin. *Logický analyzátor* [online]. dratek, 2022 [cit. 2023-04-16]. Dostupné z: https://dratek.cz/arduino/2187-logicky-analyzator-spi-i2c-1-wire-can-dmx-512-hdlc-i2s-jtag-midi-modbus.html?gclid=CjwKCAjwue6hBhBVEiwA9YTx8NYE-6bvHbnNipDuYNUoNN3ZNeKCGehNk98mewkOEDZTJMexXh2rLhoCSggQAvD_BwE.

[6] Bartoš, V., Budina, M., Friedeckỳ, B., Kratochvíla, J., Springer, D. et al. Doporučení k vyjadřování nejistot kvantitativních vỳsledk měření ve zdravotnickỳch laboratořích. *Klinická biochemie a metabolismus*. 2021, s. 84, [cit. 2023-04-16].

[7] Ben Ari, M. a Mondada, F. Robotic Motion and Odometry. In:. Leden 2018, s. 63–93 [cit. 2023-04-15]. DOI: 10.1007/978-3-319-62533-1$_5$. $ISBN 978-3-319-62532-4$.

[8] Bhandari, P. *Normal Distribution | Examples, Formulas, Uses* [online]. 2020 [cit. 2023-05-05]. Dostupné z: https://www.scribbr.com/statistics/normal-distribution/.

[9] BLDC, M. B. T. P. Mikrokontrolérem řízenỳ regulátor třífázového BLDC motoru. [cit. 2023-04-16].

[10] Brejčák, P. *Student ČVUT vytvořil mozek autonomního robota, který může hrát významnou roli při záchranných akcích* [online]. cc, 2019 [cit. 2023-04-15]. Dostupné z: https://cc.cz/student-cvut-vytvoril-mozek-autonomniho-robota-ktery-muze-hrat-vyznamnou-roli-pri-zachrannych-akcich/.

[11] ChaN. *FatFs Module Application Note* [online]. 2022 [cit. 2023-02-05]. Dostupné z: http://elm-chan.org/fsw/ff/doc/appnote.html.

[12] DEE. *Nastavení podvozku X. (Ackermann)* [online]. casterfusion, 2011 [cit. 2023-04-12]. Dostupné z: http://www.casterfusion.funsite.cz/2011/11/nastaveni-podvozku-ackermann/.

[13] Dellaert, F., Fox, D., Burgard, W. a Thrun, S. Monte Carlo localization for mobile robots. In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*. 1999, sv. 2, s. 1322–1328 vol.2 [cit. 2023-04-15]. DOI: 10.1109/ROBOT.1999.772544.

[14] Eisele, R. *Obvod kruhu* [online]. xarg, 2023 [cit. 2023-04-13]. Dostupné z: https://www.xarg.org/book/kinematics/ackerman-steering/.

[15] flyability. *WHAT IS SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)?* [online]. flyability, 2022 [cit. 2023-04-12]. Dostupné z: https://www.flyability.com/simultaneous-localization-and-mapping.

[16] Freescale. *KL25 Sub-Family Reference Manual* [online]. Rev. 3, September 2012 [cit. 2023-10-02]. Dostupné z: https://spivey.oriel.ox.ac.uk/dswiki/images-digisys/5/56/KL25-refman.pdf.

[17] GÄRTNER, J. Provedení rešeršní studie v oblasti metod simultánní lokalizace a mapování. *Brno: Vysoké učení technické v Brně*. 2008, [cit. 2023-04-15].

[18] Horský Jiří, H. P. *Vyjadřování výsledků měření*. Časopis Elektro, 2010 [cit. 2023-04-16]. Dostupné z: http://www.odbornecasopisy.cz/res/pdf/42353.pdf.

[19] Hrbáček, J., Ripel, T. a Krejsa, J. Ackermann mobile robot chassis with independent rear wheel drives. In:. říjen 2010, s. T5–46. DOI: 10.1109/EPEPEMC.2010.5606853.

[20] Inc., T. *TCS3200, TCS3210 datasheet* [online]. 2009 [cit. 2023-25-01]. Dostupné z: https://www.mouser.com/catalog/specsheets/tcs3200-e11.pdf.

[21] Ing. Josef Strnadel, P. D. *Měření ve vybraných aplikacích*. Faculty of Information Technology, 2022.

[22] Ing. Josef Strnadel, P. D. *Úvod do měření*. Faculty of Information Technology, 2022.

[23] Jirka, I. V. *Zkušenosti s prvním nasazením polních robotů v ČR* [online]. agromanual, 2021 [cit. 2023-04-15]. Dostupné z: https://www.agromanual.cz/cz/clanky/technologie/precizni-zemedelstvi/zkusenosti-s-prvnim-nasazenim-polnich-robotu-v-cr.

[24] Joner, E. *How Brushless Motors Work & How to Test Them* [online]. tytorobotics, 2022 [cit. 2023-04-16]. Dostupné z: https://www.tytorobotics.com/blogs/articles/how-brushless-motors-work.

[25] Maulana, E., Muslim, M. A. a Zainuri, A. Inverse kinematics of a two-wheeled differential drive an autonomous mobile robot. In: *2014 Electrical Power, Electronics, Communicatons, Control and Informatics Seminar (EECCIS)*. 2014, s. 93–98 [cit. 2023-04-15]. DOI: 10.1109/EECCIS.2014.7003726.

[26] Milan Vašíček, J. G. *Roboti ve válce: Hrozba nebo naděje pro Evropu?* [online]. 2019.

[27] Neznámý. *Ultrasonic doc* [online]. - [cit. 2023-25-01]. Dostupné z: https://docs.m5stack.com/en/unit/sonic.

[28] neznámý. *Geometrie řízení Ackermann* [online]. hmn.wiki [cit. 2023-04-12]. Dostupné z: https://hmn.wiki/cs/Ackermann_steering_geometry.

[29] Neznámý. *Autonomní robotickou sekačku STIGA* [online]. stiga, 2022 [cit. 2023-04-15]. Dostupné z: https://www.stiga.com/cz/robot-sekacka-trava-caste-dotazy.

[30] pazout. *Obvod kruhu* [online]. geogebra [cit. 2023-04-13]. Dostupné z: https://www.geogebra.org/m/dVjkCvze.

[31] Saman, A. B. S. a Lotfy, A. An implementation of SLAM with extended Kalman filter. *2016 6th International Conference on Intelligent and Advanced Systems (ICIAS)*. 2016, s. 1–4.

[32] sdělení, N. K. *Autonomní roboti ušetří tisíce hodin neproduktivní skladové práce ročně* [online]. systemylogistiky, 2020 [cit. 2023-04-15]. Dostupné z: https://www.systemylogistiky.cz/2020/07/31/autonomni-roboti-usetri-tisice-hodin-neproduktivni-skladove-prace-rocne/.

[33] Singh, S. *WHAT IS SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)?* [online]. gizmochina, 2019 [cit. 2023-04-12]. Dostupné z: https://www.gizmochina.com/2019/02/21/amarey-a900-robot-vacuum-packing-visual-slam-technology-launching-soon/.

[34] Slouka, D. *Donášky jídla až do domů pomocí autonomních robotů: Ford a vize budoucnosti* [online]. insmart, 2019 [cit. 2023-04-15]. Dostupné z: https://insmart.cz/donasky-jidla-az-do-domu-pomoci-autonomnich-robotu/.

[35] STMICROELECTRONICS. *VL53L0X Datasheet* [online]. 2016 [cit. 2023-25-01]. Dostupné z: https://pdf1.alldatasheet.com/datasheet-pdf/view/948120/STMICROELECTRONICS/VL53L0X.html.

[36] Vdoleček, F., Palenčár, R. a Halaj, M. Nejistoty v měření I: vyjadřování nejistot. *Automa*. 2001, sv. 7, s. 50–54, [cit. 2023-04-16].

[37] Čápka, D. *Geometrie řízení Ackermann* [online]. itnetwork [cit. 2023-04-12]. Dostupné z: https://www.itnetwork.cz/maturitni-otazka-fyzika-kinematika.

Příloha A

Manual

Tato kapitola popisuje soubory obsažené na CD. Jak nainstalovat a nastavit prostředí, ve kterém se systém kompiluje, je popsáno v *README.txt*, které se nachází v kořenovém adresáři.

Složka s programem

Zdrojové kódy[1] jsou uloženy ve složce *Bachelor_thesis_xjahnf00*, přičemž struktura této složky je ukázána zde:

```
Bachelor_thesis_xjahnf00
├── CMSIS ..... SDK knihovny pro KL25Z
├── board ..... Konfigurace pinů, hodin atd.
│   └── pin_mux.h ..... Konfigurace pinů
├── common
│   ├── delay.h ..... Knihovna pro zpoždění
│   └── hash_table.h ..... Funkce pro práci s hashovací tabulkou
├── drivers ..... Vygenerované knihovny pro práci s periferiemi
├── map ..... Mapovací knihovna
│   ├── map_operations.h ..... Knihovní funkce pro práci s mapou
│   ├── mapping.h ..... Výpočty pohybu a dat, které se mají zanést do mapy
│   └── save_map.h ..... Knihovné funkce pro uložení mapy na SD kartu
├── motors ..... Řídící knihovna
│   └── engines.h
├── sd ..... Knihovna pro práci s SD kartou
├── source ..... Hlavní složka
│   ├── main.c
│   ├── control_unit.h ..... Řídicí logika
│   ├── global_macros.h ..... Všeobecná makra pro celý systém
│   ├── globalio.c ..... Nejdůležitější data pro běh systému
│   ├── interrupts.c ..... Veškerá přerušení
│   └── routine.h ..... Hlavní řídicí smyčka
├── startup ..... Složka pro inicializaci
│   ├── startup_board.h ..... Inicializace hodin, pinů, periferií, zapnutí přerušení
│   └── startup_peripherals.h ..... Inicializace senzorů, motorů
└── utilities ..... Knihovna pro ladicí výpisy
```

[1] Github repozitář je k nahlédnutí zde: https://github.com/Filda99/Bachelor_thesis.

Hlavní složka, kterou program začíná, je *source*. Ta skýtá *main*, kterým začíná vykonávání programu. Při inicializaci se přechází do složky *startup*. Další složky jsou oddělené "knihovny", které jsou, po přípustné minimální úpravě, samostatně použitelné. Byla snaha psát je způsobem, aby byly vyjmutelné a po vložení do jiného řešení fungovaly bez větších zásahů.

Ostatní

Zmíněné *README* je v kořenovém adresáři, stejně jako tato zpráva.

Zdrojové kódy technické zprávy, včetně obrázků atp., jsou umístěny ve složce *BP_Filip_Jahn.zip*[2].

Arduino projekt je vložen do složky s názvem *arduino*, která obsahuje pouze jeden soubor s příponou *.ino*. Ten po vložení do Arduino IDE studia je možno nahrát na arduino desku.

[2]Omlouvám se za kombinaci anglického a českého jazyka. Rovněž to nemám rád, avšak programy vždy píši anglicky, proto co se týče kódu, je v angličtině.